雲端服務環境之程序隱藏型 Rootkit 偵測機制研究

曹偉駿[1]　吳佳鑫[2]　杞承樺[1]
[1]大葉大學資訊管理學系
[2]大葉大學管理學院博士班
[1]wjtsaur@yahoo.com.tw
[2]joker0301@gmail.com

摘要

雲端服務發展日趨成熟，其雲端之優點也提供了駭客便利的途徑，使其發展出複雜而細膩的攻擊手法。而這些手法不難發現皆有 Rootkit 的蹤跡，其中又以木馬型 Rootkit 危害最為嚴重。這種結合 Rootkit 隱藏的技術中，以移除雙向鏈結與使用系統服務表最難以偵測，因此常常令使用者不自覺的下載，並開啟檔案使其逐漸擴散於鄰近的系統與網路。且潛伏等待時機發動攻擊，其攻擊的發起方式是由遠端伺服器控制，傳達指令後透過偽裝成正常程序或執行緒的木馬，再經由網路傳輸回送給攻擊者，以竊取重要資料。這種手法一般又稱之為 APT (Advanced Persistent Threat)，其對於雲端環境有強大的威脅。雖然有知名偵測軟體亦能偵測出隱藏於程序型的 Rootkit，但面對混合型 Rootkit 時，卻常常無法有效偵測。因此，本研究將於雲端作業系統上開發出 Process-hidden rootkit 偵測機制，目的在於能有效偵測出利用移除雙向鏈結與使用系統服務表做 Rootkit 隱藏的混合型木馬程式，進而達到防範雲端之 APT 攻擊手法的一環，並可協助防毒軟體與雲端系統服務商，建構出完整防禦 Rootkit 攻擊機制。

關鍵詞：Rootkit，木馬程式，雲端服務，Windows 作業系統，進階持續性攻擊。

Abstract

Since cloud service's development becomes mature, the advantages of cloud service also give hackers easy way to create complicated and exquisite techniques of attacks. Rootkit always be used in these techniques and exquisite one is Trojan-based rootkits. In this rootkit-combined technique, "removing double linked list" and "using system services" are very hard to detect, which is why it always let users download data unconsciously and spread to contiguous systems and networks gradually by opening files. The way of attack is hiding to wait opportunities, and is controlled by a remote server. And pretends to be proper procedures or threads after conveying instructions, and steals important information by network transfer back to the attacker. The above-mentioned trick is called the technique of "APT" (Advanced Persistent Threat) which becomes a big menace to cloud services. Although famous anti-virus software can detect process-hidden rootkits, they still cannot work when confronting to mixed rootkits. Therefore, this research will develop a mechanism for detecting process-hidden rootkits in cloud operating systems to avoid APT attacks on clouds, which can effectively detect mixed rootkits of "removing double linked list" and "using system services". Moreover, the proposed mechanism can help anti-virus software and cloud systems service provider develop a complete protection mechanism against rootkit attacks.

Keywords: Rootkit, Trojans, Cloud Services, Windows OS, Advanced Persistent Threat

1. 前言

探討雲端服務之安全議題中，本研究從文獻[1]中發現 APT (Advanced Persistent Threat) 的攻擊手法是近幾年最為複雜與棘手，其中大部分皆以 RATs(Remote Administration Toolkits/Remote Access Toolkits)結合 rootkit 作為主要的攻擊手法之一[1]，而這種手法就是木馬程式與 rootkit 結合成為具有隱藏能力與控制的 APT 雲端攻擊。因此，當使用者一旦不自覺的下載開啟了檔案，這種 rootkit 將隱藏於系統中，逐漸擴散於鄰近的系統與網路，在等待時機發動攻擊。

