# Using Trace Replayer Debugger and Managing Traces in IDA

Copyright 2014 Hex-Rays SA

# Table of contents

Introduction	2
Quick Overview	2
Following this tutorial	
Supplied files	
Replaying and managing traces	
Recording traces	
Working with traces	
Loading an overlay and viewing differences in flow	9
Diffing traces	
Reverting the diff	
Replaying traces	20
Summary	
Summary	24

## Introduction

### **Quick Overview**

The trace replayer is an IDA pseudo debugger plugin that appeared first in IDA 6.3. This plugin can replay execution traces recorded with any debugger backend in IDA, such as local Win32 or Linux debuggers, WinDbg, remote GDB debugger, etc...

### Following this tutorial

This tutorial was created using the Linux version of IDA and a Linux binary as target. However, it can be followed on any supported platform (MS Windows, Mac OS X and Linux) by setting up remote debugging. Please refer to the <u>IDA online help</u> for more information regarding <u>remote debugging</u>.

#### Supplied files

Among with the tutorial the following files are also provided at <u>http://www.hex-rays.com/products/ida/support/tutorials/replayer/ida-replayer-tutorial.tar.gz</u>

File name	SHA1	Description
intoverflow.c	6424d3100e3ab1dd3fceae53c7d925364cea75c5	Program's source code.
intoverflow.elf	69a0889b7c09ec5c293702b3b50f55995a1a2daa	Linux ELF32 program.
no_args.trc	773837c2b212b4416c8ac0249859208fd30e2209	IDA binary trace file version 1
second_run.trc	4e0a5effa34f805cc50fe40bc0e19b78ad1bb7c4	IDA binary trace file version 1
crash.trc	f0ee851b298d7709e327d8eee81657cf0beae69b	IDA binary trace file version 1

# **Replaying and managing traces**

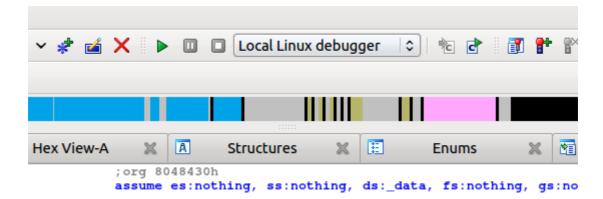
#### **Recording traces**

Before using the trace replayer plugin we will need to record an execution trace of a program. We will use the following toy vulnerable program as an example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int foo(char *arg, int size)
{
    char *buf;
```

