科技部『系統測試報告書』

System Testing Document
Of
NSC Project

雲端與端點應用之連續資安防禦與分析系統
MOST 106-2221-E-002 -009 -

研究團隊

主　持　人　：孫雅麗教授(臺灣大學資訊管理學系)

子計畫主持人：謝錫堃教授(成功大學電機工程學系)

郝方副教授(政治大學資訊管理學系)

文件版本修正履歷表

版次	變更項目	變更日期
1.1	初版	2018/05/10

總計畫與子計畫一：雲端應用暨 IoT 端點執行行為之資安分析與鑑識系統

(Cloud Applications and IoT Devices Execution Behavior: Security Analysis and Forensics System)

計畫主持人：孫雅麗 教授

國立臺灣大學 資訊管理學系

文件版本修正履歷表

版次	變更項目	變更日期
1.0	第一版	2018.05.10

1 簡介(Introduction)

(專案目的、專案範圍、測試目標)

本研究計畫的目標環境是在能利用動態側寫的行為分析分類結果，用來即時的偵測惡意程式的存在以及記錄其行為，並保護其個人電腦主機或是物聯網設備的安全，建立一個「雲端應用暨 IoT 端點執行行為之資安分析與鑑識系統」。此研究計畫成果適合惡意程式研究者、資安研究者或是企業組織內部的資訊安全使用，藉此達到既深且廣的資安研究或資安防禦。下圖為系統架構圖。

1.1.1 測試目的(Scope of Testing)

本系統的測試在系統整合前，必須先確認所有的設計元件均可正確的輸出，在此第一年我們著重於單元測試 (unit testing)。而第二年我們進行系統整合(system integration)，完成所有單元的實作以及整合，並進行惡意程式行為側寫的實驗。本文件內容將依據系統需求規格書與系統設計文件，描述單元測試以及系統整合與實驗的相關計畫與內容。並希望透過此文件之描述與實踐，達到順利進行測試與整合工作之目的。

1.1.2 接受準則(Acceptance Criteria)

本測試計畫需要滿足下列的測試接受準則：

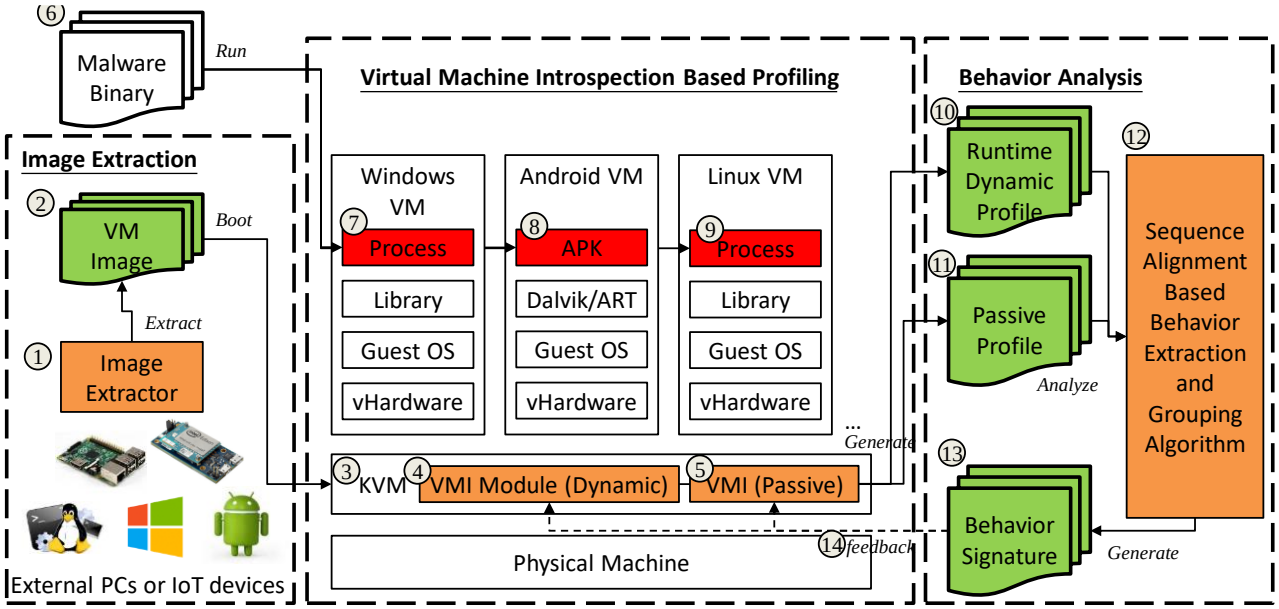
- 本系統需要對所有列為必要(Critical、Important、Desirable)之需求作完整測試。
- 測試程序需要依照本測試計畫所訂定的程序進行，所有測試結果需要能符合預期測試結果方能接受。

2 測試環境(Testing Environment)

2.1 子系統

2.1.1 運行環境(Operational Environment)

對於本系統進行系統測試階段之環境說明，請參考測試環境圖所示：



2.1.2 硬體規格(Hardware Specification)

依據測試環境圖內容，關於測試環境所需的硬體規格說明，如下列所示：

- 伺服器：Inter® i-Core™ 7 Quad CPU 3370S @ 3.1GHz
  - 支援 Intel VT-x 技術之處理器
  - RAM 為 8 GB
  - 硬碟空間為 500G 以上
  - 網路卡需包含 100/1000 Mbps 乙太網路卡
  - 網路協定：TCP/IP、HTTP、SSH、SMB
  - 以上硬體設備共四台。

2.1.3 軟體規格(Software Specification)

依據測試環境圖內容，關於測試環境所需的軟體規格說明，如下列所示：

- 用支援硬體加速之 KVM 及 QEMU 建構雲端平台
  - Ubuntu 12.04 LTS 64-bit
  - Linux Kernel 3.11.0.26
  - QENU/KVM
- 虛擬機器之作業系統與硬體配置：
  - 虛擬電腦配置 Microsoft Windows XP SP2 作業系統
  - Inter® i-Core™ 7 CPU 3370S @ 3.1GHz 單核心處理器
  - RAM 為 1 GB
  - 硬碟空間為 20G 以上
  - 網路卡需包含 100/1000 Mbps 乙太網路卡
  - 網路協定：TCP/IP、SMB
  - 特殊軟體：Wireshark 1.12 (Tshark)
  - 程式軟體：C、C++、Python、Linux Shell
  - 特殊通訊協定：QMP (QEMU Monitor Protocol)

2.1.4 測試資料來源(Test Data Source)

關於測試期間所需的測試資料來源及數量，因本測試系統所需之測試資料皆已自動化，自動化程式將於資料庫中一次取出惡意程式並以進行測試。測試資料來源如下表所示：

	正常程式對照組資料集	台灣大學資料集	國網中心資料集	北卡大學資料集
惡意程式數量	12	40	357	1260 (*980)
家族數	N/A	4	N/A	49 (*40)
程序或執行序數目	12	40	420	16301 (*9855)
資料蒐集日期	2011.11	2011.11	2009.08~2014.10	2010.08~2011.10

	國網中心資料集 II	Android Malware II	國網中心資料集 III
惡意程式數量	976	150 (*104)	1000
家族數	N/A	N/A	N/A
程序或執行序數目	420	N/A	N/A

資料蒐集日期	~ 2015.06.09	~ 2015.05.31	~ 2016.02.14

Year 3	國網中心資料集 IV	國網中心資料集 V	
惡意程式數量	10000	> 650 GB	
家族數	~100	N/A	
程序或執行序數目	~12000	N/A	
資料蒐集日期	~ 2016.03.02	~ 2018	

3 測試時程、程序和責任(Testing Schedule, Procedure, and Responsibility)

3.1 子系統

3.1.1 測試時程(Testing Schedule)

- 第一年時程

任務名稱	開始時間	完成時間
Image Extractor (Windows)	2017/07/01	2018/06/30
建立虛擬化執行環境	2017/07/01	2017/10/31
建立虛擬化平台	2017/07/01	2017/12/31
建立動態側寫模組	2017/11/01	2018/01/31
蒐集惡意程式	2017/07/01	2018/06/30
產生動態側寫檔案	2018/02/01	2018/06/30
進行行為分群演算	2018/04/01	2018/06/30

- 第一年查核點

任務名稱	查核時間
Image Extractor (Windows)	2018/06/30
建立虛擬化執行環境	2017/10/31
建立虛擬化平台	2017/12/31
建立動態側寫模組	2018/01/31
蒐集惡意程式	2018/06/30
產生動態側寫檔案	2018/06/30
進行行為分群演算	2018/06/30

3.1.2 測試程序(Testing Procedure)

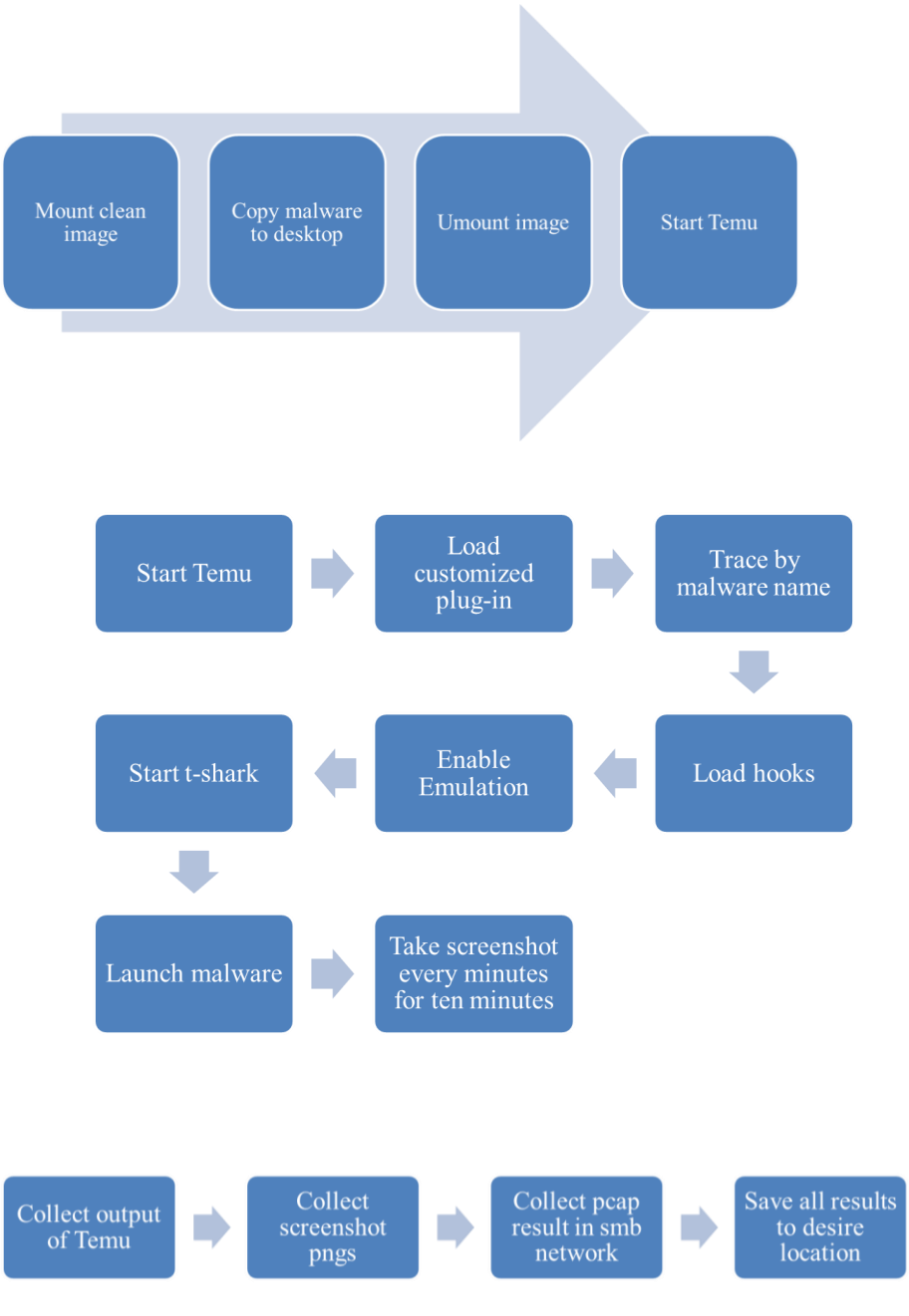
3.1.2.1 子系統驗證(Subsystem Validation)

各子系統之內部元件的測試，交由各子系統開發負責人完成，在此我們著重於本系統最後完成惡意程式自動化側寫的場景(scenario)進行整體測試。

3.1.2.2 整合測試(Integration Testing) (Incremental Testing)

(整合測試圖)

本自動化側寫系統共分三部分：(A) 準備階段、(B) 執行階段、(C) 資料收集階段。其執行流程圖如下所示。



圖二　　三階段整合系統測試

3.1.2.3 接受測試(Acceptance Testing)(Alpha Testing)

本系統須達下列功能，如下表所示：

- 功能性需求 (Functional Requirements)

需求編號	優先權	需求描述
PL1-FUR-001	1	本系統可監視惡意程式的程序執行行為。
PL1-FUR-002	1	本系統可監管惡意程式於磁碟的輸入/輸出行為。
PL1-FUR-003	1	本系統可監管惡意程式於網路的輸入/輸出行為。
PL1-FUR-004	1	本系統可監管惡意程式使用函式庫之行為。
PL1-FUR-005	1	本系統可追蹤惡意程式跨程序的執行行為。

需求編號	優先權	需求描述
PL1-FUR-006	1	本系統可產生惡意程式的側寫檔案(XML)
PL1-FUR-007	1	本系統可自動化側寫惡意程式
PL1-FUR-008	1	本系統可產生惡意程式的側寫檔案(hooklog)

需求編號	優先權	需求描述
PL1-FUR-011	1	本系統可自動產生 Common sequence
PL1-FUR-012	1	本系統可自動產生 Unique sequence
PL1-FUR-013	1	本系統可自動產生 Def-use chain
PL1-FUR-014	1	本系統可自動產生 Behavior group
PL1-FUR-015	1	本系統可自動產生行為樹圖

