

Forum:[Hack Forums](#)

Thread: [Protect Process](#)

Diallix - October 2, 2018, 12:23 am

Hi. I spent a lot of time finding articles about Protect Process via set privileges or IAT hooking ZwTerminateProcess but without succesfull.

Is here somebody, who has code to protect process? My System is win8 - 64 bit.

Thanks very much.

3DES - October 2, 2018, 1:19 am

Inject shellcode / function to remote process that monitors your process and viceversa.

Or inject shellcode/dll to every process and hook NtQuerySystemInformation & NtOpenProcess

Diallix - October 2, 2018, 1:36 am

Do you have some example? Please.

Xamarin - October 2, 2018, 2:13 am

(Oct 2, 2018 01:36 AM)Diallix Wrote: [\[CITE\]](#) Do you have some example? Please.

Here you are:

<http://bfy.tw/K9qK>

<http://bfy.tw/K9qT>

But seriously, when you are asking for people to google things for you, this tells me that you are not mature enough to create this kind of stuff. Anyway, happy c+p.

Diallix - October 2, 2018, 2:31 am

I try to find on google, but not working for me.

oxidebuffalo - October 2, 2018, 7:40 am

Here are the common ways of protecting a process.

1. Kernel-Mode software which will patch the Windows kernel and thus block the termination operation for the process which should be protected. Limitations of this is that on x64 it will not work unless you use HV, otherwise you will get a bug check - I guess you can subvert all of that too but it is just effort for something so small.

2. Kernel-Mode software which will leverage officially documented and supported kernel-mode callbacks, like ObRegisterCallbacks, to restrict handle creation and/or duplication to the protected process.

3. User-Mode API hooking through patching virtual memory of other processes.

The simplest way would be the third one I mentioned, because it is extremely straight-forward to do and doesn't take much effort at all. With the kernel-mode examples, you'll have more work to do... because of kernel-mode requirements nowadays and on x64 systems since Vista there is PatchGuard anyway.

If you go with the third route, do not bother using IAT hooking and targeting ZwTerminateProcess. Instead, use a hotpatching technique by actually altering the instructions in memory at the target address to redirect to your own callback routine, and target a routine like NtOpenProcess instead, which will provide you with a larger process protection surface (e.g. if you cannot open the handle then you cannot use NtTerminateProcess, NtAllocateVirtualMemory, NtWriteVirtualMemory and so on).

The best thing you can do is stay file-less in the first place and not have to worry about hiding or protecting processes. Rely on file-less RCE and find a way to keep the persistence with the functionality and you'll be good to go.

If you have XMPP. Just send me your Jabber thingy and I'll add you.