而現今雲端服務架構以虛擬化技術為基礎[2]的狀況下，各大企業紛紛投入於虛擬化技術，但其依然是建構在雲端作業系統之上，一旦雲端作業系統遭受攻擊，則建構在雲端作業系統上的虛擬機器亦即成為迅速擴散的媒介，因此雲端作業系統之安全就顯得格外重要。文獻[3]中指出，目前市面上的防毒軟體並無法完全偵測出此種木馬型的 rootkit，原因在於其載入的途徑為正常程序，故防毒軟體視為正常之應用程式。因為這種木馬型 rootkit 使用 IAT Hooking 技術，使其不易判斷是否真的為惡意軟體，除此之外移除雙向鏈結與使用系統服務表 SSDT(System Services Descriptor Table)，偽裝成正常的 Process 的方式更難以偵測。

本研究將在雲端作業系統上開發出 Process-hidden rootkit 偵測機制於 Host OS 與 Guest OS，並使用移除雙向鏈結與使用系統服務表的技術研製出未知的木馬型 rootkit，目的在於能夠有效偵測出混合型與現存的木馬型 rootkit，進而達到防範雲端 APT 攻擊手法中的一環。

2. 文獻探討

2.1 雲端服務環境安全技術

本研究可以知道雲端服務具有共享資源以及極佳的擴充性，因此成為了駭客攻擊的絕佳環境。根據文獻[4]指出，許多的雲端安全性議題中以 APT 是雲端攻擊最為嚴重的一項，而 APT 的手法中又以 rootkit 隱藏技術為主要的隱藏手法，如果駭客想要竊取重要資料時，首先就需在實體機器中安置惡意軟體。然而，透過木馬型 rootkit，滲透到對方的主機系統裡，就能達到遠端操作目標主機的目的[5][6]。其破壞力之大，是絕不容忽視的，以下本研究將針對木馬型 rootkit 進行分析。

木馬型 rootkit 可概分為兩種，一種是隱藏木馬的伺服器端，可以偽隱藏，也可以是真隱藏[7]。偽隱藏，就是指程式的進程仍然存在，只不過是讓他消失在程序清單裡。真隱藏則是讓程式徹底的消失，不以一個程序或者服務的方式工作。

當程序為真隱藏的時候，那麼這個木馬的伺服器部分程式運行之後，就不具備一般程序，也不具備服務，其完全的與系統的核心結合。所以本研究可以不做成一個應用程式，而把將其做為一個執行緒，一個其他應用程式的執行緒，把自身注入其他應用程式的位址空間。由圖 1 所示，可以知道真隱藏型的木馬感染途徑，對於系統來說會誤認為一個絕對安全的程式，因此其達到了徹底隱藏的效果，而這樣的結果也導致了偵測木馬程式難度的增加。

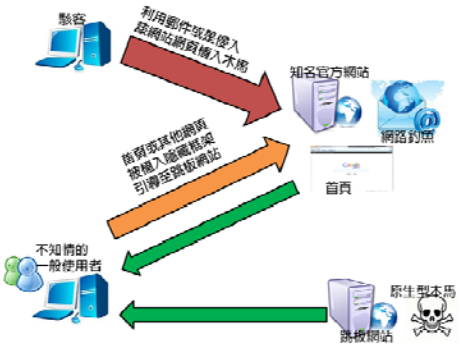

圖 1 木馬攻擊方式[8]

2.2 基於程序的 Rootkit 隱藏技術

在木馬程式中所使用的隱藏方式有程序與執行緒及服務的方法，而具體的方式則是利用 Rootkit 隱藏技術中的移除雙向鏈結以及使用 SSDT 的方式實現 [9,10]，而以下將介紹此兩種方法。

移除雙向鏈結將實體的文件位址更改為虛擬的位址，從 Windows Object Management 裡我們可以知道所有物件皆存在 Object Directory 的樹狀架構中。在圖 2 中，Object Directory 有 37 個 HashBuckets。每一個物件皆會指向一個 Object Directory Entry 且該 Object Directory Entry 的 ChainLink 會指向另一個 Object Directory Entry。Object Drivers 至少都有一個 Object Device 指向他們自己，所以當開發一個 Driver-hidden Rootkit 時皆需要考慮，是否可以經由 Object Directory 去找到 Object Drivers 和 Object Devices，如果皆已移除便可以達到隱藏的效果。

