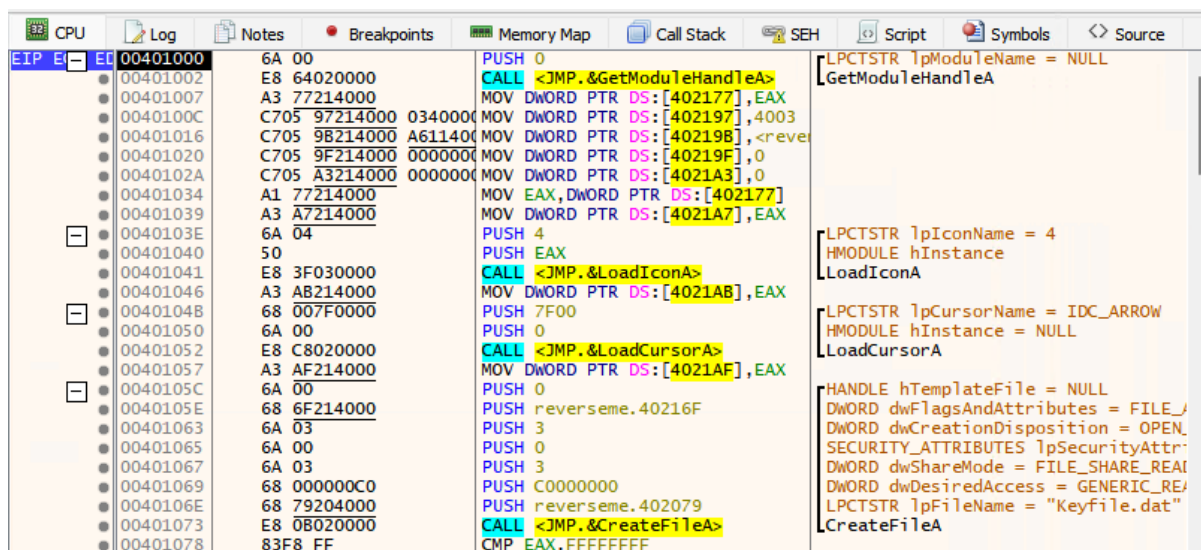


# X32Dbg/X64Dbg – Debugger Guide

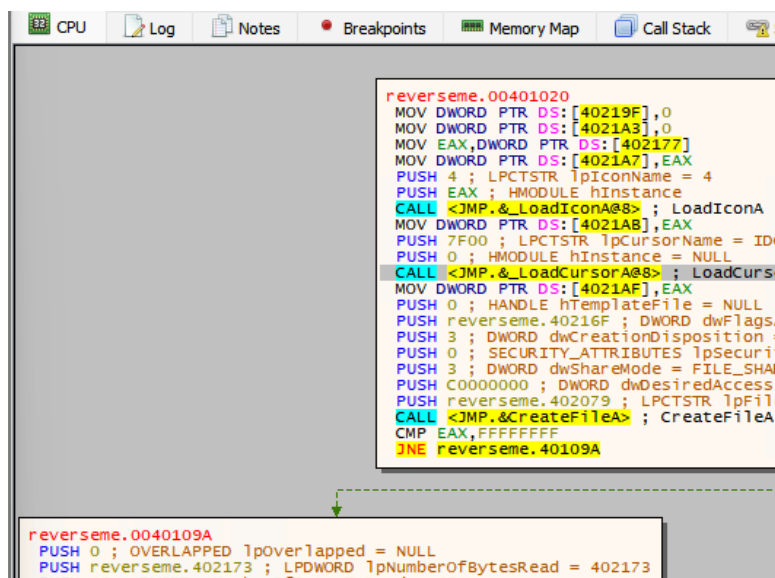
## CPU - Main Screen:



## Left – Right (Sections):

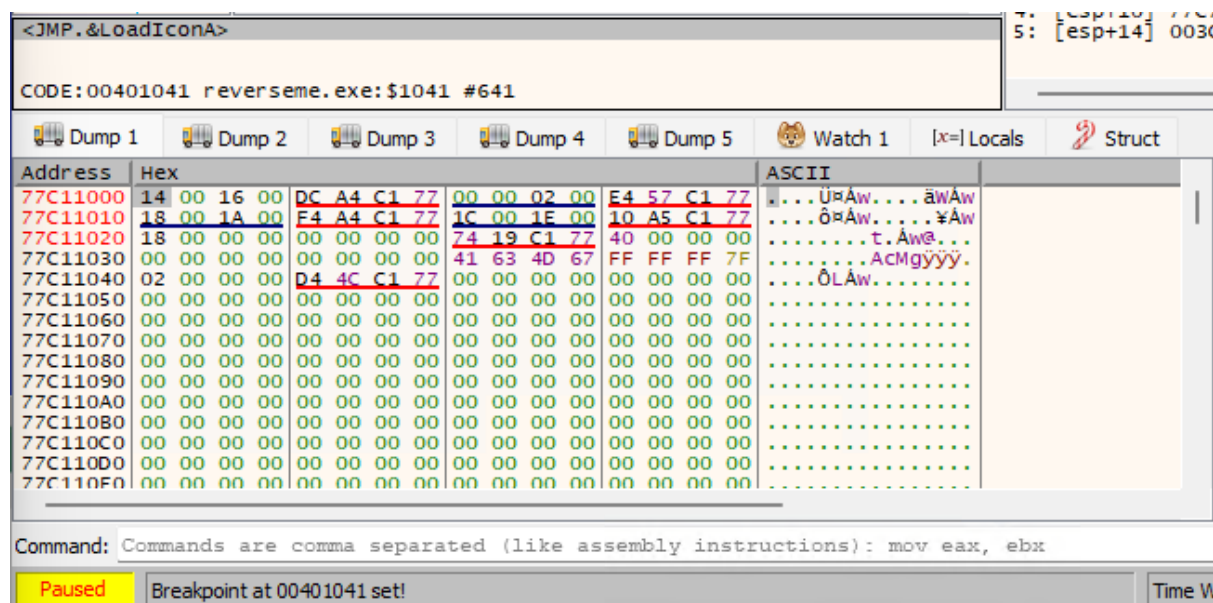
([Breakpoint/Registers Location](#)) | ([VAS – Virtual Address space](#)) |  
([Opcode](#)) | ([Disassembly](#)) | (Comments / API's ([xAnalyzer](#)))

## CPU - Main Screen (Graph Mode):



## X32Dbg/X64Dbg – Debugger Guide

CPU - Bottom Screen:



Top – Bottom (Sections):

(Execution Window)

([5 Hex Dumps](#)) | ([Watch 1](#)) | (Locals) | ([Struct](#))

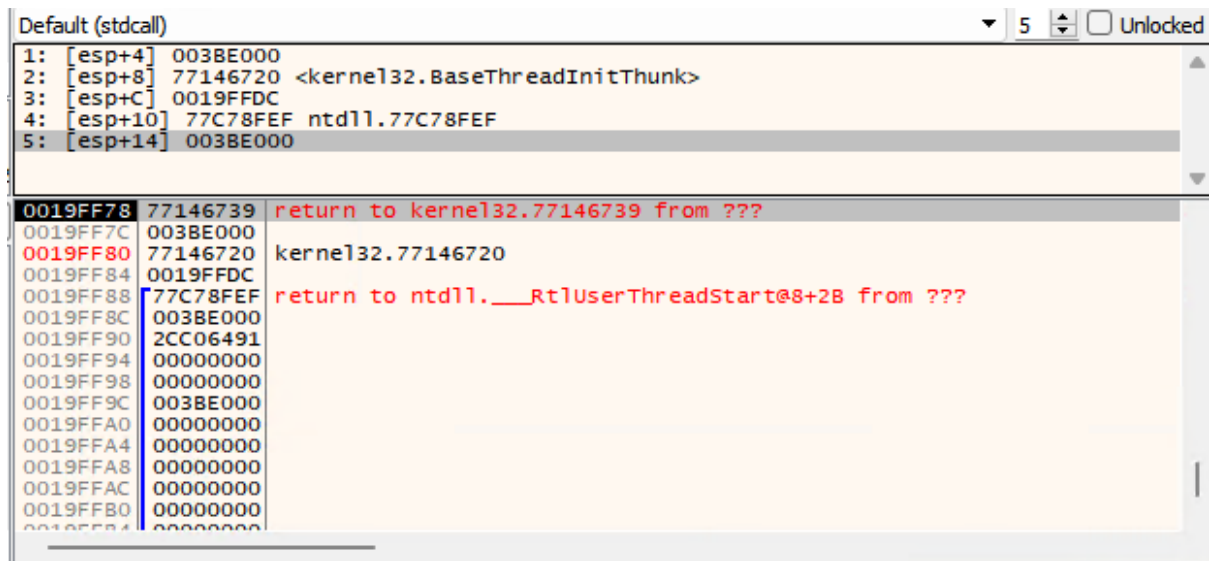
([Command Bar](#))

(Status Bar/ Info Bar/ Time Debugging Bar)

Notes: Struct is use to import header file Structs (Check Link Above). Parse header (Select File) + Visit Type (Struct name) + (nan) Function

## X32Dbg/X64Dbg – Debugger Guide

### CPU - Stack Screen:



### Top – Bottom (Sections):

([Stack Trace](#) - Menu bar) | (No. of [Stack Traces](#) (Default: 5)) | (Stack Tracer Status: (Unlocked/Calls/Locked))