```
if ( strlen(arg) > size )
    {
        printf("Too big!\n");
        return 1;
    }
    buf = malloc(size);
    strcpy(buf, arg);
    printf("Buffer is %s\n", buf);
    free(buf);
    return 0;
}
int main(int argc, char **argv)
{
    if ( argc != 3 )
    {
        printf("Invalid number of arguments!\n");
        return 2;
    }
    return foo(argv[1], atoi(argv[2]));
```

Please compile this sample program (in this example, we used GCC compiler for Linux) or use the supplied ELF binary, open the binary in IDA and wait until the initial analysis completes. When done, select a suitable debugger from the drop down list ("Local Linux debugger", or "Remote Linux debugger" if you're following this tutorial from another platform):



We have two ways of telling IDA to record a trace:

- 1. Break on process entry point and manually enable tracing at this point.
- 2. Or put a trace breakpoint at the very first instruction of the program.

In the case we prefer the first approach we will need to click on the menu "Debugger  $\rightarrow$  Debugger Options" and then mark the check box "Stop on process entry point" as shown bellow:

🕅 Debugger setup 🗵
Events <ul> <li><u>Stop on debugging start</u></li> </ul>
Stop on process entry point
Stop on thread start/exit
Stop on library load/unload
Stop on debugging message
Event condition
Log <ul> <li>Segment modifications</li> </ul>
✓ Thread start/exit
✓ Library load/unload
Breakpoint
Debugging message
Options <ul> <li>Re<u>c</u>onstruct the stack</li> </ul>
Show debugger <u>b</u> reakpoint instructions
<u>Autoload PDB files</u>
<u>E</u> dit exceptions <u>R</u> eload exceptions
Help Cancel O <u>K</u>

After checking this option press OK and run the program pressing F9. When the entry point is reached, we can select from the menu "Debugger  $\rightarrow$  Tracing" one of the following three options:

- 1. Instruction tracing: All instructions executed will be recorded.
- 2. Function tracing: Only function calls and returns will be recorded.
- 3. Basic block tracing: Similar to instruction tracing but, instead of single stepping instruction by instruction, IDA will set temporary breakpoints in the end of every known basic block, as well as on function calls.

For this example we will select "Instruction tracing". Check this option and let the program continue by pressing F9. The program will resume execution and finish quickly. Now, we have a recorded trace! To see it, select "Debugger  $\rightarrow$  Tracing  $\rightarrow$  Trace Window". A new tab will open with a content similar to the following:

DA View	-A 🗶 🗊 Trace window 🗶	🖸 Hex View-A 🗶 🖪 Structures 🗶 🗄
Thread	Address	Instruction
000047F8	.text:_start	Memory layout changed: 33 segments
💷 000047F8		
💷 000047F8	.text:_start	xor ebp, ebp
🗐 000047F8	.text:_start+2	pop esi
🗐 000047F8	.text:_start+3	mov ecx, esp
🗐 000047F8	.text:_start+5	and esp, 0FFFFFF0h
💷 000047F8	.text:_start+8	push eax
💷 000047F8	.text:_start+9	push esp
💷 000047F8	.text:_start+A	push edx
💷 000047F8	.text:_start+B	push offsetlibc_csu_fini
💷 000047F8	.text:_start+10	<pre>push offsetlibc_csu_init</pre>
💷 000047F8	.text:_start+15	push ecx
💷 000047F8	.text:_start+16	push esi
💷 000047F8	.text:_start+17	push offset main
💷 000047F8	.text:_start+1C	calllibc_start_main
J 🗐 000047F8	.plt:libc_start_main	jmp ds:off_804A004

As previously stated, there are two ways to record traces: enabling it manually, or using an "Enable tracing" breakpoint. To set such a breakpoint we will go to the program's entry point (Ctrl+E) and put a breakpoint (F2) in the very first instruction. Then right click on the new breakpoint and select "Edit breakpoint". In the dialog check the option "Enable tracing" and then select the desired "Tracing type" (for this example, we'll use "Instructions"):

1	Breal	kpoint settings	;
Location	0x8048430		~
Con <u>d</u> ition			~
Settings ☑ <u>E</u> nabled	I	Actions	
🗆 <u>H</u> ardwa	re	□ <u>T</u> race	
□ <u>M</u> odule	relative	Refres	h debugger memory
🗆 Symboli	c	✓ Enable	tracing
Source o	code	Disable	e tracing
Low leve	el condition	T <u>r</u> acing ty	pe Instructions 😂
Hardware b	oreakpoint mode		
🗆 <u>W</u> rite			
□ Execute		Size 0x1	~
Help Cancel O <u>K</u>			

Remove the "Stop on process entry point" option we set in the prior example and press F9 to run the program.

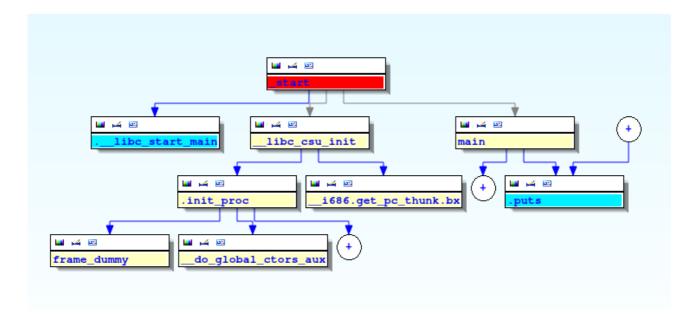
This way is more convenient than the first because the tracing is turned on automatically and does not need manual intervention.

#### Working with traces

Now we have a new recorded trace, no matter which method we used. What can we do with it? First, we can check which instructions were executed, as they are highlighted in the disassembly, like in the screenshot bellow:

	.text:08048557	public main
	.text:08048557 main	proc near ; DATA XREF: _start+17îo
	.text:08048557	
	.text:08048557 arg_0	= dword ptr 8
	.text:08048557 arg_4	= dword ptr 0Ch
	.text:08048557	
•	.text:08048557	push ebp
•	.text:08048558	mov ebp, esp
•	.text:0804855A	and esp, OFFFFFFOh
•	.text:0804855D	sub esp, 10h
•	.text:08048560	cmp [ebp+arg_0], 3
	.text:08048564	jz short loc_8048579
•	.text:08048566	mov dword ptr [esp], offset aInvalidNumber0 ; "Invalid number of arguments!"
	.text:0804856D	call puts
•	.text:08048572	mov eax, 2
	.text:08048577	jmp short locret_804859D
	.text:08048579 ;	
	.text:08048579	
1.1	.text:08048579 loc 8048579:	; CODE XREF: main+D1j
- <b>-</b>	.text:08048579	mov eax, [ebp+arg_4]
