INTRODUCTION TO ASSEMBLY X86

INTRODUCTION

- Avinash Kumar Thapa
- Working with EmiratesNBD as Senior InfoSec SME.
- OSCP and OSCE
- Bug hunter on Hackerone, Synack

CREDITS & REFERENCES

▶ All credit to <u>opensecuritytraining.info</u>

AGENDA

The intent of this class is to expose you to the most commonly generated assembly instructions, and the most frequently dealt with architecture hardware.

WHAT ARE YOU GOING TO LEARN

```
#include <stdio.h>
int main(){
    printf("Hello World!\n");
    return 0x1234;
}
```

IS SAME AS...

```
.text:00401730 main
   .text:00401730
                                        ebp
                                push
                                        ebp, esp
  .text:00401731
                                mov
                                        offset aHelloWorld; "Hello
  .text:00401733
                                push
  world\n"
  .text:00401738
                                        ds: imp printf
                                call
  .text:0040173E
                                add
                                        esp, 4
  .text:00401741
                                        eax, 1234h
                                mov
  .text:00401746
                                        ebp
                                pop
  .text:0040174
                                retn
```

Windows Visual C++, /GS (buffer overflow protection) option turned off Disassembled with IDA Pro 4.9 Free Version

IS SAME AS...

Ubuntu Disassembled with "objdump -d"

```
08048374 <main>:
 8048374:
                 8d 4c 24 04
                                                   0x4(%esp),%ecx
                                           lea
                                                   $0xfffffff0,%esp
 8048378:
                83 e4 f0
                                           and
                                                    -0x4(%ecx)
 804837b:
                ff 71 fc
                                            pushl
804837e:
                                                    %ebp
                55
                                            push
 804837f:
                89 e5
                                                    %esp,%ebp
                                            mov
 8048381:
                 51
                                                    %ecx
                                            push
                83 ec 04
 8048382:
                                            sub
                                                    $0x4,%esp
                                                    $0x8048460,(%esp)
 8048385:
                c7 04 24 60 84 04 08
                                            movl
                                                    80482d4 <puts@plt>
804838c:
                e8 43 ff ff ff
                                            call
 8048391:
                b8 2a 00 00 00
                                                    $0x1234, %eax
                                            mov
 8048396:
                                                    $0x4, %esp
                83 c4 04
                                            add
 8048399:
                59
                                                    %ecx
                                            pop
804839a:
                5d
                                                    %ebp
                                            pop
 804839b:
                8d 61 fc
                                                   -0x4(%ecx),%esp
                                           lea
 804839e:
                c3
                                            ret
 804839f:
                 90
                                            nop
```

IS SAME AS ..!

Mac OS Disassembled from command line with "otool -tV"

```
main:
00001fca
         pushl
                %ebp
00001fcb
         movl
               %esp,%ebp
00001fcd
         pushl
                %ebx
               $0x14,%esp
00001fce
         subl
         calll
00001fd1
                0x00001fd6
00001fd6
         popl
               %ebx
               0x000001a(%ebx),%eax
00001fd7
         leal
00001fdd
         movl
               %eax,(%esp)
00001fe0
         calll
                            ; symbol stub for: puts
                0 \times 00003005
00001fe5
               $0x00001234,%eax
         movl
               $0x14,%esp
00001fea
         addl
         popl
00001fed
               %ebx
00001fee leave
00001fef
         ret
```

- •By one measure, only 14 assembly instructions account for 90% of code!
- •I think that knowing about 20-30 (not counting variations) is good enough that you will have the check the manual very infrequently
- You've already seen 11 instructions, just in the hello world variations!

ARCHITECTURE - REGISTERS

- Registers are small memory storage areas built into the processor (still volatile memory)
- •8 "general purpose" registers + the instruction pointer which points at the next instruction to execute
- •On x86-32, registers are 32 bits long
- •On x86-64, they're 64 bits