(SP/ESP/RSP: Stack pointer for top address of the stack) | ([VAS – Virtual Address space](#)) | (Comments)

## X32Dbg/X64Dbg – Debugger Guide

CPU - Registers Screen:

X32Dbg / X64Dbg

Show FPU	Show FPU
EAX 0019FFCC	RAX 0000000000937B30 <hxd.EntryPoi
EBX 00284000	RBX 0000000000000000
ECX 00401000 <reverseme.EntryPoint>	RCX 00000000003AE000
EDX 00401000 <reverseme.EntryPoint>	RDX 0000000000937B30 <hxd.EntryPoi
EBP 0019FF84	RBP 0000000000000000
ESP 0019FF78	RSP 000000000014FF28
ESI 00401000 <reverseme.EntryPoint>	RSI 0000000000000000
EDI 00401000 <reverseme.EntryPoint>	RDI 0000000000000000
EIP 00401000 <reverseme.EntryPoint>	R8 00000000003AE000
EFLAGS 00000244	R9 0000000000937B30 <hxd.EntryPoi
ZF 1 PF 1 AF 0	R10 00007FFE5F737C90 kernel32.0000
OF 0 SF 0 DF 0	R11 0000000000000000
CF 0 TF 0 IF 1	R12 0000000000000000
LastError 0000012A (ERROR_TOO_MANY_POSTS)	R13 0000000000000000
LastStatus C0000047 (STATUS_SEMAPHORE_LIM	R14 0000000000000000
GS 002B FS 0053	R15 0000000000000000
ES 002B DS 002B	RIP 0000000000937B30 <hxd.EntryPoi
CS 0023 SS 002B	RFLAGS 0000000000000244
DR0 00000000	ZF 1 PF 1 AF 0
DR1 00000000	OF 0 SF 0 DF 0
DR2 00000000	CF 0 TF 0 IF 1
DR3 00000000	LastError 00000000 (ERROR_SUCCESS)
DR6 00000000	LastStatus C0000023 (STATUS_BUFFER_TOO_S
DR7 00000000	GS 002B FS 0053
	ES 002B DS 002B
	CS 0033 SS 002B
	DR0 0000000000000000
	DR1 0000000000000000
	DR2 0000000000000000
	DR3 0000000000000000
	DR6 0000000000000000
	DR7 0000000000000000

General Purpose Registers:

- AL/AH/AX/EAX/RAX: Accumulator
- BL/BH/BX/EBX/RBX: Base index (for use with arrays)
- CL/CH/CX/ECX/RCX: Counter (for use with loops and strings)
- DL/DH/DX/EDX/RDX: Extend the precision of the accumulator (e.g. combine 32-bit EAX and EDX for 64-bit integer operations in 32-bit code)
- BP/EBP/RBP: Stack base pointer for holding the address of the current [stack frame](#).
- SP/ESP/RSP: Stack pointer for top address of the stack.
- SI/ESI/RSI: *Source index* for [string](#) operations.
- DI/EDI/RDI: *Destination index* for string operations.
- R8 – R15(x64 only): Eight additional 64-bit general registers.
- IP/EIP/RIP: Instruction pointer. Holds the [program counter](#), the address of next instruction.

# X32Dbg/X64Dbg – Debugger Guide

## Control Registers:

- [ZF – Zero Flag](#)
- [OF – Overflow Flag](#)
- [CF – Carry Flag](#)
- [PF – Parity Flag](#)
- [SF – Sign Flag](#)
- [TF – Trap Flag](#)
- [AF – Adjust Flag](#)
- [DF – Direction Flag](#)
- [IF – Interrupt Enable Flag](#)

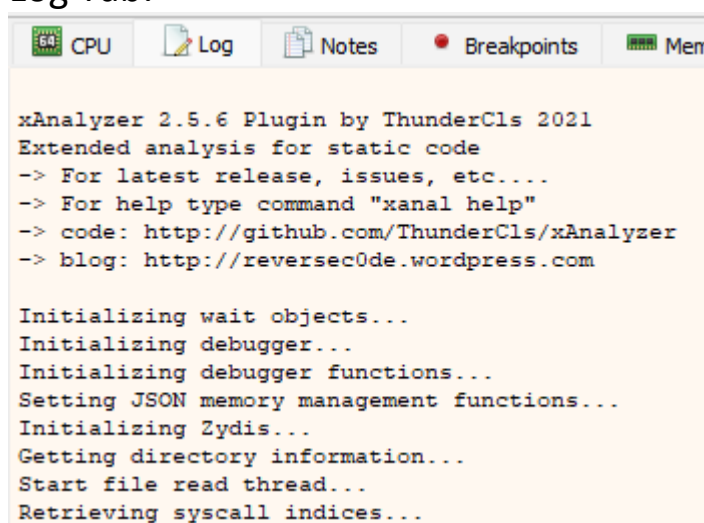
## Segment Registers:

- CS: Code
- DS: Data
- SS: Stack
- ES: Extra data
- FS: Extra data #2
- GS: Extra data #3

## Debug Registers:

- DR0 -> DR3: Breakpoints
- DR6: Debug Status
- DR7: Debug Control

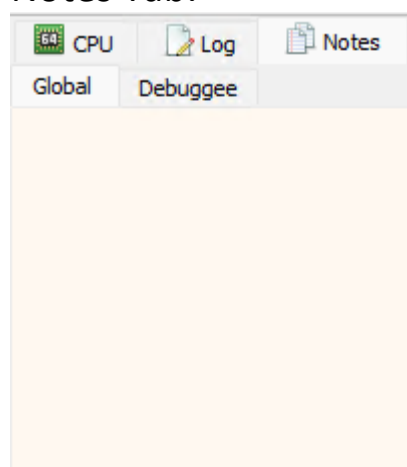
## Log Tab:



```
xAnalyzer 2.5.6 Plugin by ThunderClis 2021
Extended analysis for static code
-> For latest release, issues, etc....
-> For help type command "xanal help"
-> code: http://github.com/ThunderClis/xAnalyzer
-> blog: http://reversecode.wordpress.com

Initializing wait objects...
Initializing debugger...
Initializing debugger functions...
Setting JSON memory management functions...
Initializing Zydis...
Getting directory information...
Start file read thread...
Retrieving syscall indices...
```

## Notes Tab:



[GUI Manual - Notes](#)

## [GUI Manual - Log](#)

## Breakpoints Tab:

Type	Address	Module/Label/Exception	State	Disassembly	Hits	Summary
Software	0000000000937883	hxd.exe	Enabled	CALL <hxd.sub_74EAE0>	0	
Hardware	0000000000937892	hxd.exe	Enabled	CALL <hxd.sub_6C12B0>	0	execute()

There are 2 types of breakpoints: (Software / Hardware).  
Hardware only has (4 breakpoint Slots).

## [GUI Manual – Conditional Breakpoints](#)

# X32Dbg/X64Dbg – Debugger Guide

## Memory Map Tab:

Address	Size	Info	Content	Type	Protection	Initial
0000000000010000	0000000000001000	Reserved Thread 2160 Stack		MAP	-R---	-R---
0000000000020000	0000000000001000			MAP	-R---	-R---
0000000000030000	0000000000001F000			MAP	-R---	-R---
0000000000050000	000000000000F8000			PRV	-RW--	-RW--
00000000000148000	0000000000008000			PRV	-RW-G	-RW--
00000000000150000	0000000000004000	\Device\HarddiskVolume3\Win		MAP	-R---	-R---
00000000000160000	0000000000002000			MAP	-R---	-R---
00000000000170000	0000000000002000			PRV	-RW--	-RW--
00000000000180000	00000000000011000			MAP	-R---	-R---
000000000001A0000	00000000000011000			MAP	-R---	-R---
000000000001C0000	0000000000003000	Reserved (00000000001D0000)		MAP	-R---	-R---
000000000001D0000	0000000000002000			PRV	-RW--	-RW--
000000000001D2000	0000000000008000			PRV	-RW--	-RW--
000000000001E0000	00000000000010000			MAP	-RW--	-RW--
000000000001F0000	0000000000003000			MAP	-R---	-R---
00000000000200000	0000000000014D000	Reserved		PRV	-RW--	-RW--
0000000000034D000	0000000000009000	PEB		PRV	-RW--	-RW--
00000000000356000	000000000000AA000	Reserved (0000000000200000)		PRV	-RW--	-RW--
00000000000400000	0000000000001000	hxd.exe		IMG	-R---	ERWC-
00000000000401000	000000000000537000	".text"	Executable code	IMG	ER---	ERWC-
00000000000938000	000000000000C8000	".data"	Initialized data	IMG	-RW-	ERWC-
00000000000A00000	000000000000D0000	".bss"	Uninitialized data	IMG	-RW-	ERWC-
00000000000A0D000	0000000000006000	".idata"	Import tables	IMG	-RW-	ERWC-
00000000000A13000	00000000000001000	".didata"		IMG	-RW-	ERWC-
00000000000A14000	00000000000001000	".tls"	Thread-local storage	IMG	-RW-	ERWC-
00000000000A15000	00000000000001000	".rdata"	Read-only initialized data	IMG	-R---	ERWC-
00000000000A16000	0000000000003D000	".pdata"	Exception information	IMG	-R---	ERWC-
00000000000A53000	00000000000055000	".rsrc"	Resources	IMG	-R---	ERWC-

## Memory Map:

In [computer science](#), a **memory map** is a structure of data (which usually resides in memory itself) that indicates how [memory](#) is laid out.