Diallix - October 2, 2018, 10:46 am

Im try this:

```
#include <iostream>
#include <Windows.h>
```

```
#include <AccCtrl.h>
#include <AclAPI.h>
#include <tchar.h>
```

```
DWORD NtHookInstall(LPVOID lpTargetAddress, LPVOID lpCallbackAddress)
{
    if (lpTargetAddress == 0 || lpCallbackAddress == 0) return 0; //Misc. Check
```

```
    DWORD dwOldProtection = 0;
    if (VirtualProtect(lpTargetAddress, 7, PAGE_EXECUTE_READWRITE, &dwOldProtection) == 0)
        return 0;
```

```
    *(BYTE*)(lpTargetAddress) = 0xE9; // Opcode for JMP
    *(long*)((LPBYTE)lpTargetAddress + 1) = ((DWORD)lpCallbackAddress -
        (DWORD)lpTargetAddress + 5));
    VirtualProtect(lpTargetAddress, 7, dwOldProtection, &dwOldProtection); //reinstate original
    protection
    return 1; // SUCCESS
}
```

```
void Cals()
{
    ::std::cout << "vipnutie";
}
static const bool ProtectProcess()
{
```

```
    HANDLE hProcess = GetCurrentProcess();
    EXPLICIT_ACCESS denyAccess = { 0 };
    DWORD dwAccessPermissions = GENERIC_WRITE | PROCESS_ALL_ACCESS | WRITE_DAC |
        DELETE | WRITE_OWNER | READ_CONTROL;
    BuildExplicitAccessWithName(&denyAccess, _T("CURRENT_USER"), dwAccessPermissions,
        DENY_ACCESS, NO_INHERITANCE);
```

```

PACL pTempDacl = NULL;
DWORD dwErr = 0;
dwErr = SetEntriesInAcl(1, &denyAccess, NULL, &pTempDacl);
// check dwErr...
dwErr = SetSecurityInfo(hProcess, SE_KERNEL_OBJECT, DACL_SECURITY_INFORMATION,
NULL, NULL, pTempDacl, NULL);
// check dwErr...
LocalFree(pTempDacl);
CloseHandle(hProcess);
return dwErr == ERROR_SUCCESS;

}
BYTE MOV[10] = { 0x48, 0xB8, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0 };
BYTE JMP_RAX[2] = { 0xFF, 0xE0 };
#define BuffSizeX64 (sizeof(MOV) + sizeof(JMP_RAX))

BOOL Hook_Det_x64(char LibName[], char API_Name[], LPVOID NewFun) {
DWORD OldProtect;
DWORD64 OrgAddress = (DWORD64)GetProcAddress(LoadLibraryA(LibName), API_Name);
if (OrgAddress == NULL) return 0;

memcpy(&MOV[2], &NewFun, 8);
VirtualProtect((LPVOID)OrgAddress, BuffSizeX64, PAGE_EXECUTE_READWRITE, &OldProtect);
memcpy((LPVOID)OrgAddress, MOV, sizeof(MOV));
memcpy((LPVOID)(OrgAddress + sizeof(MOV)), JMP_RAX, sizeof(JMP_RAX));
VirtualProtect((LPVOID)OrgAddress, BuffSizeX64, OldProtect, &OldProtect);

return 1;
}

int WINAPI MessageBoxAX(
HWND hWnd,
LPCSTR lpText,
LPCSTR lpCaption,
UINT uType) {

MessageBoxExA(0, "Hooked ...", "Mahmoud", 0, 0);
return 999;
}

int main()
{
using namespace std;
int abfrage;

ProtectProcess();
while (1)
{
Hook_Det_x64("Kernel32.dll", "TerminateProcess", MessageBoxAX);

NtHookInstall(GetProcAddress(GetModuleHandleW(L"Kernel32.dll"), "TerminateProcess"), Cals);
NtHookInstall(GetProcAddress(GetModuleHandleW(L"ntdll.dll"), "ZwTerminateProcess"), Cals);
}
}

```

```

NtHookInstall(GetProcAddress(GetModuleHandleW(L"ntdll.dll"), "ZwTerminateThread"), Cals);
NtHookInstall(GetProcAddress(GetModuleHandleW(L"ntdll.dll"), "TerminateProcess"), Cals);
NtHookInstall(GetProcAddress(GetModuleHandleW(L"Kernel32.dll"), "NtTerminateProcess"), Cals);
NtHookInstall(GetProcAddress(GetModuleHandleW(L"Kernel32.dll"), "NtTerminateThread"), Cals);
NtHookInstall(GetProcAddress(GetModuleHandleW(L"ntdll.dll"), "NtQuerySystemInformation"),
Cals);
NtHookInstall(GetProcAddress(GetModuleHandleW(L"ntdll.dll"), "NtOpenProcess"), Cals);
NtHookInstall(GetProcAddress(GetModuleHandleW(L"ntdll.dll"), "ZwQuerySystemInformation"),
Cals);
NtHookInstall(GetProcAddress(GetModuleHandleW(L"ntdll.dll"), "ZwOpenProcess"), Cals);
}
return EXIT_SUCCESS;
}

```

but, i can terminate from taskmgr.

Please have some code for implement?