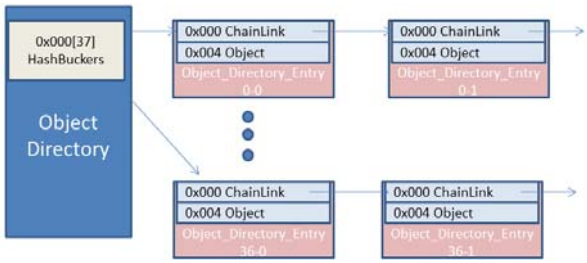

圖 2 Object_Directory 與其成員的關係[10]

使用系統服務表，在 SSDT 中有內核可以調用的函數區域。從圖 3 可見，當用戶層調用 FindNextFile 函數時，最終會調用內核層的 NtQueryDirectoryFile 函數，而這個函數的區域就在 SSDT 表中，如果我們事先把這個區域竄改成我們特定函數的區域，那麼就可達成隱藏的目的[11]。

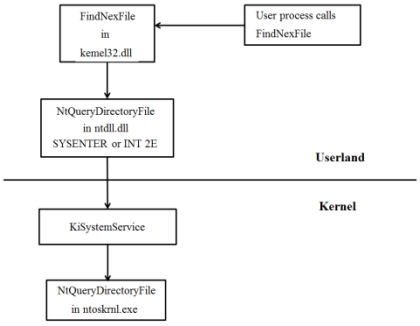

圖 3 調用 NtQueryDirectoryFile 函數[11]

2.3 現有程序型 Rootkit 的偵測技術

從上述中可以知道結合木馬的 Rootkit 大部分使用修改系統核心程式或是藏在系統較深的目錄中，並將其命名與系統程式相似檔名，來達到隱藏[12]。此外木馬型 Rootkit 可以感染 explore.exe 和 USERINIT.EXE regedit.exe，然後依此類推將系統做全面性感染，且此種木馬可以實現自動啟動將 SSDT 返回到原來的狀態，這導致某些防毒軟體的防禦和入侵偵測系統的錯誤。因此，一般系統管理者並無法用簡單的軟體發覺其存在。以下將針對其主要隱藏方法，移除雙向鏈結與 SSDT 方式做一探討。

1.移除雙向鏈結

雙向鏈結 (doubly linked list)乃是每個節點皆

具有三個欄位，由圖 4 所示，可以知道其鏈結之關係，因此若要偵測其是否被移除鏈結達到隱藏時，其方法為核心驅動程式被載入 Windows 作業系統時，驅動程式會被分派到實體記憶體中，故所有的驅動程式都將會被載入到記憶體中，此時只要搜索我們所定義的記憶體範圍便可以找到所有的 Object Drivers，依照相對應的驅動程式之所在記憶體位置進行搜尋即可偵測出移除雙向鏈結的 Rootkit。

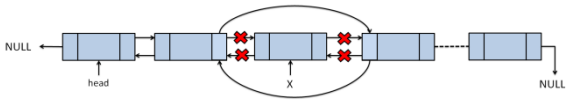

圖 4 刪除此雙向鏈結

2.SSDT

SSDT (System Service Descriptor Table) ，此技術是利用 Ring3 下的一些 API，其對應於 Ntdll.dll 裡一個 Ntxxx 的函數，因此其偵測方法是把核心檔案當作 dll 載入進記憶體，設置 DONT_RESOLVE_DLL_REFERENCES 參數，確保系統不會替我們進行重新定義指向處理，之後獲取到 KeServiceDescriptorTable 的 RVA，接著我們讀取.reloc 區塊，手動處理重新定義指向資料，直到找到對 KeServiceDescriptorTable 的重新定義指向處理，通過跟我們得到的 RVA 比對，如果相等就是有對 KeServiceDescriptorTable 重新定義指向處理，我們得到 KeServiceDescriptorTable 的位址是 0x005C9DCE, 此時我們判斷 0x005C9DCC，即 0x005C9DCE - 2 是否為 0x05c7，如果是，我們可以確定我們現在是在 mov ds：KeServiceDescriptorTable,offset_KiServiceTable 的指令當中，0x005C9DD2，即 0x005C9DCE + 4，就是我們需要定位的 KiServiceTable 位址，至此 KiServiceTable 位址已經獲取到。之後的處理與 Ring0 相似，也需要利用 ImageBase 進行修正。對於 Ring3 下獲取內核載入基址可以使用為檔案化的函數 NtQuerySystemInformation。

