改良式軟體自我保護技術

謝文恭
中國文化大學資訊管理學系所
wgshieh@staff.pccu.edu.tw

董正談
中央警察大學資訊管理學系所
tung@mail.cpu.edu.tw

藍俊雄
南華大學管理科學所
chlan@mail.nhu.edu.tw

余彥傑
南華大學管理科學所
ycyu6886@yahoo.com.tw

黃顯舒
長庚大學資訊管理所
erichuang823@msn.com

顏志平
國立中央大學資訊工程所
peter@mail.cpu.edu.tw

高佩弦
中央警察大學資訊管理學系所

鄧少華
中央警察大學資訊管理學系所
pdeng@mail.cpu.edu.tw

摘要

電腦病毒的攻擊，透過網際網路無時無刻威脅著全世界的電腦，但是目前對抗病毒的主要工具－防毒軟體，卻無法察知所有的電腦病毒，也就是說，使用的防毒軟體不管版本多麼新，病毒碼資料庫內容涵蓋多麼廣，還是很有可能被病毒入侵。我們想提出一個改良式的軟體自我保護技術，其原理主要是利用軟體加密技術、程式變形技術、以及軟體程式之間，相互協助監控驗證的機制，以達到軟體自我保護之目的。本研究所提出之方法，不但能夠使軟體具有自我偵測及修復的功能，且可防禦病毒直接攻擊我方這種自我保護機制的企圖；此外，本研究之方法，尚可不需依賴任何病毒辨識碼，而直接有

效的對抗過去所未知的各類新種病毒。因此，我們所建議的方法，可彌補現有防毒軟體功能之不足，使得使用者能夠真正免於電腦病毒的威脅。

關鍵詞: 軟體自我防衛(Software Self-defense)、一般性攻擊(Generic Attack)、雙層軟體防衛(Two-phase Software Defense)、千面人病毒(Polymorphic virus)、數位簽章(Digital Signature)

一、緒論

資訊科技的日新月異，推動了全球化的產業革命；網際網路的蓬勃發展，促進了人與人之間的溝通，但卻有一項無形的隱憂，隨時伺機而動，準備對任何疏於防範的電腦進行攻擊，在今天，一連串來自於電腦病毒的威脅，就如同電腦硬體以及網際網路的成長一般急速增加，憑藉著網際網路的優點，電腦病毒漫游在全球資訊網中，對全世界的電腦進行攻擊，全球每年必須花費數百億美元來預防以及修復這些來自於電腦病毒的損失，傳統上來說，我們使用各類的防毒軟體來偵測病毒，這些防毒軟體基本上都是建立在所謂的病毒特徵比較技術之上，也就是由技術人員分析病毒，分析出每隻病毒特有，用以辨識病毒的程式碼，我們稱爲病毒辨識碼，然後放進防毒軟體的病毒碼資料庫，掃瞄檔案時比對資料庫內的病毒碼，來分辨出受感染的檔案與正常檔案，然而，根據我們一般人的經驗以及文獻調查發現，這些防毒軟體無法偵測出所有電腦病毒，特別是那些從未見過、加密過、或者是千面人病毒(Polymorphism Virus)，除此之外，就算擁有這些病毒的特徵辨識碼，仍然有可能發生判斷錯誤的情形。

本研究延續鄧少華教授所帶領團隊之研究，繼續針對上述防毒軟體的缺點，探討彌補之方法。包括：改正原提出研究論文不足之處，並提出改良式的新方法，評估病毒對實施此防治法後之系統攻擊是否有效。

二、問題描述與文獻探討

2.1 電腦病毒的特性

所謂的電腦病毒其實就是一個程式，但是它不以獨立的型態存在，專門寄生在其他正常程式中，是沒有自己專屬檔案屬性的秘密程式，他這樣做的目的是藉以避開使用者的注意，當使用者以爲自己正使用沒有問題的檔案時，便藉此感染其他系統或檔案，甚至是藉由網路感染其他電腦，通常電腦病毒具有以下的特性：

(1) 感染性及複製性：電腦病毒具有感染其他正常程式的特性，並藉此受感染的檔案執行時感染其他更多檔案，而且除了被動的利用程式之外，還可透過磁碟複製或網際網路等傳輸方式，不知不覺傳給其他使用者，因此電腦病毒才會抓不勝抓、防不勝防。

(2) 潛伏性：通常電腦病毒的目的不是馬上發作，而是經由使用者的一些特定的行爲，或是在特定的日期才會觸發病毒的發作，進行惡意行爲，以米開朗基羅病毒(Michelangelo Virus)爲例，此病毒固定於每年的三月六日發作，潛伏期長達一年。

(3) 破壞性：電腦病毒作者當然不會毫無目的製造電腦病毒，或許是中斷你手邊的作業，然後顯示一行聖誕快樂的訊息，或是將你的硬碟格式化，使資料消失殆盡，但是不管如何電腦病毒一定會對使用者造成或多或少的不良影響。

2.2 電腦病毒的偵測方式

光明與黑暗自古以來就是永遠的對立，有人製造電腦病毒就一定會有人挺身而出，希望能將病毒的傷害減至最低，以下是一些常見的電腦病毒偵測及預防方法：

(1) 病毒掃描法：將病毒透過人工的方式進行分析，取得每隻病毒獨特且可用以分辨其身分的病毒辨識碼，加入資料庫中，在執行掃毒任務時，便從資料庫中的資料對檔案做比對，可以快速的找出病毒，但缺點是無法發現未知的、加密過的病毒。

(2) 加總比對(Checksum)：將檔案的檔案大小、名稱、日期及內容，加總爲一個檢查碼，然後將檢查碼存於程式的後面，或是資料庫中，然後利用比對檢查碼的方式來檢驗此程式是否遭到修改，缺點是容易遭受病毒作者刻意的設計而失效。

(3) 人工智慧陷阱(Rule-based)：這個方法的原理是將病毒的行爲做分析，並總和歸納，當系統中有程式有任何不當行爲，則可判斷此程式有可能已受到病毒的感染，或爲惡意程式。此法優點爲速度快、理論是可以偵測到各式各樣的病毒，缺點爲程式設計困難，而且不容易考慮到所有的病毒行爲。

(4) 軟體模擬掃描法：利用軟體模擬CPU執行，在其虛擬的作業環境系統下執行病毒，藉此分析病毒並解碼，可說是千面人病毒的剋星。

(5) 先知掃描法(VICE-Virus Instruction Code Emulation)：此法是將工程師分析是否有病毒存在的方法，加以分析之後歸納爲專家系統知識庫，再利用軟體工程的模擬技術，在安全的環境下假執行病毒，藉以分析出新的病毒，用以防制以後的新病毒。

