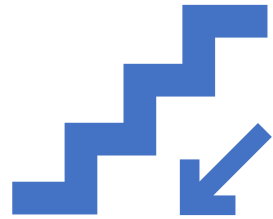




# 리버싱 이론 : 엔디언과 스택

이지훈

# 목차



## Byte Ordering

Last Week

Big Endian & Little Endian

Little Endian.exe



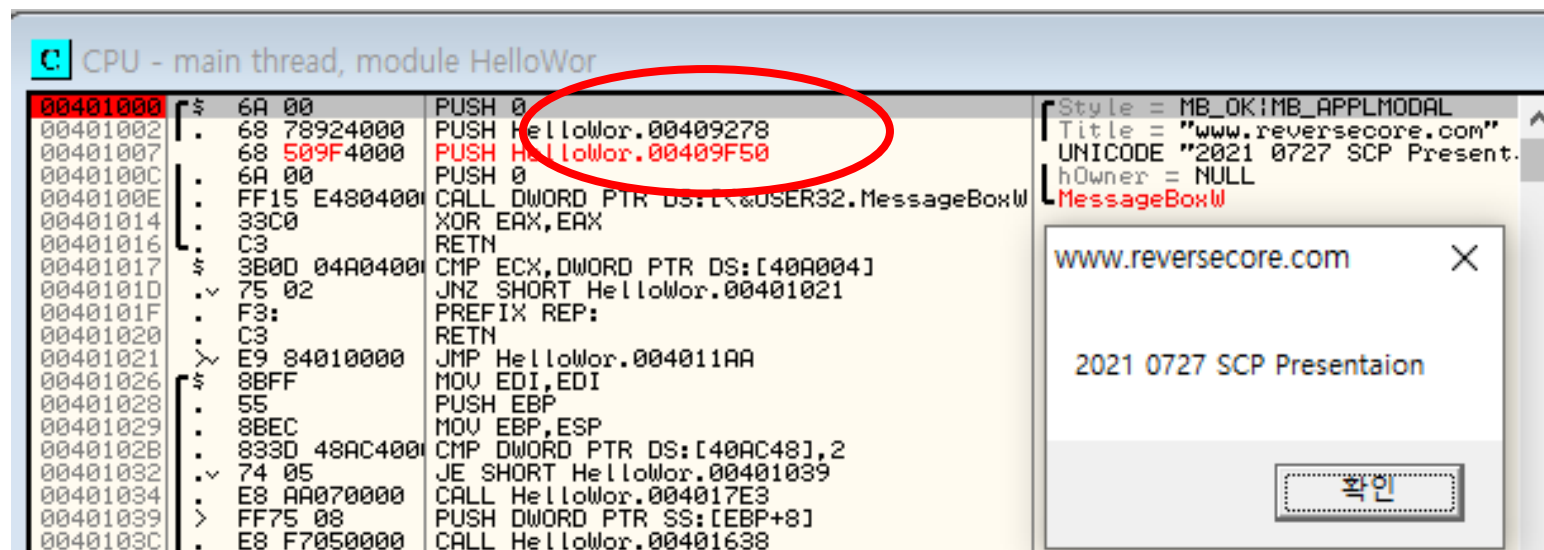
## Stack

Stack

Stack.exe

Next week

# Last Week



The screenshot shows a debugger window titled "CPU - main thread, module HelloWor". The assembly list on the left shows instructions at various addresses. A red circle highlights the instruction at address 00401007: `PUSH HelloWor.00409F50`. The instruction at 0040100E is `CALL DWORD PTR DS:[40A004], MessageBoxW`. On the right, the "Properties" pane shows the `MessageBoxW` function's parameters: `Style = MB_OK!MB_APPLMODAL`, `Title = "www.reversecore.com"`, `UNICODE "2021 0727 SCP Presentaion"`, and `hOwner = NULL`. Below this, a `MessageBoxW` dialog box is displayed with the title "www.reversecore.com" and the message "2021 0727 SCP Presentaion". The dialog has a single button labeled "확인" (OK).

