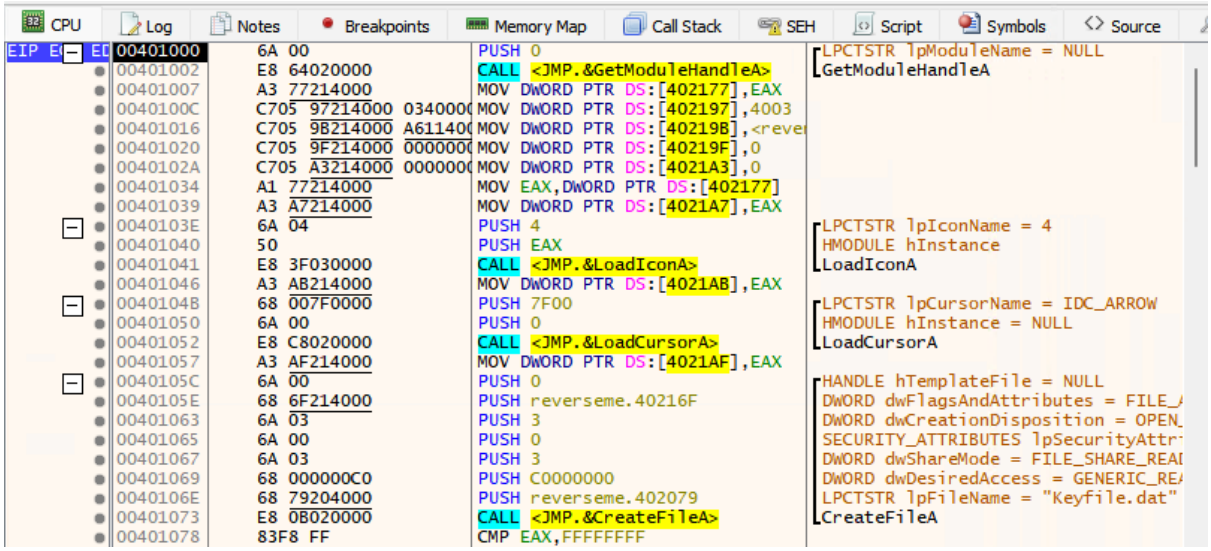
X32Dbg/X64Dbg – Debugger Guide

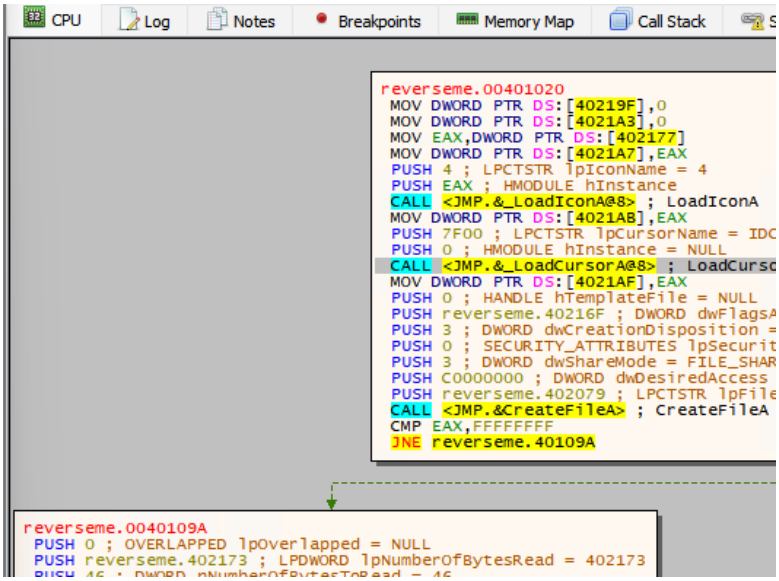
CPU - Main Screen:



Left – Right (Sections):