2.3 電腦病毒的分類

電腦病毒根據其運作環境、作用演算法、破壞性等等分類條件，大致可以分爲以下幾種：

(1) 檔案型病毒(File Virus)：這類病毒會嘗試去感染系統中的可執行檔，例如 *.com 或是 *.exe等檔案，並寄生於這些檔案中，根據其傳染方式的不同，又可再分爲常駐型與非常駐型兩類：

I. 常駐型病毒：這類病毒啓動之後，便躲藏在系統記憶體中進行冬眠，等到系統發生符合條件時便發作，通常常駐型病毒會攔截程式執行的中斷向量(例如INT 21h)，只要有程式執行就會使病毒開始運作。

II. 非常駐型病毒：非常駐型病毒會將自己寄生於可執行檔中，藉著受感染的執行檔執行時，病毒也會隨之執行。

(2) 開機型病毒(Boot Virus)：當電腦開機時，電腦必須在硬式磁碟機或是開機磁片上讀取開機時所需的檔案，開機型病毒就是對開機磁區或是分割區表格進行感

染，可能是修改開機磁區內容，將自己的惡性程式碼附於開機程式中，或是將開機檔案搬移至其他磁區並加以隱藏，自己則取代原本開機檔案之位置，也就是必須等到病毒執行完畢之後，才由病毒導向開機檔案進行開機，開機型病毒皆屬常駐型病毒。

(3) 巨集型病毒(Macro Virus)：巨集是一種軟體提供的功能，可以記錄操作過程的動作，然後設定快速鍵，只要按下快速鍵就可以替我們執行一連串紀錄的動作，簡化操作程序，例如Word、Excel、AmiPro等軟體皆有提供此功能，而巨集型病毒便是利用軟體的巨集語言，在使用者的電腦中留下程式，進而感染程式或啓動磁區。

(4) 網路病毒(Network Virus)：網路病毒的特性就是會自動透過網際網路的通訊協定連接阜(Port)或是電子郵件，對遠端的電腦或是工作站來散撥自己；或是利用Java、ActiveX等支援網頁功能的語言來撰寫惡意程式，使人上網瀏覽這些網頁時就會執行其中的惡意程式。

從傳統上來說，最原始的病毒只有兩種，也就是檔案型病毒與開機型病毒，後來隨著各種軟體的出現與普及，以及網路的興起，漸漸出現所謂的巨集型病毒以及網路病毒，但是就其目的來說，巨集型病毒與網路病毒不外乎也是以感染開機啓動磁區或是檔案爲目標，因此亦可將巨集型病毒與網路病毒歸納爲開機型或是檔案型病毒的子集合來討論，而我們的檔案自我保護技術是以保護檔案爲主要目的，在此前提之下，我們接下來將對檔案型病毒的感染方式與結構進行分析。

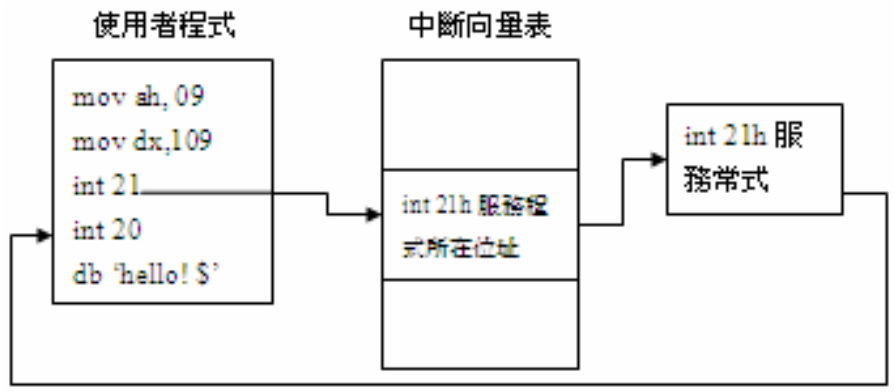
2.4 檔案型病毒的感染方式

檔案型病毒是目前數量最多且最爲廣泛的一種病毒，其攻擊主要對象就是*.com 及*.exe 這兩種可執行檔，因爲可執行檔於執行時會得到系統的控制權，所以病毒如果能感染可執行檔，那就可以藉著程式的執行取得系統控制權，然後在使用者完全不知情的情況下進行感染、複製甚至破壞行爲。

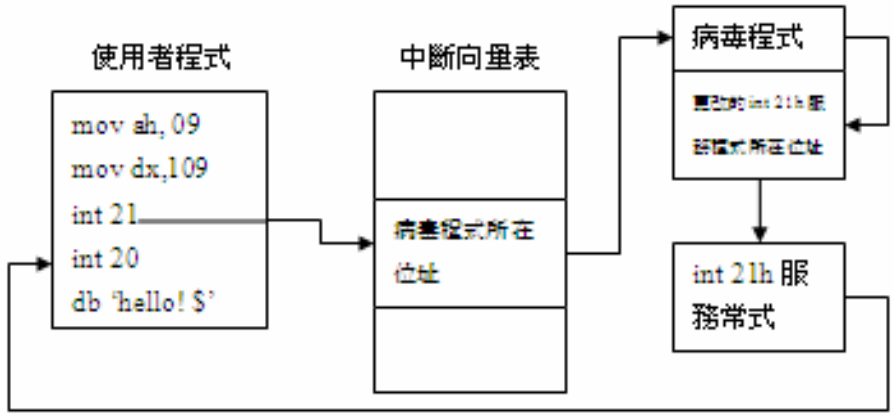
一般程式根據其執行之後的記憶體處理情形可以分爲兩種，也就是常駐型程式(TSR，Terminate and Stay Resident)與非常駐型程式(又稱穿透式，Transient)，常駐型程式在執行完畢，將控制權交回給作業系統時並不如傳統程式一般，也會同時把記憶體的空間釋放，反而會佔據所使用的記憶體空間，因爲已經將控制權交還給系統，所以這個程式就像是進入冬眠一樣，直到滿足喚醒它的條件才會甦醒，而使用這種技術的病毒我們稱之爲常駐型病毒，常駐型病毒通常都會攔截INT 21h 這個中斷向量，因爲大部分的程式執行時都會呼叫這個中斷向量，也就是說只要有程式執行，就會喚醒病毒，圖一、圖二解釋了這種情形：

由圖一、圖二我們可以清楚的了解到常駐型病毒，藉由修改記載了中斷服務位址的中斷向量表，來達到攔截int 21h這個中斷向量之目的，每當有程式要求中斷服務時，就會先執行病毒程式，才執行原本的中斷服務常式，就使用者的觀點來看，使用者還是成功的執行了本身所要求的服務，但是在這中間其實已經受到病毒攻擊，因此常常使人毫不自覺受到病毒感染，如圖三。

非常駐型病毒的發作，主要是靠著受感染的程式執行，然後藉由程式執行時系統控制權的轉移，來達到執行病毒程式碼的目的，一次執行只會做一次的惡意行爲，例如感染其他檔案或是破壞等等，至於一次感染的檔案數量與破壞程度則視寫作者設計而定，我們以圖四來簡單說明非常駐型病毒的感染方式：