## Call Stack Tab:

Thread	Address	To	From	Size	Comment	Party
8544	000000000014FF28	00007FFE5F6854E0	00007FFE5F6C4100	30	return to kernel32.00007FFE5F6854E0 from kernel32	System
	000000000014FF58	00007FFE6068485B			return to ntdll.00007FFE6068485B from ???	System
6980	0000000000EBFC38	00007FFE60696FDF	00007FFE60727A50	2F0	return to ntdll.00007FFE60696FDF from ntdll.00007	System
	0000000000EBFF28	00007FFE5F6854E0	00007FFE5F6C4100	30	return to kernel32.00007FFE5F6854E0 from kernel32	System
	0000000000EBFF58	00007FFE6068485B			return to ntdll.00007FFE6068485B from ???	System
4344	0000000000DBFC38	00007FFE60696FDF	00007FFE60727A50	2F0	return to ntdll.00007FFE60696FDF from ntdll.00007	System
	0000000000DBFF28	00007FFE5F6854E0	00007FFE5F6C4100	30	return to kernel32.00007FFE5F6854E0 from kernel32	System
	0000000000DBFF58	00007FFE6068485B			return to ntdll.00007FFE6068485B from ???	System

A **call stack** is a [stack data structure](#) that stores information about the active [subroutines](#) of a [computer program](#)

## GUI Manual – Call Stack



# X32Dbg/X64Dbg – Debugger Guide

## Structured Exception Handler (SEH) Chain Tab:

CPU	Log	Notes	Breakpoints	Memory Map	Call Stack	SEH
Address	Handler	Module/Label	Comment			

[Wikipedia – SEH mechanism](#)

## Script Tab:

CPU	Log	Notes	Breakpoints	Memory Map	Call Stack	SEH	Script
Line	Text	Info					
0001	dbh						
0002	ret						

## Symbols Tab:

all Stack	SEH	Script	Symbols	Source	References	Threads	Handles	Trace	Snowman
Base	Module	Address	Type	Ordinal	Symbol	Symbol			
0000000004000000	hxd.exe	00000000A0E778	Import		advapi32.GetFileSecurityW				
00007FFE424F0000	oleacc.dll	00000000A0E780	Import		advapi32.FreeSid				
00007FFE46B00000	wininet.dll	00000000A0E788	Import		advapi32.EqualSid				
00007FFE51520000	comctl32.dll	00000000A0E790	Import		advapi32.AllocateAndInitializeSid				
00007FFE51920000	winmm.dll	00000000A0E798	Import		advapi32.AdjustTokenPrivileges				
00007FFE54C70000	winpool.drv	00000000A0E7A8	Import		user32.MessageBoxA				
00007FFE59A20000	version.dll	00000000A0E7B0	Import		user32.CharNextW				
00007FFE5DB70000	ucrtbase.dll	00000000A0E7B8	Import		user32.LoadStringW				
00007FFE5DD00000	win32u.dll	00000000A0E7C0	Import		user32.SetClassLongPtrW				
00007FFE5DDF0000	kernelbase.dll	00000000A0E7C8	Import		user32.GetClassLongPtrW				
00007FFE5E360000	msvcrt.dll	00000000A0E7D0	Import		user32.SetWindowLongPtrW				
00007FFE5E400000	gdi32.dll	00000000A0E7D8	Import		user32.GetWindowLongPtrW				
00007FFE5E580000	sechost.dll	00000000A0E7E0	Import		user32.CreateWindowExW				
00007FFE5E990000	gdi32.dll	00000000A0E7E8	Import		user32.WindowFromPoint				
00007FFE5EE50000	shlwapi.dll	00000000A0E7F0	Import		user32.WaitMessage				
00007FFE5EF60000	ole32.dll	00000000A0E7F8	Import		user32.ValidateRect				
00007FFE5F110000	advapi32.dll	00000000A0E800	Import		user32.UpdateWindow				
00007FFE5F1C0000	oleaut32.dll	00000000A0E808	Import		user32.UnregisterClassW				
00007FFE5F2A0000	combase.dll	00000000A0E810	Import		user32.UnhookWindowsHookEx				
00007FFE5F6A0000	kernel32.dll	00000000A0E818	Import		user32.TranslateMessage				
00007FFE5F7D0000	msvcrt.dll	00000000A0E820	Import		user32.TranslateMDISysAccel				
00007FFE5F880000	shcore.dll	00000000A0E828	Import		user32.TrackPopupMenu				
00007FFE5F970000	imm32.dll	00000000A0E830	Import		user32.ToAscii				
00007FFE5F9D0000	shell32.dll	00000000A0E838	Import		user32.SystemParametersInfoW				
00007FFE60210000	comdlg32.dll	00000000A0E840	Import		user32.SubtractRect				
00007FFE60300000	user32.dll	00000000A0E848	Import		user32.ShowWindowAsync				
00007FFE60520000	rpcrt4.dll	00000000A0E850	Import		user32.ShowWindow				
00007FFE60680000	ntdll.dll	00000000A0E858	Import		user32.ShowScrollBar				
		00000000A0E860	Import		user32.ShowOwnedPopups				
		00000000A0E868	Import		user32.ShowCaret				
		00000000A0E870	Import		user32.SetWindowRgn				