- 效能需求(Performance Requirement)

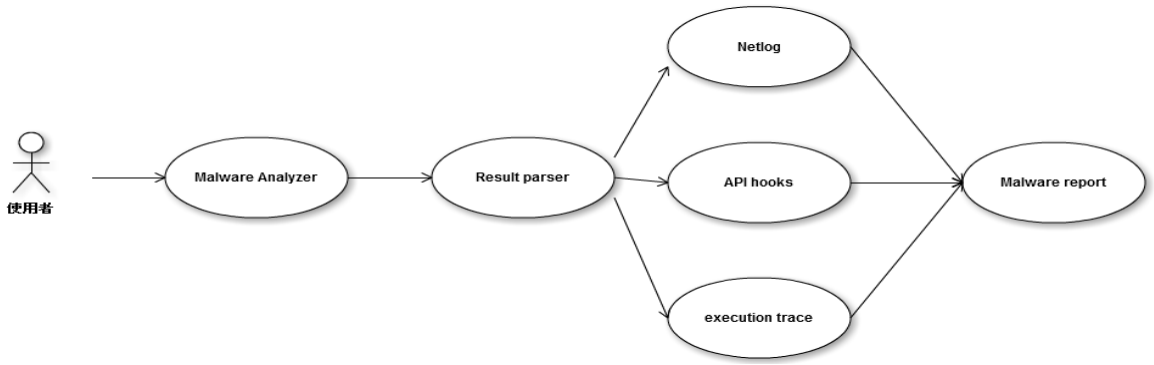
需求編號	優先權	用途描述
PL1-PER-001	4	當使用者執行輸入惡意程式執行碼功能時,分析及產生報告之執行時間需小於 10 分鐘。

需求編號	優先權	用途描述
PL1-PER-002	1	當使用者於網頁輸入惡意程式時,分析及產生報告之執行時間需小於 15 分鐘。

- 環境需求(Environment Requirement)

需求編號	優先權	用途描述
PL1-ENR-001	1	將使用支援硬體加速之 KVM 以建構雲端平台，。其上可以支援多組 Guest OSs，且彼此互不干擾。
PL1-ENR-002	1	預計在 QEMU 當中，攔截 I/O 相關的行為，同時支援檔案類型的汙點分析的功能。

針對本系統之需求，本系統設計時期之使用案例(Use Case)如下圖所示，本系統



圖三　　系統使用案例圖

3.1.3 人員職責分配(Personnel and Responsibility)

本系統之測試人員姓名及職責如下列所示。

表一　　人員職責分配表

成員名單與縮寫對照表					
縮寫	姓名	縮寫	姓名	縮寫	姓名
YL	孫雅麗	SW	蕭舜文	CY	薛筑允
HC	李鴻鈞	WC	邱偉志	WSW	吳尚偉
CH	彭証鴻	TY	陳廷易	ALL	全體人員

任務名稱	參與人員/負責人
Image Extractor (Windows)	CY, WC, HC / YL
建立虛擬化執行環境	CY, HC / YL
建立虛擬化平台	CY, HC / YL
建立動態側寫模組	CY, HC / YL
蒐集惡意程式	SW / YL
產生動態側寫檔案	CY / YL
進行行為分群演算	WC, SW, CH, TY, WSW / YL

4　測試案例(Test Case)

4.1　子系統

4.1.1 整合測試案例(Integration Testing Cases)(Incremental testing)

本計畫針對 Windows 惡意程式進行側寫，其實驗範例如下。本實驗的目的及為了驗證側寫程式之有效性，並且產出側寫檔案以做為第二年度的分析資料。

輸入：惡意程式可執行檔案

輸出：

惡意程式側寫檔案

執行呼叫紀錄

網路封包記錄檔

螢幕截圖共十張

XML側寫檔案

家族樹

Common sequence

Unique sequence

Def-use chain

Behavior group

行為樹圖

輸出一：側寫檔案

992 malware.exe
#213710000
RegQueryValue
hKey=HKEY_LOCAL_MACHINE\SYSTEM\Setup\SystemSetupInProgress

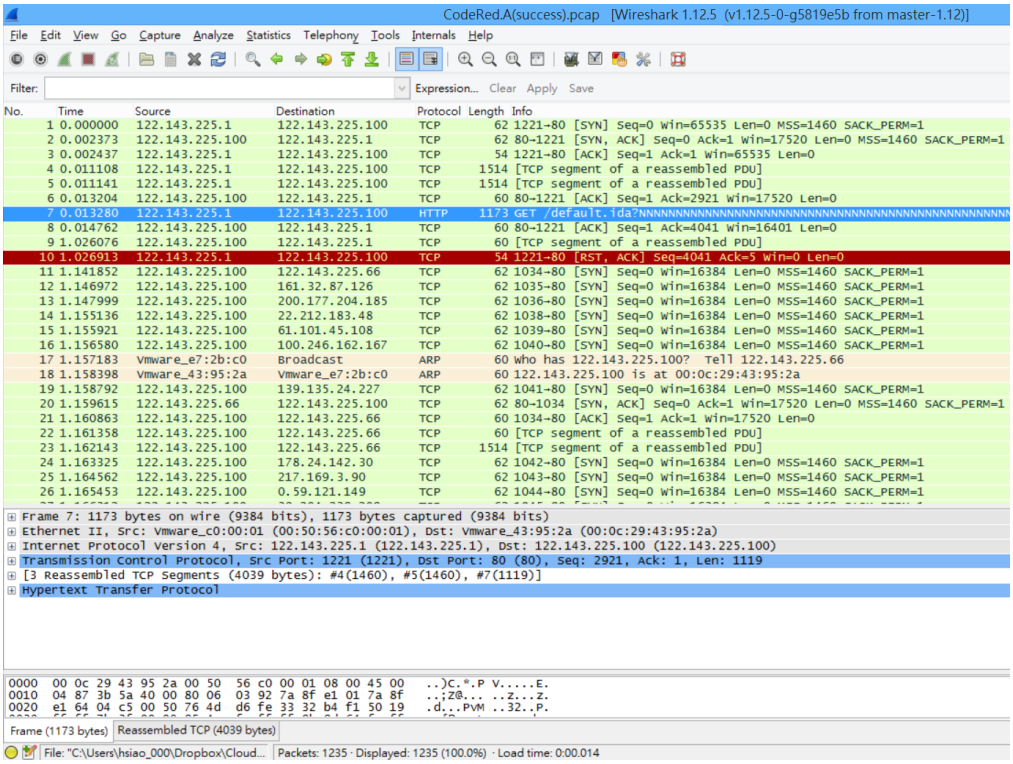
Return=SUCCESS
type=REG_DWORD
data=0
#214820000
CreateFile
hName=C:\WINDOWS\WindowsShell.Manifest
desiredAccess=GENERIC_READ
creationDisposition=OPEN_EXISTING
Return=SUCCESS
#215090000
RegQueryValue
hKey=HKEY_CURRENT_USER\Control Panel\Desktop\SmoothScroll
result=FAILURE
#215110000
RegQueryValue
hKey=HKEY_CURRENT_USER\software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\EnableBalloonTips
result=FAILURE
#214040000
LoadLibrary
lpFileName=comctl32.dll
Return=SUCCESS
#216050000
RegQueryValue
hKey=HKEY_CURRENT_USER\Control Panel\Desktop\SmoothScroll
result=FAILURE

輸出二：執行呼叫紀錄

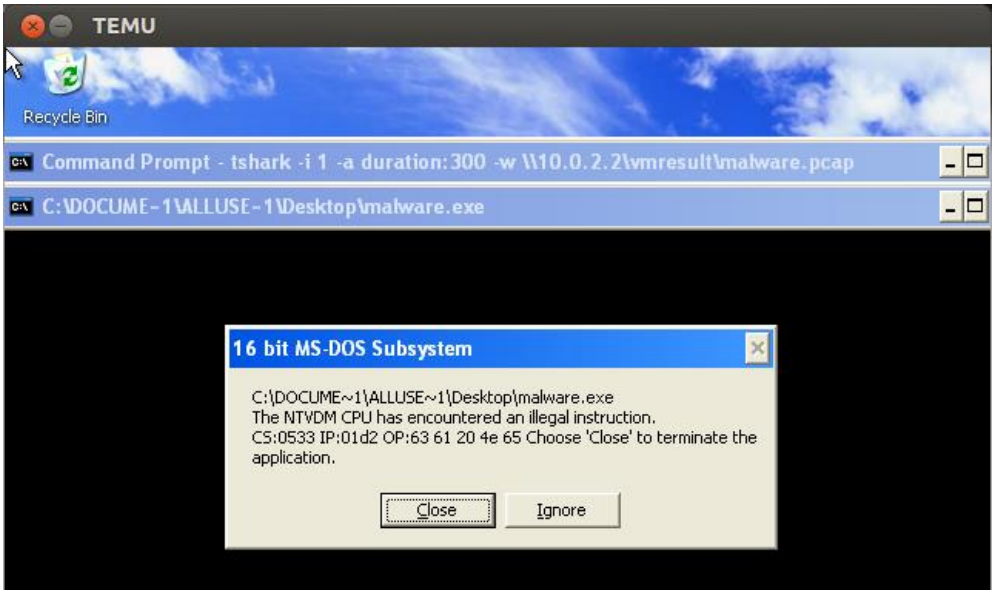
Process 1360 TID: 1340 -> ntdll.dll::RtlImageDirectoryEntryToData @ EIP: 0x7c910326
Process 1360 TID: 1340 -> ntdll.dll::RtlImageNtHeader @ EIP: 0x7c910319
Process 1360 TID: 1340 -> ntdll.dll::RtlAllocateHeap @ EIP: 0x7c9100a4
Process 1360 TID: 1340 -> ntdll.dll::RtlEnterCriticalSection @ EIP: 0x7c901000
Process 1360 TID: 1340 -> ntdll.dll::RtlLeaveCriticalSection @ EIP: 0x7c9010e0
Process 1360 TID: 1340 -> ntdll.dll::RtlImageNtHeader @ EIP: 0x7c910319
Process 1360 TID: 1340 -> ntdll.dll::RtlUpcaseUnicodeChar @ EIP: 0x7c9103c0
Process 1360 TID: 1340 -> ntdll.dll::ZwClose @ EIP: 0x7c90cfd0
Process 1360 TID: 1340 -> ntdll.dll::KiFastSystemCall @ EIP: 0x7c90e4f0
Process 1360 TID: 1340 -> ntdll.dll::KiFastSystemCallRet @ EIP: 0x7c90e4f4
Process 1360 TID: 1340 -> ntdll.dll::RtlInitUnicodeString @ EIP: 0x7c901295
Process 1360 TID: 1340 -> ntdll.dll::RtlEqualUnicodeString @ EIP: 0x7c912e9b
Process 1360 TID: 1340 -> ntdll.dll::RtlGetActiveActivationContext @ EIP: 0x7c91c5ab
Process 1360 TID: 1340 -> ntdll.dll::RtlActivateActivationContextUnsafeFast @ EIP: 0x7c901198
Process 1360 TID: 1340 -> ntdll.dll::RtlImageDirectoryEntryToData @ EIP: 0x7c910326
Process 1360 TID: 1340 -> ntdll.dll::RtlImageDirectoryEntryToData @ EIP: 0x7c910326
Process 1360 TID: 1340 -> ntdll.dll::RtlInitAnsiString @ EIP: 0x7c90125d
Process 1360 TID: 1340 -> ntdll.dll::RtlAnsiStringToUnicodeString @ EIP: 0x7c90eb1b
Process 1360 TID: 1340 -> ntdll.dll::RtlMultiByteToUnicodeN @ EIP: 0x7c90ec9a
Process 1360 TID: 1340 -> ntdll.dll::strchr @ EIP: 0x7c90e7ed
Process 1360 TID: 1340 -> ntdll.dll::RtlInitUnicodeString @ EIP: 0x7c901295
Process 1360 TID: 1340 -> ntdll.dll::RtlDosApplyFileIsolationRedirection_Ustr @ EIP: 0x7c91597b
Process 1360 TID: 1340 -> ntdll.dll::RtlFindCharInUnicodeString @ EIP: 0x7c915d41
Process 1360 TID: 1340 -> ntdll.dll::RtlValidateUnicodeString @ EIP: 0x7c915e4a
Process 1360 TID: 1340 -> ntdll.dll::RtlValidateUnicodeString @ EIP: 0x7c915e4a
Process 1360 TID: 1340 -> ntdll.dll::RtlFreeUnicodeString @ EIP: 0x7c910446
Process 1360 TID: 1340 -> ntdll.dll::RtlFindActivationContextSectionString @ EIP: 0x7c9154f1
Process 1360 TID: 1340 -> ntdll.dll::bsearch @ EIP: 0x7c9151d3
Process 1360 TID: 1340 -> ntdll.dll::RtlHashUnicodeString @ EIP: 0x7c91563d
Process 1360 TID: 1340 -> ntdll.dll::bsearch @ EIP: 0x7c9151d3
Process 1360 TID: 1340 -> ntdll.dll::bsearch @ EIP: 0x7c9151d3
Process 1360 TID: 1340 -> ntdll.dll::RtlAddRefActivationContext @ EIP: 0x7c90fbb2
Process 1360 TID: 1340 -> ntdll.dll::RtlFreeUnicodeString @ EIP: 0x7c910446
Process 1360 TID: 1340 -> ntdll.dll::RtlUpcaseUnicodeChar @ EIP: 0x7c9103c0
Process 1360 TID: 1340 -> ntdll.dll::RtlEqualUnicodeString @ EIP: 0x7c912e9b

