2010 3rd International Conference on Computer and Electrical Engineering (ICCEE 2010)
IPCSIT vol. 53 (2012) © (2012) IACSIT Press, Singapore
DOI: 10.7763/IPCSIT.2012.V53.No.1.61

Design and Implementation of a Process-Protecting System in Windows NT

Guo Chun[+] and Jiang Chao-Hui

School of Computer Science and Information,
Guizhou University,
Guiyang 550025, China

Abstract. One of the pressing problems facing security software today is that the defense strength for its process is not high, which makes the self-protection of software so inadequate that malicious application can terminate it by some ways. After analyzed the current status of the technology of process protection, a process-protecting project which is proposed in this paper, utilizes SSDT hook, inline hook and technique of two processes and three threads to construct a process-protecting system of high strength and multi-layered. This system contains three prevention mechanisms——monitor, guard, restore, can remedy the deficiency in a lot of security software that strength of software self-defense is not high, and enhance the ability of self-protection about software. Finally, experimental testing validate that the system has a strong ability to protect the process and terminate the malicious program to eliminate its threat in Windows. Therefore, it is effective for this system to further safeguard the safe operation of security software.

Keywords: SSDT; hook;inline hook; process protection

1. Introduction

The process of security software does not shut down unexpectedly by illegal operation when running, is the base that it plays ability, which raises the importance of process protection. With the rapid development of information technology, the technology of process protection is upgraded frequently. Especially in the past two years, the research and application in RootKit is popular, drive the rapid development in technologies of process protection and forced termination of process. So far, the mainstream technologies of process protection are mainly as follows:

- Hidden process [1]. It makes the process of software do not appear in the list of various application-tools about process management through thread injection technique or Hook API technology.
- Technique of two processes and three threads. Using the code injection to make local and remote threads monitor each other, as long as one of them is terminated, the other thread would start it immediately [2].
- SSDT hook [3]. SSDT (System Services Descriptor Table) is a role which relates the Win32 API of ring3 to kernel API of ring0. It would protect process effectively By SSDT hook NtOpenProcess () or NtTerminateProcess().
- Inline hook. Inline hook is more bottom than SSDT hook, there are studies using inline hook KiInsertQueueApc () to achieve a high strength protection of process.

Experimental results show that the above techniques have strong ability to protect process ,which can counter the feature of "the process of forced termination" in some software of process management, however, today's emerging technologies of forced termination of process, make the existing technology of process protection facing with a very strong challenge:

- Process with protection by technique of two processes and three threads, will be terminated when closing the protected process and the daemon at the same time, so it will lose the protection when in the face of the

[+] Corresponding author.
E-mail address: chung1@163.com.

technique of "strong kill multi-threaded" which can also terminate the daemon process and protected process at one time.

- Process protection by using SSDT hook is achieved through hook the function of NtTerminateProcess(), but because its implementation does not bottom enough, so that malicious program easy to bypass, Such as calling the API of Psp-series which are not directly derived from ntosXXX (mainly include two methods which are PspTerminateProcess and PspTerminateThreadByPointer), they can release its protection.
- After inline hook KiInsertQueueApc, it can withstand “strong kill” which including calling NtTerminateProcess and methods of PspTerminateProcess or PspTerminateThreadByPointer. However, the current study only inline hook function of KiInsertQueueApc to filter out the APC which malicious program tries to insert, and no measures to terminate the malicious program. Therefore, the threat that malicious program bring to protected process still does not eliminate thoroughly.

Nowadays, security software in achieving self-protection always uses one of several kinds of process-protecting technologies the above mentioned, which makes its protective measures easy to bypass by malicious program. It results the software in a lack of self-protecting, and can not eliminate the existing threats giving by malicious program. Given the importance of process protection, this paper presents an integrated project which using the latest technology to protect the process of selected software. It makes use of SSDT hook to hide process and to monitor when new process creating, inline hook KiInsertQueueApc and PsLookupThreadByThreadId to guard process and to kill the process of the malicious program which intend to end protected process, what’s more, this system makes use of technique of two processes and three threads to restore the protected process which shut down unexpectedly. The three techniques construct a process-protecting system of high strength and multi-layered, which contain three prevention mechanism——monitor (in advance), guard (in the process), restore (after be ended).