•	.text:0804857C	add eax, 8
•	.text:0804857F	mov eax, [eax]
•	.text:08048581	mov [esp], eax ; nptr
•	.text:08048584	call atoi
	.text:08048589	mov edx, [ebp+arg_4]
	.text:0804858C	add edx, 4
	.text:0804858F	mov edx, [edx]
	.text:08048591	mov [esp+4], eax ; size
	.text:08048595	mov [esp], edx ; src
	.text:08048598	call foo
	.text:0804859D	
	.text:0804859D locret 804859D:	; CODE XREF: main+201j
	.text:0804859D	leave
•	.text:0804859E	retn
	.text:0804859E main	endp
	.text:0804859E	

(the highlight color can be changed in "Debugger  $\rightarrow$  Tracing  $\rightarrow$  Tracing Options") Highlighting makes it clear which instructions have been executed. We can also check what functions have been executed (instead of instructions) by opening the "Trace Window" via "Debugger  $\rightarrow$  Tracing  $\rightarrow$  Trace Window", right clicking on the list and then selecting "Show trace call graph":



Now let's inspect the register values in order to understand why the check at 0x0848566 doesn't pass. Please select "Debugger  $\rightarrow$  Switch debugger" and in the dialog box click on the "Trace replayer" radio button:

🕈 Select a debugger 🛛 🗙
Available debuggers O Local <u>B</u> ochs debugger
<ul> <li>Local <u>L</u>inux debugger</li> </ul>
O Remote <u>G</u> DB debugger
O Remote Linux debugger
Trace replayer
Default debuggers (autoselected for new databases): NONE Set as <u>d</u> efault debugger
Cancel O <u>K</u>

Click OK and press F4 in the first instruction of the "main" function.

The trace replayer will suspend execution at the "main" function and display the register values that were recorded when the program was executed:

📂 🖬 🗢 🗸 🔿 🗸 🆓 👘	x 🔺 👄 📾 💣 🕈 🛹 🛩 🖆 🗙 🕨 💷	🗖 Trace replayer 🔯 👘 👔 » 🗊
IDA View-EIP		<ul> <li>Image: Second sec</li></ul>
.text:08048557		EAX FFEB6F04 🤟 OF
.text:08048557 public main .text:08048557 main proc near	; DATA XREF: start+17 <sup>†</sup> 0	EBX F76FBFF4 🦌 DF
.text:08048557	, DATA AREFSCALCTITIO	ECX 7E557C3E
.text:08048557 arg_0= dword ptr 8		EDX 00000001 \
text: 08048557 arg 4= dword ptr 0Ch		SF SF
.text:08048557		
.text:08048557 push ebp		EDI 00000000 🦌 AF
.text:08048558 mov ebp, esp		EBP FFEB6ED8 SPF
.text:0804855A and esp, 0FFFFFFF0h		ESP FFEB6E5C 🖌 CF
• .text:0804855D sub esp, 10h		EIP 08048557 🗣 main
.text:08048560 cmp [ebp+arg_0], 3		EFL 00000246
.text:08048564 jz short loc 8048579		
	offset aInvalidNumberO ; "Invalid number of argum	ents!"
.text:0804856D call puts		
.text:08048572 mov eax, 2		
.text:08048577 jmp short locret 8048	59D	
.text:08048579 ;		
.text:08048579		
.text:08048579 loc_8048579:	; CODE XREF: main+Dfj	
<pre>.text:08048579 mov eax, [ebp+arg_4]</pre>		
.text:0804857C add eax, 8		
.text:0804857F mov eax, [eax]		
<pre>.text:08048581 mov [esp], eax</pre>	; nptr	
.text:08048584 call _atoi		
<pre>.text:08048589 mov edx, [ebp+arg_4]</pre>		
.text:0804858C add edx, 4		
<pre>.text:0804858F mov edx, [edx]</pre>		
<pre>.text:08048591 mov [esp+4], eax</pre>	; size	
.text:08048595 mov [esp], edx	; src	
.text:08048598 call foo		
.text:0804859D		
.text:0804859D locret_804859D:	; CODE XREF: main+20 <b>î</b> j	
.text:0804859D leave		
.text:0804859E retn		
.text:0804859E main endp		

We can single step by pressing F7, as usual. Let us keep pressing F7 until the "jz" instruction is reached:

IDA View-EIP			🙆 🗵 👿 General registers 🛛
.text:08048557			EAX FFEB6F04
.text:08048557 publ	ic main		EBX F76FBFF4
.text:08048557 main	proc near	; DATA XREF: _start+17fo	
.text:08048557			ECX 7E557C3E
.text:08048557 arg_			EDX 00000001 🤟
.text:08048557 arg_	4= dword ptr 0Ch		ESI 0000000 🦌
.text:08048557			EDI 00000000 4
.text:08048557 push	ebp		EBP FFEB6E58
.text:08048558 mov	ebp, esp		
.text:0804855A and	esp, OFFFFFFF0h		ESP FFEB6E40 5
.text:0804855D sub			EIP 08048564 😽 main+D
.text:08048560 cmp			EFL 00000293
.text:08048564 jz			
.text:08048566 mov		t aInvalidNumberO ; "Invalid number of an	rguments!"
.text:0804856D call			
.text:08048572 mov	eax, 2		
text:08048577 jmp	short locret_804859D		
.text:08048579 ;			
.text:08048579			
.text:08048579 loc_		; CODE XREF: main+Dfj	
• .text:08048579 mov	eax, [ebp+arg_4]		
.text:0804857C add			
.text:0804857F mov			
.text:08048581 mov	[esp], eax	; nptr	
.text:08048584 call			
.text:08048589 mov	edx, [ebp+arg_4]		
.text:0804858C add	edx, 4		
.text:0804858F mov	edx, [edx]		
.text:08048591 mov	[esp+4], eax	; size	
.text:08048595 mov	[esp], edx	; src	
.text:08048598 call	foo		
.text:0804859D			
.text:0804859D locr		; CODE XREF: main+20 <b>1</b> j	
.text:0804859D leav	-		
.text:0804859E retr			
.text:0804859E main	endp		

The comparison "cmp [ebp+arg\_0], 3" was not successful (ZF=0) so the check does not pass. We need to give to the program two arguments to pass this check and record a new trace.

### Loading an overlay and viewing differences in flow

Before doing another run, let's save the first trace to a file. Select "Debugger  $\rightarrow$  Tracing  $\rightarrow$  Trace Window", right click in the middle of the newly opened tab, and select "Save trace" from the popup menu:

/usii		LUI -
ous' ous :all	Copy Copy all	Ctrl+C Ctrl+Shift+Ins SP=
	Quick filter Modify filters	Ctrl+F Ctrl+Shift+F <sup>:SP=</sup>
งนร	Show trace info Clear trace Load trace	Ctrl+I SP= Ctrl+X AX= Ctrl+L SP=
no	Save trace	Ctrl+S BP=
ous	Change trace description	Ctrl+E SP=
ous ous	Show trace call graph	Ctrl+G SP= SP=
all nov	Export trace to text file Overlay epx, [esp+o]	SP= SP= EBX= FSP=

