**Writing your own Tool Debugger**

**Why Debugging?**

Debugging is usually used to detect application bugs and traces its execution and also helps to know what went wrong in job execution. Also, it is used in reverse engineering and analyzing application when you don’t have the source code of this application. Reverse engineering is used mainly for detecting vulnerabilities, analyzing malware or cracking applications.

**Security Research and Development Framework**

This is a free open source Development Framework created to support writing security tools and malware analysis tools. And to convert the security researches and ideas from the theoretical approach to the practical implementation.

This development framework created mainly to support the malware field to create malware analysis tools and anti- virus tools easily without reinventing the wheel and inspire the innovative minds to write their researches on this field and implement them using SRDF.

In User-Mode part, SRDF gives you many helpful tools … and they are:

a. Assembler and Disassembler

b. x86 Emulator

c. Debugger

d. PE Parser, ELF Parser, PDF and Android Parsers

e. Process Analyzer (Loaded DLLs, Memory Maps … etc)

f. MD5, SSDeep and Wildlist Scanner (YARA)

g. API Hooker, IAT Hooker and Process Injection

h. Backend Database, XML Serializer

i. Pcap File Anaylzer

j. Packet Analysis

k. Protcol Analysis like: TCP, UDP, ICMP, HTTP, DNS and so on

l. Network Flow Analysis and Network Separation

And many more

In the Kernel-Mode part, it tries to make it easy to write your own filter device driver (not with WDF and callbacks) and gives an easy, object oriented (as much as we can) development framework with these features:

a. Object-oriented and easy to use development framework

b. Easy IRP dispatching mechanism

c. SSDT Hooker

d. Layered Devices Filtering

e. TDI Firewall

f. File and Registry Manager

g. Kernel Mode easy to use internet sockets

h. Filesystem Filter

Still the Kernel-Mode in progress and many features will be added in the near future.

**Gather Information about process**

If you decided to debug a running application or you start an application for debugging. You need to gather information about this process that you want to debug like:

1. Allocated Memory Regions inside the process
2. The Application place in its memory and the size of the application in memory
3. Loaded DLLs inside the application’s memory
4. Read a specific place in memory Also, if you need to attach to a process already running … you will also need to know the Process Filename and the command line of this application.

**Begin the process Analysis**

To gather the information about a process in the memory, you should create an object of cProcess class given the ProcessId of the process that you need to analyze.

cProcess myProc(792);

If you only have the process name and don’t have the process id, you can get the process Id from the ProcessScanner in SRDF like this:

cProcessScanner ProcScan;

And then get the hash of process names and Ids from ProcessList field inside the cProcessSanner Class …and this item is an object of cHash class. cHash class is a class created to represent a hash from key and value … the relation between them are one-to-many … so each key could have many values. In our case, the key is the process name and the value is the process id. You could see more than one process with the same name running on your system. To get the first ProcessId for a process “Explorer.exe” for example … you will do this:

ProcScan.ProcessList["explorer.exe"]

This will return a cString value includes the ProcessId of the process. To convert it into integer, you will use atoi() function … like this:

atoi(ProcScan.ProcessList["explorer.exe"])

**Getting Allocated Memory**

To get the allocated memory regions, there’s a list of memory regions named MemoryMap the typeof this Item is cList. cList is a class created to represent a list of buffers with fixed size or array of a specific struct. It has a function named GetNumberOfItems and this function gets the number of items inside the list. In the following code, we will see how to get the list of Memory Regions using cList Functions.

for(int i=0; i<(int)(myProc->MemoryMap.GetNumberOfItems()) ;i++)

{

cout<<"Memory Address "<< ((MEMORY\_MAP\*)myProc->MemoryMap.GetItem(i))->Address;

cout << " Size: "<<hex<<((memory\_map\*)myproc->MemoryMap.GetItem(i))->Size << endl;

}

The struct MEMORY\_MAP describes a memory region inside a process … and it’s:

struct MEMORY\_MAP

{

DWORD Address;

DWORD Size;

DWORD Protection;

};

In the previous code, we loops on the items of MemoryMap List and we get every memory region’s address and size.