2. Design of Process-Protecting System Based on Windows Kernel

2.1 Design Ideas

The design concept of this system is based on protection of three layers:

Monitor. First protective layer will intercept any new process which intends to build in operating system, and judge the creation of new process by users whether agree or reject its establishment with the purpose of preventing the illegal process from performing. Meanwhile, in order to make user easy to use this function, this module increases two feature modules of blacklist of process and white-list of process. When a new process creates at the first time, if user selects the button of "ban", the process would be blocked from loading, at the same time, the information adds to the blacklist of process, when detect the process would create one more time, it will directly return to failure; If user selects the button of "allow", the process information will be added to the white-list of process, and it does not need to re-certification next time.

Guard. The second protective layer makes use of inline hook to protect process at Kernel-level. For the process which user selects to protect, this layer will provide it with high strength protection, which make the process would not be obligatorily interrupted by other software during operation. Meanwhile, try to end the process of procedures which intent to terminate the protected process. Therefore, this layer can keep the protected process running normally.

Restore. The implementation of the third protective layer is that injecting the protection-code to a daemon created by author with technique of two processes and three threads. When the previous two layers of protection have been breached, the process will restart immediately through daemon.

2.2 System Design

Designing to achieve the functions of monitoring and protecting in Windows system, the key is intercepting the relative operations of process which are implemented by Windows at appropriate areas, and filtering the operations of process according to rules of user settings. Filtering operation on the process needs to achieve in kernel mode, and implementation of the user rules is set in the user mode. Therefore, this system combine the technology of SSDT hook and inline hook in kernel mode with inserting API in user mode, two modes primarily connect each other through the function of DeviceIoControl. Concrete implementation of this system is that its driver is developed by WDK7600.16385.0 version, and the driver loading, unloading, interacts with user, etc, using VC6.0 development environment to develop interface of application program. The overall architecture of the tool is shown in Figure1.

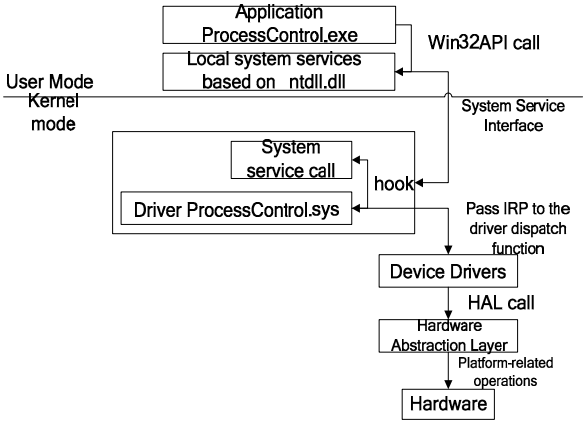


Figure 1. System Architecture

Accordingly, top-level modules of the system include monitoring module which monitor new process created real-time, the guard module giving guard on process, the recovery module with technique of two processes and three threads. System module structure is shown in Figure2.

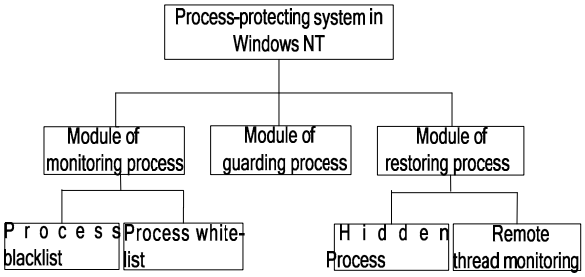


Figure 2. System module diagram

3. Implementation of Process-Protecting System Based on Windows Kernel

3.1 Implementation of Module of Monitoring Process

In Windows systems, the creation of process includes the following steps [4]:①open .exe file with FILE_EXECUTE access;②load executable image into memory;③establish executable object of the process (EPROCESS, KPROCESS and PEB structures);④allocate address space for the new process;⑤build executable object of thread for main thread of the new process (ETHREAD, KTHREAD and TEB structures);⑥allocate stack for main thread of new process;⑦build executable context for main thread of the new process;⑧inform the new process to subsystem of Win32.

At each step of process creating, it needs to call local API function. Therefore, in order to effectively interrupt creation of process, you can use the native API hook to hook any of the local API function which can not skip in the process of process created, which can gain full control over the process created. The function of ZwCreateProcess in ntdll.dll is an API function which can not skip during the process of creating. Therefore, this module would find ID number of ZwCreateProcess in system from ntdll.dll at the application layer, then passes it to the driver, and changes the content of ID number of this service in the system [5].