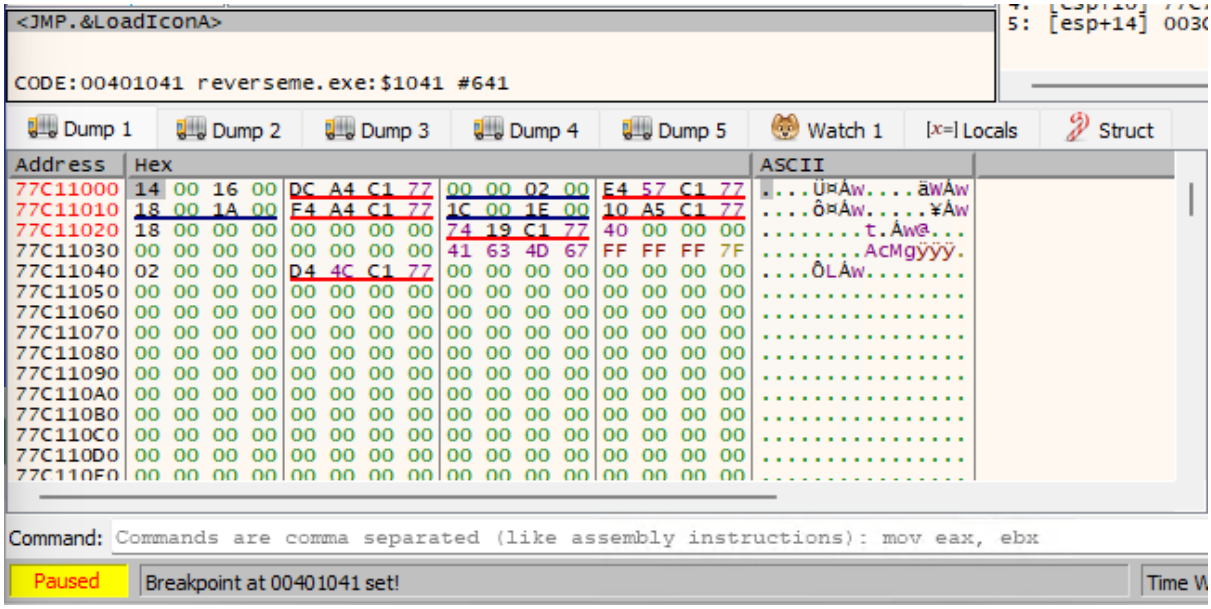
([Breakpoint](#)/[Registers Location](#)) | ([VAS – Virtual Address space](#)) | ([Opcode](#)) | ([Disassembly](#)) | (Comments / API's ([xAnalyzer](#)))

CPU - Main Screen (Graph Mode):



X32Dbg/X64Dbg – Debugger Guide

CPU - Bottom Screen:



Top – Bottom (Sections):

(Execution Window)

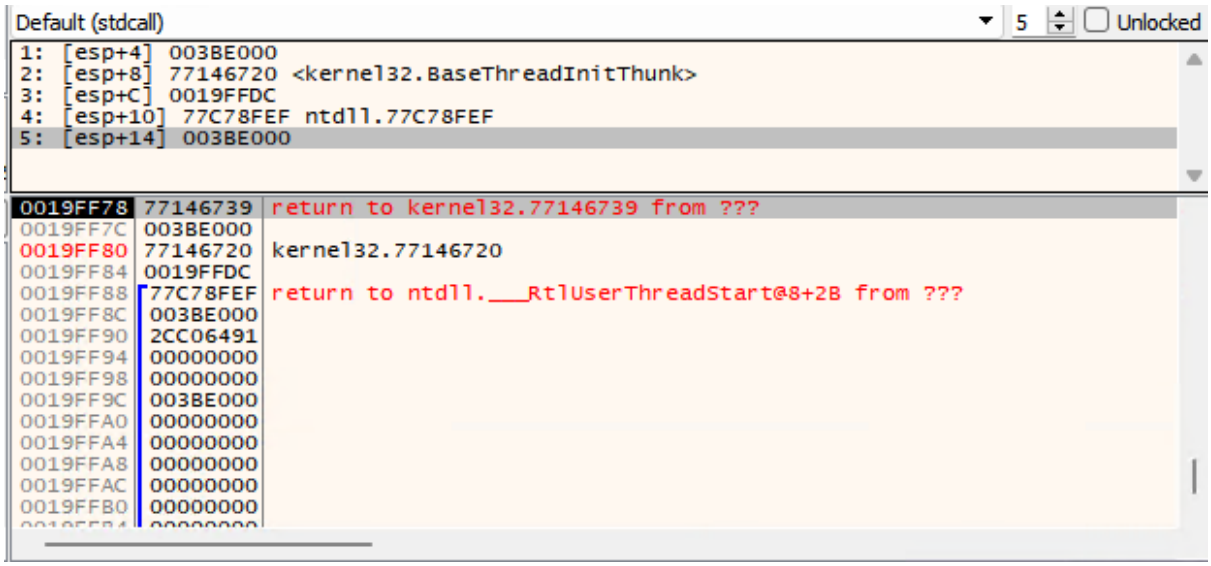
(5 Hex Dumps) | (Watch 1) | (Locals) | (Struct)