## X32Dbg/X64Dbg – Debugger Guide

### References Tab:

All Modules (Calls)		
Address	Disassembly	Destination
00000000005517DC	call <JMP.&ImageList_LoadImageW>	<comctl32.ImageList_LoadImageW>
000000000068A8C8	call <JMP.&ImageList_LoadImageW>	<comctl32.ImageList_LoadImageW>
00000000008353F1	call <JMP.&LoadBitmapW>	<user32.LoadBitmapW>
000000000069518E	call <JMP.&LoadBitmapW>	<user32.LoadBitmapW>
000000000083574D	call <JMP.&LoadBitmapW>	<user32.LoadBitmapW>
000000000081D3CD	call <JMP.&LoadBitmapW>	<user32.LoadBitmapW>
00000000008BE61C	call <JMP.&LoadBitmapW>	<user32.LoadBitmapW>
00000000006936A6	call <JMP.&LoadBitmapW>	<user32.LoadBitmapW>
000000000058617D	call <JMP.&LoadCursorW>	<user32.LoadCursorW>
00000000006E6886	call <JMP.&LoadCursorW>	<user32.LoadCursorW>
000000000071AA87	call <JMP.&LoadCursorW>	<user32.LoadCursorW>
000000000074F451	call <JMP.&LoadCursorW>	<user32.LoadCursorW>
000000000041BF06	call <JMP.&LoadCursorW>	<user32.LoadCursorW>
00000000006BA571	call <JMP.&LoadCursorW>	<user32.LoadCursorW>
00000000006E756B	call <JMP.&LoadCursorW>	<user32.LoadCursorW>
00000000006B9E0B	call <JMP.&LoadCursorW>	<user32.LoadCursorW>
00000000006B9E84	call <JMP.&LoadCursorW>	<user32.LoadCursorW>
00000000006E5326	call <JMP.&LoadCursorW>	<user32.LoadCursorW>
0000000000704DF8	call <JMP.&LoadCursorW>	<user32.LoadCursorW>
000000000071AEF3	call <JMP.&LoadCursorW>	<user32.LoadCursorW>
00000000006E7512	call <JMP.&LoadCursorW>	<user32.LoadCursorW>
0000000000746DCE	call <JMP.&LoadCursorW>	<user32.LoadCursorW>
000000000074F412	call <JMP.&LoadCursorW>	<user32.LoadCursorW>
000000000083E462	call <JMP.&LoadCursorW>	<user32.LoadCursorW>
00000000007047CA	call <JMP.&LoadCursorW>	<user32.LoadCursorW>

### Threads Tab:

Threads							
Number	ID	Entry	TEB	RIP	Suspend Count	Priority	Wait Reason
Main	3684	0000000000937B30	0000000000334000	0000000000937B30	1	Normal	Executive
1	4896	00007FFE60696D00	0000000000336000	00007FFE60727A64	1	Normal	Suspended
2	4824	00007FFE60696D00	0000000000338000	00007FFE60727A64	1	Normal	Suspended



# X32Dbg/X64Dbg – Debugger Guide

## Handles Tab:

all Stack SEH Script Symbols Source References Threads Handles

Proc	Handle	Title

Search:

▼Handles

Type	Type num	Handle	Access	Name
Event	12	4	1F0003	
Event	12	8	1F0003	
ALPC Port	31	C	1F0001	
Key	2F	10	9	\REGISTRY\MACHINE\SOFTWARE\Micro:
Event	12	14	1F0003	
WaitCompletionPa	26	18	1	
IoCompletion	25	1C	1F0003	
TpWorkerFactory	20	20	F00FF	
IRTimer	17	24	100002	
WaitCompletionPa	26	28	1	
IRTimer	17	2C	100002	
WaitCompletionPa	26	30	1	
	35	34	804	ERROR_NOT_SUPPORTED
	35	38	804	ERROR_NOT_SUPPORTED
	35	3C	804	ERROR_NOT_SUPPORTED
	35	40	804	ERROR_NOT_SUPPORTED
Directory	3	44	3	\KnownDlls
Event	12	48	1F0003	
Event	12	4C	1F0003	
File	27	50	100020	\Device\HarddiskVolume3\Windows
Event	12	54	1F0003	
Event	12	58	1F0003	
Directory	3	5C	3	\KnownDlls32
Key	2F	60	9	\REGISTRY\MACHINE\SOFTWARE\Micro:
Event	12	64	1F0003	
WaitCompletionPa	26	68	1	
IoCompletion	25	6C	1F0003	
TnWorkerFactory	20	70	F00FF	