經過以上兩種方式，可以知道現有程序型偵測機制之方法，但仍不足以偵測出木馬程序[13]，原因是上述之方法只針對於 Rootkit 的隱藏，而木馬型 Rootkit 多了擴散與偽裝於正常程序內[6]，因此本研究將針對 Rootkit 如何與木馬之結合與其真偽藏的方式做出其有效的偵測機制。

3. 建構基於 Windows 雲端之 Process-based Rootkit 偵測機制

3.1 偵測機制

本研究所提出之偵測機制，是基於[10]所設計之 Rootkit 偵測機制的經驗中，發現許多應用程式都依賴 Windows 核心函數(Kernel Function)來獲得所要使用的函數程式，例如可從中找到 SystemProcessInformation、ProcessId、ProcessName 等核心函數，他們是 Rootkit 常要呼叫的核心函數，用以執行隱藏的動作。故我們所設計的偵測機制先載入核心函數，從 SystemProcessInformation、ProcessId、 ProcessName 等特定核心函數來檢測，若為程序型 Rootkit，則加以警告，如圖 5 所示。茲將其五個步驟作一說明如下：

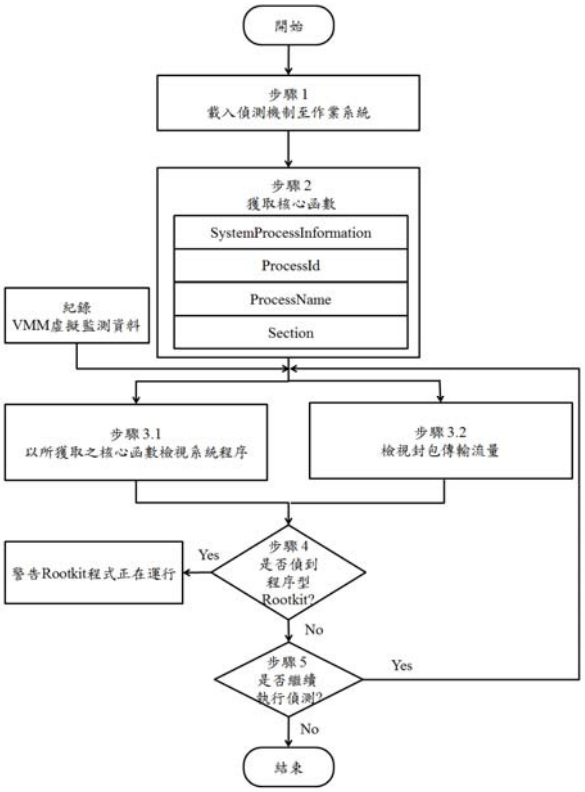

圖 5 偵測機制系統流程

步驟 1：將製作的偵測機制常駐到記憶體中，以得到系統中的 SystemProcessInformation、ProcessId、 ProcessName 等各相關核心函數。

步驟 2：呼叫 SystemProcessInformation、ProcessId、ProcessName 等核心函數後，再呼叫 Section 函數，並進一步檢測相關可疑程式是否有寫入此 Section 核心函數。由於 Section1 表裡包含著 FullDllName 函數，而 FullDllName 包含 LIST_ENTRY 結構的訊息，此訊息擁有著前後 EPROCESS 之記憶體位置。為了要隱藏驅動程式，必需先有前後兩個 LIST_ENTRY 鏈結的記憶體位置，才可以做到修改鏈結的動作，如圖 4 所示。因此，若有寫入此 Section 核心函數的動作，則可以判斷為程序型 rootkit。