(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

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)

CPU - Registers Screen:

X32Dbg / X64Dbg

```
Show FPU

EAX   0019FFCC
EBX   00284000
ECX   00401000     <reverseme.EntryPoint>
EDX   00401000     <reverseme.EntryPoint>
EBP   0019FF84
ESP   0019FF78
ESI   00401000     <reverseme.EntryPoint>
EDI   00401000     <reverseme.EntryPoint>

EIP   00401000     <reverseme.EntryPoint>

EFLAGS   00000244
ZF 1  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 0  IF 1

LastError   0000012A (ERROR_TOO_MANY_POSTS)
LastStatus  C0000047 (STATUS_SEMAPHORE_LIM

GS 002B  FS 0053
ES 002B  DS 002B
CS 0023  SS 002B

DR0 00000000
DR1 00000000
DR2 00000000
DR3 00000000
DR6 00000000
DR7 00000000
```

```
Show FPU

RAX   0000000000937B30     <hxd.EntryPoi
RBX   0000000000000000
RCX   00000000003AE000
RDX   0000000000937B30     <hxd.EntryPoi
RBP   0000000000000000
RSP   000000000014FF28
RSI   0000000000000000
RDI   0000000000000000

R8    00000000003AE000
R9    0000000000937B30     <hxd.EntryPoi
R10   00007FFE5F737C90     kernel32.0000
R11   0000000000000000
R12   0000000000000000
R13   0000000000000000
R14   0000000000000000
R15   0000000000000000

RIP   0000000000937B30     <hxd.EntryPoi

RFLAGS   0000000000000244
ZF 1  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 0  IF 1

LastError   00000000 (ERROR_SUCCESS)
LastStatus  C0000023 (STATUS_BUFFER_TOO_S

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

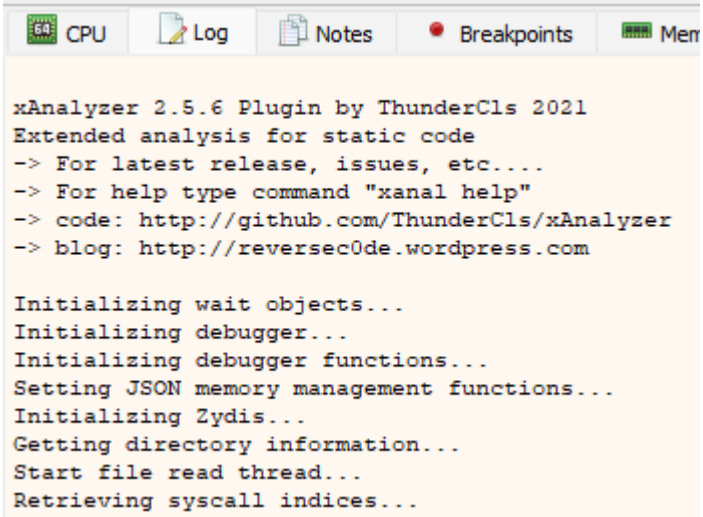
Segment Registers:

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

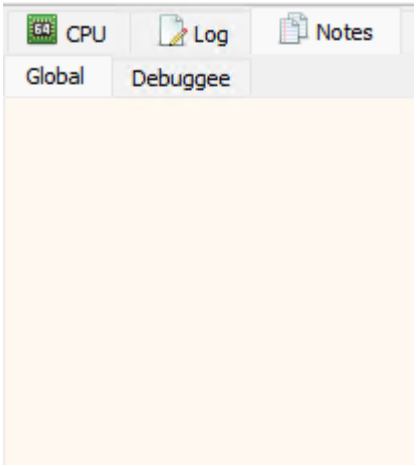