Search:

▼TCP Connections

Remote address	Local address

## X32Dbg/X64Dbg – Debugger Guide

### Trace Tab:

all Stack	SEH	Script	Symbols	Source	References	Threads	Handles	Trace	Snowman
00	00401000	6A 00		push 0				EAX: 19FFCC-	
01	00401002	E8 64020000		call <JMP.&GetModuleHandleA>				EAX: 19FFCC-	Show FPU
02	00401007	A3 77214000		mov dword ptr ds:[402177],eax					EAX 0019FFCC
03	0040100C	C705 98214000 03400000		mov dword ptr ds:[402197],4003					EBX 0021A000
04	00401016	C705 98214000 A6114000		mov dword ptr ds:[40219B],<reverseme.su					ECX 00401000
05	00401020	C705 9F214000 00000000		mov dword ptr ds:[40219F],0					EDX 00401000
06	0040102A	C705 A3214000 00000000		mov dword ptr ds:[4021A3],0					EBP 0019FF84
07	00401034	A1 77214000		mov eax,dword ptr ds:[402177]					ESP 0019FF78
08	00401039	A3 A7214000		mov dword ptr ds:[4021A7],eax					ESI 00401000
09	0040103E	6A 04		push 4				ESP: 19FF78-	EDI 00401000
0A	00401040	50		push eax				ESP: 19FF74-	EIP 00401000
0B	00401041	E8 3F030000		call <JMP.&LoadIconA@8>				EAX: 400000-	
0C	00401046	A3 AB214000		mov dword ptr ds:[4021AB],eax					
0D	0040104B	68 007F0000		push 7F00				ESP: 19FF78-	
0E	00401050	6A 00		push 0				ESP: 19FF74-	
0F	00401052	E8 C8020000		call <JMP.&LoadCursorA@8>				EAX: 2E405E9	
10	00401057	A3 AF214000		mov dword ptr ds:[4021AF],eax					EFLAGS 00000244
11	0040105C	6A 00		push 0				ESP: 19FF78-	ZF 1 PF 1 AF 0
12	0040105E	68 6F214000		push reverseme.40216F				ESP: 19FF74-	OF 0 SF 0 DF 0
13	00401063	6A 03		push 3				ESP: 19FF70-	CF 0 TF 0 IF 1
14	00401065	6A 00		push 0				ESP: 19FF6C-	LastError 000001
15	00401067	6A 03		push 3				ESP: 19FF68-	LastStatus 000000
16	00401069	68 000000C0		push C0000000				ESP: 19FF64-	
17	0040106E	68 79204000		push reverseme.402079				ESP: 19FF60-	GS 0028 FS 0053
18	00401073	E8 08020000		call <JMP.&CreateFileA>				EAX: 10003->	ES 0028 DS 0028
19	00401078	83F8 FF		cmp eax,FFFFFFFF					CS 0023 SS 0028
1A	0040107B	75 1D		jne reverseme.40109A					
1B	0040107D	6A 00		push 0				ESP: 19FF78-	DR0 00000000
1C	0040107F	68 00204000		push reverseme.402000				ESP: 19FF74-	DR1 00000000
1D	00401084	68 17204000		push reverseme.402017				ESP: 19FF70-	DR2 00000000
1E	00401089	6A 00		push 0				ESP: 19FF6C-	DR3 00000000
1F	0040108B	E8 D7020000		call <JMP.&MessageBoxA@16>					DR6 00000000
									DR7 00000000