Address	Disassembly
00401000	<code>PUSH 0</code>
00401002	<code>PUSH HelloWor.00409278</code>
00401007	<code>PUSH HelloWor.00409F50</code>
0040100C	<code>PUSH 0</code>
0040100E	<code>CALL DWORD PTR DS:[40A004], MessageBoxW</code>
00401014	<code>XOR EAX, EAX</code>
00401016	<code>RETN</code>
00401017	<code>CMP ECX, DWORD PTR DS:[40A004]</code>
0040101D	<code>JNZ SHORT HelloWor.00401021</code>
0040101F	<code>PREFIX REP;</code>
00401020	<code>RETN</code>
00401021	<code>JMP HelloWor.004011AA</code>
00401026	<code>MOV EDI, EDI</code>
00401028	<code>PUSH EBP</code>
00401029	<code>MOV EBP, ESP</code>
0040102B	<code>CMP DWORD PTR DS:[40AC48], 2</code>
00401032	<code>JE SHORT HelloWor.00401039</code>
00401034	<code>CALL HelloWor.004017E3</code>
00401039	<code>PUSH DWORD PTR SS:[EBP+8]</code>
0040103C	<code>CALL HelloWor.00401638</code>





# Big Endian & Little Endian

**바이트 오더링** : 데이터를 저장하는 방식, 빅 엔디언과 리틀 엔디언 방식이 있다.

**빅 엔디언** : 사람이 보는 방식과 동일하게 앞에서부터 순차적으로 저장  
: 사람이 보기에 직관적이라는 장점, RISC 계열의 CPU에서 많이 사용

**리틀 엔디언** : 저장되는 바이트 순서가 멀티 바이트의 경우 역순으로 저장  
: 산술 연산과 데이터의 타입이 확장/축소될 때 효율적, Intel x86 CPU에서 사용





# Big Endian & Little Endian

문자열의 경우 char 배열이므로  
각 바이트를 하나씩 연속해서 저장하는 방식  
-> 리틀 엔디언 방식에서도 순차적으로 저장된다.

TYPE	Name	Size	빅 엔디언	리틀 엔디언
BYTE	b	1	[12]	[12]
WORD	w	2	[12][34]	[34][12]
DWORD	dw	4	[12][34][56][78]	[78][56][34][12]
char []	str	6	[61][62][63][64][65][00]	[61][62][63][64][65][00]



# Little Endian.exe

[EBP-D] = BYTE  
[EBP-C] = WORD  
[EBP-8] = DWORD  
[EBP-4] = char[]

CPU - main thread, module LittleEn			
00401000	5E	PUSH EBP	
00401001	8BEC	MOV EBP,ESP	
00401003	83EC 10	SUB ESP,10	
00401006	A0 40AC4000	MOV AL,BYTE PTR DS:[40AC40]	
0040100B	8845 F3	MOV BYTE PTR SS:[EBP-D],AL	
0040100E	66:8B0D 44AC	MOV CX,WORD PTR DS:[40AC44]	
00401015	66:894D F4	MOV WORD PTR SS:[EBP-C],CX	
00401019	8B15 48AC4000	MOV EDX,DWORD PTR DS:[40AC48]	
0040101F	8955 F8	MOV DWORD PTR SS:[EBP-8],EDX	
00401022	C745 FC 4CAC	MOV DWORD PTR SS:[EBP-4],LittleEn.0040A	ASCII "abode"
00401029	33C0	XOR EAX,EAX	
0040102B	8BE5	MOV ESP,EBP	
0040102D	5D	POP EBP	
0040102E	C3	RETN	