…

輸出三：網路封包記錄檔



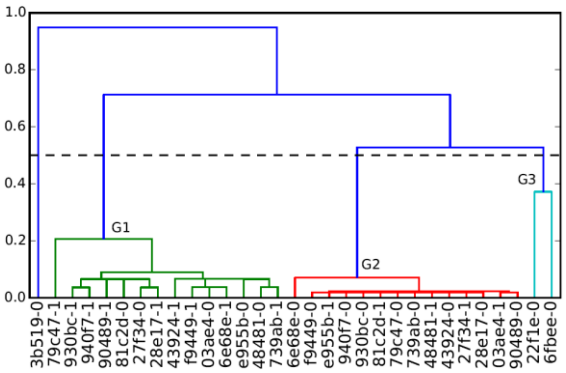
輸出四：螢幕截圖



輸出五：XML側寫檔案

<profile>
<meta>
<hash_id>03ae42959b733f11a168e268</hash_id>
<process_id>1264</process>
<duration>300</duration>
</meta>
<execution>
<CreateFile hName="C:\WINDOWS\WindowsShell.Manifest" desiredAccess="GENERIC_READ"
creationDisposition="OPEN_EXISTING" Return="SUCCESS" Time="347220000" />
<RegQuery ValuehKey="HKEY_CURRENT_USER\Control Panel\Desktop\SmoothScroll" result="FAILURE"
Time="347500000" />
<LoadLibrary lpFileName="comctl32.dll" Return="SUCCESS" Time="348300000" />
<OpenProcess desiredAccess="PROCESS_DUP_HANDLE" dwProcessId="1896" Return="SUCCESS"
procName="explorer.exe" Time="706440000" />
...
</execution>
</profile>

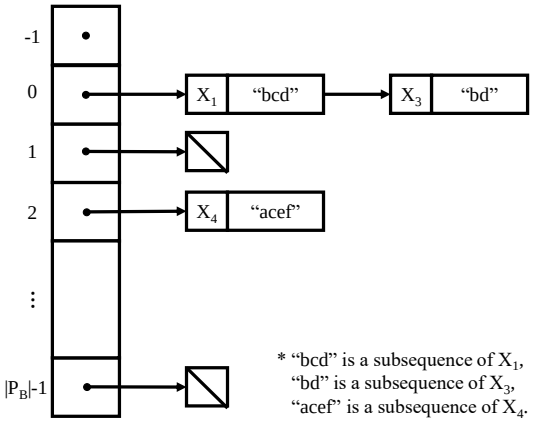
輸出六：家族樹



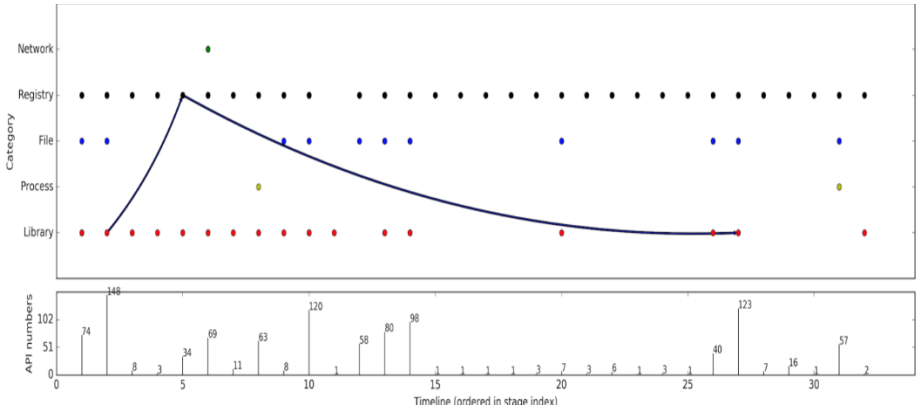
輸出七：Common Sequence

	G_1	G_3	...	$G_{|G|}$
P_B	“ab”	“”		
P_{X1}	“ab”	“”		
P_{X2}	“cd”	“ghi”		
⋮				
P_{XN-1}	“ef”	“gh”		

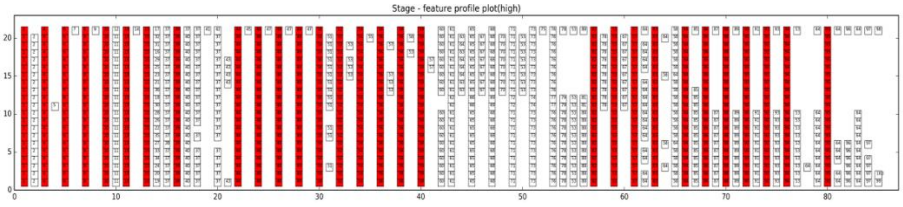
輸出八：Unique Sequence



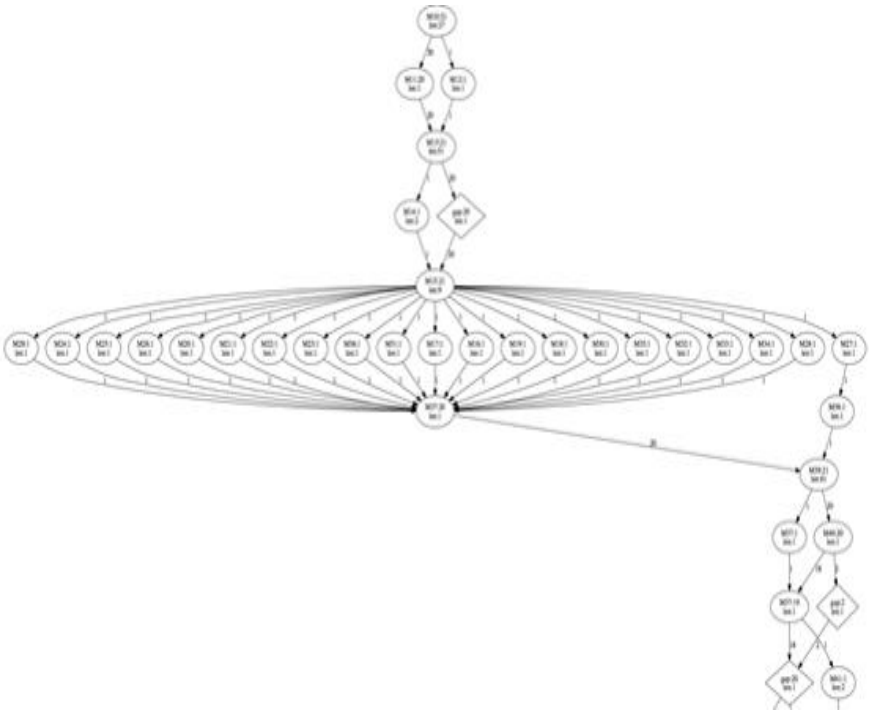
輸出九： Def-use chain



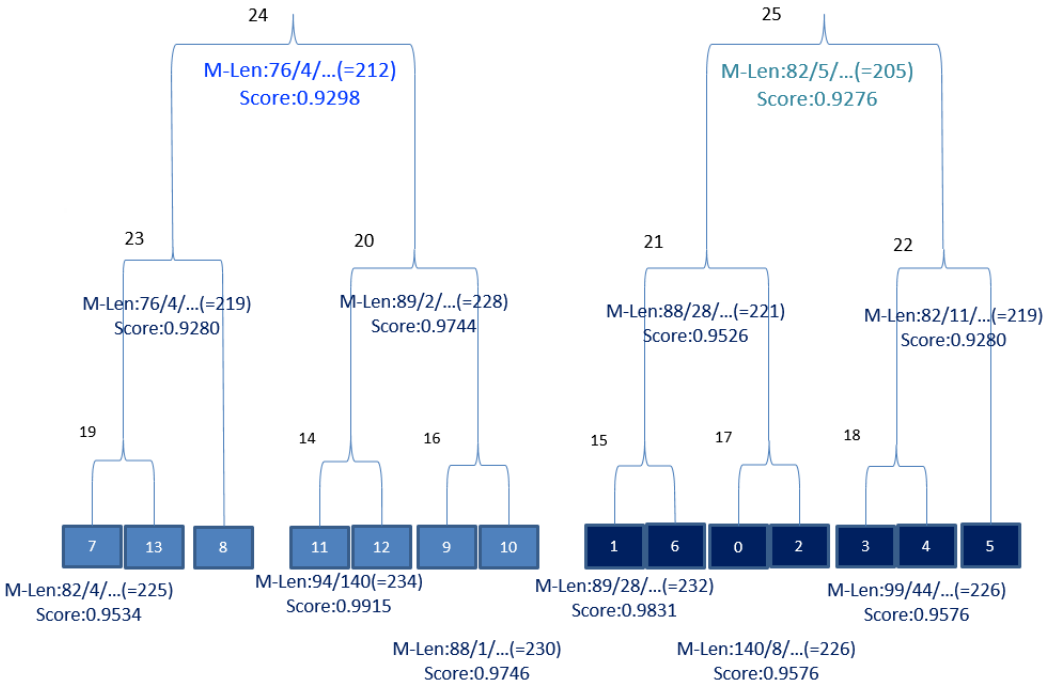
輸出十：Behavior group



輸出十一：行為樹圖



輸出十二：RASMMA Tree



5 測試結果與分析(Test Result and Analysis)

5.1 子系統

5.1.1 整合子系統測試案例(Integration Testing Cases)

- 功能性需求 (Functional Requirements)

需求編號	優先權	需求描述	結果
PL1-FUR-001	1	本系統可監視惡意程式的程序執行行為。	V
PL1-FUR-002	1	本系統可監管惡意程式於磁碟的輸入/輸出行為。	V
PL1-FUR-003	1	本系統可監管惡意程式於網路的輸入/輸出行為。	V
PL1-FUR-004	1	本系統可監管惡意程式使用函式庫之行為。	V
PL1-FUR-005	1	本系統可追蹤惡意程式跨程序的執行行為。	V

5.1.2 接受測試案例(Acceptance Testing Cases)

需求編號	優先權	需求描述	結果
PL1-FUR-006	1	本系統可產生惡意程式的側寫檔案(XML)	V
PL1-FUR-007	1	本系統可自動化側寫惡意程式	V
PL1-FUR-008	1	本系統可產生惡意程式的側寫檔案(hooklog)	V

需求編號	優先權	需求描述	結果
PL1-FUR-011	1	本系統可自動產生 Common sequence	V
PL1-FUR-012	1	本系統可自動產生 Unique sequence	V
PL1-FUR-013	1	本系統可自動產生 Def-use chain	V
PL1-FUR-014	1	本系統可自動產生 Behavior group	V
PL1-FUR-015	1	本系統可自動產生行為樹圖	V

- 效能需求(Performance Requirement)

需求編號	優先權	用途描述	結果
PL1-PER-001	4	當使用者執行輸入惡意程式執行碼功能時，分析及產生報告之執行時間需小於 10 分鐘。	V

需求編號	優先權	用途描述	結果
PL1-PER-002	1	當使用者於網頁輸入惡意程式時，分析及產生報告之執行時間需小於 15 分鐘。	V

- 環境需求(Environment Requirement)

需求編號	優先權	用途描述	結果
PL1-ENR-001	1	將使用支援硬體加速之 KVM 以建構雲端平台，。其上可以支援多組 Guest OSs，且彼此互不干擾。	V
PL1-ENR-002	1	預計在 QEMU 當中，攔截 I/O 相關的行為，同時支援檔案類型的汙點分析的功能。	V

6 子計畫

6.1 子系統 vs. 測試案例(Subsystem vs. Test Cases)

需求編號	優先權	用途描述	環境測試結果
PL1-ENR-001	1	將使用支援硬體加速之 KVM 以建構雲端平台，。其上可以支援多組 Guest OSs，且彼此互不干擾。	V
PL1-ENR-002	1	預計在 QEMU 當中，攔截 I/O 相關的行為，同時支援檔案類型的汙點分析的功能。	V

6.2 接受度測試結果(Acceptance Testing Cases)

需求編號	優先權	需求描述	功能性測試結果
PL1-FUR-001	1	本系統可監視惡意程式的程序執行行為。	V
PL1-FUR-002	1	本系統可監管惡意程式於磁碟的輸入/輸出行為。	V
PL1-FUR-003	1	本系統可監管惡意程式於網路的輸入/輸出行為。	V
PL1-FUR-004	1	本系統可監管惡意程式使用函式庫之行為。	V
PL1-FUR-005	1	本系統可追蹤惡意程式跨程序的執行行為。	V
PL1-FUR-006	1	本系統可產生惡意程式的側寫檔案(XML)	V
PL1-FUR-007	1	本系統可自動化側寫惡意程式	V
PL1-FUR-008	1	本系統可自動產生惡意程式的家族樹	V

需求編號	優先權	需求描述	功能性測試結果
PL1-FUR-011	1	本系統可自動產生 Common sequence	V
PL1-FUR-012	1	本系統可自動產生 Unique sequence	V
PL1-FUR-013	1	本系統可自動產生 Def-use chain	V
PL1-FUR-014	1	本系統可自動產生 Behavior group	V
PL1-FUR-015	1	本系統可自動產生行為樹圖	V

需求編號	優先權	用途描述	效能測試結果
PL1-PER-001	4	當使用者執行輸入惡意程式執行碼功能時，分析及產生報告之執行時間需小於 10 分鐘。	V

需求編號	優先權	用途描述	效能測試結果

PL1-PER-002	1	當使用者於網頁輸入惡意程式時，分析及產生報告之執行時間需小於 15 分鐘。	V

Appendix：Glossary

(名詞) (解釋)

VMI （Virtual Machine Introspection）虛擬機器內省技術係指在虛擬機器外部利用 hypervisor 或 emulator 可以操控虛擬硬體之管理功能。用以解析低階硬體資料並重新橋接 semantic gap 導出具有高階意涵的資訊，而利用高階資訊進行有價值的應用 (例如：資安檢測、鑑識分析、稽核控管...等)。

Appendix C：Reference

[1] Stephanie Forrest et al., “A Sense of Self for Unix Processes,” in Proc. IEEE Symposium on Security and Privacy (S&P), May 1996, pp. 120-128.

[2] Heng Yin, Dawn Song, Manuel Egele, Christopher Kruegel, and Engin Kirda, ”Capturing System-Wide Information Flow for Malware Detection and Analysis,” in Proc. ACM Conference on Computer and Communications Security (CCS), 2007, pp. 116-127.

[3] Engin Kirda, Christopher Kruegel, Greg Banks, Giovanni Vigna, and Richard A. Kemmerer, “Behavior-Based Spyware Detection,” in Proc. USENIX Security Symposium, 2006, pp. 273-288.

[4] David Wagner and Drew Dean, “Intrusion Detection via Static Analysis,” in Proc. IEEE Symposium on Security and Privacy (S&P), May 2001, pp. 156-168.