3DES - October 2, 2018, 11:51 am

(Oct 2, 2018 10:46 AM)Diallix Wrote: [\[CITE\]](#) Im try this:

```

#include <iostream>
#include <Windows.h>

```

```

#include <AccCtrl.h>
#include <AclAPI.h>
#include <tchar.h>

```

```

DWORD NtHookInstall(LPVOID lpTargetAddress, LPVOID lpCallbackAddress)
{
if (lpTargetAddress == 0 || lpCallbackAddress == 0) return 0; //Misc. Check

```

```

DWORD dwOldProtection = 0;
if (VirtualProtect(lpTargetAddress, 7, PAGE_EXECUTE_READWRITE, &dwOldProtection) == 0)
return 0;

```

```

*(BYTE*)(lpTargetAddress) = 0xE9; // Opcode for JMP
*(long*)((LPBYTE)lpTargetAddress + 1) = ((DWORD)lpCallbackAddress -
(DWORD)lpTargetAddress + 5));
VirtualProtect(lpTargetAddress, 7, dwOldProtection, &dwOldProtection); //reinstate original
protection
return 1; // SUCCESS
}

```

```

void Cals()
{
::std::cout << "vipnutie";
}
static const bool ProtectProcess()
{

```

```

HANDLE hProcess = GetCurrentProcess();

```

```

EXPLICIT_ACCESS denyAccess = { 0 };
DWORD dwAccessPermissions = GENERIC_WRITE | PROCESS_ALL_ACCESS | WRITE_DAC |
DELETE | WRITE_OWNER | READ_CONTROL;
BuildExplicitAccessWithName(&denyAccess, _T("CURRENT_USER"), dwAccessPermissions,
DENY_ACCESS, NO_INHERITANCE);
PACL pTempDacl = NULL;
DWORD dwErr = 0;
dwErr = SetEntriesInAcl(1, &denyAccess, NULL, &pTempDacl);
// check dwErr...
dwErr = SetSecurityInfo(hProcess, SE_KERNEL_OBJECT, DACL_SECURITY_INFORMATION,
NULL, NULL, pTempDacl, NULL);
// check dwErr...
LocalFree(pTempDacl);
CloseHandle(hProcess);
return dwErr == ERROR_SUCCESS;

}
BYTE MOV[10] = { 0x48, 0xB8, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0 };
BYTE JMP_RAX[2] = { 0xFF, 0xE0 };
#define BuffSizeX64 (sizeof(MOV) + sizeof(JMP_RAX))

BOOL Hook_Det_x64(char LibName[], char API_Name[], LPVOID NewFun) {
DWORD OldProtect;
DWORD64 OrgAddress = (DWORD64)GetProcAddress(LoadLibraryA(LibName), API_Name);
if (OrgAddress == NULL) return 0;

memcpy(&MOV[2], &NewFun, 8);
VirtualProtect((LPVOID)OrgAddress, BuffSizeX64, PAGE_EXECUTE_READWRITE, &OldProtect);
memcpy((LPVOID)OrgAddress, MOV, sizeof(MOV));
memcpy((LPVOID)(OrgAddress + sizeof(MOV)), JMP_RAX, sizeof(JMP_RAX));
VirtualProtect((LPVOID)OrgAddress, BuffSizeX64, OldProtect, &OldProtect);

return 1;
}

int WINAPI MessageBoxAX(
HWND hWnd,
LPCSTR lpText,
LPCSTR lpCaption,
UINT uType) {

MessageBoxExA(0, "Hooked ...", "Mahmoud", 0, 0);
return 999;
}

int main()
{
using namespace std;
int abfrage;

ProtectProcess();
while (1)