```
Then save the file:
```

1	Save trace file		_ 0 ×
	guments_execution.trc	C	reate Fo <u>l</u> der
Places Search Secently Used ↑ aundro Desktop	Name arguments_execution.trc arguments_execution_2.trc	<ul> <li>✓ Size</li> <li>53.5 KB</li> <li>33.5 KB</li> </ul>	
File System		Cancel	*.trc ♀

You will also be offered a chance to give the trace a description:

Trace description	- • ×		
Enter description (or leave empty) Running without argument	s. 🗸		
Cancel O <u>K</u>			

Now let's record a new trace but this time we will pass two command line arguments to the program. Select "Debugger  $\rightarrow$  Process Options" and set "AAAA 4" as the arguments:

Debug application setup: linux 🗵										
Application /home/joxean/Documentos/hexrays/replayer/intoverflow										
Input file /home/joxean/Documentos/hexrays/replayer/intoverflow										
Directory /home/joxean/Documentos/hexrays/replayer										
Parameters AAAA 4										
ostname V Po <u>r</u> t 23946 V	<u>H</u> ostname									
ass <u>w</u> ord	Pass <u>w</u> ord									
Save network settings as default										
Help Cancel O <u>K</u>										

Close the dialog, revert to the "Local Linux debugger", and press F9. A new trace will be recorded. If we check the "main" function we will see that different instructions have been executed this time:

DA V	'iew-A	×	<b>Ş</b>	Trace window	×	Hex View-A	XA	Stru	uctures	×	Ħ	Enums	×	1	Imports	×
	.text:				public											
	.text: .text:			main	proc n	ear		DATA	XREF :	_star	t+17 <b>î</b> o					
	.text: .text:				- due v	d ptr 8										
	.text:					d ptr 0Ch										
	.text:			arg_4	- 4001	a per och										
•	.text:				push	ebp										
•	.text:				mov	ebp, esp			1							
•	.text:				and	esp, OFFFFF	FFOh									
•	.text:				sub	esp, 10h										
•	.text:	08048	560		cmp	[ebp+arg 0]	, 3									
	.text:	08048	564		jz	short loc_8	048579									
	.text:	08048	566		mov	dword ptr [	esp],	offset	: aInva	alidNu	mber0 ;	"Invalio	i numbe	er of	arguments!	**
	.text:	08048	56D		call	_puts									-	
	.text:	08048	572		mov	eax, 2										
	.text:				jmp	short locre	t_8048	59D								
	.text:			;												
1.1	.text:															
4.0				loc_8048579:				CODE	XREF :	main+	DTj					
118	.text:				mov	eax, [ebp+a	rg_4]									
- 1	.text:				add	eax, 8										
- 1	.text: .text:				mov	eax, [eax]										
	.text:				mov call	[esp], eax atoi		nptr								
	.text:				mov	_acor edx, [ebp+a	ng 41									
- el	.text:				add	edx, [ebp+a edx, 4	49_41									
	.text:				mov	edx, [edx]										
. •	.text:				mov	[esp+4], ea	x :	size								
	.text:				mov	[esp], edx		src								
. •	.text:	08048	598		call	foo										
	.text:	08048	59D													
	.text:	08048	59D	locret_804859	D:		;	CODE	XREF :	main+	20 <b>1</b> j					
	.text:			_	leave						-					

Let's check which instructions are different between the first and the second run.

1	5	
pushoffsetlibc_csu	_init	ESP=FFDEBC6C
Сору	Ctrl+C	ESP=FFDEBC68
Copy all	Ctrl+Shift+Ins	ESP=FFDEBC64
W Quick filter	Ctrl+F	ESP=FFDEBC60
		ESP=FFDEBC5C
🍟 Modify filters	Ctrl+Shift+F	
Show trace info	Ctrl+I	ESP=FFDEBC58
Clear trace	Ctrl+X	
Load trace	Ctrl+L	ESP=FFDEBC54
Save trace	Ctrl+S	EAX=FFDEBC84 EBX=F77
		ESP=FFDEBBD8
Change trace description	Ctrl+E	EBP=FFDEBBD8
Show trace call graph	Ctrl+G	ESP=FFDEBBD4
Export trace to text file		ESP=FFDEBBD0
Overlay	$\rightarrow$	Load overlay Ctrl+Shift+L
calli686_get_pc_thu	unk_bx	ESP=FFDEBBC8

First, we will need to load the previous trace as "overlay":

Select the trace we saved:

1		Please select a trace			- • ×
The test of te	tra	aces			
Location: no_ar	gun	nents_execution.trc			
<u>P</u> laces		Name	~	Size	Modified
🖹 Search	≡	arguments_execution.trc		53.5 KB	12:40
🕙 Recently U		arguments_execution_2.trc		33.5 KB	12:41
🛧 aundro		no_arguments_execution.trc		716 bytes	12:46
🔳 Desktop					
📃 File System					
🔜 storage					
🔜 intel_120g	~				\
0 -					*.trc 🗘
			8	<u>C</u> ancel	<u>ျံ</u> Open

Note that we have now other options in the 'Overlay' submenu, now that there is an overlay present:

t+1C	call	lıbc_stari	t_mair	1	ESP=FFD8	BC5C
:_st:     Copy :3A4    Copy all		Ct Ctrl+Shift	rl+C +Ins		ESP=FFDE	BC58
13AF 1384 🍞 Quick filt 1384 ữ Modify fil		Ct Ctrl+Shi	rl+F ft+F		ESP=FFDE EAX=FFD	
36_g Show tra	ce ce	Ct Ct Ct	trl+1 rl+X rl+L rl+S rl+E rl+G	ınk_bx	ESP=FFDE EBP=FFDE ESP=FFDE ESP=FFDE ESP=FFDE EBX=080 ESP=FFDE	BBD8 BBD4 BBD0 BBCC BBC8 485AB
c_cs <u>Overlay</u> c_csu_init+11 c_csu_init+14 proc proc+1	sub call push mov	esp, 1Ch _init_proc ebp ebp, esp	> ; _in	Show overlay info Clear overlay Load overlay Subtract overlay	Ctrl+Shift+L Ctrl+Shift+L Ctrl+Shift+L Ctrl+D EBP=FFDE	9FF4 F 3BB0 / 3BAC 3BA8 BBA8

Now go back to the disassembly view and check how the disassembly code is highlighted in three different colors:

IDA \	/iew-A	×	· ·	Trace window	× [	0	Hex Vie	ew-A	×	A	Stru	ctures	×	<b></b>	Enums	×	1	Imports	×
	.text: .text:																		
	.text:																		
	.text:	08048	557		pu	blic	main												
	.text:	08048	557	main	pr	oc ne	ear												
	.text:																		
				arg_0			d ptr												
				arg_4	=	dwor	d ptr					1							
	.text:																		