圖一：使用者要求中斷服務



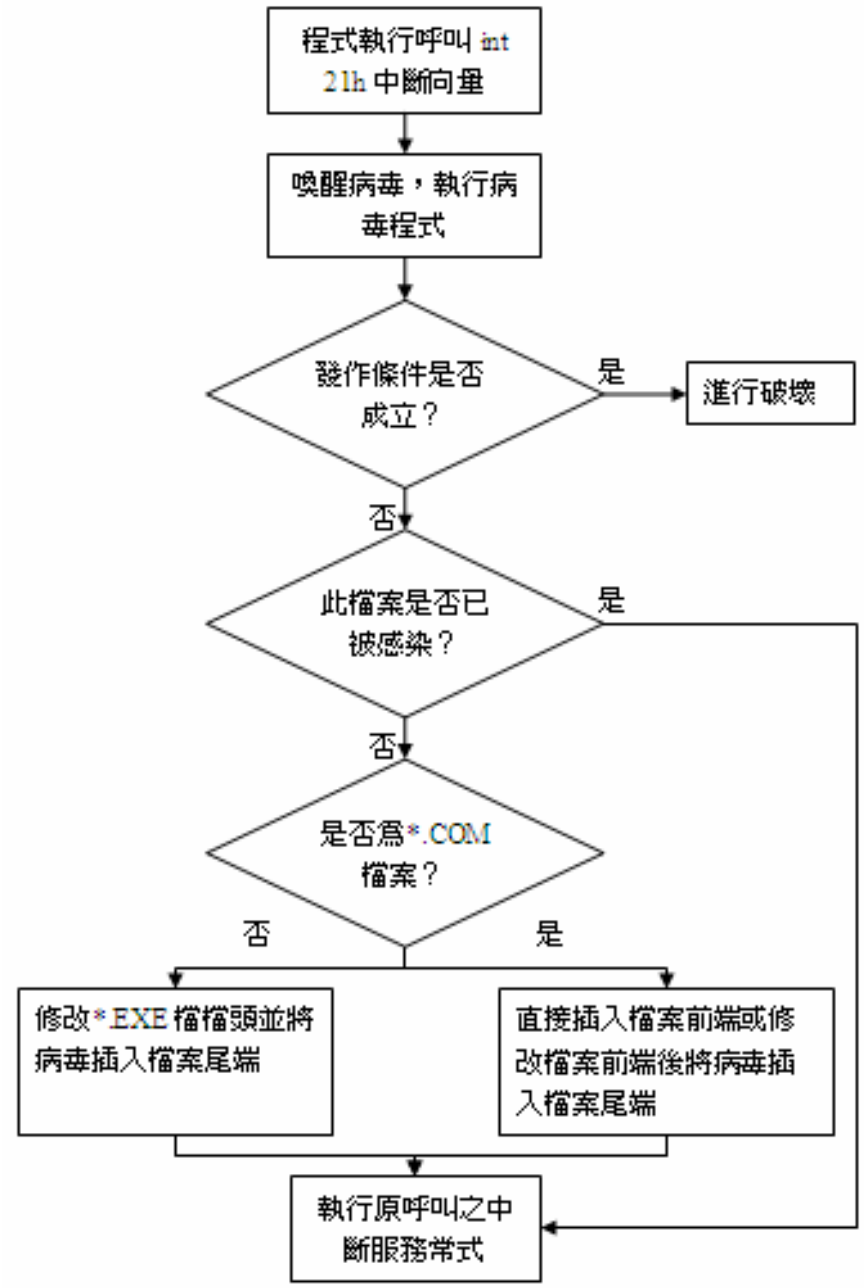
圖二：病毒攔截中斷服務

2.5 COM檔案型病毒的結構分析

了解檔案型病毒感染與破壞的流程之後，我們接下來將更深入病毒內部分析病毒如何複製、寄生及對檔案結構的改變，最重要的是了解病毒如何將程式的控制權神不知鬼不覺地轉移到自己手中。

在檔案型病毒的攻擊對象中，以 *.COM 檔及 *.EXE檔爲主，其實COM檔屬於EXE檔的特例，爲了要讓可執行檔的執行與載入速度更快，所以特別讓檔案DS(Data Segment)、ES(Extra Segment)、CS(Code Segment)及SS(Stack Segment)這四個記憶節區總合小於64K Bytes的EXE檔獨立出來，去掉檔頭，在執行時直接載入記憶體中執行，也就是說COM檔儲存在磁碟上的樣子跟載入記憶體之後一樣，圖五是COM檔案載入記憶體之後的結構：

當DOS在執行程式時，首先會建立一個256Bytes的程式前置區(PSP，Program Segment Prefix)，接著才載入程式、建立各暫存器初值，然後才把執行權交給所載入的程式，DOS