Debug Registers:

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

Log Tab:



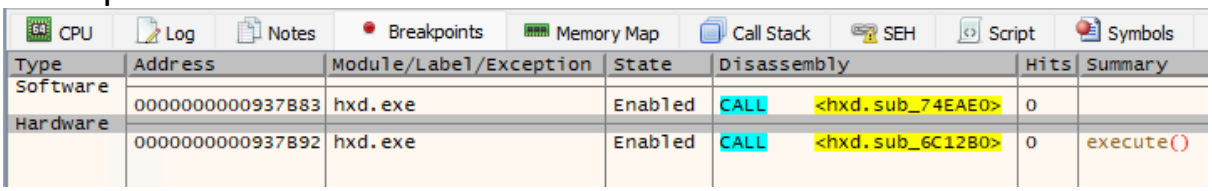
Notes Tab:



GUI Manual - Notes

GUI Manual - Log

Breakpoints Tab:



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:

CPU | Log | Notes | Breakpoints | Memory Map | Call Stack | SEH | Script | Symbols | Source

Address	Size	Info	Content	Type	Protection	Initial
0000000000010000	0000000000001000			MAP	-R---	-R---
0000000000020000	0000000000001000			MAP	-R---	-R---
0000000000030000	000000000001F000			MAP	-R---	-R---
0000000000050000	00000000000F8000	Reserved		PRV		-RW--
0000000000148000	0000000000008000	Thread 2160 Stack		PRV	-RW-G	-RW--
0000000000150000	0000000000004000			MAP	-R---	-R---
0000000000160000	0000000000002000			MAP	-R---	-R---
0000000000170000	0000000000002000			PRV	-RW--	-RW--
0000000000180000	0000000000011000	\Device\HarddiskVolume3\Win		MAP	-R---	-R---
00000000001A0000	0000000000011000	\Device\HarddiskVolume3\Win		MAP	-R---	-R---
00000000001C0000	0000000000003000	\Device\HarddiskVolume3\Win		MAP	-R---	-R---
00000000001D0000	0000000000002000			PRV	-RW--	-RW--
00000000001D2000	00000000000B8000	Reserved (00000000001D0000)		PRV		-RW--
00000000001E0000	0000000000010000			MAP	-RW--	-RW--
00000000001F0000	0000000000003000	\Device\HarddiskVolume3\Win		MAP	-R---	-R---
0000000000200000	00000000014D000	Reserved		PRV		-RW--
000000000034D000	0000000000009000	PEB		PRV	-RW--	-RW--
0000000000356000	00000000000AA000	Reserved (0000000000200000)		PRV		-RW--
0000000000400000	0000000000001000	hxd.exe		IMG	-R---	ERWC-
0000000000401000	0000000000537000	".text"	Executable code	IMG	ER---	ERWC-