[5] S. Kumar, Classification and Detection of Computer Intrusions, PhD thesis, Department of Computer Sciences, Purdue University, August 1995.

[6] R. A. Kemmerer and G. Vigna, “Intrusion Detection: A Brief History and Overview,” Computer, vol. 35, no. 4, pp. 27-30, April 2002.

[7] P. Garcia-Teodoro, J. Diaz-Verdejoa, G. Macia-Fernandeza and E. Vazquez, “Anomaly-based Network Intrusion Detection: Techniques, Systems and Challenges,” Computers & Security, vol. 28, pp. 18-28, 2009.

[8] D. E. Denning, “An Intrusion Detection Model,” IEEE Transactions on Software Engineering, vol. 13, no. 2, pp. 222-232, Feb. 1987.

[9] S. A. Hofmeyr, S. Forrest, and A. Somayaji, “Intrusion Detection using Sequences of System Calls,” Journal of Computer Security, vol. 6, pp. 155–180, 1998.

[10] Wenke Lee and Salvatore J. Stolfo, “Data Mining Approaches for Intrusion Detection,” in Proc. USENIX Security Symposium, July 1998.

[11] C. Kruegel, D. Mutz, F. Valeur, and G. Vigna, “On the Detection of Anomalous System Call Arguments,” in Proc. European Symposium on Research in Computer Security (ESORICS’03), 2003, pp. 101-118.

[12] Ulrich Bayer, Christopher Kruegel, and Engin Kirda, “TTAnalyze: A Tool for Analyzing

Malware," in Proc. European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference, April 2006.
[13] C. Willems et al., "Toward Automated Dynamic Malware Analysis Using CWSandbox," IEEE Security & Privacy, vol. 5, no. 2, pp. 32--39, 2007.
[14] L. Liu et al., "BotTracer: Execution-Based Bot-Like Malware Detection," in Proc. Int. Conf. on Information Security (ISC), 2008, pp. 97-113.
[15] Shun-Wen Hsiao, Yi-Ning Chen, Yeali S. Sun, and Meng Chang Chen, "A Cooperative Botnet Profiling and Detection in Virtualized Environment," in Proc. IEEE Conference on Communications and Network Security (IEEE CNS), Washington, D.C., Oct. 2013.
[16] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda, "Scalable, Behavior-Based Malware Clustering," in Proc. Network and Distributed System Security Symposium (NDSS), 2009.
[17] M. Christodorescu, Somesh Jha, and Christopher Kruegel, "Mining Specifications of Malicious Behavior," in Proc. India Software Engineering Conference (ISEC), 2008, pp. 5-14.
[18] Konrad Rieck et al., "Learning and Classification of Malware Behavior," in Proc. International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA), 2008, pp. 108-125.
[19] James Newsome and Dawn Song "Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software," in Proc. Network and Distributed System Security Symposium (NDSS), Feb. 2005.
[20] R. Sekar et al., "Specification-Based Anomaly Detection: a New Approach for Detecting Network Intrusions," in Proc. ACM Conference on Computer and Communications Security (CCS), 2002, pp. 265–274.
[21] G. Vigna and R. A. Kemmerer, "NetSTAT: A Network-Based Intrusion Detection Approach," Journal of Computer Security, vol. 7, no. 1, pp. 37–71, 1999.
[22] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated Generation and Analysis of Attack Graphs," in Proc. IEEE Symposium on Security and Privacy (S&P), Oakland, CA, May 9 2002, pp. 273-284.
[23] Clemens Kolbitsch et al., "Effective and Efficient Malware Detection at the End Host," in Proc. USENIX Security Symposium, 2009, pp. 351–366.
[24] Tal Garfinkel and Mendel Rosenblum, "A Virtual Machine Introspection Based Architecture for Intrusion Detection," in Proc. Network and Distributed Systems Security Symposium (NDSS), 2003, pp. 191-206.
[25] X. Jiang et al., "Stealthy Malware Detection through VMM-based 'out-of-the-box' Semantic View Reconstruction," in Proc. ACM conference on Computer and Communications Security (CCS), 2007, pp. 128-138.
[26] Bryan D. Payne, "Simplifying Virtual Machine Introspection Using LibVMI," Sandia National Laboratories, Albuquerque, NM and Livermore, CA, Tech. Rep. SAND2012-7818, Sept. 2012.
[27] Lok-Kwong Yan and Heng Yin, "DroidScope: Seamlessly Reconstructing OS and Dalvik Semantic Views for Dynamic Android Malware Analysis," in Proc. USENIX Security Symposium, Aug. 2012.
[28] Peter M. Chen and Brian D. Noble, "When virtual is better than real," in Proc. 8th Workshop on

Hot Topics in Operating Systems (HotOS), May 2001, pp. 133-138.
[29] G. W. Dunlap et al., “ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay,” in Proc. Symposium on Operating Systems Design and Implementation (OSDI), 2002, pp. 211-224.
[30] Fabrice Bellard, “QEMU, a Fast and Portable Dynamic Translator,” in Proc. USENIX Annual Technical Conference, 2005, pp. 41-46.
[31] Paul Barham et al. “Xen and the Art of Virtualization,” in Proc. ACM Symposium on Operating Systems Principles (SOSP), 2003, pp. 164-177.
[32] Avi Kivity, “kvm: the Linux Virtual Machine Monitor,” in Proc. Linux Symposium, July 2007, pp. 225-230.
[33] Android Emulator, http://developer.android.com/tools/devices/emulator.html
[34] Dawn Song et al., “BitBlaze: A New Approach to Computer Security via Binary Analysis,” in Proc. International Conference on Information Systems Security (ICISS), 2008, pp. 1-25.
[35] Bryan D. Payne, Martim D. P. de A. Carbone and Wenke Lee, “Secure and Flexible Monitoring of Virtual Machines,” in Proc. Computer Security Applications Conference (ACSAC), Dec. 2007, pp. 385–397.
[36] Bryan D. Payne et al., “Lares: An Architecture for Secure Active Monitoring Using Virtualization,” in Proc. IEEE Symposium on Security and Privacy (S&P), May 2008, pp. 233–247.
[37] Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee, “Ether: Malware Analysis via Hardware Virtualization Extensions,” in Proc. ACM Conference on Computer and Communications Security (CCS), 2008, pp. 51–62.
[38] Lok-Kwong Yan and Heng Yin, “DroidScope: Seamlessly Reconstructing OS and Dalvik Semantic Views for Dynamic Android Malware Analysis,” in Proc. USENIX Security Symposium, Aug. 2012.
[39] J. Pfoh, C. Schneider, and C. Eckert, “Nitro: hardware-based system call tracing for virtual machines,” in *Proc. the 6th International Conference on Advances in Information and Computer Security (IWSEC’11*), Springer-Verlag, Berlin, Heidelberg, pp. 96-112, 2011.
[40] Sharif, MI, Lee, W, Cui, W, Lanzi, A, “Secure in-vm monitoring using hardware virtualization,” in *Proc. the 16th ACM conference on Computer and communications security(CCS ’09*), ACM, New York, NY, USA, pp. 477-487, 2009.
[41] Dolan-Gavitt, B, Leek, T, Zhivich, M, Giffin, J, Lee, W, “Virtuoso: narrowing the semantic gap in virtual machine introspection, in *Proc. the 2011 IEEE Symposium on Security and Privacy (SP ’11),* IEEE Computer Society, Washington, DC, USA, pp. 297-312, 2011.

子計畫二：跨區域近即時通用型殭屍網路聯偵平台
(Cross-Regional Multivariated Botnet Detection Cloud)

子計畫主持人：謝錫堃教授

國立成功大學 電機工程學系

文件版本修正履歷表

版次	變更項目	變更日期
1.0	第一版	2018.04.27

5 簡介(Introduction)

(專案目的、專案範圍、測試目標)

一個基於 Flink 平台並使用串流方式的 P2P 殭屍網路近即時偵測系統，專案的主要目的在於攻擊前的潛伏階段偵測出殭屍網路，以期達到快速防範與預警之目標。

本次測試將依據本系統下列各項子系統，進行測試：

- 預處理子系統(Pre-processing)
- 會話合成子系統(Session Merging)
- 快篩子系統(Quick Detection)

1.1.3 測試目的(Scope of Testing)

本文件主要是在網路環境中建置近即時 P2P 殭屍網路偵測系統(P2P Botnet Quick Detection)，以下簡稱本系統的測試計畫。必須確認所有的設計元件均可正確的輸出，在此我們著重於子系統驗證(Subsystem Validation)及整合系統測試(Integration Test)。本文件內容將依據系統需求規格書與系統設計文件，描述關於整合測試的相關計畫與內容。並希望透過此文件之描述與實踐，達到順利進行測試工作之目的。

1.1.4 接受準則(Acceptance Criteria)

本測試計畫需要滿足下列的測試接受準則：

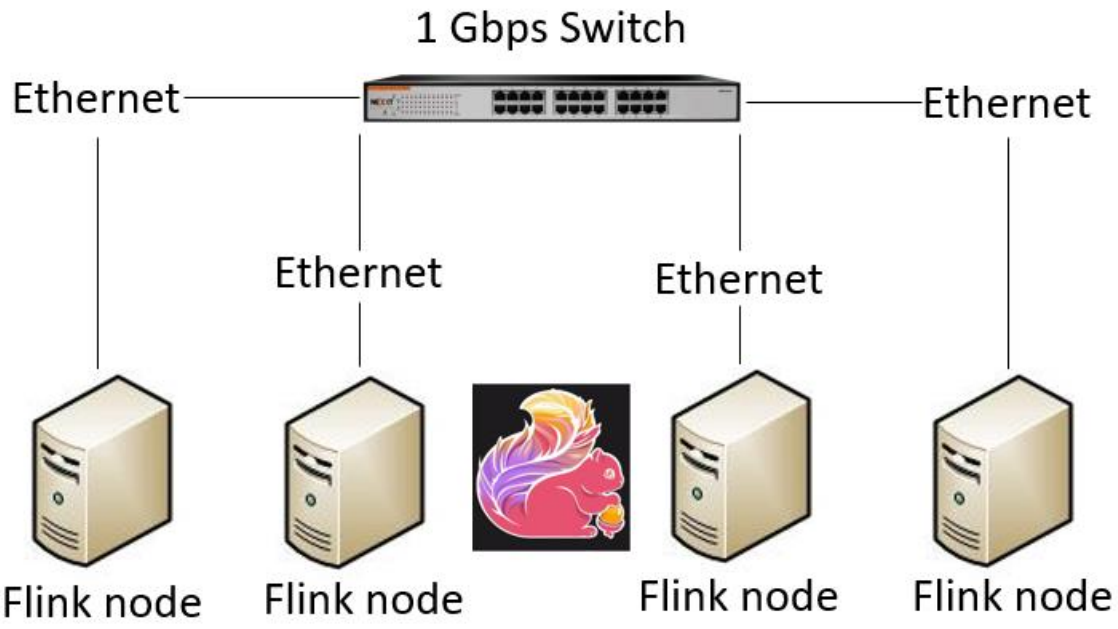
- 本系統需要對所有列為必要(Critical、Important、Desirable)之需求作完整測試。
- 測試程序需要依照本測試計畫所訂定的程序進行，所有測試結果需要能符合預期測試結果方能接受。

6 測試環境(Testing Environment)

6.1 子系統

6.1.1 運行環境(Operational Environment)

對於本系統進行系統測試階段之環境說明，請參考測試環境圖所示：



6.1.2 硬體規格(Hardware Specification)

依據測試環境圖內容，關於測試環境所需的硬體規格說明，如下列所示：

- 伺服器：Intel® Xeon® E3-1240 3.30GHz，128KiB Level 1，1MiB Level 2，8MiB Level 3 快取，記憶體為 16GiB，硬碟空間 1TB，網路卡 Intel® 82574L Gigabit Ethernet Controller。伺服器數量 4 部，網路速度 1Gbps。

6.1.3 軟體規格(Software Specification)

依據測試環境圖內容，關於測試環境所需的軟體規格說明，如下列所示：

- 作業系統：

  伺服器需配置 Ubuntu 14.04。

  個人電腦需配置 Microsoft Windows 10 作業系統。
- JAVA 版本：

  JAVA Runtime Environment 8 (JRE 8)
- Flink 版本：

  1.3.0

6.1.4 測試資料來源(Test Data Source)

關於測試期間所需的測試資料來源及數量 ，說明如下：(文字或表格)

1. 國家高速網路中心提供成大網路流量 Log – 105/07/01

7 測試時程、程序和責任(Testing Schedule, Procedure, and Responsibility)

7.1 子系統

7.1.1 測試時程(Testing Schedule)

- 時程
  - 各子系統之內部元件整合測試 (Module Test) (106/9/14~107/3/8)
  - P2P Botnet Quick Detection 系統整合測試 (Integration Test) (107/3/13~107/3/29)
  - P2P Botnet Quick Detection 系統接受度測試 (Acceptance Test) (107/4/4~107/4/13)
- 查核點
  - 各子系統之內部元件整合測試(107/3/11)
  - P2P Botnet Quick Detection 系統整合測試(107/4/2)
  - P2P Botnet Quick Detection 系統接受度測試(107/4/16)

7.1.2 測試程序(Testing Procedure)