圖三： 常駐型病毒感染流程

藉由PSP與使用者溝通，做一些必要而基本的訊息交換，PSP總共可以大致可分爲三個區域：

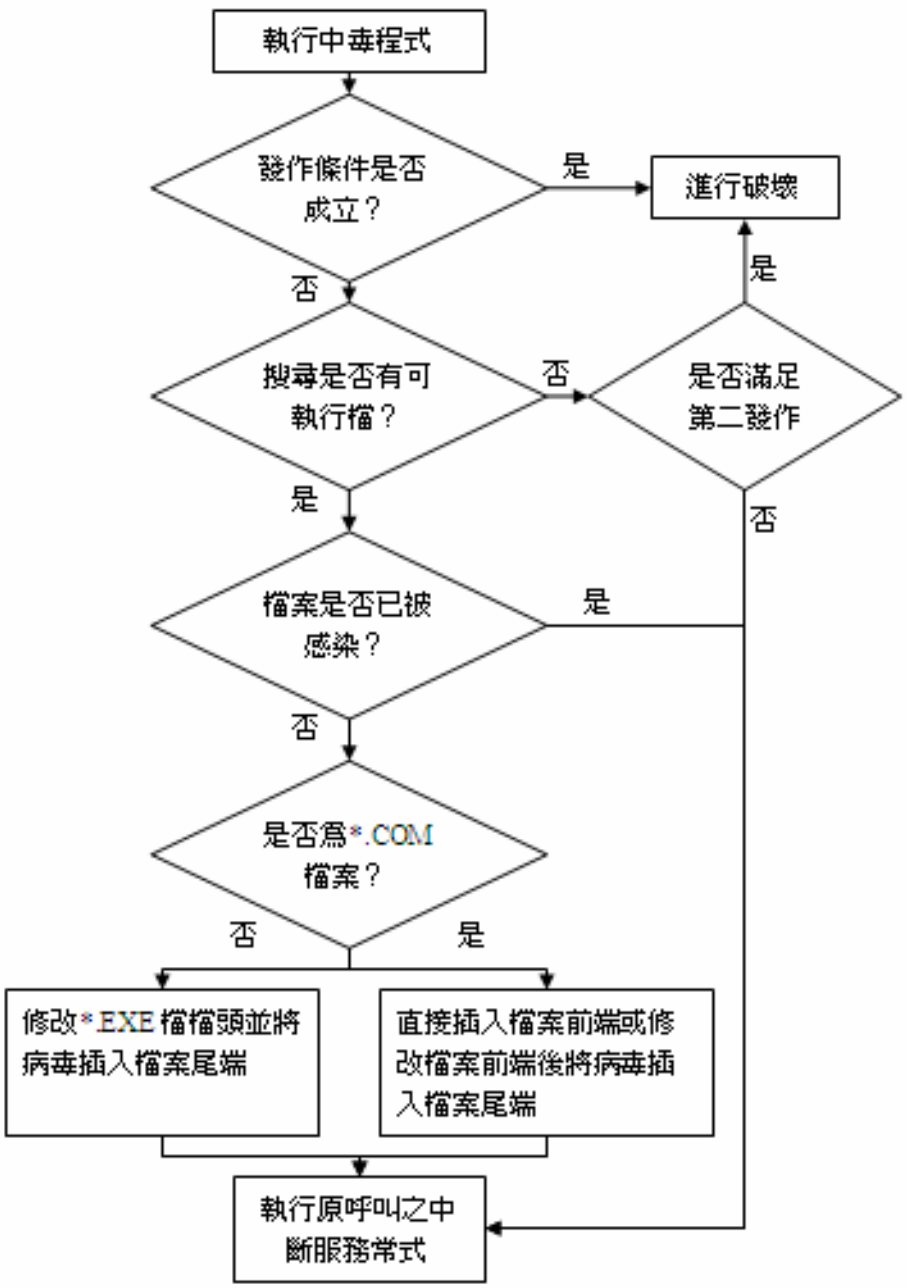
(1) 00H - 5BH：相關資訊及位址區

(2) 5CH - 7FH：FCB使用區

(3) 80H - FFH：DTA或參數區

因PSP與我們所要討論的內容沒有直接的相關，因此暫不做介紹。

溝通協調完成之後，CPU內的暫存器(Register)就會分別填入所需資訊，例如程式指位器(IP，Instruction Pointer)，當CPU執行完一行指令之後，IP指向下一指令的位址，CPU



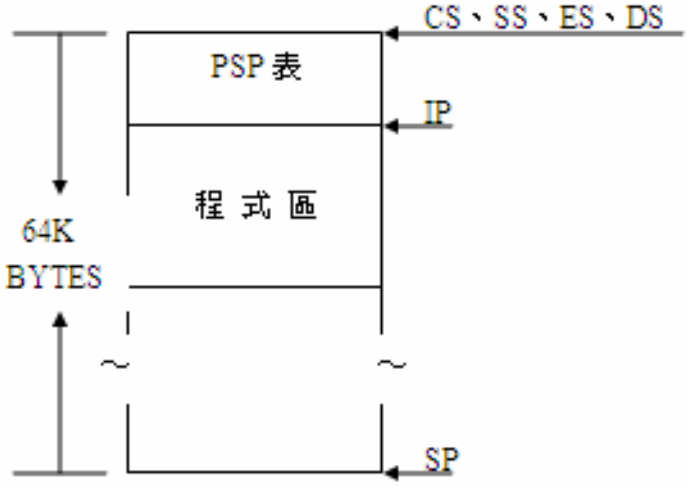
圖四： 非常駐型病毒感染流程

便往IP所指的位址去讀取所要執行的指令。就COM檔案來說，在DOS藉由PSP與使用者程式溝通完之後，便會指向程式區的開頭，也就是第一個指令所在位址，因爲COM檔沒有檔頭，所以扣除PSP所使用的256 Bytes之後就是程式區，也就是說IP此時會固定指向記憶體中100H的位址，因爲COM檔有這種特性，所以使得病毒入侵之後要取得控制權就要從此處著手，使程式一開始就會執行病毒的惡意程式碼或是利用指令使程式先執行完病毒程式碼才會到使用者程式完成剩下的流程，因爲這個想法所以會有圖六的四種COM檔病毒感染檔案的可能。

需要特別注意的一點是，以上所代表的情況爲儲存於磁碟中的檔案，並不是已經載入記憶體中待執行的情形，只因爲COM檔在磁碟中資料情況與載入記憶體中會一致，所以不需要再另外說明載入記憶體中的情形。

2.6 EXE檔案型病毒結構分析

EXE執行檔與COM執行檔不同之處就是EXE檔多了一段長度不一的檔頭，在執行時，檔頭並不會載入記憶體，系統在將程式載入記憶體時，會根據儲存於檔頭中的各個參數，來設定程式的起始狀態，也就是說，EXE檔在載入記憶體之後與儲存在檔案中的型態不同，IP也不再與COM檔一般指向固定的位址。



圖五： COM檔載入記憶體之結構

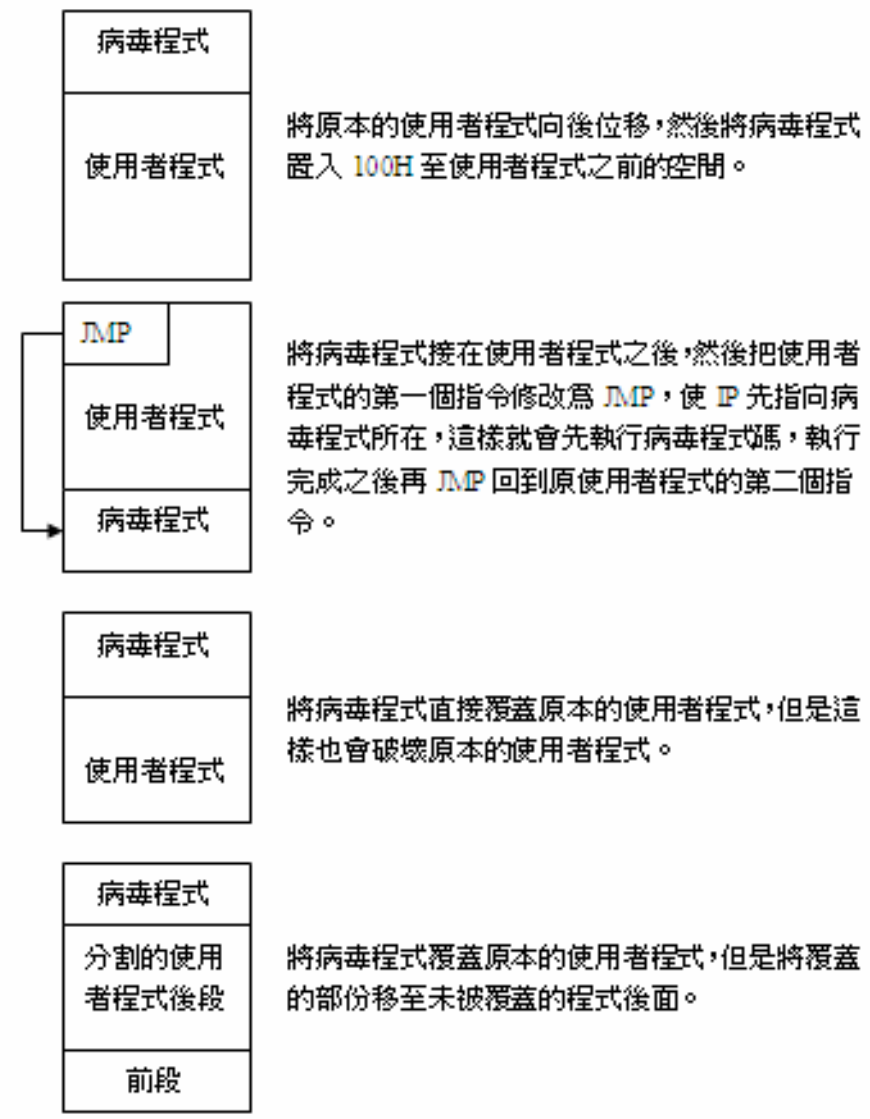
如果病毒想要對原本的程式碼動手腳就會十分困難，很可能在修改之後造成檔案無法執行，當然也無法執行到病毒本身的惡意程式碼，這樣看來，唯一會成爲攻擊目標的就是檔頭了，表一是EXE檔的檔頭資料：