```

```

{
Hook_Det_x64("Kernel32.dll", "TerminateProcess", MessageBoxAX);

NtHookInstall(GetProcAddress(GetModuleHandleW(L"Kernel32.dll"), "TerminateProcess"), Cals);
NtHookInstall(GetProcAddress(GetModuleHandleW(L"ntdll.dll"), "ZwTerminateProcess"), Cals);
NtHookInstall(GetProcAddress(GetModuleHandleW(L"ntdll.dll"), "ZwTerminateThread"), Cals);
NtHookInstall(GetProcAddress(GetModuleHandleW(L"ntdll.dll"), "TerminateProcess"), Cals);
NtHookInstall(GetProcAddress(GetModuleHandleW(L"Kernel32.dll"), "NtTerminateProcess"), Cals);
NtHookInstall(GetProcAddress(GetModuleHandleW(L"Kernel32.dll"), "NtTerminateThread"), Cals);
NtHookInstall(GetProcAddress(GetModuleHandleW(L"ntdll.dll"), "NtQuerySystemInformation"),
Cals);
NtHookInstall(GetProcAddress(GetModuleHandleW(L"ntdll.dll"), "NtOpenProcess"), Cals);
NtHookInstall(GetProcAddress(GetModuleHandleW(L"ntdll.dll"), "ZwQuerySystemInformation"),
Cals);
NtHookInstall(GetProcAddress(GetModuleHandleW(L"ntdll.dll"), "ZwOpenProcess"), Cals);
}
return EXIT_SUCCESS;
}

```

but, i can terminate from taskmgr.

Please have some code for implement?I wont help you copy paste, fuck off

lunacyrcus - October 2, 2018, 8:12 pm

(Oct 2, 2018 12:23 AM)Diallix Wrote: [\[CITE\]](#) Hi. I spent a lot of time finding articles about Protect Process via set privileges or IAT hooking ZwTerminateProcess but without succesfull.

Is here somebody, who has code to protect process? My System is win8 - 64 bit.

Thanks very much.
for better ways, write a minifilter driver

Diallix - October 3, 2018, 1:02 am

(Oct 2, 2018 01:19 AM)3DES Wrote: [\[CITE\]](#) Inject shellcode / function to remote process that monitors your process and viceversa.

(Oct 2, 2018 11:51 AM)3DES Wrote: [\[CITE\]](#) I wont help you copy paste, fuck off
It is not my code. I try it, but it is not working

oxidebuffalo - October 3, 2018, 9:21 am

(Oct 2, 2018 08:12 PM)lunacyrcus Wrote: [\[CITE\]](#) for better ways, write a minifilter driverThat isn't going to help him unless he wants to protect the files on disk, unless you're recommending OP to use ObRegisterCallbacks, which can be done from a normal KMDF driver anyway.

Furthermore, Windows 10 has requirements such as an EV certificate unless you have a zero-day privilege escalation exploit for kernel mode execution, or you have sufficient privileges to enable Test Mode on the machine. Previous versions of Windows since Windows Vista have a requirement for a normal certificate which supports kernel mode signing. Of course, this is all only for x64 versions.

It's easy to just tell OP to use a KM driver but that isn't going to help them because clearly they are not on any level to be dealing with one.

Mr. Kim - October 3, 2018, 9:19 pm

(Oct 3, 2018 09:21 AM)oxidebuffalo Wrote: [\[CITE\]](#) That isn't going to help him unless he wants to protect the files on disk, unless you're recommending OP to use ObRegisterCallbacks, which can be done from a normal KMDF driver anyway.

Furthermore, Windows 10 has requirements such as an EV certificate unless you have a zero-day privilege escalation exploit for kernel mode execution, or you have sufficient privileges to enable Test Mode on the machine. Previous versions of Windows since Windows Vista have a requirement for a normal certificate which supports kernel mode signing. Of course, this is all only for x64 versions.

It's easy to just tell OP to use a KM driver but that isn't going to help them because clearly they are not on any level to be dealing with one.

Or just use an old, vulnerable driver to get kernel-mode code running? I mean technically you're not wrong, but win10 still allows old signed drivers to be loaded, and there's lots of vulnerable drivers that allow arbitrary code execution in kernel-mode.

As for OP not being ready for writing kernel mode shellcode, this is probably true. OP why not focus on remaining undetected instead of being 'protected' from removal? The only 'real' way to protect from process termination is to have kernel mode code running and protecting your process, which brings along it's own problems (especially on win10).