步驟 3：利用以下步驟 3.1 以及步驟 3.2，並使用步驟 2 數據所得到的核心函數與程序做比對，且加上檢視網路封包流量，監測

是否異常。

步驟 3.1：主要是向系統獲取 SystemProcessInformation 、ProcessId、ProcessName 等系統核心函數，來進行檢視系統內程序。

步驟 3.2：對於網路的封包進行流量的檢視。

步驟 4：綜合步驟 3.1 與步驟 3.2，將比對結果與流量監測記錄，偵測判斷是否為程序型 rootkit，若是則跳出視窗警告。

步驟 5：偵測結果為正常，則繼續下一個程序。而整個機制由 VMM 進行監測以完整記錄其運行資料。

3.2 機制模組設計

在本機制中我們採用 SSDT(System Services Descriptor Table) 技術來取得核心函數，因其 SSDT 這個表裡面存放著各個系統核心函數的位址，其目的是讓 Windows 系統掌握其相關核心函數真正的位址。如圖 6 所示，可看出本研究之偵測機制流程分為 User mode 與 Kernel mode，其唯一溝通是經由 Ntdll.dll 檔案，這個檔案是做為當驅動程式要呼叫核心函數時從使用者層轉換到核心層的一個橋梁，在轉換過後，驅動程式會帶著必要資料來到 SSDT (System Service Descriptor Table) 尋找所要呼叫的核心函數真實記憶體位址，而擁有了真實記憶體位址才可以正確的驅動其相關核心函數。

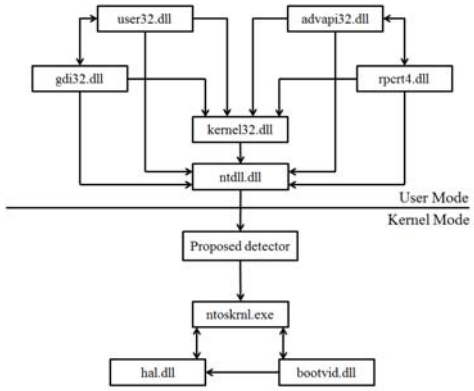

圖 6 偵測機制流程示意

有了 SSDT 之後，就擁有權限可以探索各種核心函數並且加以使用，本研究偵測機制所需要的 SystemProcessInformation 核心函數資訊則處於 KeServiceDescriptorTable。經由 Tsaur and Chen [10] 所設計之 DKOM Rootkit 偵測機制的經驗中，可以發現 DKOM 型的 Rootkit 利用當中 ZwQuerySystemInformation 來獲得所要的資訊，而我們也發現 ZW 一系列的核心函數當中，都有 System 名詞，因此本偵測機制便針對這點，搜尋「System」關鍵點，獲取所有的 System 相關資訊，來達到判斷是否為 DKOM 型 Rootkit 關鍵。

為了搜尋關鍵資料，本段程式使用了 GetProcessName 函數， GetProcessNameOffset 主要是搜索 Process Environment Block 基址，得到每一個 ProcessName 裡有關 System 的資料。

在找到所需要的 System 資料之後，便可以將資料複製到緩衝區，供驅動程式使用，如下所示，而 IRQL 是 Windows 系統的中斷請求，讓新的 IRQL 替代舊的 IRQL，並取得每一個 Process PID，以獲得實體路徑的判別。

PEPROCESS：指向 EPROCESS 數扯結構指針
Char *name ptr：指向當前 Process name
ULONG：長整數
Strncpy：替換新的 Process

Process Name

```
BOOL GetProcessName (PCHAR Name)
{
PEPROCESS          curproc;
char               *name ptr;
ULONG              i;
KIRQL              oldirql;
If(GetProcessNameOffset)
{
    curproc = PsGetCurrentProcess();
    nameptr = (PCHAR) curproc + GetProcessNameOffset;
    strncpy (Name, nameptr, NT_PROCNAMELEN );
    return TRUE;
}
return FALSE;}
```