在與使用者藉由PSP協調完成之後，系統就會根據檔頭內容來配置記憶體並設定各個暫存器的值，如前所述，IP存放了系統執行程式第一個指令的位址，因此只要修改檔頭中IP的值，就會使系統自IP指向的指令開始執行，並非如COM檔一般一定從程式區第一個指令開始，所以，病毒感染EXE執行檔時，除了將自身的惡意程式寫於檔案最後之外，還需修改檔頭資訊中的IP及CS(Code Segment，CPU暫存器，指向程式區的開頭)，使開始執行程式之前，病毒就取得控制權，先將惡意程式部分執行完畢，圖七是EXE檔案中毒前與中毒後的比較。

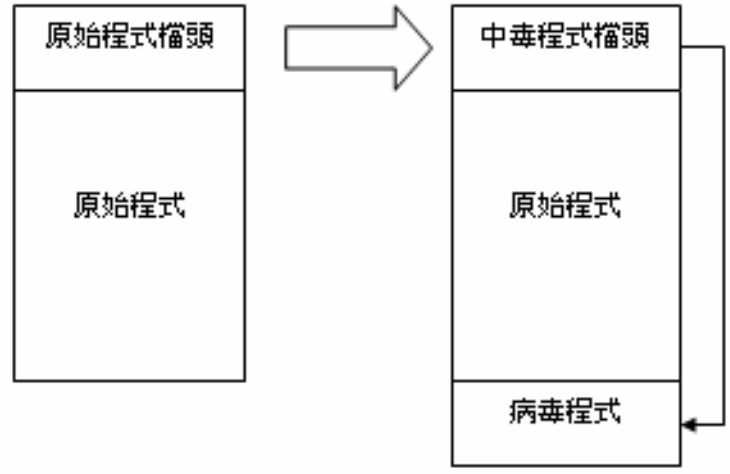
三、軟體自我保護的基本概念

3.1 軟體自我保護之概念

假設展示於圖八(a)中的F1，是一個我們想要保護，使其免於受到病毒攻擊的檔案，而P則是儲存在檔案F1內的程式，軟體自我保護這個技術，最基本的想法就是在P前面放入一段額外的新程式碼Ⅰ，來形成一支新的程式，我們叫她IP，如圖八(b)，接著，開始



圖六： COM檔遭病毒感染之各情形



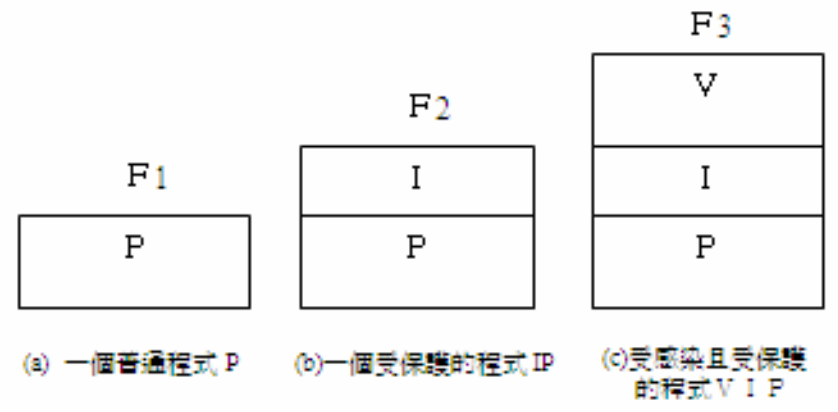
圖七： EXE檔遭病毒入侵情形

表一：EXE檔頭資料表

相對位置	長度	內容
00h	2	EXE檔案的標誌(4D5A)
02h	2	檔案長度除以512的餘數
04h	2	檔案長度除以512的商(若有餘數再加1)
06h	2	移位表項數
08h	2	檔頭長度
0ah	2	程式之後額外所需最少記憶體大小
0ch	2	程式之後額外所需最多記憶體大小
0eh	2	程式SS值
10h	2	程式SP值
12h	2	查核字元
14h	2	程式IP值
16h	2	程式CS值
18h	2	檔案內第一個移位單位的距離
1ah	1	後續檔案編號
1bh		保留區
		移位表
		保留區
		程式區

執行IP之後，I會對IP進行檢查，如果程式Ｉ發現到IP已經受到病毒的感染，Ｉ將給使用者發送一個警告訊息，接著把IP還原，而Ｉ所使用的偵測方法，則可能參考各種不同的檢查技術，例如總和檢查（Checksum）、程式檔案大小檢查（Program File Size Check）、最後更新時間檢查（Latest Update Date Check）、循環冗數檢查（CRC，Cyclic Redundancy Check）……等等方式。

當IP遭到病毒V的感染，此時IP轉變成新的程式，我們用程式VIP稱之如圖八(c)，執行VIP時，首先會執行病毒程式V，接著執行Ｉ，此時Ｉ進行偵測必定會產生錯誤，也就是說程式IP受到病毒或是其他惡意程式的改變或感染，接著Ｉ就會利用事先備份的原始程式拷貝，將受感染的VIP復原成乾淨而且受保護的程式IP，我們稱Ｉ爲保護碼(Protection Code)，IP(Protected Program)爲受保護的程式，而VIP爲受感染但受保護的程式。