0000000000938000	00000000000C8000	".data"	Initialized data	IMG	-RWC-	ERWC-
0000000000A00000	00000000000D000	".bss"	Uninitialized data	IMG	-RWC-	ERWC-
0000000000A0D000	0000000000006000	".idata"	Import tables	IMG	-RWC-	ERWC-
0000000000A13000	0000000000001000	".didata"		IMG	-RWC-	ERWC-
0000000000A14000	0000000000001000	".tls"	Thread-local storage	IMG	-RW--	ERWC-
0000000000A15000	0000000000001000	".rdata"	Read-only initialized data	IMG	-R---	ERWC-
0000000000A16000	00000000003D000	".pdata"	Exception information	IMG	-R---	ERWC-
0000000000A53000	0000000000055000	".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:

CPU | Log | Notes | Breakpoints | Memory Map | Call Stack | SEH | Script | Symbols | Source | References | Thr

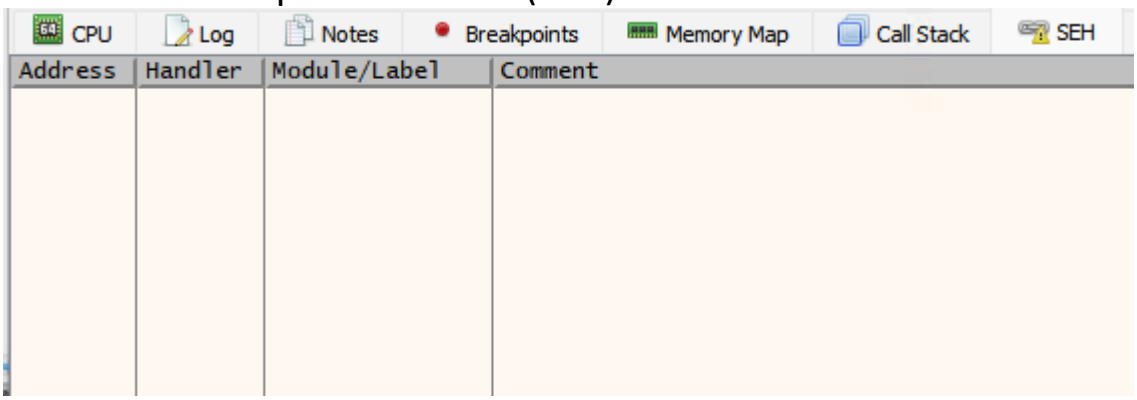
Thread	Address	To	From	Size	Comment	Party
8544						
	000000000014FF28	00007FFE5F6B54E0	00007FFE5F6C4100	30	return to kernel32.00007FFE5F6B54E0 from kernel32	System
	000000000014FF58	00007FFE6068485B			return to ntdll.00007FFE6068485B from ???	System
6980						
	0000000000EBFC38	00007FFE60696FDF	00007FFE60727A50	2F0	return to ntdll.00007FFE60696FDF from ntdll.00007	System
	0000000000EBFF28	00007FFE5F6B54E0	00007FFE5F6C4100	30	return to kernel32.00007FFE5F6B54E0 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	00007FFE5F6B54E0	00007FFE5F6C4100	30	return to kernel32.00007FFE5F6B54E0 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

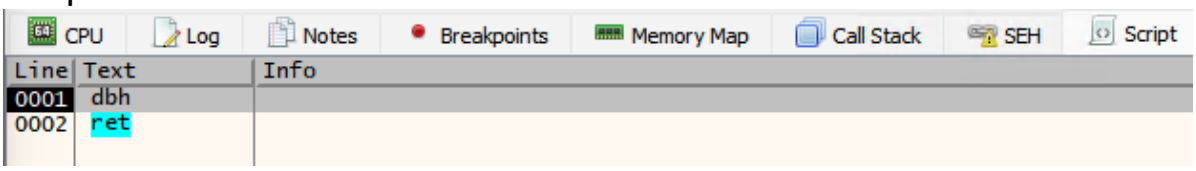
X32Dbg/X64Dbg – Debugger Guide

Structured Exception Handler (SEH) Chain Tab:

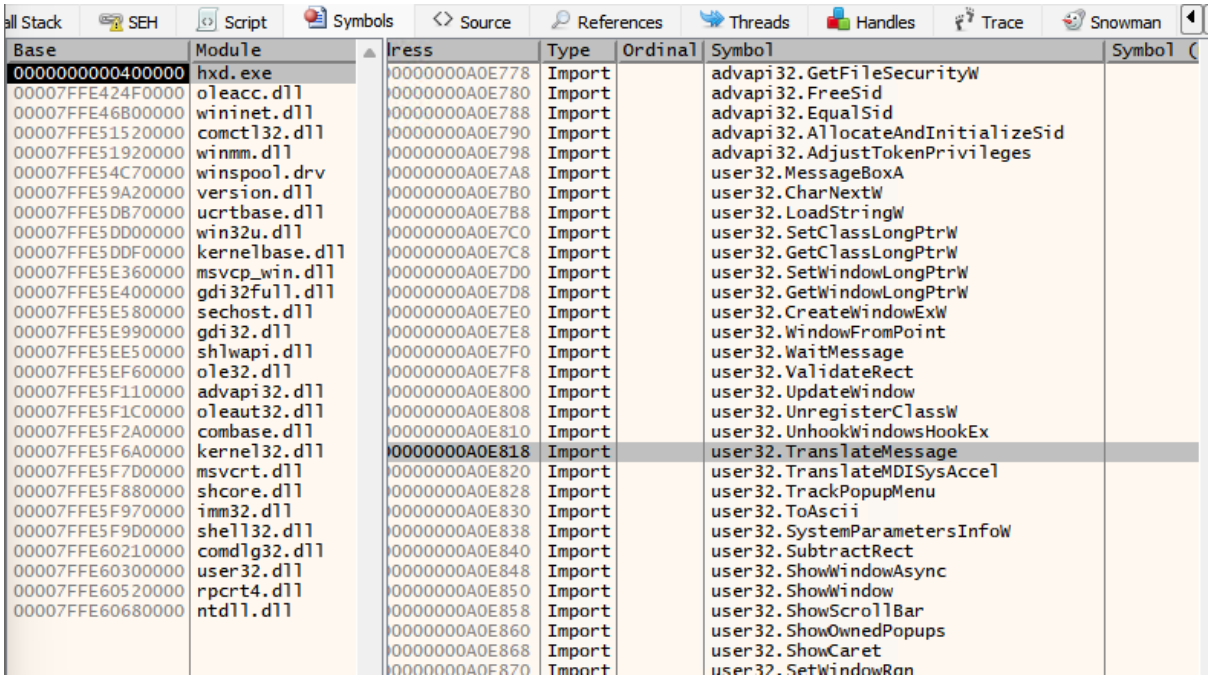


[Wikipedia – SEH mechanism](https://)

Script Tab:



Symbols Tab:



References Tab:

ll Stack | SEH | Script | Symbols | Source | References | Threads | Handles | Tr

All Modules (Calls)

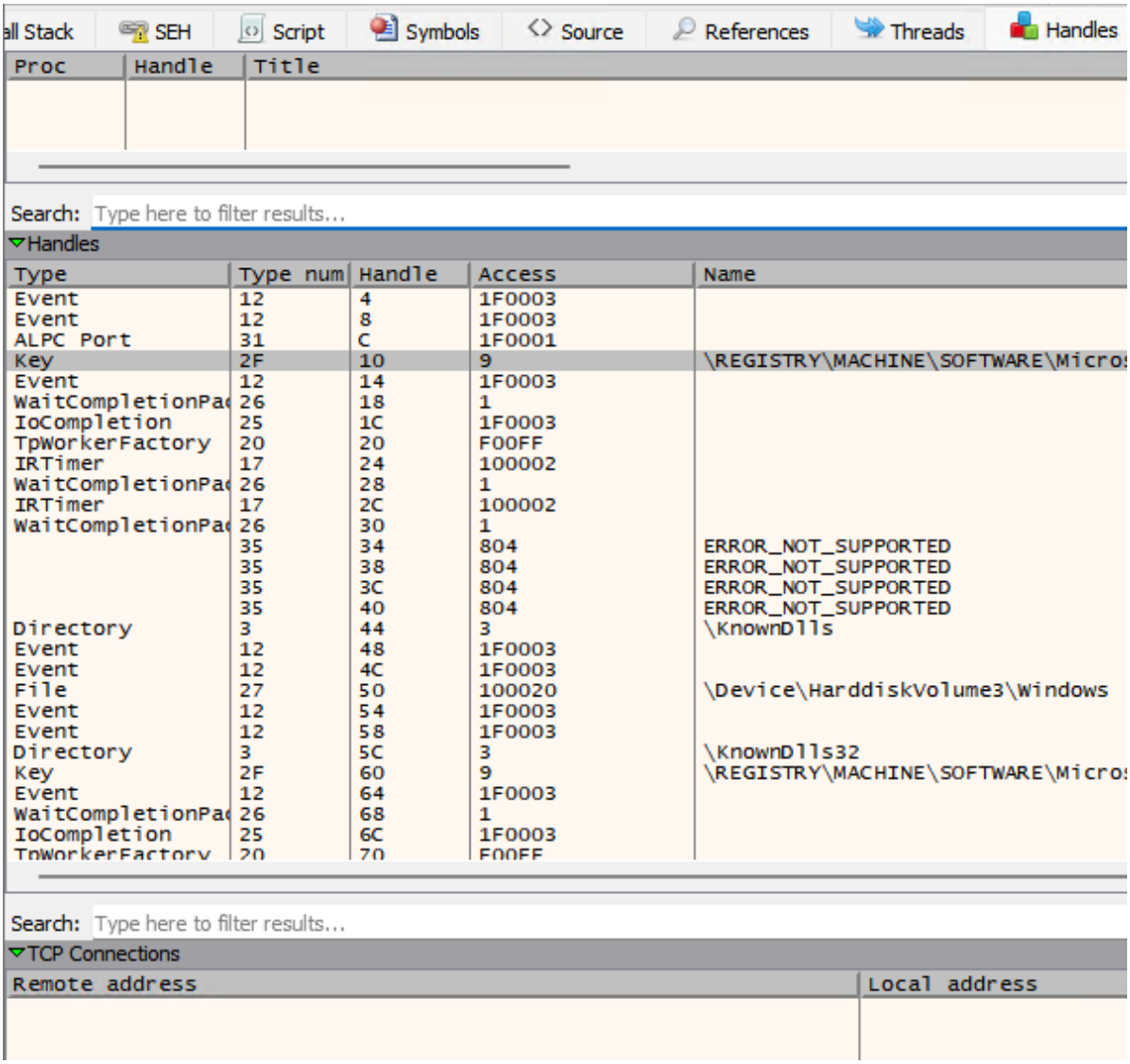
Address	Disassembly	Destination
00000000005517DC	call <JMP.&ImageList_LoadImageW>	<comctl32.ImageList_LoadImageW>
0000000000068A8C8	call <JMP.&ImageList_LoadImageW>	<comctl32.ImageList_LoadImageW>
00000000008353F1	call <JMP.&LoadBitmapW>	<user32.LoadBitmapW>
000000000069518E	call <JMP.&LoadBitmapW>	<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>

Threads Tab:

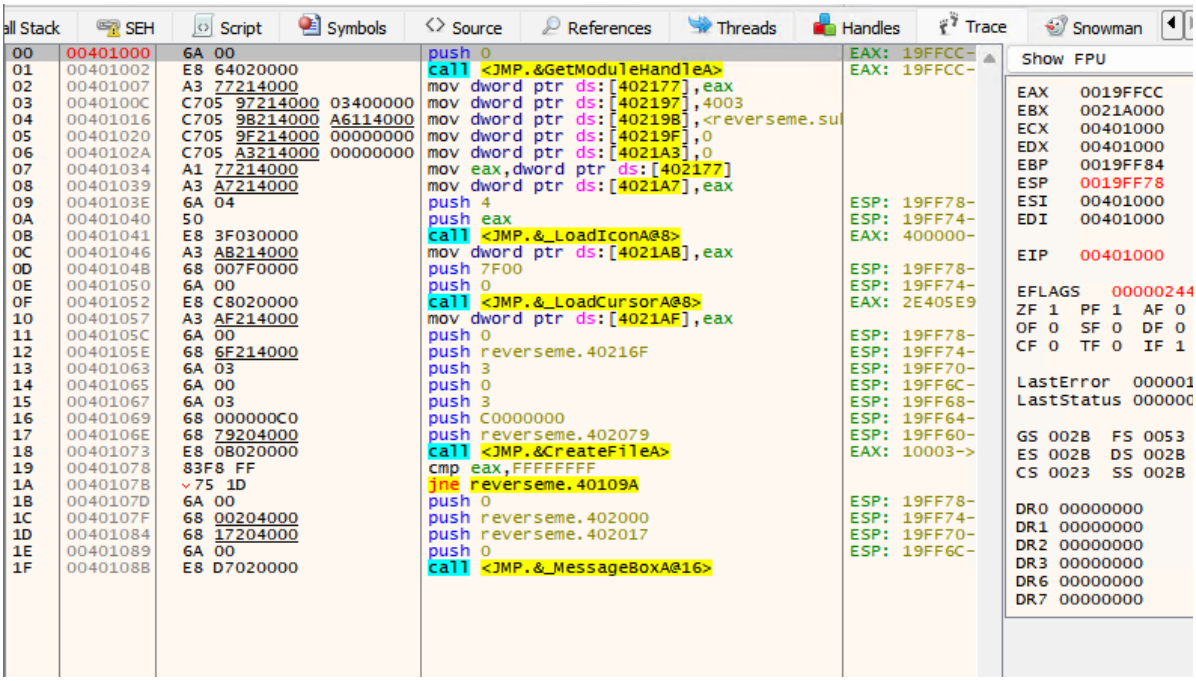
ll Stack | SEH | Script | Symbols | Source | References | Threads | Handles | Trace | Snowman

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

Handles Tab:


all Stack
SEH
Script
Symbols
Source
References
Threads
Handles
Proc
Handle
Title
Search: Type here to filter results...
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\Micros
Event 12 14 1F0003
WaitCompletionPac 26 18 1
IoCompletion 25 1C 1F0003
TpWorkerFactory 20 20 F00FF
IRTimer 17 24 100002
WaitCompletionPac 26 28 1
IRTimer 17 2C 100002
WaitCompletionPac 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\Micros
Event 12 64 1F0003
WaitCompletionPac 26 68 1
IoCompletion 25 6C 1F0003
Search: Type here to filter results...
TCP Connections
Remote address
Local address

Trace Tab:



Left – Right (Sections):
(Steps) | (VAS – Virtual Address space) | (Opcode) | (Disassembly) | (Comments / API's (xAnalyzer))
GUI Manual - Trace

Snowman Tab:

Call Stack | SEH | Script | Symbols | Source | References | Threads | Handles | Trace | Snowman

File Analyse View Help

Instructions