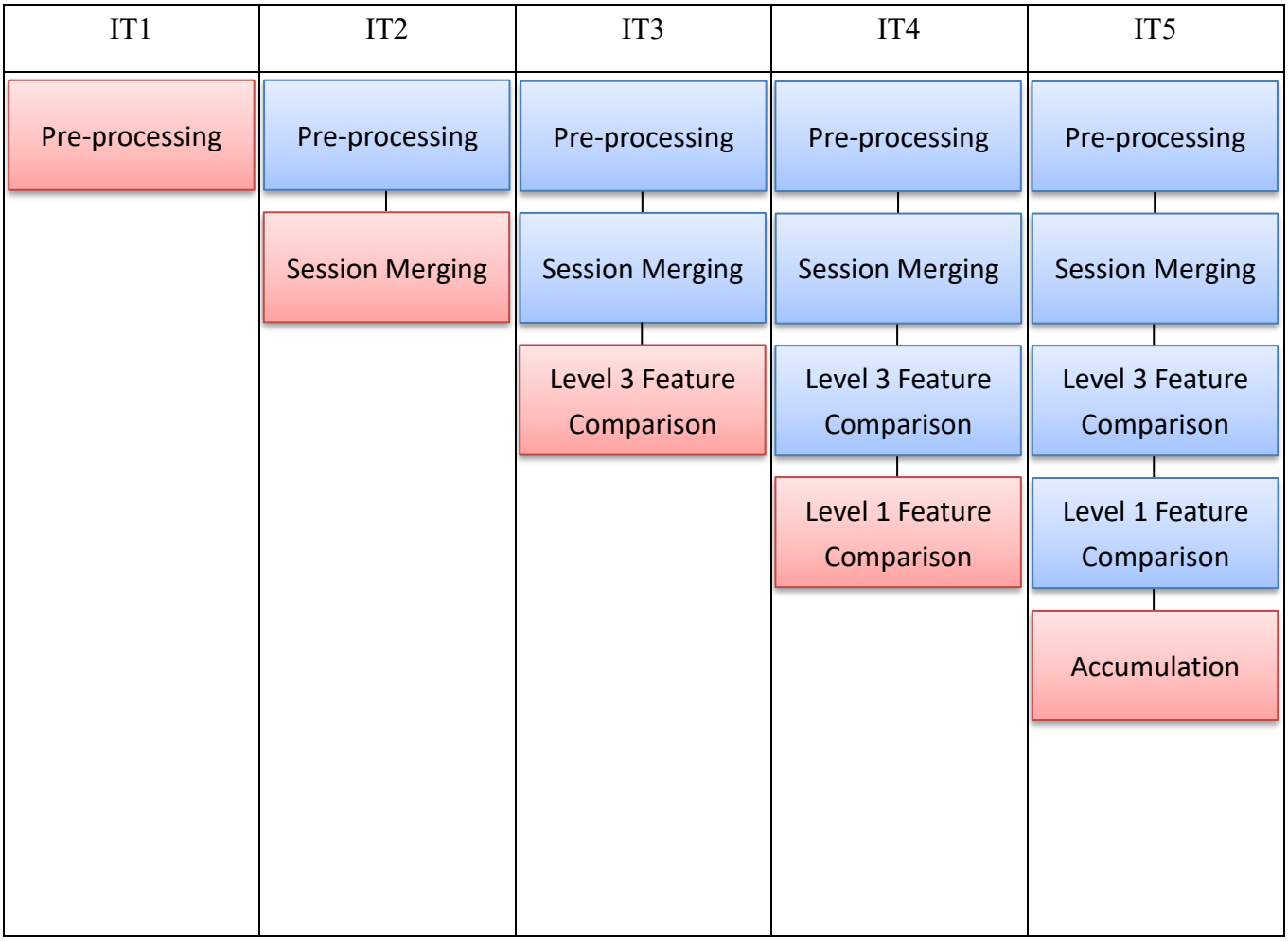
7.1.2.1 子系統驗證(Subsystem Validation)

針對近即時 P2P 殭屍網路偵測各階段驗證，以期達到有效殭屍網路偵測模組

7.1.2.2 整合測試(Integration Testing) (Incremental Testing)

(整合測試圖)

系統整合測試之次序為：IT1→IT2→IT3→IT4→IT5



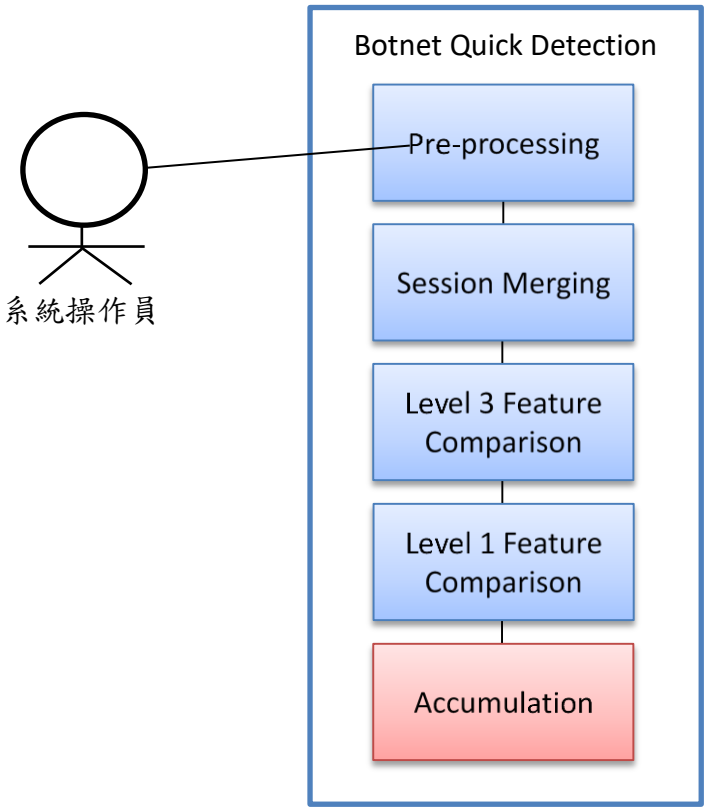
圖二　　整合系統測試

7.1.2.3 接受測試(Acceptance Testing)(Alpha Testing)

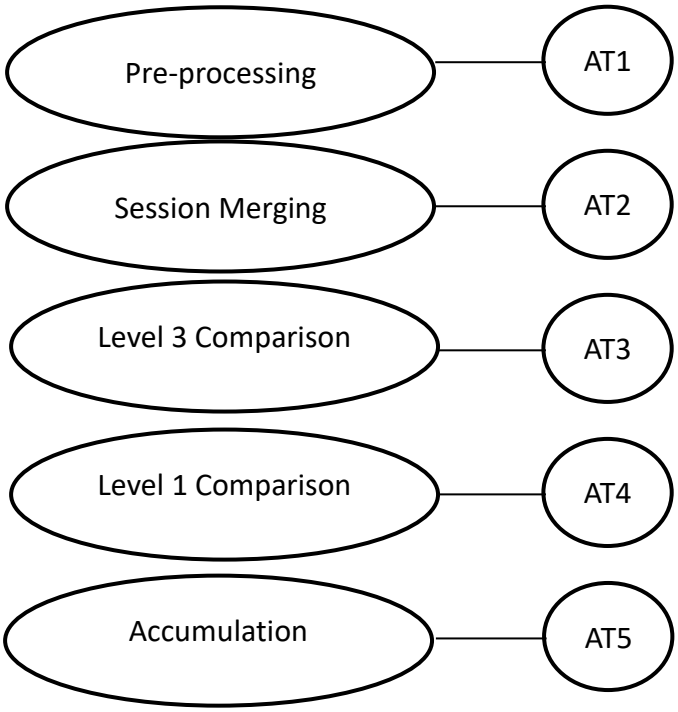
本系統須達到需求規格書所列之所有功能，如下表所示：

需求編號	優先順序	需求描述
TAF-DIC-001	1	通用型近即時 P2P 殭屍網路偵測演算法，各階段程式採用 Flink 框架為基礎，採用 Scala 為開發語言。

針對本系統之需求，本系統設計時期之使用案例(Use Case)如下圖所示，本系統須達成使用案例所列之所有功能。



圖三　　Botnet Detection 系統使用案例圖



圖四　　系統接受度測試

7.1.3 人員職責分配(Personnel and Responsibility)

本系統之測試人員姓名及職責如下列所示。

表一　　人員職責分配表

Testing Activities	Personnel
IT1	卓峰民
IT2	歐奇隴
IT3	歐奇隴

IT4	陳冠中
IT5	陳冠中
AT1	張育恩
AT2	呂宗達
AT3	呂宗達
AT4	葉家宏
AT5	葉家宏

8 測試案例(Test Case)

8.1 子系統

8.1.1 整合測試案例(Integration Testing Cases)(Incremental testing)

8.1.1.1 IT1 Test Case -將白名單IP過濾。

輸入	NetFlow 格式的網路流量log
輸出	不是白名單IP之流量Log
步驟	將使用者定義的可信任名單(白名單、DNS、熱門網站)中的IP從輸入流量log中過濾掉，以節省資源。
預期結果	輸出 Pre-processing 後 Log

8.1.1.2 IT2 Test Case -將流量合併成session。

輸入	IT1 Test Case之輸出Log
輸出	將流量合成為session之Log
步驟	將IT1 Test Case的輸出Log當做Session Merging指令的輸入，依照相同來源IP，目的地IP以及通訊協定合成Session。
預期結果	輸出Session Merging後的Log

8.1.1.3 IT3 Test Case -將session與BotCluster所產生的Level 3分群特徵進行比較。

輸入	IT2 Test Case之輸出Log
輸出	與BotCluster Level3 Group中相近的session
步驟	將IT2 Test Case的輸出Log當做Level 3 Feature Comparison指令的輸入，將輸入之session與Botcluster的Level3 Group的各個分群比較距離，找出一定範圍內之session
預期結果	輸出Level 3 Feature Comparison後的Log

8.1.1.4 IT4 Test Case -將session與BotCluster所產生的Level 1分群特徵進行比較。

輸入	IT3 Test Case之輸出Log
輸出	與BotCluster Level1 Group中相近的session
步驟	將IT3 Test Case的輸出Log當做Level 1 Feature Comparison指令的輸入，將輸入之session與Botcluster的Level1 Group的各個分群比較距離，找出相近之session。
預期結果	輸出Level 1 Feature Comparison後的Log

8.1.1.5 IT5 Test Case –進行該惡意行為出現次數計算。

輸入	IT4 Test Case之輸出Log
輸出	惡意行為之IP
步驟	將IT4 Test Case的輸出Log當做Accumulation指令的輸入，依照設定好的出現次數與時間之標準，判定是否為惡意IP。
預期結果	輸出Accumulation後的IP

4.1.2 接受測試案例(Acceptance Testing Cases)(alpha testing)

因我們進行測試時即是在User Site使用Develop Site的環境，因此IT與AT是同時進行的測試的。因此完成測試4.1.1之內容同時也完成AT測試。

7 測試結果與分析(Test Result and Analysis)

7.1 子系統

7.1.1 整合子系統測試案例(Integration Testing Cases)

表二 整合子系統測試案例

Test Case #	Results (PASS/FAIL)	Comment
IT1	PASS	
IT2	PASS	
IT3	PASS	
IT4	PASS	
IT5	PASS	

7.1.2 接受測試案例(Acceptance Testing Cases)

表三 接受測試案例

Test Case #	Results (PASS/FAIL)	Comment
AT1	PASS	
AT2	PASS	
AT3	PASS	
AT4	PASS	
AT5	PASS	

Appendix A：Traceability

8 子計畫

8.1 子系統 vs. 測試案例(Subsystem vs. Test Cases)

表四 Subsystems vs. Test Cases Traceability Table

Subsystems / Test Cases	Pre-processing	Session Merging	Level 3 Feature Comparison	Level 1 Feature Comparison	Accumulation
IT1	V				
IT2		V			
IT3			V		
IT4				V	
IT5					V

8.2 求 vs. 測試案例(Requirements vs. Test Cases)

表五 Requirements vs. Test Cases Traceability Table

Test Cases / Requirements	IT1	IT2	IT3	IT4	IT5
TAF-DIC-001	V	V	V	V	V

Test Cases / Requirements	AT1	AT2	AT3	AT4	AT5
TAF-DIC-001	V	V	V	V	V

Appendix B：Glossary

(名詞) (解釋)

Appendix C：Reference

子計畫三：行動資安：應用程式的控制流程與字串分析技術研發與安全軟體開發策略違背偵測技術之研究 (Static Flow and String Analysis of Mobile Applications for Automatic Security Policy Violation Detection)

子計畫主持人：郁方 副教授

國立政治大學資訊管理系

9 簡介(Introduction)

行動應用程式被廣泛的應用在日常生活之中，系統性的自動化分析與檢測刻不容緩。任何在資安方面的改善與資訊揭露都可能有顯著的影響。本計劃探討 iOS 應用程式分析技術與工具開發, 實現自動且系統性的行動應用靜態分析與惡意軟體偵測，從原始下載的程式碼解析,函示控制流程建立,惡意行為偵測到資訊公開揭露。期能在蘋果的黑箱外，提供額外的程式分析資訊。 在技術研發上, 有別於多數的 Android 應用程式分析技術, 我們以 iOS 行動應用程式本身為主體，研發二元程式碼的靜態流分析技術，透過應用程式執行檔的反組譯工程解析應用程式的控制流程與跨函示的呼叫關聯，再利用字串分析技術解析以字串變數動態載入的系統類別與執行程式，從而架構出一健全(sound)的 iOS 應用程式行為分析模型。

(專案目的、專案範圍、測試目標)

9.1 子系統

1.1.5 測試範圍(Scope of Testing)

本文件主要是使用字串分析揭露 iOS 執行檔之動態載入類別(Static Detection of API Call Vulnerabilities in iOS Executables, Binflow)的測試計畫。Binflow 共分為七大模組，分別為

1. App Collector 模組
2. App Loader 模組
3. App Disassembler 模組
4. Segment Information Extraction 模組
5. Control Flow Graph Construction 模組
6. String Dependency Graph Construction 模組
7. Property Checking 模組

本測試計劃著眼於在系統整合前，必須先確認所有的設計元件均可正確的輸出，在此我們著重於整合系統測試(Integration Test) 及接受度測試(Acceptance Test)。本文件內容將依據系統需求規格書與系統設計文件，描述關於整合測試的相關計畫與內容。並希望透過此文件之描述與實踐，達到順利進行測試工作之目的。

1.1.6 驗收標準(Acceptance Criteria)

本測試計畫需要滿足下列的測試接受準則：

- 本系統需要對所有為必要(Critical、Important、Desirable)之需求作完整測試。
- 測試程序需要依照本測試計畫所訂定的程序進行，所有測試結果需要能符合預期測試結果方能接受。
- 以測試案例為單位，當測試未通過時，需要進行該單元的測試，其接受的準則與前一項規定相同。

10 測試環境(Testing Environment)