After interrupting creation of the process, this module calls system service ZwCreateProcess to complete the work that obtaining process information at the kernel space [6]. The prototype of NtCreateProcess is as follows:

```
NtCreateProcess
(PHANDLE phProcess,
ACCESS_MASK DesiredAccess,
POBJECT_ATTR IBUTESObject Attributes,
HANDLE hParentProcess,
BOOLEAN bInheritParentHandles,
HANDLE hSection OPTIONAL,
HANDLE hDebugPortOPTIONAL,
HANDLE hExceptionPortOPTIONAL)
```

As the project intercepts the process creating at step ② in process of creation, and now step ① has opened the image which will be implemented, parameters of hSection are result of Step ①, and the parameters associate with the

file object, Therefore, the process can get the path information which is about created process. Concrete implementation of the program in the kernel is as follows: Call ObReferenceObjectByHandle to obtain the pSection that is a section object pointer which corresponding to the hSection, it can find pointer of SEGMENT structure through the data structure of section object, SEGMENT object in the data structure has a pointer of the structure CONTROL_AREA, what we find is file object pointer PFILEOBJECT in structure CONTROL_AREA. Finally, it can get the full path that the process is created through PFileObject calling ObQueryNameString, also can use PFileObject-> FileName to get the file name and relative path ,and use the drive getting by PFileObject-> DeviceObject to get he full file name of the process which is created.

After intercepting the creation of process in the kernel, it hangs the process to manage by the application layer at first. Then passes result to the driver if the result is legitimate , adds information of the process in white-list of process and executes the hijacked function of ZwCreateProcess, finally ,the result goes back and restores the suspended process; Otherwise, stops the process in kernel space, and adds its records in the blacklist of process to prevent the process re-creation next time.

3.2 Implementation of Module of Guarding Process

Recent studies in Windows kernel have shown that the reason of a process ended, is process called PspExitThread itself to end thread in running through KelnitializeApc / KelnsertOueueApc inserting APC into the kernel mode or user mode in system. For this reason, the process-protecting system filters out the thread of protected process to combat APC through inline hook KilnsertQueueApc and PsLookupThreadByThreadId. Meanwhile, in order to enhance the ability of guarding process, this module adds "anti-kill" code in the driver to kill the process of malicious software which intends to terminate the protected process. Therefore, this module can eliminate the threats from malicious software thoroughly.

The principle of inline hook is that analyzing several instructions at the beginning of function, copy the instructions to save at the array, then call a custom function to replace the several instructions, if intending to perform the original function, you must finish custom function firstly, then execute the several instructions which are saved up, finally, jumping to the address where we take instructions to execute [7].

According to the principle of inline hook, the implementation of inline hook is as follows:

Location address of KilnsertQueueApc. As KilnsertQueueApc is a function that does not export from kernel directly, it must determine its position according to the known KeInsertQueueApc or process of other functions. After have found the start address of KiInsertQueueApc, it will jump to the address and save the original function at the beginning of 5 bytes.

Modify KiInsertQueueApc at the beginning of 5 bytes. Remove the Write-protect in system, change the 5 bytes at the beginning of KiInsertQueueApc and jump to the custom function of FakeKiInsertQueueApc.

To achieve a custom function of FakeKiInsertQueueApc. The function needs to get the process name. Here use IoThreadToProcess ((PETHREAD) ExitApc-> Thread) to convert thread into process, which can determine the process's name. The key code to achieve the function is as follows:

```
  BOOLEAN _stdcall FakeKiInsertQueueApc（INPKAPC apc,IN PVOID SystemArgument1,IN
PVOID SystemArgument2,INKPRIORITY PriorityBoost）
  {
  PEPROCESS pTargetProcess;
  PUCHAR pTargetProcessName;
  ULONG XXThread;
  // Covert thread into the process
  pTargetProcess=
  IoThreadToProcess((PETHREAD)Apc->Thr-ead);
  // Get the process name
  pTargetProcessName=PsGetProcessImage-FileName (pTargetProcess);
  // Judge whether the system process will be ended through process name.
  if((strcmp(pTargetProcessName,"ProcessProtect.exe"))||PriorityBoost!=2||Apc!=S
ystemArgument1)
  {goto Call_KeInsertQueueApc;}
  else // location CrossThreadFlags address
  {
  XXThread=
 (ULONG) Apc->Thread + g_ThreadFlagsOffset;
```

```
// Amendment thread's CrossThread Flags in order to ensure that it can insert APC next time.
_asm{
mov eax,XXThread
and [eax],0xfffffffe
}
// Change thread in the APC into the pointers which system performs operation of inserting APC currently in order to achieve "anti-kill".
Apc->Thread=KeGetCurrentThread();
goto Call_KeInsertQueueApc;
}
}
```

Write uninstall function of UnFakeKiInsertQueueApc. It needs to implement that restoring Write-protect and reducing IRQL, and other operations.