	.text:					sh	ebp												
	.text: .text:				mo an		ebp,	esp OFFFI											
	.text:				su		esp,		rrro										
	.text:				cm			+arg_(	01.3										
	.text:				iz			t loc		579									
•	.text:				mo						ffset	aInva	lidNu	mber0	: "Invalid	l numbe	r of	arguments!'	
	.text:	08048	56D		ca	11	put			• •								-	
•	.text:	08048	572		mo	v	eax,	2											
	.text:	08048	577		jm	<b>p</b>	shor	t loci	ret_8	04859	9D								
	.text:			;															
	.text:																		
1 i				loc_8048579:							CODE	XREF :	main+	DTj					
	.text: .text:				mo ad			[ebp-	arg_	4]									
	.text:						eax,												
	.text:				mo mo			[eax] ], eax			nptr								
	.text:				ca		ato		•		iper								
	.text:				mo			[ebp+	arg a	41									
	.text:	08048	58C		ad		edx,												
	.text:	08048	58F		mo	v		[edx]	1										
•	.text:	08048	591		mo	v		+4], e		; ;	size								
•	.text:	08048	595		mo	v	[esp	], ed)	ĸ	; :	src								
•	.text:	08048	598		ca	11	foo												
	.text:																		
				locret_804859						; (	CODE	XREF :	main+	20 <b>1</b> j					
	.text:	08048	59D		le	ave													

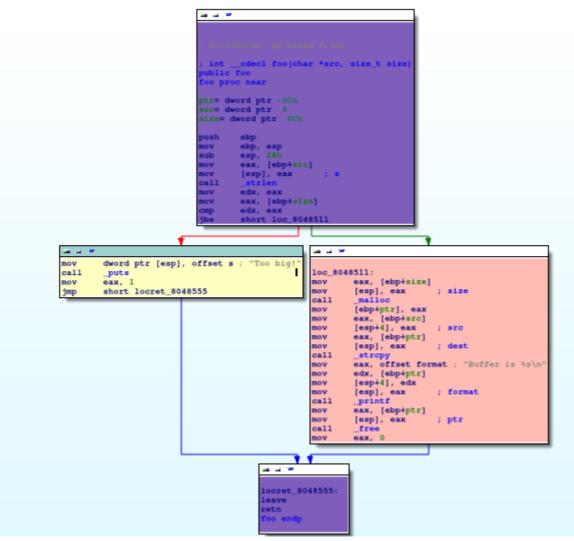
The code highlighted in yellow is the code executed in the current trace (the one listed in the "Trace Window"). The pink code was executed only in the overlay trace. And the code in purple is the code common to both traces. We can immediately see that there is some new code that have been executed, like the calls to **atoi** and **foo**.

Let's go to the "foo" function and see what happened here:

IDA V	/iew-A 🔰	ت	Trace window	× 🖸 I	Hex View-A 🛛 💥	Structure	es 🗶 🗉	Enums	×
	.text:080								
	.text:080				l ptr -0Ch				
	.text:080				lptr 8				
	.text:080		size	= dword	lptr OCh				
	.text:080				- h				
- 1	.text:080			push	ebp				
	.text:080			mov sub	ebp, esp esp, 28h				
	.text:080			mov	eax, [ebp+src				
•	.text:080			mov	[esp], eax	·] ; s			
•	.text:080			call	strlen	, .			
•	.text:080			mov	edx, eax				
•	.text:080			mov	eax, [ebp+siz	el.		1	
•	.text:080			cmp	edx, eax			•	
	.text:080			jbe	short loc 804	8511			
•	.text:080	0484FE		mov		<pre>p], offset s ;</pre>	"Too biq!"		
	.text:080	48505		call	puts		2		
•	.text:080	04850A		mov	eax, 1				
	.text:080	04850F		jmp	short locret	8048555			
	.text:080	048511	;						
1	.text:080	048511							
	.text:080	048511	loc_8048511:			; CODE XREF	: foo+18 <b>1</b> j		
<b>₩</b>	.text:080			mov	eax, [ebp+siz	:e]			
	.text:080			mov	[esp], eax	; size			
	.text:080			call	_malloc				
	.text:080			mov	[ebp+ptr], ea				
	.text:080			mov	eax, [ebp+src	-			
	.text:080			mov	[esp+4], eax	; src			
	.text:080			mov	eax, [ebp+ptr				
	.text:080			mov	[esp], eax	; dest			
	.text:080			call	_strcpy	annat i UDirece	n is sala"		
	.text:080			mov		ormat ; "Buffe	r is Ss/U.		
	.text:080			mov	edx, [ebp+ptr [esp+4], edx	1			
	.text:080			mov	[esp+4], eax [esp], eax	; format			
	.text:080			call	printf	, IOIMac			
•	.text:080			mov	eax, [ebp+ptr	-1			
				In o v	can, tempther				

The code in yellow tells us that the check for the size at 0x800484FC passed and the calls to **malloc**, **strcpy** and **printf** were executed. Let's save this trace for later analysis and comparison with the future runs. As before, go to the trace window, right click on the list and select "Save trace". Set the trace's description to 'Correct execution'.

It's time to record another trace with different arguments to see what happens. For this new execution, we will longer command line arguments (eight "A" characters instead of four). Let's change the arguments in "Debugger  $\rightarrow$  Process Options", switch back to the "Local Linux debugger", and run it. We have a new trace. Let's compare it against the previously recorded one. As we did before, go to the "Trace Window", right click on the list, select "Overlay", then "Local overlay", and select the trace with description "Correct execution".



As we see, the code that alerts us the about a too big string was executed (it's highlighted in yellow). Let's save this recorded trace with the "String too big" description. Now we will record one more trace but this time we will use the number "-1" as the second command line argument.

Change the arguments in "Debugger  $\rightarrow$  Process Options" as shown bellow:

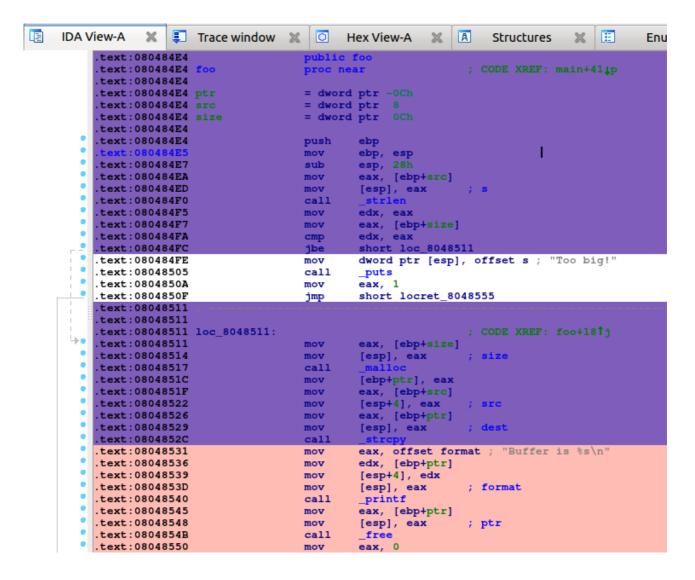
🕅 Debug application setup: linux 🗙									
Application /home/joxean/Documentos/hexrays/replayer/intoverflow									
Input file /home/joxean/Documentos/hexrays/replayer/intoverflow									
Directory /home/joxean/Documentos/hexrays/replayer									
Parameters AAAAAAAA-1									
Hostname Command-line arguments to pass to the application									
Password V									
Save network settings as default									
Help Cancel O <u>K</u>									

Then switch back again, to the "Local Linux debugger" (or to "Remote Linux debugger" if needed) and run it by pressing F9. The process will crash somewhere in the call to **strcpy**:

1	Warning
	F7622578: got SIGSEGV signal (Segmentation violation) (exc.code b, tid 21828)

Stop the debugger and save this trace (let's call it "Crash"). Then diff this trace against the "Correct execution" trace.