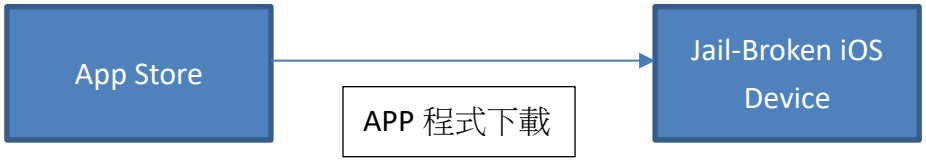
10.1 子系統

10.1.1 運行環境(Operational Environment)

本系統共分為七個模組：分別為 App Collector 模組、App Loader 模組、App Disassembler 模組、Segment Information Extraction 模組、Control Flow Graph Construction 模組、String Dependency Graph Construction 模組、Property Checking 模組

本節將敘述對於本系統各模組間的環境說明，各個場景之測試環境圖如下圖所示：

- App Collector 模組

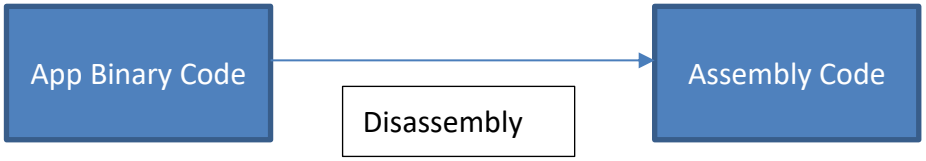


在 App Collector 模組中，我們首先從 App Store 下載並安裝應用程式到一個越獄後的 iOS 設備，藉此我們才可以直接訪問 iOS 設備中的文件系統並獲取我們所需要的二進制程式碼。

- App Loader 模組

在 App Loader 模組中，會從 iOS 設備獲取我們所需要的 App 二進制程式碼，但由於該二進制程式碼會被加密，故需要進行解密才可以讀取二進制程式碼程式碼。 而解密二進制程式碼有兩種方式：1) 可以在執行之前插入斷點，或 2) 應用第三方二進制程式碼解密工具生成解密的二進制程式碼。 解密後的二進制程式碼可以使用反彙編工具（如 IDA pro）進行分析。

- App Disassembler 模組



在 App Disassembler 模組中，我們使用 IDA 工具來生成純文本格式的 assembly code。 在這裡要注意的是，一個 iOS 應用程序的二進制程式碼是一個 Mach 對象（Mach-O），而它的 assembly code 會被拆分為多個包含各種 meta 資訊的段。也就是除了子程序的主體內容外，其他像是子例程條目，外部調用，常量字符串，映射表等也會包含在 assembly code 中。

- Segment Information Extraction 模組

Segment Information Extraction 模組為分析過程中的第一步，也為前處理階段。在此模組中，我們要從 ARMv7 架構的 iOS assembly code 中，藉由 code 中的每個 segment，來提取子程序的進入點、常量字符串

的信息，做為之後的分析使用。

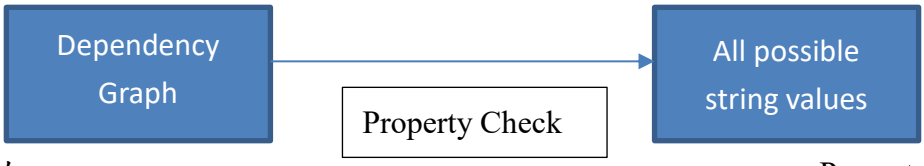
- Control Flow Graph Construction 模組

在 Control Flow Graph Construction 模組中，我們的目標是為每個子程序構建 Control Flow Graph，並且解決間接鏈接跳轉到這些例程的暫存器值。 在構建 CFG 期間，我們也會為每個彙編語句的暫存器標記依賴關係。 為了找到 sensitive functions，我們會先找到 call-external-C-function node 以及 call-external-method node，並處理它們的暫存器值，藉此來確認它們是否與 sensitive functions 有關。

- String Dependency Graph Construction 模組

在 String Dependency Graph Construction 模組中，對於每個 Sink (Sink 代表的是目標函數的參數值)，我們會構建它的 String Dependency Graph 來呈現輸入值到 Sink 之間的關係。對於每個 Dependency Graph，如果它包含外部輸入，我們就會在這裡指出有一個漏洞。

- Property Checking 模組



在 Property Checking 模組中，我們會在 Dependency Graph 中執行前向的字串分析去找出 sink node 中可能造成危害的字串值輸入特徵。我們採用 automata-based 的字串分析搭配輸入特徵，來找出 sink node 的參數中，所有可能造成危害的字串值

10.1.2 硬體規格(Hardware Specification)

依據測試環境圖內容，關於測試環境所需的硬體規格說明，如下列所示：

- App Collector 模組/ App Loader 模組
  - Server
    - RAM 16GB 或以上
    - CORE 8 核心 或以上
    - HD 2TB 或以上

- iOS Device
  - RAM 2GB 或以上
  - armv7 架構的處理器
  - SSD 32GB 或以上

- App Disassembler 模組
  - CPU 為 Intel Core I5 2.5GHz 或以上
  - RAM 16GB 或以上
  - HD 3TB 或以上
- Segment Information Extraction 模組
  - CPU 為 Intel Core I5 2.5GHz 或以上
  - RAM 8GB 或以上
  - HD 3TB 以上
- Control Flow Graph Construction 模組
  - CPU 為 Intel Core I5 2.5GHz 或以上
  - RAM 16GB 或以上
  - HD 3TB 以上
- Dependency Graph Construction 模組
  - CPU 為 Intel Core I5 2.5GHz 或以上
  - RAM 24GB 或以上
  - HD 6TB 以上
- Property Checking 模組
  - CPU 為 Intel Core I5 2.5GHz 或以上
  - RAM 24GB 或以上
  - HD 6GB 以上

10.1.3 軟體規格(Software Specification)

依據測試環境圖內容，關於測試環境所需的軟體規格說明，如下列所示：

Server 主機:

- Ubuntu14.04LTS 以上
- Java SE Development Kit 7 或以上
- IDA Pro
- C/C++
- DFS
- OpenSSH-server
- VIM
- Jenkins
- PostgresSQL

iOS 手機

- iOS 9

、 網路環境：

- 有線網路頻寬－測試 10 次平均速率

上傳速率：64.85 Mbps、下載速率：103.43 Mbps

•網路協定：TCP/IP、HTTP、SSH、

10.1.4 測試資源(Test Data Source)

- App Store 上所有 App 程式(1.5 million apps)

11 測試時程、程序和責任(Testing Schedule, Procedure, and Responsibility)

11.1 子系統

11.1.1 測試時程(Testing Schedule)

- 時程
  - Binflow 各子系統之內部元件整合測試 (Module Test) (2017/10/3~2017/11/10)
  - Binflow 各子系統測試(Integration Test) (2017/10/11~2017/11/30)
  - Binflow 各子系統接受度測試(Integration Test) (2017/10/11~2017/11/30)
- 查核點
  - Binflow 各子系統之內部元件整合測試 (Module Test) (2017/10/21~2017/11/30)
  - Binflow 各子系統測試(Integration Test) (2017/10/21~2017/11/30)
  - Binflow 各子系統接受度測試(Integration Test) (2017/10/21~2017/11/30)

11.1.2 測試程序(Testing Procedure)

11.1.2.1 子系統驗證(Subsystem Validation)

- App Collector 模組
  確認與 app store 上的 app 資訊相符
- App Loader 模組
  驗證加解密間是否異常
- App Disassembler 模組
  測試 IDA 程序是否正常運行
- Segment Information Extraction 模組
  確認特徵值是否需要增加或修改
- Control Flow Graph Construction 模組
  確認 CFG 之執行流程正確與否
- String Dependency Graph Construction 模組
  透過測試輸入，確保 Dependency Graph 建立正確
- Property Checking 模組
  確認每個 Property Check 功能正確與否

11.1.2.2 整合測試(Integration Testing)

- 模組內整合測試：
  - App Collector 模組
    本分析模組為單一模組，無需整合測試
  - App Loader 模組
    本分析模組為單一模組，無需整合測試
  - App Disassembler 模組

本分析模組為單一模組，無需整合測試

- Segment Information Extraction 模組
  確認各 Segment 中所需截取的資訊是否正確
- Control Flow Graph Construction 模組
  本分析模組為單一模組，無需整合測試
- String Dependency Graph Construction 模組
  本分析模組為單一模組，無需整合測試
- Property Checking 模組
  測試每個 Property Check 所找到的 Property

- 模組間整合測試：
  - App Collector 模組
    觀察 App 下載狀況
  - App Loader 模組
    觀察 App 是否成功解密
  - App Disassembler 模組
    觀察 所產生的 assembly code 格式正確與否
  - Segment Information Extraction 模組
    觀察 所帶的特徵值
  - Control Flow Graph Construction 模組
    觀察 Control Flow Graph 的正確產生與否
  - String Dependency Graph Construction 模組
    觀察 Dependency Graph 的正確產生與否

  - Property Checking 模組
    觀察 所找到的特徵值是否符合
- 驗收測試(Acceptance Testing)

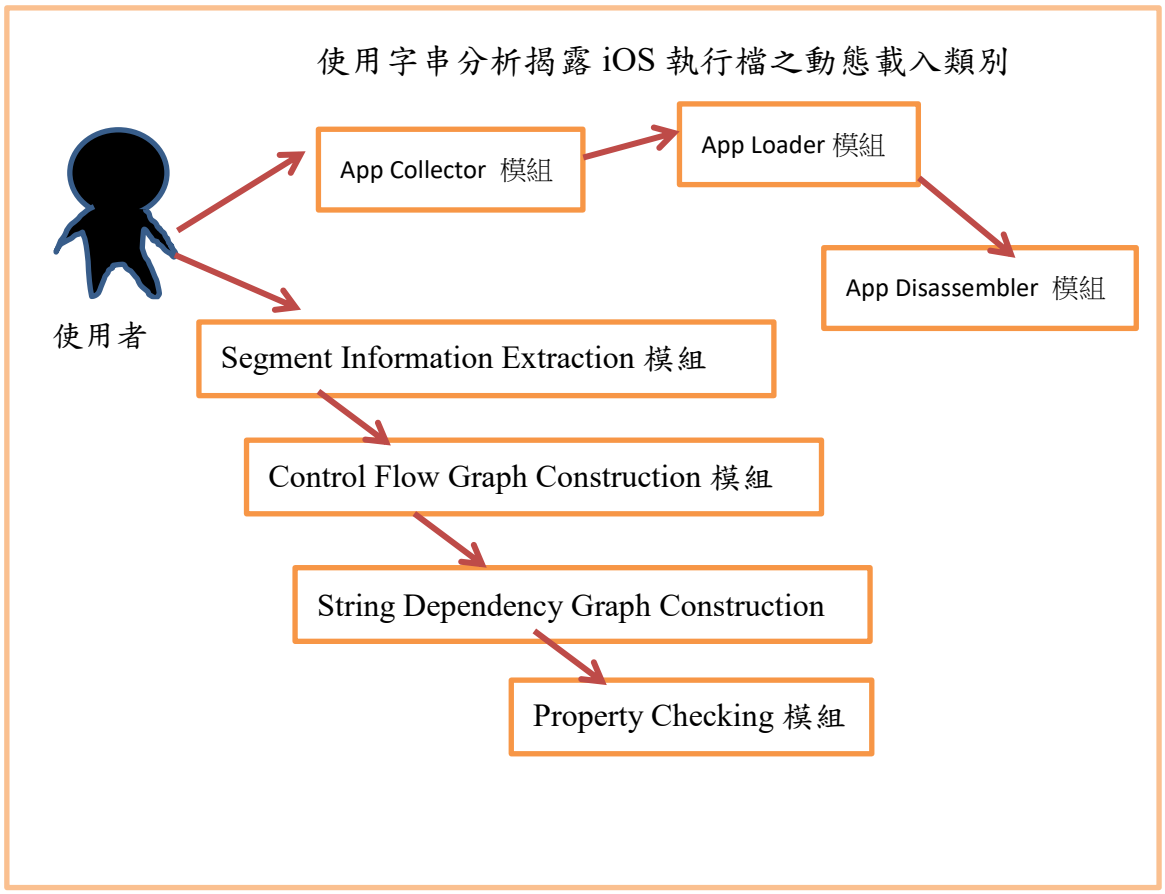
本系統須達到需求規格書所列之所有功能，如下表所示：

需求編號	優先順序	需求描述
App Collector 模組	1	從 App Store 下載並安裝應用程式，並獲得所需要的二進制程式碼。
App Loader 模組	1	解密下載完的二進制程式碼，讓之後可利用反彙編工具（如 IDA pro）進行分析。
App Disassembler 模組	1	生成純文本格式的 assembly code，包含多個 meta 資訊
Segment Information	1	提取每個 segment 中子程序的進入點、常量字符串的信息，用於

Extraction 模組		之後的分析使用
Control Flow Graph Construction 模組	1	為每個子程序構建 Control Flow Graph，並處理每個彙編語句的暫存器值
String Dependency Graph Construction 模組	1	構建 String Dependency Graph 來呈現輸入值到 Sink 之間的關係，
Property Checking模組	1	採用 automata-based 的字串分析搭配輸入特徵，來找出 sink node 的參數中，所有可能造成危害的字串值

針對本系統之需求，本系統設計時期之使用案例(Use Case)如下圖所示，本系統須達成使用案例所列之所有功能。

(使用案例圖)



11.1.3　人員與職責(Personnel and Responsibility)

本 Binflow 系統之測試人員姓名及職責如下列所示。(表)

11.1.4

成員名單與職責對照表		
縮寫	姓名	職責
TA	林君翰	Segment Information Extraction 模組、Control Flow Graph Construction 模組、String Dependency Graph Construction 模組、Property Checking 模組
TB	黃仲瑀	App Loader 模組、App Disassembler 模組
TC	黃存宇	App Collector 模組

Testing Activities	Personnel
IT1	TC
IT2	TB
IT3	TB

IT4	TA
IT5	TA
IT6	TA
IT7	TA

12 (Test Case)

12.1 子系統

12.1.1 (Integration Testing Cases)