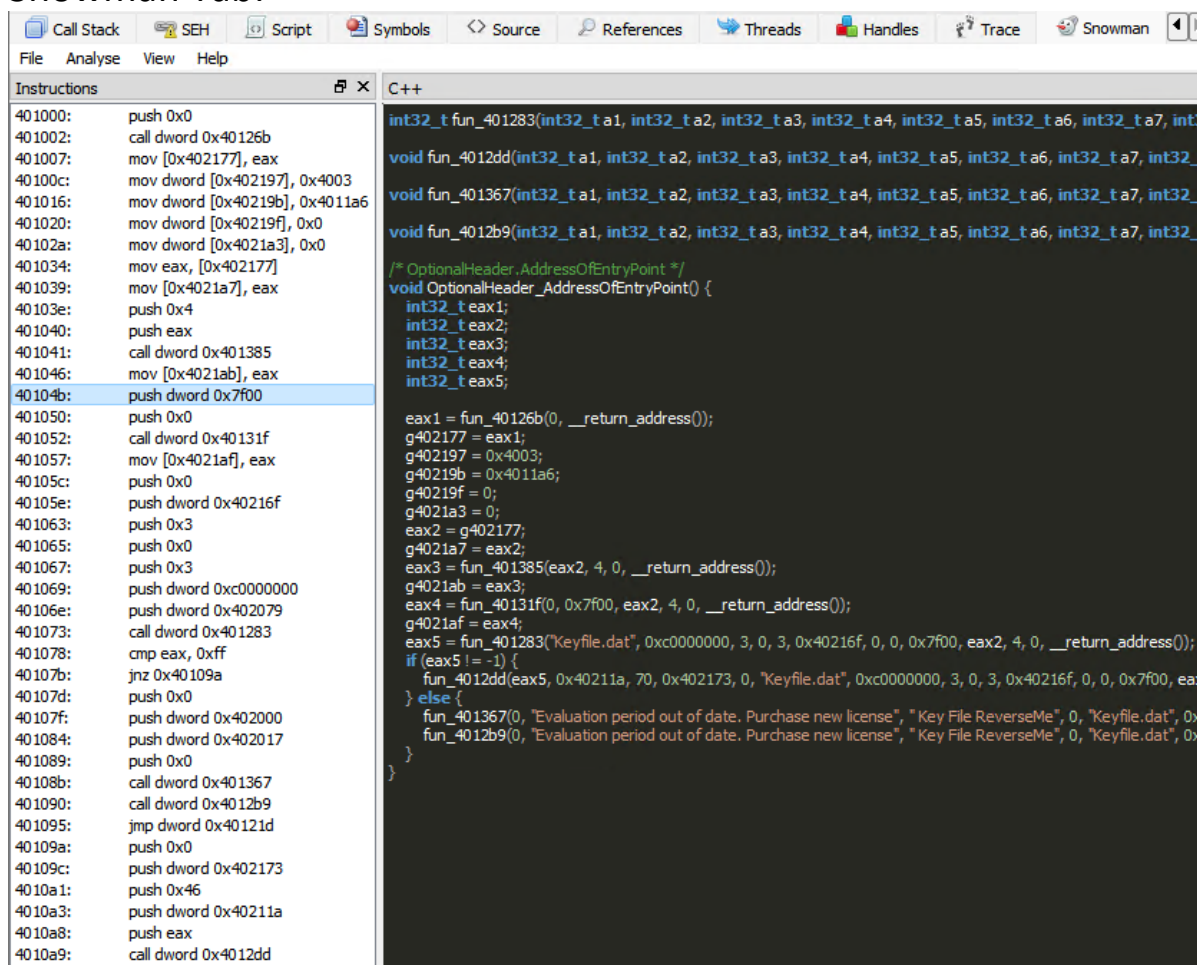
### Left – Right (Sections):

(Steps) | ([VAS – Virtual Address space](#)) | ([Opcode](#)) | ([Disassembly](#))  
| (Comments / API's ([xAnalyzer](#)))

[GUI Manual - Trace](#)

# X32Dbg/X64Dbg – Debugger Guide

## Snowman Tab:



The screenshot displays the Snowman tab interface, which is used for decompiling assembly code into C++ source code. The interface is divided into two main panes: 'Instructions' on the left and 'C++' on the right. The 'Instructions' pane shows a list of assembly instructions with their addresses, such as '401000: push 0x0' and '401002: call dword 0x40126b'. The 'C++' pane shows the corresponding decompiled C++ code, including function definitions like 'int32\_t fun\_401283(int32\_t a1, int32\_t a2, int32\_t a3, int32\_t a4, int32\_t a5, int32\_t a6, int32\_t a7, int32\_t a8)' and 'void fun\_4012dd(int32\_t a1, int32\_t a2, int32\_t a3, int32\_t a4, int32\_t a5, int32\_t a6, int32\_t a7, int32\_t a8)'. The C++ code includes comments and variable declarations, such as '/\* OptionalHeader.AddressOfEntryPoint \*/' and 'int32\_t eax1; int32\_t eax2; int32\_t eax3; int32\_t eax4; int32\_t eax5;'. The interface also includes a menu bar with options like 'File', 'Analyse', 'View', and 'Help', and a toolbar with icons for various debugging functions.

## Snowman - C++ Decompiler (Plugin)

<https://github.com/x64dbg/snowman/releases/tag/plugin-v1>