**Getting the Application Information**

To get the application place in memory … you will simply get the Imagebase and SizeOfImage fields inside cProcess class like this:

cout<<"Process: "<< myProc->processName<< endl;

cout<<"Process Parent ID: "<< myProc->ParentID << endl;

cout<< "Process Command Line: "<< myProc->CommandLine << endl;

cout<<"Process PEB:\t"<< myProc->ppeb<< endl;

cout<<"Process ImageBase:\t"<<hex<<>ImageBase<< endl;

cout<<"Process SizeOfImageBase:\t"<<dec<<>SizeOfImage<< " bytes"<< endl;

</dec<<>

As you see, we get the most important information about the process and its place in memory (Imagebase) and the size of it in memory (SizeOfImage).

**Loaded DLLs and Modules**

The loaded Modules is a cList inside cProcess class with name “modulesList” and it represents an array of struct “MODULE\_INFO” and it’s like this:

struct MODULE\_INFO

{

DWORD moduleImageBase;

DWORD moduleSizeOfImage;

cString\* moduleName;

cString\* modulePath;

};

To get the loaded DLLs inside the process, this code represents how to get the loaded DLLs:

for (int i=0 ; i<(int)( myProc->modulesList.GetNumberOfItems()) ;i++)

{

cout<<"Module "<< ((MODULE\_INFO\*)myProc->modulesList.GetItem(i))->moduleName->GetChar();

cout <<" ImageBase: "<< hex <<((MODULE\_INFO\*)myProc->modulesList.GetItem(i))->moduleImageBase << endl;

}

**Read, Write and Execute on this Process**

To read a place on the memory of this process, the cProcess class gives you a function named Read(…) which allocates a space into your memory and then reads the specific place in the memory of this process and copies it into your memory (the new allocated place in your memory).

DWORD Read(DWORD startAddress,DWORD size)

For writing to the process, you have another function name Write and it’s like this:

DWORD Write (DWORD startAddressToWrite ,DWORD buffer ,DWORD sizeToWrite)

This function takes the place that you would to write in, the buffer in your process that contains the data you want to write and the size of the buffer.

If the startAddressToWrite is null … Write() function will allocate a place in memory to write on and return the pointer to this place.

To only allocate a space inside the process … you can use Allocate() function to allocate memory inside the process and it’s like that:

Allocate(DWORD preferedAddress,DWORD size)

You have also the option to execute a code inside this process by creating a new thread inside the process or inject a DLL inside the process using these functions

DWORD DllInject(cString DLLFilename)

DWORD CreateThread (DWORD addressToFunction , DWORD addressToParameter)

And these functions return the ThreadId for the newly created thread.

**Debugging an Application**

To write a successful debugger, you need to include these features in your debugger:

Could Attach to a running process or open an EXE file and debug it

Could gather the register values and modify them

Could Set Int3 Breakpoints on specific addresses

Could Set Hardware Breakpoints (on Read, Write or Execute)

Could Set Memory Breakpoints (on Read, Write or Execute on a specific pages in memory)

Could pause the application while running

Could handle events like exceptions, loading or unloading dlls or creating or terminating a thread.

**Open Exe File and Debug… or Attach to a Process**

To open and Exe File and Debug it

cDebugger\* Debugger = new cDebugger("C:\\upx01.exe");

Or with command line

cDebugger\* Debugger = new cDebugger("C:\\upx01.exe","xxxx");

If the file opened successfully, you will see IsFound variable inside cDebugger class set to TRUE. If any problems happened (file not found or anything) you will see it equal FALSE. Always check this field before going further.  
If you want to debug a running process … you will create a cProcess class with the ProcessId you want and then attach the debugger to it:

cDebugger\* Debugger = new cDebugger(myProc);

to begin running the application … you will use function Run() like this:

Debugger->Run();

Or you can only run one instruction using function Step() like this:

Debugger->Step();

This function returns one of these outputs (until now, could be expanded):

DBG\_STATUS\_STEP

DBG\_STATUS\_HARDWARE\_BP

DBG\_STATUS\_MEM\_BREAKPOINT

DBG\_STATUS\_BREAKPOINT

DBG\_STATUS\_EXITPROCESS

DBG\_STATUS\_ERROR

DBG\_STATUS\_INTERNAL\_ERROR

If it returns DBG\_STATUS\_ERROR, you can check the ExceptionCode Field and the debug\_event Field to ge more information.