```
401000:  push 0x0
401002:  call dword 0x40126b
401007:  mov [0x402177], eax
40100c:  mov dword [0x402197], 0x4003
401016:  mov dword [0x40219b], 0x4011a6
401020:  mov dword [0x40219f], 0x0
40102a:  mov dword [0x4021a3], 0x0
401034:  mov eax, [0x402177]
401039:  mov [0x4021a7], eax
40103e:  push 0x4
401040:  push eax
401041:  call dword 0x401385
401046:  mov [0x4021ab], eax
40104b:  push dword 0x7f00
401050:  push 0x0
401052:  call dword 0x40131f
401057:  mov [0x4021af], eax
40105c:  push 0x0
40105e:  push dword 0x40216f
401063:  push 0x3
401065:  push 0x0
401067:  push 0x3
401069:  push dword 0xc0000000
40106e:  push dword 0x402079
401073:  call dword 0x401283
401078:  cmp eax, 0xff
40107b:  jnz 0x40109a
40107d:  push 0x0
40107f:  push dword 0x402000
401084:  push dword 0x402017
401089:  push 0x0
40108b:  call dword 0x401367
401090:  call dword 0x4012b9
401095:  jmp dword 0x40121d
40109a:  push 0x0
40109c:  push dword 0x402173
4010a1:  push 0x46
4010a3:  push dword 0x40211a
4010a8:  push eax
4010a9:  call dword 0x4012dd
```

C++