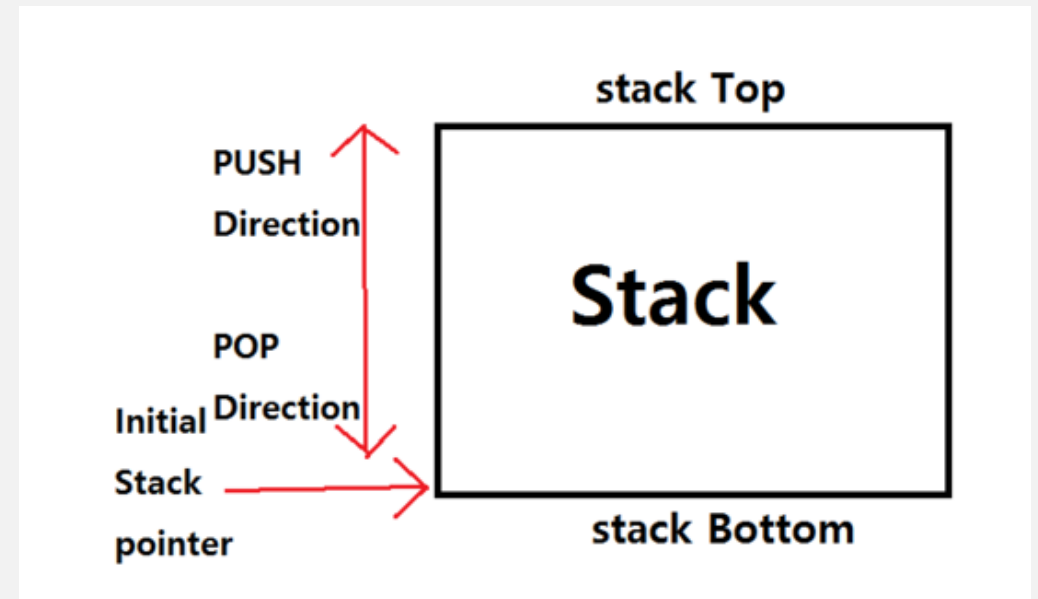


# Little Endian.exe

0040AC40	12	00	00	00	34	12	00	00	78	56	34	12	61	62	63	64	#...4#...xU4#abcd
0040AC50	65	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	e.....
0040AC60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0040AC70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....



# Stack



**스택 메모리 : 함수 내의 로컬 변수 임시 저장, 함수 파라미터 전달, 복귀 주소 저장 등의 역할**

**: 스택의 FILO(First In Last Out) 구조가 위 역할 수행에 유용하기 때문**

**: 스택은 거꾸로 자란다.**





# Stack.exe

CPU - main thread, module Stack

Address	Hex dump	Assembly
00401000	68 00010000	PUSH 100
00401005	58	POP EAX
00401006	90	NOP
00401007	90	NOP
00401008	90	NOP
00401009	90	NOP
0040100A	90	NOP
0040100B	90	NOP
0040100C	90	NOP

Local call from <ModuleEntryPoint>+3C

Stack.<ModuleEntryPoint>

Address	Hex dump	Assembly
0040C000	01 00 00 00 59 B7 68 DF B1 19 BF	
0040C010	A0 DA 40 00 00 00 00 00 A0 DA 40	
0040C020	00 00 00 00 00 00 00 00 00 10 00	
0040C030	00 00 00 00 00 00 00 00 00 00 00	
0040C040	01 00 00 00 00 00 00 00 00 00 00	
0040C050	00 00 00 00 00 00 00 00 00 00 00	

Registers (FPU)

Register	Value	Comment
EAX	0019FFCC	
ECX	00401000	Stack.<ModuleEntryPoint>
EDX	00401000	Stack.<ModuleEntryPoint>
EBX	002BB000	
ESP	0019FF74	
EBP	0019FF80	
ESI	00401000	Stack.<ModuleEntryPoint>
EDI	00401000	Stack.<ModuleEntryPoint>
EIP	00401000	Stack.<ModuleEntryPoint>
ES	002B 32B1	0(FFFFFFFF)
EAX	0019FF74	7689FA29 RETURN to KERNEL32.7689FA29
ECX	0019FF78	002BB000
EDX	0019FF7C	7689FA10 KERNEL32.BaseThreadInitThunk
EBX	0019FF80	0019FFDC
ESP	0019FF84	77107A7E RETURN to ntdll.77107A7E
EBP	0019FF88	002BB000
ESI	0019FF8C	51B5C34A