圖八： 軟體自我防衛系統圖解

3.2 三個大問題

爲了建立一個健全又可靠的軟體自我防衛系統，我們必須先解決下面三個問題，第一個問題，病毒可能會修改我們的保護碼I，使得IP可以接受甚至繁殖這隻病毒，例如破壞I的內容使之無法正常運行，或者直接將I從程式IP中移除。

第二個問題就是偵測有可能會失敗，舉個例子來說，如果你使用總和檢查（checksum）、修改時間檢查、循環冗數檢查（CRC），或是其他複合式的檢查演算法，這些方式都有可能在很短的時間之內，遭到十分簡單的演算法攻擊，例如找到你存放checksum、CRC的位址然後修改它們，我們應該想辦法讓這些不法者的非法行爲變的更困難。

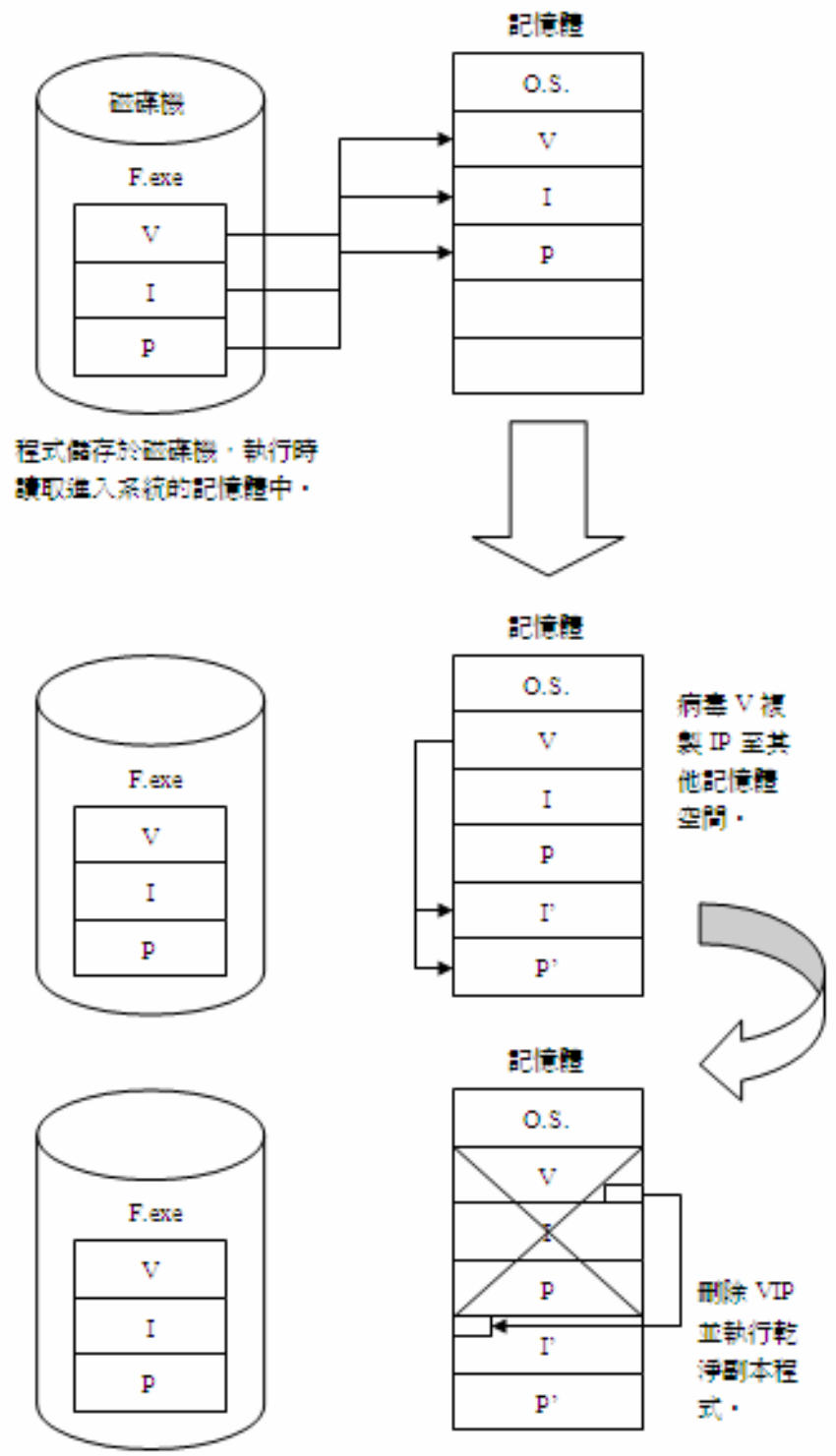
第三個問題稱爲對自我防護機制的一般性攻擊(Genetic Attack)，用一般性稱之是因爲不管你使用任何一種自我防護技術，這種攻擊肯定有效，一種攻擊，全部適用，假設程式VIP是一個受感染但受保護的程式，一般性攻擊病毒碼V將會有以下的行爲（如圖九）：

(1) 程式VIP載入主記憶體並開始執行。
(2) 程式VIP執行程式碼V：
    (a) 執行V之惡意函式。
    (b) 複製一份程式IP’至其他記憶體空間中。
    (c) 完成其他惡意行爲。
    (d) 刪除VIP。
    (e) 行程結束開始執行副本IP。
(3) 正常結束乾淨的程式IP。

雖然IP執行時使用者感覺不到病毒，但病毒其實已經達到了目的。

3.3 三個問題的解決方案

目前所提供的解決方案，出自我們要改良的論文，我們將提供他所敘述的解決方案，接著第四章再提出我們的改良方案。要解決第一及第二個問題，可以截長補短，利用一些現有技術的優點來克服，首先，在受保護的程式IP中的保護碼I可以利用數位簽章的技術來產生，我們可以在I中存放一個數位簽章H(IP)，H是一個雜湊函數而IP是一個受保



圖九： Generic Attack圖解

護的程式，因此，當程式IP執行時，整個程式包括I及P都被檢查到了，除此之外，爲了達到資訊隱藏的目的，大部分的程式碼I都經過加密，甚至經過壓縮，除了一小段放在開頭用來說明的函式，最後，根據多態性病毒概念的優點，每次程式IP執行時，保護碼I會藉由其他的加密或壓縮函式將自己改變成爲I’，接著產生一個新的程式IP’來覆蓋原本的程式IP，換句話說，IP’是由新的保護碼I’及程式P的程式碼來組成。