PID：Process 識別
PsGetProcessId：返回 Process ID
ObDereferenceObject：保留 Process

Process PID

```
if (PsGetProcessId)
{Pid = PsGetProcessId((PEPROCESS)EProcess);
ObDereferenceObject((PVOID)EProcess);}
```

為了尋找木馬型 DKOM Rootkit 的關鍵，便載入封包流量監控以及所製作新型程序型 rootkit 偵測機制，其在載入之後可發現其通訊封包的發送與 System 變動跡象。因此，我們便可加以判斷這個程序有可能是 rootkit。

4. 實驗設計與分析

4.1 偵測機制測試

本研究依照所提出 Process-based rootkit 偵測機制進行系統實作與實驗，其主要是基於 Tsaur and Chen [10]提出之 Driver-hidden rootkit 偵測機制加入

主程式控制、交叉察看、特徵比對、卸載模組加以改良，且增加木馬行為判斷進而研發出一套 Process-based rootkit 偵測機制。

以下將測試本研究所研發之 Process-based rootkit 偵測機制。在使用本研究所研發之 rootkit 偵測機制時，首先設定系統環境為 Windows Hyper-v Server 2008 R2 如圖 7 所示。

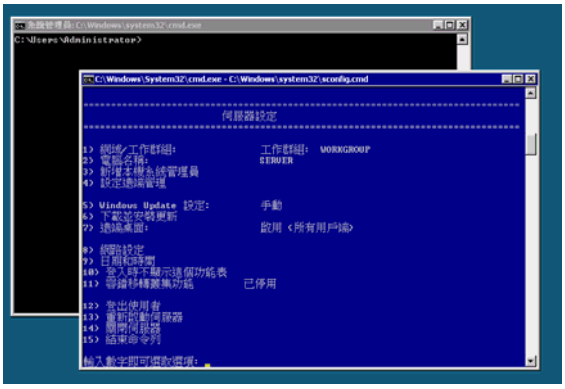

圖 7 Windows server 2008 R2 (包含Hyper-v)環境設定

當環境設定完成後，再來便是機制的執行流程。本偵測機制包含兩個部份，需先載入驅動程式，再啟動執行檔，如圖 8 所示，在 QUICK NT Driver Loader 工具中出現「Driver is actived」字樣，並使用 Xuter 工具，可發現我們載入後的驅動程式名稱為 antirootkit.sys，如圖 9 所示，其證明了 Anti-Rootkit 成功載入作業系統。接著啟動 antirootkit.exe 進行偵測，如圖 10，本偵測器執行後會列出目前所有已載入的驅動程式，而本偵測器所偵測到的 rootkit，將會於程式末端中顯示出 rootkit 所在 PID，如圖 11，使用者可以依此 PID 移除 rootkit。