We will see the following in the disassembly view:



As we see, pretty much the same code as in the previous run was executed until the call to **strcpy**. It's time to replay this last execution and see what happened.

#### **Diffing traces**

When both a "main trace" and an "overlay trace" are present, the context menu item "Overlay  $\rightarrow$  Subtract overlay" becomes available.

This allows one to subtract the list of events (e.g., instructions) that are present in the overlay, from the main trace:

Thread	Address		Instru	ction		Result
00007288	.text:_start		Memo	orv lavout changed: 1	5 segments	Memory layout changed: 15 segn
100007288		Сору		Ctrl+C		ST0=FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
1 00007288		Copy all		Ctrl+Shift+Ins		ZF=1 SF=0
	.text:_start+2	W Quick filter		Ctrl+F	-	ESI=00000003 ESP=FFDEBC84
	.text:_start+3	Modify filters		Ctrl+Shift+F		ECX=FFDEBC84
1 00007288	.text:_start+5					ESP=FFDEBC80 PF=0 ZF=0 SF=
1 00007288	.text:_start+8	Show trace info	D	Ctrl+I		ESP=FFDEBC7C
	.text:_start+9	Clear trace		Ctrl+X		ESP=FFDEBC78
	.text:_start+A	Load trace		Ctrl+L		ESP=FFDEBC74
	.text:_start+B	Save trace		Ctrl+S		ESP=FFDEBC70
	.text:_start+10	Change trace (	descriptio	n Ctrl+E		ESP=FFDEBC6C
	.text:_start+15	Show trace cal		Ctrl+G		ESP=FFDEBC68
	.text:_start+16	Export trace to				ESP=FFDEBC64
	.text:_start+17	Overlay	rext file		Show overlay info	Ctrl+Shift+I
	.text:_start+1C				· · · · ·	~
	.plt:libc_start_	main	jmp	ds:off_804A004	Clear overlay	Ctrl+Shift+X
	.plt:080483AA				Load overlay	Ctrl+Shift+L 8
	.plt:080483AF				Subtract overlay	Ctrl+D
	.plt:08048384				Remove, fro	om the main trace, events that are
	plt:0804838A					Ent HDEBOOLEDIC HANDING
	.text:libc_csu_ir		push	ebp		ESP=FFDEBBD8
	.text:libc_csu_ir		mov	ebp, esp		EBP=FFDEBBD8
	.text:libc_csu_ir		push	edi		ESP=FFDEBBD4
	.text:libc_csu_ir		push	esi		ESP=FFDEBBD0
1 00007288	.text:_libc_csu_ir	nit+5	push	ebx		ESP=FFDEBBCC

#### Will give the following results:

Image: style="text-styye="text-style="text-style="text-style="text-style="text-	📃 IDA View	-A 🛛 💭 Trace window 🛛 🗊	Breakpoints 🛛 🖸 Hex View-1 🛛 🕅	Structures 🔀 語 Enu
Image: Stripper S	Thread	Address	Instruction	Result
# 00007288       text:main:loc_8048579       mov       eax, [ebp+argv]         # 00007288       text:main+25       add       eax, 8       EAX=FFDEBC8C PF=0 ZF=0         # 00007288       text:main+28       mov       eax, [eax]       EAX=FFDEDC94         # 00007288       text:main+2A       mov       [esp], eax       ; nptr         # 00007288       text:main+2A       mov       [esp], eax       ; nptr         # 00007288       text:main+2D       call       _atoi       ESP=FFDEBBBC         # 00007288       .plt:_atoi       jmp       ds:off_804A018       ESP=FFDEBBB8         # 00007288       .plt:080483FA       ESP=FFDEBBB8       EDX=FFDEBC84         # 00007288       .plt:080483FF       EDX=FFDEBC84       EDX=FFDEBC84         # 00007288       text:main+32       mov       edx, [ebp+argv]       EDX=FFDEBC88 AF=0         # 00007288       text:main+35       add       edx, 4       EDX=FFDEBC88 AF=0         # 00007288       text:main+38       mov       edx, [edx]       EDX=FFDEBC88 AF=0         # 00007288       text:main+38       mov       edx, [edx]       EDX=FFDEDC8F         # 00007288       text:main+34       mov       [esp], edx       ; size         # 00007288	00007288	.text:_start	Memory layout changed: 15 segments	Memory layout changed: 15 s
Image: Second	📒 00007288			ST0=FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Image: Sector of the sector	📒 00007288	.text:main:loc_8048579	mov eax, [ebp+argv]	
Image: Sector of the sector	📒 00007288	.text:main+25	add eax, 8	EAX=FFDEBC8C PF=0 ZF=0 :
Image: Second State Sta	📒 00007288	.text:main+28	mov eax, [eax]	EAX=FFDEDC94
Image: Second Stream of Se			mov [esp], eax ; nptr	
Image: state of the state			-	ESP=FFDEBBBC
<ul> <li>i, i, i</li></ul>			jmp ds:off_804A018	
Image: spectra spectra spectra       mov       edx, [ebp+argv]       EDX=FFDEBC84         Image: spectra spectra       add       edx, 4       EDX=FFDEBC88 AF=0         Image: spectra       mov       edx, [edx]       EDX=FFDEBC88 AF=0         Image: spectra       mov       edx, [edx]       EDX=FFDEBC8F         Image: spectra       mov       edx, [edx]       EDX=FFDEBC8F         Image: spectra       mov       [esp+4], eax ; size       Image: spectra         Image: spectra       mov       [esp], edx ; s       Image: spectra         Image: spectra       mov       [esp], edx ; s       Image: spectra         Image: spectra       mov       [esp], edx ; s       Image: spectra         Image: spectra       mov       [esp], edx ; s       Image: spectra         Image: spectra       mov       [esp], edx ; s       Image: spectra         Image: spectra       mov       [esp], edx ; s       Image: spectra         Image: spectra       mov       [esp], edx ; s       Image: spectra         Image: spectra       mov       [esp], esp       ESP=FFDEBBB8         Image: spectra       mov       ebp, esp       EBP=FFDEBBB8	· ·			ESP=FFDEBBB8