What's more, this module also inline hook PsLookupThreadByThreadId to protect the system thread is not listed, enhance the capacity of guarding process.

3.3 Implementation of Module of Restoring Process

The key of implementation of the technique of three threads is monitoring the remote thread which is built by main thread in daemon. As accessing address space of other processes directly under Windows XP is mistaken, it is necessary to inject code and the required parameters into the daemon.

1) *Create a remote thread and injected into the daemon.*

First define a remote thread of execution function.

```
NTSTATUS__stdcall RemoteThreadFunc (RemoteProcess Data *rpd)
```

As need to execute the functions of required API in the remote thread, these functions must be called by remote thread though specifying the address of the remote thread:

```
typedef DWORD (WINAPI *EWaitForSingleObject) ( HANDLE, DWORD);
hkernel32 = GetModuleHandle (_T (″kernel32.dll″));
rp.pFunWaitForSingleObject= (EWaitForSingleObject) GetProcAddress (hkernel32, ″WaitForSingleObject″);
```

rp is the custom structure which is injected into the daemon. rp Contains the function's address, the process ID and the directory to start the process, and other information.

Next, the protect code should be injected into daemon. In order to prevent the instability after injected system processes, this module creates a new process as the target to inject code, and adopt writing virtual memory as injecting method.

```
//Allocate memory space for the process
lpAddress = Virtual AllocEx(hProcess, NULL,ulFuncLen,MEM_COM2MIT|MEM_TOP_DOWN, PAGE_EXECUTE_READWR ITE);
//Modify the authority of process memory
(VirtualProtectEx(hProcess,lpAddress, ulFuncLen, PAGE_EXECUTE_READWR ITE, &dwOldProtect))
// Clear the buffer structure of the process
FlushInstructionCache(hProcess,lpAddress,ulFuncLen)
//Write lpData to memory of process, lpData is the code which intend to inject, including the function of thread and its parameters
WriteProcessMemory(hProcess,lpAddres-s,lpData,ulFuncLen,&BytesWritten)
//Resume the authority of process memory
VirtualProtectEx(hProcess,lpAddress, ulFuncLen,dwOldProtect,NULL);
//Finally, create remote thread in a daemon thread
hInjectThread=CreateRemoteThread (hRemoteProcess,&saSecAttr,0,(LPTHREAD_START_ROUTINE)lpThreadFunc,(LPVOID)lp-Parameters,0,NULL);
```

hRemoteProcess is a handle which is injected into daemon, lpThreadFunc is returned function after injected code, lpParameters are returned address of parameter after injected code.

2) *Create a local thread of assistant*

This thread is mainly used to monitor the operation of guarding thread.

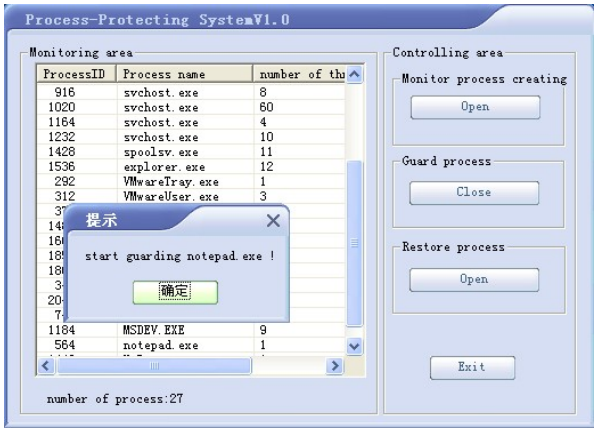
```
GetExitCodeThread(pThis->m_RemoteThreadHandle, &exitcode);
```

Setting a timer to persistently get status of protected thread, it is indicated that the remote thread has been closed if the exitcode does not equal STILL_ACTIVE, thread of assistant immediately calls function which is created by remote thread.