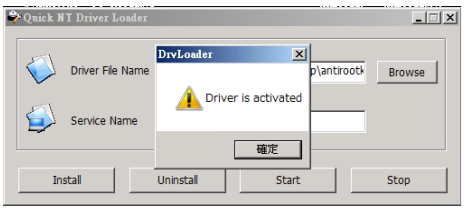

圖 8 載入驅動程式

圖 9 Anti-Rootkit成功載入作業系統

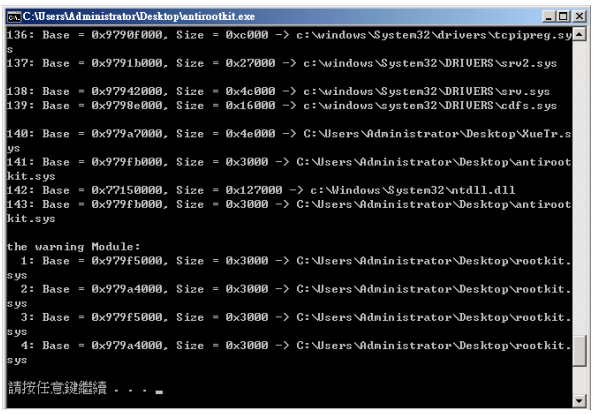

圖 10 已載入的驅動程式

圖 11 Rootkit所在PID

4.2 偵測能力與分析

為了證實本偵測機制確實優於現有知名偵測軟體，本研究從趨勢科技諮詢百科內選了近 1 年肆虐全球的著名木馬型 rootkit，如表 1 所示，並挑選 ICSA Labs 認證之國際知名防毒軟體大廠 ESET、AVAST 及趨勢科技等 rootkit 偵測軟體如表 2，進行偵測率、CPU 使用率、偵測時間等 3 項比較，其測試結果如表 3-5 所示。

網路型入侵偵測系統以原始網路封包作為資料來源，它通常運用網路卡於雜亂模式（Promiscuous mode）來偵測及分析所有過往的網路通訊。當偵測到駭客行為時，NIDS 可採多種反應方式應對，各家產品有不同的應對方式，通常都有提供通知管理者、切斷連線或記錄入侵資料作為事後稽核鑑識的證據等。因建置 NIDS 雖然成本少還是有一定量的成本，且在雲端網路型態為大的時候有多項的缺失，如表 6。

表 1 所欲測試之Rootkits

Rootkit 名稱	說明
Rootkit.tdss.TDL3	結合木馬與惡意軟體之 Rootkit
Rootkit.tdss.TDL4	結合木馬與惡意軟體之 Rootkit
Rootkit.Win32.Agent.ff	結合木馬惡意軟體之 Rootkit
RootKit.Win32.Mnless.akx	結合木馬惡意軟體之 Rootkit
Tsaur and Chen's Rootkit	Tsaur and Chen [10] 提出之新型 Driver-hidden Rootkit

表2 所欲比較之偵測軟體

代碼	說明
A	IceSword
B	AVAST Anti-rootkit
C	Trend Micro Anti-rootkit
D	ESET Anti-rootkit
E	The proposed scheme

表 3 偵測機制之比較

偵測工具 / Rootkits	A	B	C	D	E
Rootkit.tds.TDL4	×	×	○	×	○
Rootkit.tdss.TDL3	○	○	×	○	○
Rootkit.Win32.Agent.ff	○	×	×	○	○
RootKit.Win32.Mnless.akx	×	×	○	×	○
Tsaur and Chen's Rootkit	×	×	×	×	○
○代表被偵測到 / ×代表無法被偵測					

表 4 CPU使用率之比較(單位:%)

偵測工具 / Rootkits	A	B	C	D	E
Rootkit.tdss.TDL3	98	95	100	100	93
Rootkit.tdss.TDL4	95	93	100	100	98
Rootkit.Win32.Agent.ff	95	92	100	100	90
RootKit.Win32.Mnless.akx	93	90	100	100	92
Tsaur and Chen's Rootkit	97	95	100	98	69

表 5 偵測時間比較表(單位:秒)

偵測工具 / Rootkits	A	B	C	D	E
Rootkit.tdss.TDL3	4.5	70	58	67	1
Rootkit.tdss.TDL4	3	112	60	187	1.2
Rootkit.Win32.Agent.ff	3.2	98	50	102	1
RootKit.Win32.Mnless.akx	2	92	50	110	2
Tsaur and Chen's Rootkit	3	73	65	71	1.2

表 6 偵測機制比較

項目	Zeyong et al. [7]	Proposed scheme
流通封包可監控數	低	高
CPU 運算需求	高	低
記憶體空間	高	低
加密封包檢查	否	是
雲端適用情形	否	是