```
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, int
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_
void fun_401367(int32_t a1, int32_t a2, int32_t a3, int32_t a4, int32_t a5, int32_t a6, int32_t a7, int32_
void fun_4012b9(int32_t a1, int32_t a2, int32_t a3, int32_t a4, int32_t a5, int32_t a6, int32_t a7, int32_

/* OptionalHeader.AddressOfEntryPoint */
void OptionalHeader_AddressOfEntryPoint() {
  int32_t eax1;
  int32_t eax2;
  int32_t eax3;
  int32_t eax4;
  int32_t eax5;

  eax1 = fun_40126b(0, __return_address());
  g402177 = eax1;
  g402197 = 0x4003;
  g40219b = 0x4011a6;
  g40219f = 0;
  g4021a3 = 0;
  eax2 = g402177;
  g4021a7 = eax2;
  eax3 = fun_401385(eax2, 4, 0, __return_address());
  g4021ab = eax3;
  eax4 = fun_40131f(0, 0x7f00, eax2, 4, 0, __return_address());
  g4021af = eax4;
  eax5 = fun_401283("Keyfile.dat", 0xc0000000, 3, 0, 3, 0x40216f, 0, 0, 0x7f00, eax2, 4, 0, __return_address());
  if (eax5 != -1) {
    fun_4012dd(eax5, 0x40211a, 70, 0x402173, 0, "Keyfile.dat", 0xc0000000, 3, 0, 3, 0x40216f, 0, 0, 0x7f00, ea
  } else {
    fun_401367(0, "Evaluation period out of date. Purchase new license", "Key File ReverseMe", 0, "Keyfile.dat", 0x
    fun_4012b9(0, "Evaluation period out of date. Purchase new license", "Key File ReverseMe", 0, "Keyfile.dat", 0x
  }
}
```

Snowman - C++ Decomplier (Plugin)

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