# Stack.exe

CPU - main thread, module Stack

Address	Hex	dump	Instruction
00401000	68 00 00 00		PUSH 100
00401005	58		POP EAX
00401006	90		NOP
00401007	90		NOP
00401008	90		NOP
00401009	90		NOP
0040100A	90		NOP
0040100B	90		NOP
0040100C	90		NOP

Stack [0019FF70]=00000100  
EAX=0019FFCC  
Stack.<ModuleEntryPoint>+5

Address	Hex	dump
0040C000	01 00 00 00 59 B7 68 DF B1 19 BF	
0040C010	A0 DA 40 00 00 00 00 00 A0 DA 40	
0040C020	00 00 00 00 00 00 00 00 00 10 00	
0040C030	00 00 00 00 00 00 00 00 00 00 00	
0040C040	01 00 00 00 00 00 00 00 00 00 00	
0040C050	00 00 00 00 00 00 00 00 00 00 00	
0040C060	02 00 00 00 00 00 00 00 00 00 00	

Registers (FPU)

Register	Value	Comment
EAX	0019FFCC	
ECX	00401000	Stack.<ModuleEntryPoint>
EDX	00401000	Stack.<ModuleEntryPoint>
EBX	002BB000	
ESP	0019FF70	
EBP	0019FF80	
ESI	00401000	Stack.<ModuleEntryPoint>
EDI	00401000	Stack.<ModuleEntryPoint>
EIP	00401005	Stack.00401005
C 0	ES 002B 32bit 0 (FFFFFFFF)	

0019FF70 00000100  
0019FF74 7689FA29 RETURN to KERNEL32.7689FA29  
0019FF78 002BB000  
0019FF7C 7689FA10 KERNEL32.BaseThreadInitThunk  
0019FF80 0019FFDC  
0019FF84 77107A7E RETURN to ntdll.77107A7E  
0019FF88 002BB000  
0019FF8C 51B5C34A



# Stack.exe

**CPU - main thread, module Stack**

Address	Hex	dump	Instruction
00401000	68	00010000	PUSH 100
00401005	58		POP EAX
00401006	90		NOP
00401007	90		NOP
00401008	90		NOP
00401009	90		NOP
0040100A	90		NOP
0040100B	90		NOP
0040100C	90		NOP

Stack.<ModuleEntryPoint>+6

Address	Hex	dump
0040C000	01 00 00 00	59 B7 68 DF B1 19 BF
0040C010	A0 DA 40 00	00 00 00 00 A0 DA 40
0040C020	00 00 00 00	00 00 00 00 00 10 00
0040C030	00 00 00 00	00 00 00 00 00 00 00
0040C040	01 00 00 00	00 00 00 00 00 00 00
0040C050	00 00 00 00	00 00 00 00 00 00 00

**Registers (FPU)**

Register	Value	Comment
EAX	00000100	
ECX	00401000	Stack.<ModuleEntryPoint>
EDX	00401000	Stack.<ModuleEntryPoint>
EBX	002BB000	
ESP	0019FF74	
EBP	0019FF80	
ESI	00401000	Stack.<ModuleEntryPoint>
EDI	00401000	Stack.<ModuleEntryPoint>
EIP	00401006	Stack.00401006
C 0	ES 002B 32b (0)	0 (FFFFFFFF)
0019FF74	7689FA29	RETURN to KERNEL32.7689FA29
0019FF78	002BB000	
0019FF7C	7689FA29	KERNEL32.BaseThreadInitThunk
0019FF80	0019FFDC	
0019FF84	77107A7E	RETURN to ntdll.77107A7E
0019FF88	002BB000	
0019FF8C	51B5C34A	





# Next Week

EBP.. ESP... EAX .. ??

-> 레지스터! 공부해 보았으나 이해하지 못하였음..

Abex' Crackme 크랙

-> 패치와 크랙 연습





# Q & A