要妥善處理第三個問題，我們將使用三個程式來建立一個健全的軟體自我防護系統，以下爲原文演算法：

A： A program to be protected
C1 and C2： our integrity checking programs
F2＝FB || F0 || H（FA）
where symbol ”||” represents concatenation
F1＝FA || H（F2）
H： A secure hash function（one way）satisfying the following property :
Given X, it is computationally infeasible to find Y
H(F2): the hash value of file F2
K: The number of bytes in file FA
F1 and F2 together create a protected A with self-defense capability.
To run A, we execute file F1.
execute file F1: (i.e. run C1)
read file F2 into W
compute H(W) into X
read H(F2) from F1 into Y
if (X=Y) execute file F2 else print(“file F2 has problem”)
execute file F2: (i.e. run C2)
read file F2 into V
compute H(V) into X
read the first K bytes from F1 into W
if(H(W)≠H(FA))
then stop run
compute H(W||X) into Y
read file F1 into V
compute H(V)into Z
if (Y=Z) { print(“OK”); read A from F2 into file B; execute B;
modify F1 and F2 with new C1 and new C2: delete B}
else print(“file F1 has problem”)

3.4 先前論文的缺失與可改良之處

在以上解決問題的方法中，與演算法相對照之下，尚有幾個未完成的想法，或是缺失：

(1) 第一個就是對保護碼I或是我們要保護的程式IP進行安全性加密或壓縮，原文演算法之前的說明有提到這個構想，但是卻沒有在演算法中實行或演練，因此我們將提出新的構想結合千面人病毒的優點，來改良這個可以改進的地方。

(2) 第二個有缺失的地方就是，演算法與所提供的說明圖形無法配合，而且欠缺周詳，使得演算法與圖形無法配合，這也是我們將要改良的缺點。

四、改良式的軟體自我保護方法

4.1 改良方法

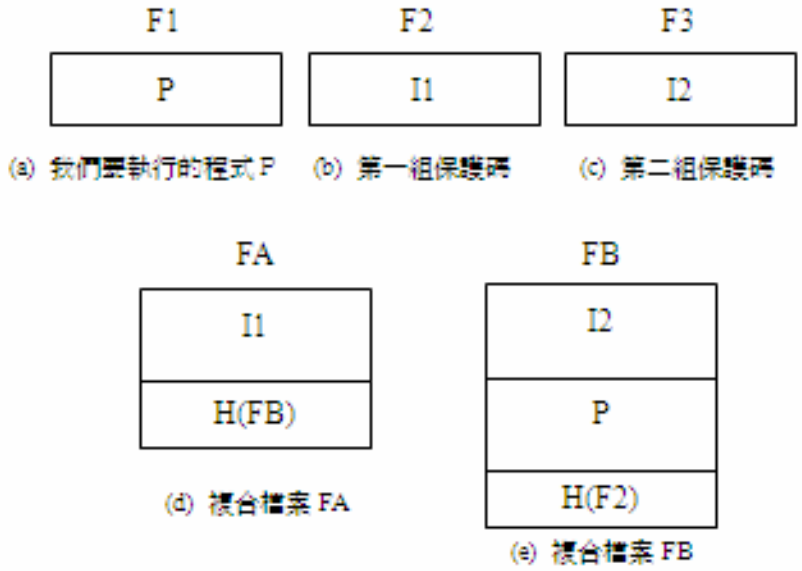
現在我們提出改良式的方法來改進以上所產生的問題，要解決第一及第二個問題，我們還是截長補短，利用一些現有技術的優點，首先，爲了達到資訊隱藏的目的，使得病毒無法對I進行破壞及對我們的檢查函數進行竄改，除了一小段放在開頭用來說明的函式，大部分的程式碼IP都經過加密或壓縮，並結合千面人病毒變形引擎概念的優點，每次程式IP執行後，IP會藉由加密或壓縮函式將自己改變成爲IP’，接著用這個新的程式IP’來覆蓋原本的程式IP，換句話說，IP原始內容不變，但是經過加密之後產生了新的外觀，而且每次都不相同。

要妥善處理第三個問題，我們使用三個程式搭配數位簽章的觀念，來建立一個健全的軟體自我防護系統，將檔案的內容視爲一串二進位數字的組合，利用一個雜湊函式，把檔案內容轉換爲一段訊息摘要，然後利用比對摘要的方式來判斷檔案的內容是不是有經過改變，這段訊息摘要可以儲存在其他檔案裡面，然後要開啓檔案的同時也開啓儲存有數位簽章的檔案來比對，這樣就可以有效解決第三個問題了。

接下來，我們將提出一個改良式演算法來說明以上解決問題的概念。

4.2 改良演算法

圖十爲改良演算法之概念圖。



圖十： 演算法概念圖

元件定義：

- H(X)代表X的數位簽章，H是一個單向安全雜湊函式(例如：MD5)，滿足給定X則趨近於不可能找到Y != X而使H(Y)=H(X)。
- 檔案F1，內容是我們預定要執行的程式P。
- 檔案F2，內容是我們的第一組保護碼I1。
- 檔案F3，內容是我們的第二組保護碼I2。
- 檔案FA，內容是由保護碼I1與檔案FB的數位簽章H(FB)所組成。
- 檔案FB，內容是由保護碼I2與預定要執行的程式P、K以及檔案FA的數位簽章H(FA)所組成。
- K代表檔案F2中保護碼I1的Byte數，可能儲存於I2中或程式結尾。

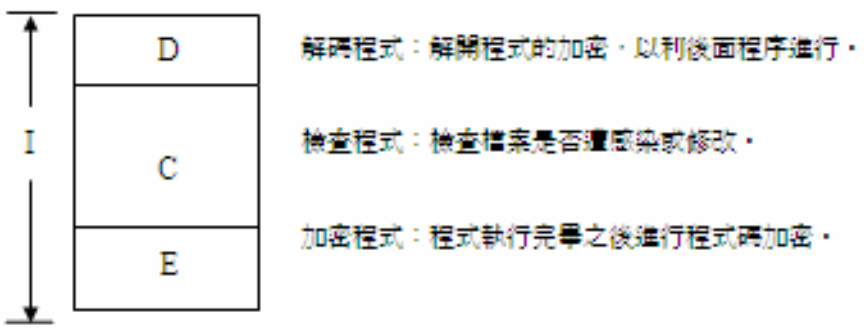
爲了執行P，我們首先讀取FA，並開始執行I1，並開始執行以下程序：

```
decode the main source code
open FB and read all data into W
compute H(W) as X1
read H(FB) as Y1
if (X1 == Y1) executeFB() ;
else {
    print (“There are problems in FB !”) ;
    recover FB ;
}

// FA執行完成無誤之後導向FB，並執行下列程序：

decode the main source code
read the first K bytes from FA into V1
compute H(V1) as X2
read all H(F2) into W1 from FB
compute H(W1) as Y2
if(X2 != Y2) {
    print (“There are problems in FA !”) ;
    recover FA ;
}
read all data from FB into V2
compute H(V2) into X3
compute H(V2 || X3) as Y3  //  symbol ”||” represents concatenation
read all data from FA into V3
compute H(V3) as Z
```

```
if (Y3 == Z) {
    print (“OK !”) ;
    execute P ;
    encode the main source code in FB ;
        encode the main source code in FA ;
}
else {
    print (“There are problems in FA !”) ;
    recover FA ;
}
```