12.1.1.1 IT1 Test Case

12.1.1.1.1 目的

App Collector 模組測試

12.1.1.1.2 輸入與輸出

輸入: 從App Store所獲得的App清單

輸出: App Binary Code

12.1.1.1.3 操作說明

執行App Collector 模組，會先從App Store獲得一份App清單作為工作列表，將工作列表中的每個app下載下來，獲得該App的Binary Code

12.1.1.2 IT2 Test Case

12.1.1.2.1 目的

App Loader模組測試

12.1.1.2.2 輸入與輸出

輸入: 加密的 Binary Code

輸出: 解密的 Binary Code

12.1.1.2.3 操作說明

執行App Loader 模組，會將下載完的Binary Code做解密的動作，使 binary code 能夠被之後分析

12.1.1.3 IT3 Test Case

12.1.1.3.1 目的

App Disassembler模組測試

12.1.1.3.2 輸入與輸出

輸入: Binary Code

輸出: Assembly Code

12.1.1.3.3 操作說明

執行App Disassembler模組時，會將 binary code 藉由IDA Pro 反組譯成 Assembly Code

12.1.1.4 IT4 Test Case

12.1.1.4.1 目的

Segment Information Extraction模組測試

12.1.1.4.2 輸入與輸出

輸入: Assembly Code

輸出: Result with Segment Information

12.1.1.4.3 操作說明

執行Segment Information Extraction模組時，會從ARMv7 架構的iOS assembly code中，提取每個segment 中的信息後，用於之後的分析。

12.1.1.5 IT5 Test Case

12.1.1.5.1 目的

Control Flow Graph Construction模組測試

12.1.1.5.2 輸入與輸出

輸入: Assembly Code & Segment Information

輸出: Control Flow Graph

12.1.1.5.3 操作說明

執行Control Flow Graph Construction模組時，藉由Assembly Code 及 Segment Information，來生成Control Flow Graph。

12.1.1.6 IT6 Test Case

12.1.1.6.1 目的

String Dependency Graph Construction模組測試

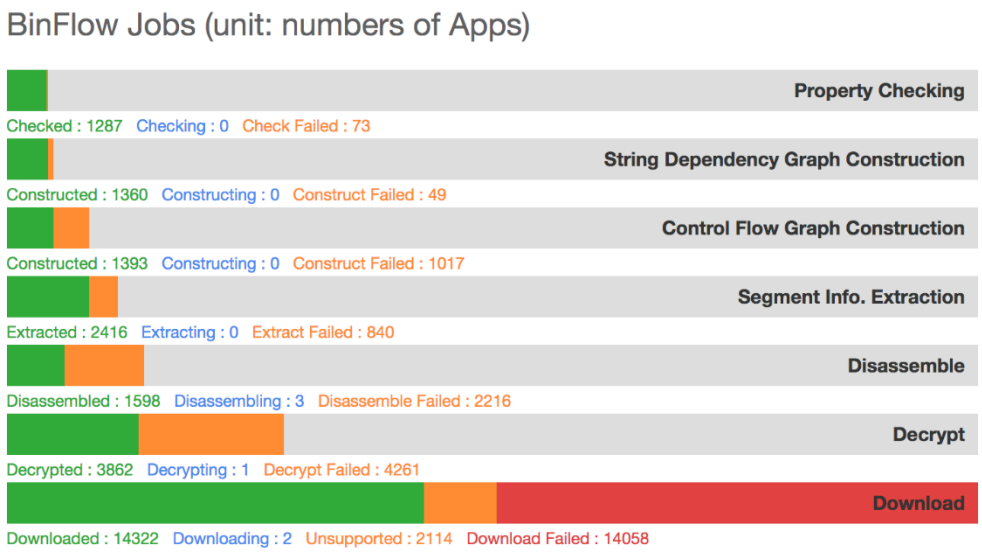
12.1.1.6.2 輸入與輸出

輸入: Control Flow Graph

輸出: String Dependency Graph

12.1.1.6.3 操作說明

執行String Dependency Graph Construction模組，藉由剛剛在CFG中所獲得的訊息，可以為每個sink中的參數建立 Dependency Graph



12.1.1.7 IT7 Test Case

12.1.1.7.1 目的

Property Checking 模組測試

12.1.1.7.2 輸入與輸出

輸入: String Dependency Graph

輸出: Property Check Result

12.1.1.7.3 操作說明

在執行Property Checking 模組時，會藉由Dependency Graph來判斷該 Property 是否存在於該 sink 之中

13 測試結果及分析(Test Result and Analysis)

5.1 資料集(一)：

總比較圖

5.1.1 App Collector 模組

5.1.1.1 實驗方法

在 App Collector 模組中，我們從 App Store 下載並安裝應用程式到一個越獄後的 iOS 設備。在這個模組中，我們總共會分成四個階段，分別為 Downloaded / Downloading / Unsupported / Downloaded Failed，以下詳述各階段所代表意義：

- Downloaded
  已經下載完的 App
- Downloading
  正在下載中的 App
- Unsupported
  若 App 被標示成此狀態，則表示其為本研究所不支援的格式。本研究是以 ARMv7 架構的處理器為主的分析方法，其在 iphone 裡頭所代表的是 32 位元處理器，若是程式採用 64 位元處理器指令集，則其處理器架構為 ARM64 架構
- Downloaded Failed
  下載失敗，可能是網路連線問題或認證問題

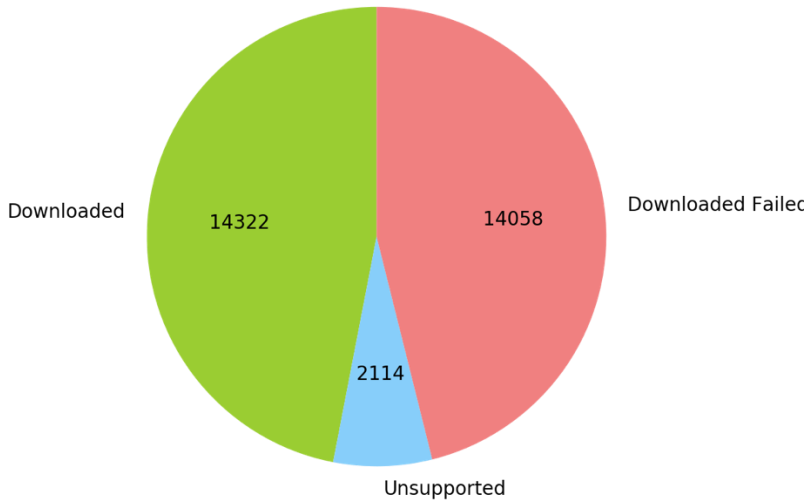
5.1.1.2 實驗結果

Downloaded 14322

Downloading: 2 (圖表省略)

Unsupported :2114

Downloaded Failed:14058



5.1.2 App Loader 模組

5.1.2.1 實驗方法

在 App Loader 模組中，會從 iOS 設備獲取我們所需要的 App 二　進制程式碼，但由於該二進制程式碼會被加密，故需要進行解密才可以讀取二進制程式碼程式碼。在這個模組中，我們總共會分成三個階段，分別為 Decrypted / Decrypting / Decrypt Failed，以下詳述各階段所代表意義：

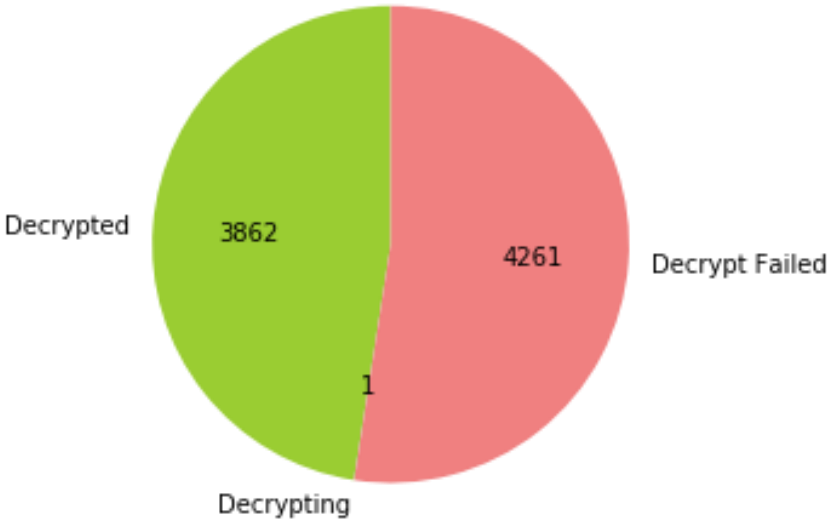
- Decrypted
  運用兩種方法中的其中一種成功解密的二進制程式碼
- Decrypting
  正在進行解密的程式碼
- Decrypt Failed
  兩種方法都解密失敗的程式碼

5.1.2.2 實驗結果

Decrypted : 3862

Decrypting : 1

Decrypt Failed : 4261



5.1.3 App Disassembler 模組

5.1.3.1 實驗方法

在 App Disassembler 模組中，我們使用 IDA 工具來生成純文本格式的 assembly code。在這個模組中，我們總共會分成三個階段，分別為 Disassembled / Disassembling / Disassemble Failed，以下詳述各階段所代表意義：

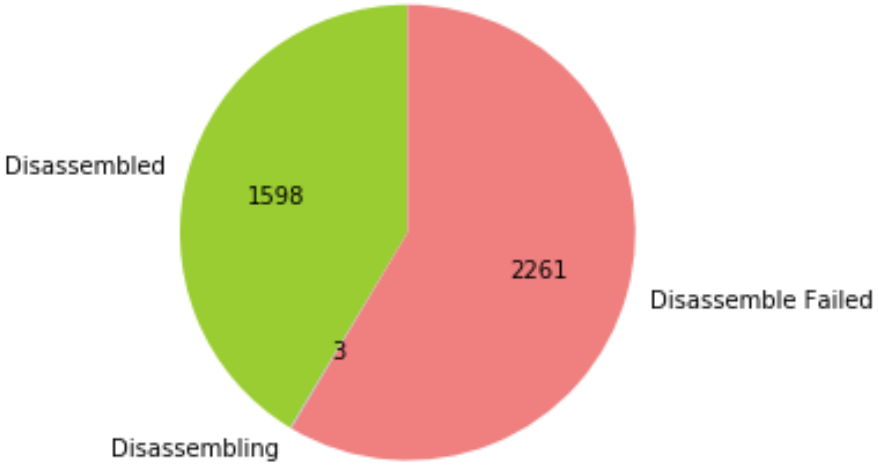
- Disassembled
  運用兩種方法中的其中一種成功解密的二進制程式碼
- Disassembling
  正在進行解密的程式碼
- Disassemble Failed
  兩種方法都解密失敗的程式碼

5.1.3.2 實驗結果

Disassembled : 1598

Disassembling : 3

Disassemble Failed : 2261



5.1.4 Segment Information Extraction 模組

5.1.4.1 實驗方法

Segment Information Extraction 模組為前處理階段，要從 ARMv7 架構的 iOS assembly code 中，藉由 code 中的每個 segment，來提取子程序的進入點、常量字符串的信息，用於之後的分析使用。在這個模組中，我們總共會分成三個階段，分別為 Extracted / Extracting / Extract Failed，以下詳述各階段所代表意義：

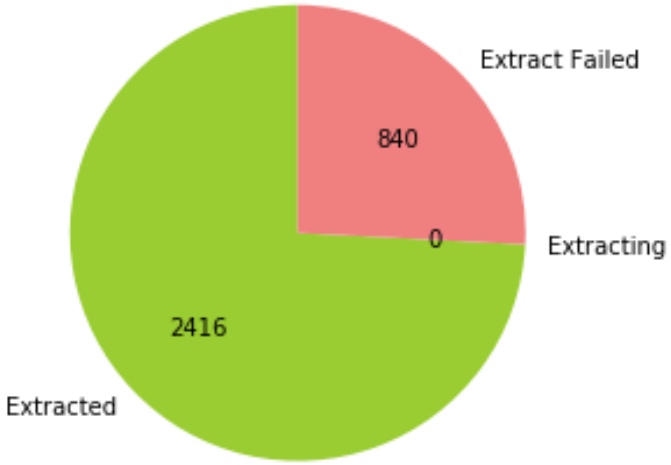
- Extracted
  成功增加 Segment 訊息
- Extracting
  正在增加 Segment 訊息
- Extract Failed
  增加 Segment 訊息失敗

5.1.4.2 實驗結果

Extracted : 2416

Extracting : 0

Extract Failed : 840



5.1.5 Control Flow Graph Construction 模組

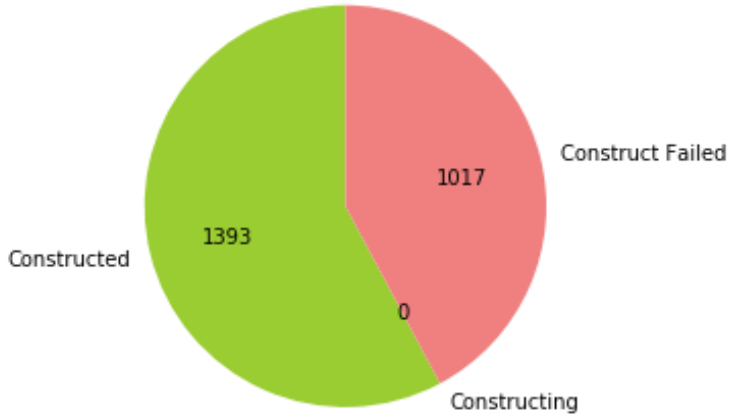
5.1.5.1 實驗方法

我們的目標是為每個子程序構建 Control Flow Graph，並且解決間接鏈接跳轉到這些例程的暫存器值。 在構建 CFG 期間，我們也會為每個彙編語句的暫存器標記依賴關係。而在這個模組中，我們總共會分成三個階段，分別為 Constructed / Constructing / Construct Failed，以下詳述各階段所代表意義：

- Constructed
  成功將 Assembly Code 轉換成為 Control Flow Graph
- Constructing
  構建 Control Flow Graph 中
- Construct Failed
  轉換成 Control Flow Graph 失敗