Also, users may doubt this unfamiliar process because the daemon is created by author. Solution is that making them transparent to the user through hiding the daemon. Hidden process is achieved through SSDT hook, implementation of function that getting process information is hooking function of ZwQueryInformationProcess which is derived from ntdll.dll. After hooking this function, when the query to the daemon, it returns a null value, which can realize the hidden daemon. Obviously, it just needs to get the handle of daemon and determine in custom function, then makes it return a null value if it is daemon. How to realize this module is similar to implementation of the module of monitoring process.

4. System Testing

The software successfully compiled and run in Windows XP Professional SP3, WDK7600.16385.0, VC + + 6.0, MFCv4.2. Software interface shown in Figure 3:



5. System Interface

The following are functional test cases in Table1, Table 2, and Table 3:

Table 1 Testing cases

Test context	***The function of monitoring process created***
Test step	(1)Open the function of "process monitoring" (2)Open an executable file at random. (3)Select "No" if the dialog popped (4)Open another executable file (5)Select "Yes" if the dialog popped
Test results	The system block the creation of new process, select bottom of "Yes" to create a new process and select bottom of "No" to forbid creation of this process.
conclusion	The system can completely control the creation of process

Table 2 Testing cases

Test context	***The function of guarding process***
Test step	(1)Close the function of "monitoring process" and "restoring process" (2)Select the process of "notepad.exe" in the list of process and open the function of "process guarding". (3)Use Task Manager to terminate the process of "notepad.exe". (4)Use software of "icesword" to terminate the process of "notepad.exe".. (5)Use software of "Yi-Tian windows Assistant" to terminate the process of "notepad.exe".
Test results	The protected process of "notepad.exe" have been running properly and task manager, icesword,Yi-Tian windows Assisant are exit.
conclusion	The system has a strong ability to guard process, and can realize the function of "anti-kill" for malicious programs.

Table 3 Testing cases

Test context	*The function of restoring process*
Test step	(1)Close the functions of "monitoring process" and "guarding process" (2)Select the process of "notepad.exe" in the list of process and open the function of "process restoring". (3)Use Task Manager to terminate the process of "notepad.exe". (4)Use software of "icesword" to terminate the process of "notepad.exe".
Test results	notepad.exe is restored immediately after be ended and ID number of its process is changed.
conclusion	The system can restore the protected process immediately when the process is terminated by accident.

6. Conclusion

The design and implementation of process-protecting system in this paper can provide high strength protection with the process of any specified procedures, which prevent the specified process from being terminated in running by malicious programs, so it can ensure that the application operates properly. Tests show that the system has a strong ability to protect process and provide assistant protection with security software in Windows system.

7. Acknowledgment

This work is supported by Science & Technology Project in Guiyang City, Guizhou Province, PRC (Grant No: ZHU KE GONG 2009-1-051)

8. References

[1] DENG Lujuan, LIU Tao, GAN Yong, XIONG Kun, "Active Defence Technologywith Virus Based on Differentiation and Hiding Process,"Computer Engineering, 2007,33 (5), pp.117-119.

[2] Ma J inxin,Yuan Ding. "ON PROCESS2 PROTECTING BASED ON WINDOWS AND ITS IMPLEMENTATION,"Computer Applications and Software, 2010,27(3),pp.18-21.

[3] Jiayuan Zhang,Shufen Liu,Jun Peng, "Techniques of user-mode detecting System Service Descriptor Table," Proceedings of the 2009 13th International Conference on Computer Supported Cooperative Work in Design.Santiago, Chile,2009,pp.96-101, doi: 10.1109/SCIS.2007.357670

[4] CHEN Chang-pin, Intercept technique of processes creation and its application in software experiments, Computer Engineering and Design, 2008.29(13), pp.3485-3487.

[5] Shen,Jianfang,Cheng,Lianglun,Fu,Xiufen, "Implementation of program behavior anomaly detection and protection using Hook technology," Proceedings - 2009 WRI International Conference on Communications and Mobile Computing. Kunming, Yunnan, China, 2009, pp.338-342, doi: 10.1109/CMC.2009.225.

[6] MENG Qing-qian, LI Qing-bao, WEI Min, "Windows, Design and Realization of Process-Monitoring Based on Windows，"Journal of Information Engineering University,2007,8(1),pp.26-29.

[7] Wang,Yulin, Shen, Yang, Pan, Jian, "Usage control based on windows kernel hook," 2009 International Conference on Information and Multimedia Technology. Jeju Island, Korea, 2009,pp.264-267, doi: 10.1109/ICIMT.2009.92