4.3 演算法解說

接下來我們將對以上演算法進行較詳細的解說，首先是我們的主角I，I是我們的保護碼，具有解密、檢查檔案、加密這三項主要的功能，雖然我們在上面的演算法中將I視爲一個元件來操作他的功能，但是詳細分析之後，I可以分爲三個部份來說明，如圖十一。



圖十一：保護碼I分析圖

當程式載入記憶體開始執行時，首先會執行I中的D，D負責將先前加密過的C及E甚至是P進行解碼的工作，如果沒有解碼，那後面的程式就只是一堆看似雜亂無章的亂碼而已，必須經過解碼才能正常執行，C則是負責利用檔案讀取與進行雜湊運算比對數位簽章的方式，來偵測檔案是否受到感染或是修改，而E則進行最後執行完畢之後的再次加密，這裡的加密技術使用到先前的研究[3]，也就是變形引擎(Mutation Engine)的概念，每次加密都可以使I或IP的外觀與先前不同，自然而然也就使病毒無法對之修改，如果病毒強行破壞的話，程式偵測發現錯誤，就會使用之前存有的備分進行復原。

我們偵測錯誤的作法是，利用數位簽章比對的方式，來判別程式是否受到病毒的感染，正如我們之前所述，遭到病毒感染的檔案，檔案內容一定會受到病毒的改變，以利病毒本身惡意行爲的進行而不會被使用者查覺，然後再透過程式互相比對的方式，來確定沒有受到之前所提的一般性攻擊(Genetic Attack)，在演算法中這兩個互相比對的檔案，我們以FA及FB來代表，而用以比對的數位簽章則分別是存放在FA中的H(FB)，以及存放在FB中的H(FA)，現在我們已經有了確定乾淨的數位簽章，只要取得程式執行

時，想要比對之檔案的數位簽章即可進行比對，我們都知道，所有程式內容都是以二進位的方式儲存於磁碟機中，讀取進入系統執行時也是以二進位的方式來解讀，也就是說，不管什麼檔案其實都可以看成是一長串的二進位數字，即使執行檔亦如是，因此我們在製作數位簽章時便是利用此條件，將欲比對之檔案內容全部視爲一串很長很長的二進位資料，然後利用雜湊函式轉換成數位簽章，而我們也可以利用同樣的方式，立即計算取得任何檔案內容的數位簽章，請看演算法中的以下幾行：

```
open FB and read all data into W
compute H(W) as X1
/*----------------------以及----------------------*/
read the first K bytes from FA into V1
compute H(V) as X2
read all data from F2 into W
compute H(W) as Y2

/*----------------------以及----------------------*/

read all data from FB into V2
compute H(V2) into X3
compute H(V2 || X3) as Y3  //  symbol ”||” represents concatenation
read all data from FA into V3
compute H(V3) as Z
```

以上所進行的程序就是將我們想要比對的資料讀取進入系統的記憶體中，然後使用雜湊函式計算出雜湊值，也就是我們的數位簽章，接著再與之前就儲存於檔案中的數位簽章做比對，即可分辨出受感染的檔案，並進行復原的動作。

將受感染檔案復原的方式，我們可以預先將乾淨的檔案，複製一份本身的拷貝，接著進行壓縮，然後儲存於磁碟中的某處，當正本受到病毒感染，即可利用我們拷貝的副本加以覆蓋，達到復原受感染檔案的目的。

4.4 問題檢視

現在用我們提出的新演算法，對之前所述的三大問題進行檢視，看看演算法是否能有效的解決這三個問題：

(1) 病毒針對保護碼 I 進行破壞：演算法中的加密或壓縮，就是針對這個破壞方式的反制措施，因爲經過加密或是壓縮之後，病毒撰寫者無法輕易的分析出我們保護碼之長度、特徵，當然也就無法撰寫出能夠有效針對我們程式的病杜，這個問題也就輕易的解決了。

(2) 病毒針對檢核值進行竄改：同樣的，演算法中的加密或壓縮技術，也可以確保我們的檢核值不會被病毒撰寫者發現我們存放檢核值的位址，病毒也就無法破壞或篡改我們的檢核值了。

(3) 一般性攻擊：在我們的演算法中，預防一般性攻擊的方法就是利用三個檔案所組合成的兩個複合檔案FA、FB，將這兩個檔案於執行時，配合數位簽章的使用，做互相比對的動作，比對完成確定檔案沒有受到病毒感染之後，才執行我們所保護的檔案P，這樣一來，就可以避免一般性病毒攻擊，發現病毒並將受感染的檔案復原。

五、結論及未來發展

我們研究的結果，成功改善了先前的演算法[3]，並且配合自我變體引擎(Mutation Engine)，加強了檔案加密的部份，使得每次加密之後，檔案所呈現的型態都不同，大幅降低了遭到破解的可能性。

在未來的努力方面，我們可以結合各項新的技術，來讓這個技術更加完備、更有效率，當加密解密技術不停精進之後，或許我們現在所使用的加密技術會變的一點都不安全，所以隨時結合最新的科技技術，對我們的構想進行提昇改良，是十分必要的。

參考文獻

[1] 張真誠、李維斌，魔鬲

[2] 施威銘，80X86組合語言實務

[3] Peter Shaohua Deng、Wen-Gong Shieh、Jau-Hwang Wang、Cheng-Tan Tung、Chih-Pin Yen，”A Robust Method for Software Self-defense”

[4] R. A. Grimes, “Malicious Mobile Code – Virus Protection for Windows”, O’Reilly, 2001.

[5] F. B. Cohen, “A Short Course on Computer Viruses”, ASP Press, 1990.

[6] J. O. Kephart, G. B. Sorkin, Morton Swimmer, and Steve R. White , “Blueprint for a Computer Immune System”, Virus Bulletin International Conference in San Francisco, California, October 1-3, 1997.

[7] S. R. White, “Open Problems in Computer Virus Research”, Virus Bulletin Conference, 1998.

[8] Gerald Tesauro, Jeffrey O. Kephart, and Gregory B. Sorkin, “Neural Networks for Computer Virus Recognition”, IEEE Expert, 11(4):5-6. IEEE Computer Society, August, 1996.

[9] Wildlist Organization, “Virus descriptions of viruses in the wild”, Online publication, November, 2001. http://www.f-secure.com/virus-info/wild.html.

[10] W. Lee, S. Stolfo, and K. Mok, “A Data Mining Framework for Building Intrusion Detection Models”, IEEE Symposium on Security and Privacy, 1999.

[11] P. Kerchen, R. Lo, J. Crossley, G. Elkinbard, and R. Olsson, “Static Analysis Virus Detection Tools for UNIX Systems”, Proceedings of the 13th National Computer Security Conference, pages 350-365, 1990.