5. 結論與未來展望

木馬嵌入式目標系統本身是很容易被現有的防毒軟體偵測出來，所以必須透過各種技術隱藏自己通過安全機制的檢測，並盡可能延長在伺服器的存活時間，然而這樣攻擊會大大的影響雲端使用者在閱覽網路的情況下成為木馬的受害者。因此，本文分析和研究，rootkit 木馬隱藏技術主要分為三類：程序、執行緒和服務，不僅是為了了解網絡攻擊的措施也掌握隱藏技術。同時，加深了解 Windows 系統 rootkit 技術之快速發展和廣泛應用，且發現 rootkit 之攻擊在電腦多層次從用戶級別，提升至操作系統到硬體的內核，甚至直接進入 BIOS。

因此，rootkit 技術是影響一個計算機系統的安全性之關鍵，這樣中性技術在現有的防毒軟體下難以偵測，所以本研究於雲端作業系統上開發出 Process-hidden rootkit 偵測機制，目的在於能夠有效偵測出利用移除雙向鏈結與使用系統服務表做 rootkit 隱藏的混合型木馬程式，進而達到防範雲端之 APT 攻擊手法的一環。並可協助防毒軟體與雲端系統服務商，建構出完整防禦 rootkit 攻擊機制。

參考文獻

[1] F. Li, A. Lai and D. Ddl, "Evidence of Advanced Persistent Threat: A Case Study of Malware for Political Espionage," 6th International Conference on Malicious and Unwanted Software (MALWARE), pp. 102-109, Oct 2011.

[2] R. S. Montero, "Building IaaS Clouds and the Art of Virtual Machine Management," Proceedings of International Conference on High Performance Computing and Simulation (HPCS), pp. 573-573, 2012.

[3] G. Bonfa, "Step-by-step Reverse Engineering Malware: ZeroAccess / Max++ / Smiscer Crimeware Rootkit", Internet:http://resources.infosecinstitute.com/step-by-step-tutorial-on-reverse-engineering-malware-the-zeroaccessmaxsmiscer-crimeware-rootkit/ [May.20, 2013]

[4] Trend Labs, "2013 Advanced Persistent Threat", Internet: http://blog.trendmicro.com.tw/?p=3883 [Jan. 15, 2013]

[5] H. Gao, Q. Li, Z. Yu, W. Wei and Z. Li, "Research on the Working Mechanism of Bootkit," Proceedings of 8th International Conference on Information Science and Digital Content Technology (ICIDT), pp. 476-479, June 2012.

[6] X. Li, Y. Wen, M. Huang and Q. Liu, "An Overview of Bootkit Attacking Approaches," Proceedings of Seventh International Conference on Mobile Ad-hoc and Sensor Networks (MSN), pp. 428-431, Dec, 2011.

[7] L. Zeyong, G. Gang and J. Jun, "Analysis and Research on Hidden Technology Based on Kernel-Level Rootkit Process," Proceedings of 2011 International Conference on Internet Technology and Applications (iTAP), pp. 1-4, Aug. 2011.

[8] E. U. Kumar "User-mode Memory Scanning on 32-bit & 64-bit Windows," Journal in Computer Virology, vol. 6, no. 2, pp. 123-141, May 2010.

[9] P. Bravo and D. F. García, "Rootkits Survey A Concealment Story," Architecture.

[10] W. J. Tsaur and Y. C. Chen, "Exploring Rootkit Detectors' Vulnerabilities Using a New Windows Hidden Driver Based Rootkit," Proceedings of IEEE Second International Conference on Social Computing (SocialCom), pp. 842-848, Aug. 2010.

[11] J. Zhang, S. Liu, J. Peng and A. Guan, "Techniques of User-mode Detecting System Service Descriptor Table," Proceedings of 13th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2009), pp. 96-101, April 2009.

[12] E. Florio and K. Kasslin, "Your Computer is Now Stoned (Again!): the Rise of MBR Rootkits," Technical Report of Symantec, 2009.

[13] I. Seo, I. Kim, J. Yoon and J. Ryou, "Detection of Unknown Malicious Codes Based on Group File Characteristics," Proceedings of the 5th International Conference on Ubiquitous Information Technologies and Applications (CUTE), pp. 1-6, Dec 2010.