5.1.5.2 實驗結果

Constructed : 1393
Constructing : 0
Construct Failed : 1017



5.1.6 String Dependency Graph Construction 模組

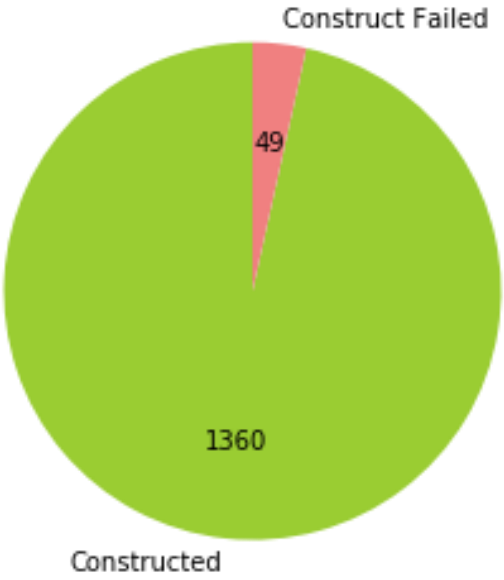
5.1.6.1 實驗方法

對於每個 Sink (Sink 代表的是目標函數的參數值)，我們會構建它的 String Dependency Graph 來呈現輸入值到 Sink 之間的關係。而在這個模組中，我們總共會分成三個階段，分別為 Constructed / Constructing / Construct Failed，以下詳述各階段所代表意義：

- Constructed
  成功將 Control Flow Graph 轉換成 String Dependency Graph 的數量(以每個 App 來做計算，因為每個 App 會有多個 String Dependency Graph)
- Constructing
  正在轉換兩者之間
- Construct Failed
  將 Control Flow Graph 轉換成 String Dependency Graph 失敗

5.1.6.2 實驗結果

Constructed : 1360
Constructing : 0 (圖表省略)
Construct Failed : 49



5.1.7 Property Checking 模組

5.1.7.1 實驗方法

我們會在 Dependency Graph 中執行前向的字串分析去找出 sink node 中可能造成危害的字串值輸入特徵。而在這個模組中，我們總共會分成三個階段，分別為 Checked / Checking / Check Failed，以下詳述各階段所代表意義：

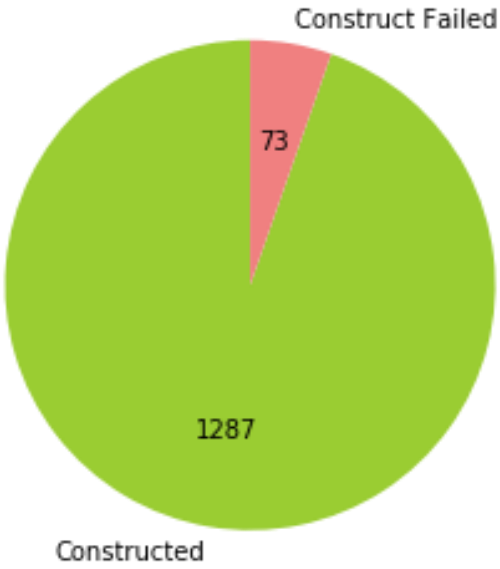
- Checked
  成功將找出在 Dependency Graph 中包含惡意的 property
- Constructing
  正在找尋與確認
- Construct Failed
  無法找出在 Dependency Graph 中包含惡意的 property

5.1.7.2 實驗結果

Constructed : 1287

Constructing : 0 (圖表省略)

Construct Failed : 73



5.1.8 研究成果已上傳分享至 github 上

github 連結：https://github.com/soslab-nccu/binflow

Glossary

(名詞) (解釋)

CFG（Control Flow Graph）控制流程圖：使用圖表符號來表示程序期間所有可能便利的路徑，控制流程圖是由 Frances E. Allen 在 Control flow analysis 這篇論文中所提出的，而 CFG 對於許多編譯器優化與靜態分析工具都很重要

SGD （String Dependency Graph）字串依賴分析圖: 從 CFG 中，我們可以找到每個可能執行外部程式的輸入值，藉由構建 SGD，可以了解輸入值之間的依賴關係，並藉由字串分析技術，透過 SGD 來解析並計算輸入值

‘

References

[1] [n. d.]. G DATA MOBILE MALWARE REPORT THREAT REPORT: Q3/2015. https://public.gdatasoftware.com/Presse/ Publikationen/Malware Reports/G DATA MobileMWR Q3 2015 EN.pdf. ([n. d.]). (Visited on 01/04/2016).

[2] [n. d.]. IDA: About - Hex-Rays. http://www.hex-rays.com/ products/ida. ([n. d.]). (Visited on 01/04/2016).

[3] [n. d.]. McAfee Labs Threats Report November 2015. http://www.mcafee.com/us/resources/reports/ rp-quarterly-threats-nov-2015.pdf. ([n. d.]). (Visited on 01/04/2016).

[4] [n. d.]. Number of apps available in leading app stores as of July 2015. http://www.statista.com/statistics/276623/ number-of-apps-available-in-leading-app-stores. ([n. d.]). (Visited on 01/04/2016).

[5] [n. d.]. Path. https://itunes.apple.com/us/app/path/ id403639508?mt=8. ([n. d.]). (Visited on 01/04/2016).

[6] [n. d.]. Path app under re for unauthorized address book up- load. http://appleinsider.com/articles/12/02/07/path app under re for unauthorized address book upload.html. ([n. d.]). (Visited on 01/04/2016).

[7]Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Luka ́ˇs Hol ́ık,AhmedRezine,PhilippRu ̈mmer,andJariStenman.2014. String constraints for veri cation. In International Conference on Computer Aided Veri cation. Springer, 150–166.

[8] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. FlowDroid: precise context, ow, eld, object-sensitive and lifecycle-aware taint analysis

for An- droid apps. In ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI ’14, Edinburgh, United Kingdom - June 09 - 11, 2014. 29.
[9] Domagoj Babi´c, Daniel Reynaud, and Dawn Song. 2011. Malware Analysis with Tree Automata Inference. In Proceedings of the 23rd International Conference on Computer Aided Veri cation (CAV’11). 116–131
[10] Ti any Bao, Jonathan Burket, Maverick Woo, Rafael Turner, and David Brumley. 2014. BYTEWEIGHT: Learning to Recognize Functions in Binary Code. In Proceedings of the 23rd USENIX Conference on Security Symposium (SEC’14). USENIX Associa- tion, 845–860.
11] David Brumley, Ivan Jager, Thanassis Avgerinos, and Edward J. Schwartz. 2011. BAP: A Binary Analysis Platform. In Computer Aided Veri cation - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings. 463–469.
[12] Aske Simon Christensen, Anders Møller, and Michael I. Schwartzbach. 2003. Precise Analysis of String Expressions. In Proc. 10th International Static Analysis Symposium (SAS) (LNCS), Vol. 2694. Springer-Verlag, 1–18. Available from http://www.brics.dk/JSA/.
[13] Lucas Davi, Alexandra Dmitrienko, Manuel Egele, Thomas Fis- cher, Thorsten Holz, Ralf Hund, Stefan Nu¨rnberger, and Ahmad- Reza Sadeghi. 2012. MoCFI: A Framework to Mitigate Control- Flow Attacks on Smartphones.. In NDSS.
[14] Zhui Deng, Brendan Saltaformaggio, Xiangyu Zhang, and Dongyan Xu. 2015. iris: Vetting private api abuse in ios ap plications. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM, 44–56.
[15] DroidBench. [n. d.]. Droidbench benchmarks. https://github.com/secure-software-engineering/DroidBench. ([n. d.]).
[16] Dynist. [n. d.]. Dynist: Tools for binary instrumentation, analysis, and modi cation. https://github.com/dyninst. ([n. d.]).
[17] Manuel Egele, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. 2011. PiOS: Detecting Privacy Leaks in iOS Applications.. In NDSS.
[18] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. 2014. TaintDroid: an information- owtracking system for realtime privacy monitoring on smartphones.ACM Transactions on Computer Systems (TOCS) 32, 2 (2014), 5.
[19] Z. R. Fang, S. W. Huang, and F. Yu. 2016. AppReco: Behavior-Aware Recommendation for iOS Mobile Applications. In 2016 IEEE International Conference on Web Services (ICWS). 492– 499.
[20] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner. 2011. A Survey of Mobile Malware in the Wild. In Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM ’11). 3–14.
[21] Carl Gould, Zhendong Su, and Premkumar Devanbu. 2004. Static checking of dynamically generated queries in database applications. In Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on. IEEE, 645–654.
[22] Hex-Rays. [n. d.]. IDAPro. https://www.hex-rays.com/products/ida. ([n. d.]).
[23] Jianjun Huang, Xiangyu Zhang, Lin Tan, Peng Wang, and Bin Liang. 2014. AsDroid: detecting stealthy behaviors in Android applications by user interface and program behavior contradic tion. In 36th International Conference on Software Engineering, Hyderabad, India - May 31 - June 07, 2014. 1036–1046.
[24] Adam Kiezun, Vijay Ganesh, Philip J Guo, Pieter Hooimeijer, and Michael D Ernst. 2009. HAMPI: a solver for string constraints. In Proceedings of the eighteenth international symposium on Software testing and analysis. ACM, 105–116.
[25] Guodong Li and Indradeep Ghosh. 2013. PASS: String solving with parameterized array and interval automaton. In Haifa Veri cation Conference. Springer, 15–31.
[26] Li Li, Tegawend´e F Bissyand´e, Damien Octeau, and Jacques Klein. 2016. DroidRA: taming re ection to support whole-program analysis of Android apps. In Proceedings of the 25th International Symposium on Software Testing and Analysis. ACM, 318–329.
[27] Tianyi Liang, Andrew Reynolds, Cesare Tinelli, Clark Barrett, and Morgan Deters. 2014. A DPLL (T) theory solver for a theory of strings and regular expressions. In International Conference on Computer Aided Veri cation. Springer, 646–662.
[28] Christopher Mann and Artem Starostin. 2012. A framework for static detection of privacy leaks in android applications. In Proceedings of the 27th Annual ACM Symposium on Applied Computing. ACM, 1457–1462.
[29] Xiaozhu Meng and Barton P. Miller. 2016. Binary Code is Not Easy. In Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016). ACM, 24–35.
[30] Yasuhiko Minamide. 2005. Static approximation of dynamically generated web pages. In Proceedings of the 14th international conference on World Wide Web. ACM, 432–441.
[31] Nicholas Nethercote and Julian Seward. 2007. Valgrind: a framework for heavyweight dynamic binary instrumentation. In ACM Sigplan notices, Vol. 42. ACM, 89–100.
[32] Thomas Reinbacher and J¨org Brauer. 2011. Precise Control Flow Reconstruction Using Boolean Logic. In Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT ’11). ACM, 117–126.
[33] Paulo de Barros SILVA FILHO. 2016. Static analysis of implicit control ow: resolving Java re ection and Android intents. (2016).
[34] Dawn Song, David Brumley, Heng Yin, Juan Caballero, Ivan Jager, Min Gyung Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam, and Prateek Saxena. 2008. BitBlaze: A New Approach to Computer Security via Binary Analysis. In Proceedings of the 4th International Conference on Information Systems Security (ICISS ’08). 1-25.
[35] Hung-En Wang, Tzung-Lin Tsai, Chun-Han Lin, Fang Yu, and Jie-Hong R. Jiang. 2016. String Analysis via Automata Manipulation with Logic Circuit Representation. Springer Inter- national Publishing, Cham, 241–260. https://doi.org/10.1007/978-3-319-41528-4 13
[36] Gary Wassermann and Zhendong Su. 2007. Sound and precise analysis of web applications for injection vulnerabilities. In

ACM Sigplan Notices, Vol. 42. ACM, 32–41.
[37] Gary Wassermann and Zhendong Su. 2008. Static Detection of Cross-site Scripting Vulnerabilities. In Proceedings of the 30th International Conference on Software Engineering (ICSE '08). ACM, New York, NY, USA, 171–180. https://doi.org/10.1145/ 1368088.1368112
[38] Tim Werthmann, Ralf Hund, Lucas Davi, Ahmad-Reza Sadeghi, and Thorsten Holz. 2013. PSiOS: bring your own privacy & se- curity to iOS devices. In Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications se- curity. ACM, 13–24.
[39] Fang Yu, Muath Alkhalaf, and Tev k Bultan. 2010. Stranger: An automata-based string analysis tool for PHP. In International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, 154–157
[40] Fang Yu, Muath Alkhalaf, and Tev k Bultan. 2011. Patching Vulnerabilities with Sanitization Synthesis. In Proceedings of the 33rd International Conference on Software Engineering (ICSE '11). ACM, New York, NY, USA, 251–260. https://doi.org/10. 1145/1985793.1985828
[41] Fang Yu, Muath Alkhalaf, Tev k Bultan, and Oscar H Ibarra. 2014. Automata-based symbolic string analysis for vulnerability detection. Formal Methods in System Design 44, 1 (2014), 44–70.
[42] Fang Yu, Tev k Bultan, and Oscar H Ibarra. 2010. Relational string veri cation using multi-track automata. In International Conference on Implementation and Application of Automata. Springer, 290–299.
[43] Fang Yu, Yuan-Chieh Lee, Steven Tai, and Wei-Shao Tang. 2013. AppBeach: Characterizing App Behaviors via Static Binary Anal- ysis. In Proceedings of the 2013 IEEE Second International Conference on Mobile Services. IEEE Computer Society, 86.
[44] Fang Yu, Ching-Yuan Shueh, Chun-Han Lin, Yu-Fang Chen, Bow- Yaw Wang, and Tev k Bultan. 2016. Optimal Sanitization Syn- thesis for Web Application Vulnerability Repair. In Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016). ACM, New York, NY, USA, 189–200. https://doi.org/10.1145/2931037.2931050
[45] Yunhui Zheng, Xiangyu Zhang, and Vijay Ganesh. 2013. Z3- str: A z3-based string solver for web application analysis. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM, 114–124.
[46] Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. 2012. Hey, You, Get O of My Market: Detecting Malicious Apps in O cial and Alternative Android Markets. In Proceedings of the 19th Annual Network & Distributed System Security Symposium (NDSS'12).