Image: Second Restriction of Second Parage       Image: Second Parage       Image: Second Parage         Image: Second Parage       Image: Second Parage       Image: Second Parage       Image: Second Parage         Image: Second Parage       Image: Second Parage       Image: Second Parage       Image: Second Parage         Image: Second Parage       Image: Second Parage       Image: Second Parage       Image: Second Parage         Image: Second Parage       Image: Second Parage       Image: Second Parage       Image: Second Parage         Image: Second Parage       Image: Second Parage       Image: Second Parage       Image: Second Parage         Image: Second Parage       Image: Second Parage       Image: Second Parage       Image: Second Parage         Image: Second Parage       Image: Second Parage       Image: Second Parage       Image: Second Parage         Image: Second Parage       Image: Second Parage       Image: Second Parage       Image: Second Parage         Image: Second Parage       Image: Second Parage       Image: Second Parage       Image: Second Parage       Image: Second Parage         Image: Second Parage       Image: Second Parage       Image: Second Parage       Image: Second Parage       Image: Second Parage         Image: Second Parage       Image: Second Parage       Image: Second Parage       Image: Second Parage       Image: Second Parage				
Image: Second Structure       Image: Second Structure <td></td> <td></td> <td></td> <td></td>				
Image: Section of the section of th				
Image: Second Rest: Main + 3E       mov [esp], edx ; s         Image: Second Rest: Main + 3E       mov [esp], edx ; s         Image: Second Rest: Main + 41       call foo         Image: Second Rest: Second				EDX=FFDEDC8F
Image: Section of the section of th				
III 00007288 .text.foopush ebpESP=FFDEBBB8III 00007288 .text.foo+1mov ebp, espEBP=FFDEBBB8				
II 00007288 .text:foo+1 mov ebp, esp EBP=FFDEBBB8				
■ 00007288 .text:foo+3 sub esp, 28h ESP=FFDEBB90				
📒 00007288 .text:foo+6 mov eax, [ebp+s] EAX=FFDEDC8F	100007288	.text:foo+6	mov eax, [ebp+s]	EAX=FFDEDC8F

 $\overline{As}$  you can see, many events that were present in both the overlay & the main trace have been removed. Only those that were only present in the main trace remain.

### Reverting the diff

The diffing operation is reversible:

0040	22/3 110	ov eax,[eph_aida]				
	ad	d eax, 8		EAX=FFDEBC8C PF=0 ZF=		
	Сору	Ctrl+C		EAX=FFDEDC94		
	Copy all	Ctrl+Shift+Ins				
	🐺 Quick filter	Ctrl+F		ESP=FFDEBBBC		
	🎢 Modify filters	Ctrl+Shift+F		ESP=FFDEBBB8		
	Show trace info	Ctrl+I				
	Clear trace	Ctrl+X		EDX=FFDEBC84		
	Load trace	Ctrl+L		EDX=FFDEBC88 AF=0		
	Save trace	Ctrl+S		EDX=FFDEDC8F		
	Change trace desc					
	Show trace call gra	ph Ctrl+G		ESP=FFDEBBBC		
	Export trace to text	: file		ESP=FFDEBBB8		
	Overlay	$\rightarrow$	Show overlay info	Ctrl+Shift+I		
	su	b esp, 28h	Clear overlay	Ctrl+Shift+X		
	ma		Load overlay	Ctrl+Shift+L		
	ma		Revert subtraction			
	ca	-	ESPEFFDEBB8C Restore main trace to its original state			
	jm	p us.on_804A00C				

This will restore the main trace as it were, before the contents of the overlay were removed from it.

### **Replaying traces**

We know that the program is crashing somewhere in the call to **strcpy** but we don't know why the check at 0x080484FC passes since -1 is smaller than the size of the string (8 bytes). Let's put a breakpoint at the call to **strlen** at 0x080484F0, switch to the "Trace replayer" debugger, and "run" the program by pressing F9. Please note that we do not really run the program, we are merely replaying a previously recorded trace.

The debugger will stop at the **strlen** call:

6		Trace replayer 🔯 🔹 🛃 » 📮 »
		k ( )
	IDA View-EIP 🗶 🐖 Call Stack 🗶	🖉 General registers 🛛 🕅
	<pre>.text:08048424 .text:08048424 ; intcdecl foo(char *src, size_t size) .text:08048424 foo proc near ; CODE XREF: main+41↓p .text:08048424 ptr= dword ptr -OCh .text:08048424 src= dword ptr 8 .text:08048424 size= dword ptr 0Ch .text:08048424 push ebp .text:08048425 mov ebp, esp .text:08048425 mov ebp, 28h .text:08048425 mov esp, 28h .text:08048425 mov esp, 28h</pre>	A         EAX FFCOF41B         GP 0           EBX FFCOF41B         DF 0           ECX 0000000 G         IF 1           EDX FFCOF41B         SF 1           ESI 0000000 G         SF 1           ESI 0000000 G         AF 0           EDI 0000000 G         AF 0           ESF FFC0D3F8         PF 0           ESF FFC0D3F0 G         CF 0           EIF 080484F0 G         Foo+C           EFL 00000282         ST
EIP+	.text:080484ED mov [esp], eax ; s .text:080484F0 call strlen	
	.text:080484F5 mov edx, eax .text:080484F7 mov edx, eax .text:080484F7 mov edx, eax .text:080484FC jbe short loc_8048511 .text:080484FC jbe short loc_8048511 .text:080484FC mov dword ptr [esp], offset s ; "Too big!" .text:0804850A mov eax, 1 .text:0804850A mov eax, 1 .text:0804850J jmp short locret_8048555 .text:08048511 ;	

In the trace replayer we can use all usual debugging commands like "run to cursor" (F4), "single step" (F7), or "step over" (F8). Let's press F8 to step over the **strlen** call and check the result:

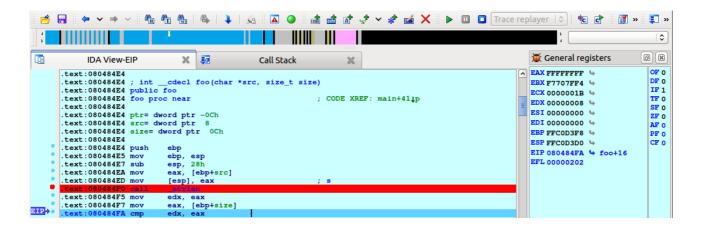
1 🖻 🔒	🗢 🗸 🔿 🗸 🖓	4 🐁   🌭   🕽	<u>k</u> (	o 💼 💼 🚺	.s <sup>‡</sup> ∽ ≉ ₫	i 🗙 🕨 💷	Trace rep	olayer 🔯 🛛 🖈	) 🛃 📑 🛃	» 🏮 »
		<b>1</b>						ь 4		\$
	IDA View-EIP	X 🗊	Call Sta	ck 🗶				👿 General reg	gisters	ØX
te te te te te te te te te te te te	xt:080484E4 xt:080484E4 ; int	foo oc near word ptr -OCh word ptr 8	] 511 ], offset s	; CODE X	REF: main+41,	ļβ		EAX 0000008 EBX F7707FF4 ECX 000001B EDX 0000000 EDI 0000000 EDI 0000000 EBP FFC0D3F8 ESP FFC0D3F8 ESP FFC0D3F8 EIP 00000202	6 6 6 6 6 6 6 6 6 6	OF 0 DF 0 IF1 FF 0 SF 0 ZF 0 AF 0 PF 0 CF 0

It returns 8 as expected. Now move to the address 0x080484FC and press F4 or right click on this address, select "Set IP", and press F7 (we need to inform the replayer plugin that we changed the current execution instruction in order to refresh all the register values). The difference between "Run to" (F4) and "Set IP" is that "Run to" will replay all events happened until that point but "Set IP" will directly move to the nearest trace event happened at this address (if it's in the recorded trace, of course).

Regardless of how we moved to this point IDA will display the following:

6	☴ ⇐ ▾ ⇒ ヾ ♣ ♣ ⓑ	🏝   🗼   🖗 🔺 🗶 📾 📾 🗗 🖈 🛩 🦨 🗙 🕨 🔲 🗖 Trace	replayer 🔯 👘 🛃 »	› 📮 »
			b	\$
	IDA View-EIP 🗙	5 Call Stack	👿 General registers	ØX
	<pre>.text:080484E4 .text:080484E4 ; intcdecl .text:080484E4 ; public foo .text:080484E4 foo proc near .text:080484E4 foo proc near .text:080484E4 ptr= dword pt .text:080484E4 size= dword pt .text:080484E4 size= dword pt .text:080484E4 mov ebp, .text:080484E7 sub esp, .text:080484E7 mov eax, .text:080484EA mov eax,</pre>	; CODE XREF: main+411p -OCh 8 c OCh sp sh bh bbp+src]	▲         EAX FFFFFFFF         ↓           EBX F7707F4         ↓         E           EDX 0000001         ↓         E           EDX 00000000         ↓         E           EDI 00000000         ↓         E           EDF FFCOD3P8         ↓         E           EIP 080484FC         ↓         foo+18           EFL 00000217         ↓	OF 0 DF 0 IF 1 TF 0 SF 0 ZF 0 AF 1 PF 1 CF 1
	.text:080484FA cmp edx, .text:080484FE mov dword text:080484FE mov dword text:08048505 call _puts .text:08048505 mov eax, .text:08048501 jmp short .text:08048511 ;	ax sbp+size]		

As we see, the jump was taken because CF was set to 1 in the previous instruction ("cmp edx, eax"). Let's step back to this instruction to see what values were compared. Select "Debugger  $\rightarrow$  Step back" from the menu:



The flags are reset to 0 and we can see that EAX (0xFFFFFFF) and EDX (8) are compared. Press F7 to step one instruction again and you will notice CF changes to 1. The instruction JBE performs an unsigned comparison between 8 and 0xFFFFFFFF and, as  $8 \le 0$ xFFFFFFFF, the check passes. We just discovered the cause of the bug.

Let's continue analyzing it a bit more. Scroll down until the call to malloc at 0x08048517, right click, choose "Set IP", and press F7 (or simply press F4). As we see, the argument given to malloc is 0xFFFFFFF (4 GB).

Press F8 to step over the function call:

6		🏝 🦀   🏝   🖡	🧟 🔼 🥥 📠	i 🖬 🗗 求 🛩 🖈 🖆	X 🕨 M 🔲 Trace re	player 🗘 🔁 🛃	» 🏮 v
4						4	\$
	IDA View-EIP	×	Call Stack	×		👿 General registers	ðx
	.text:080484E4				4		OF 0
	.text:080484E4 ; int		rc, size_t size	)		EBX F7707FF4 5	DF 0
	.text:080484E4 public					ECX 0000001 9	IF 1
	.text:080484E4 foo pr	roc near		; CODE XREF: main+41	P _	EDX FFFFFFFF \$	TF O
	.text:080484E4 .text:080484E4 ptr= 0	hund at a OCh				ESI 00000000 \	SF 0
	.text:080484E4 ptr= 0					EDI 00000000 4	ZF 1
	.text:080484E4 size=						AF 0
	.text:080484E4	anora por oon				EBP FFC0D3F8	PF 1
	.text:080484E4 push	ebp				ESP FFC0D3D0 4	CF 0
•	.text:080484E5 mov	ebp, esp				EIP 0804851C 🗣 foo+38	
	.text:080484E7 sub	esp, 28h				EFL 00000246	
•	.text:080484EA mov	eax, [ebp+src]					
	.text:080484ED mov	[esp], eax		; s			
	.text:080484F0 call	strlen					
	.text:080484F5 mov	edx, eax					
	.text:080484F7 mov	eax, [ebp+size]					
	.text:080484FA cmp	edx, eax short loc 8048511					
5.7	.text:080484FC jbe .text:080484FE mov	dword ptr [esp],	ffaat a	. "The big!"			
	.text:08048505 call	puts	STISEC S	, "Too big!"			
	.text:0804850A mov	eax, 1					
<u> </u>	.text:0804850F jmp	short locret 8048	555				
	.text:08048511 ;						
	.text:08048511						
	.text:08048511 loc_80			; CODE XREF: foo+18fj			
- <b>-</b>	.Cexc.08048511 MOV	eax, [ebp+size]					
	.text:08048514 mov	[esp], eax		; size			
EIP	.text:08048517 call	_malloc					
	.text:0804851C mov	[ebp+ptr], eax					
	.text:0804851F mov	eax, [ebp+src]					
	.text:08048522 mov .text:08048526 mov	[esp+4], eax eax, [ebp+ptr]		; src			
	.text:08048529 mov	[esp], eax		; dest			
	.text:0804852C call	[esp], eax _strcpy		, 4636			
- III -	CONCLOSED OF CALL	_serepy					

Obviously, **malloc** can not allocate so much memory and returns NULL. However, the program does not check for this possibility and tries to copy the contents of the given buffer to the address 0, resulting in a crash.

# Summary

In this tutorial we showed you the basics of trace management and the trace replayer module in IDA. We hope you enjoy this new feature. Happy debugging!