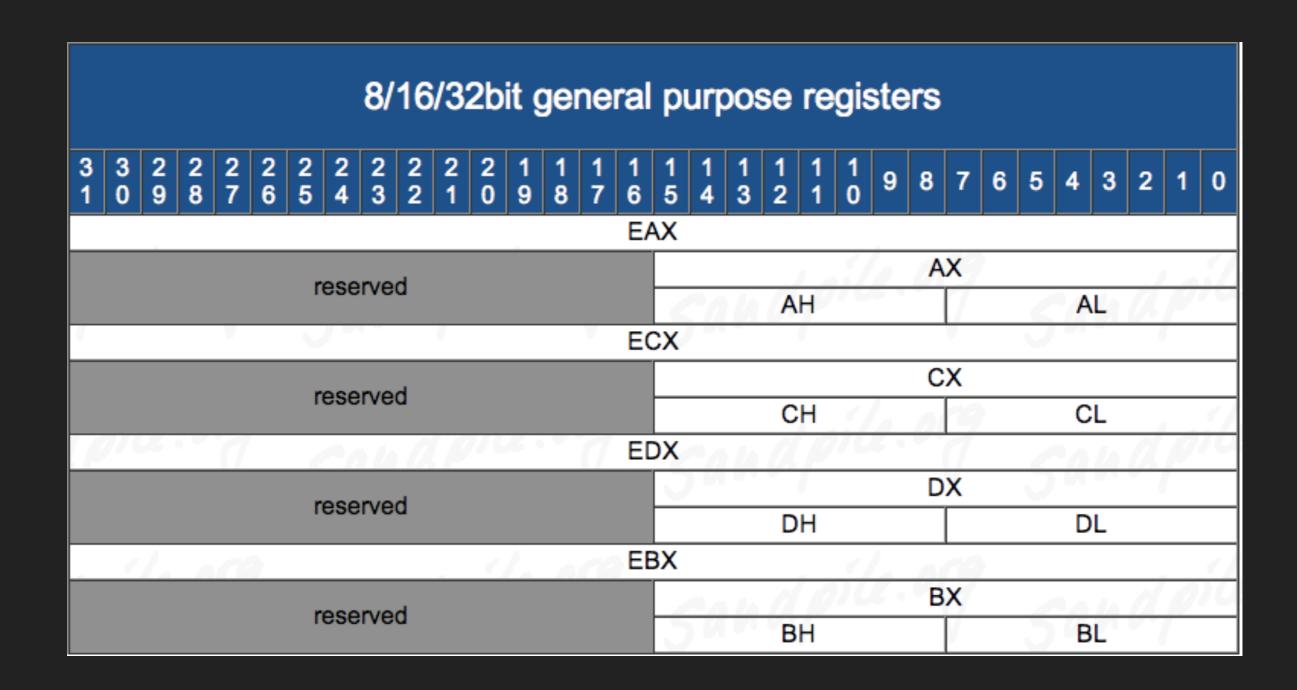
ARCHITECTURE - REGISTERS CONVENTIONS -1

- •These are Intel's suggestions to compiler developers (and assembly handcoders). Registers don't have to be used these ways, but if you see them being used like this, you'll know why. But I simplified some descriptions. I also colour coded as **GREEN** for the ones which we will actually see in this class (as opposed to future ones), and **RED** for not.
- •EAX Stores function return values
- •EBX Base pointer to the data section
- •ECX Counter for string and loop operations
- •EDX I/O pointer

ARCHITECTURE- REGISTERS - CONVENTION 2

- •ESI Source pointer for string operations
- •EDI Destination pointer for string operations
- •ESP Stack pointer
- •EBP Stack frame base pointer
- •EIP Pointer to next instruction to execute ("instruction pointer")

ARCHITECTURE - REGISTERS - 8/16/32 BIT ADDRESSING



ARCHITECTURE - REGISTERS - 8/16/32 BIT ADDRESSING-2

								1	6/3	32	bit	g	en	era	al	pu	rpo	08	e r	eć	jis	ter	s								
3	3 0		2 8	2 7	2 6	2 5	2 4	2	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1	1 0	9	8	7	6	5	4	3	2	1	0
7		1/2										6		10	ES	SP						6	7/1	16							11
						r	ese	rve	d								4/	//a	A	70	776	6 .	S	Р			4/1	h	A	10	10
7				V		- 0) 6			1				V	EE	3P	Į,			1				V		U	U				
						r	ese	rve	d														В	Р							
		1/.		18	7						10	7.	- 4	110	E	SI					1	/_	40	77							11
						r	ese	rve	d								0,1	100	1	11	776	6 .	S	SI			0.4	//a	d	0	16
7				V		T) U	J.A.	U.	1				V	Е	DI	χĽ,	A	0	/				٧		0	4	A			
						r	ese	rve	d)I							

ARCHITECTURE- EFLAGS

- •EFLAGS register holds many single bit flags. Will only ask you to remember the following for now.
- **–Zero Flag (ZF)** Set if the result of some instruction is zero; cleared otherwise.
- -Sign Flag (SF) Set equal to the most-significant bit of the result, which is the sign bit of a signed integer. (0 indicates a positive value and 1 indicates a negative value.)

YOUR FIRST INSTRUCTION - NOP

- NOP No Operation! No registers, no values, no nothin'!
- Just there to pad/align bytes, or to delay time
- Bad guys use it to make simple exploits more reliable. But that's another class;)
- "The one-byte NOP instruction is an alias mnemonic for the XCHG (E)AX, (E)AX instruction."

PUSH INSTRUCTION – 2ND INSTRUCTION

- •For our purposes, it will always be a DWORD (4 bytes).
- -Can either be an immediate (a numeric constant), or the value in a register
- •The push instruction automatically decrements the stack pointer, esp, by 4.

Registers Before

eax	0x00000003
esp	0x0012FF8C

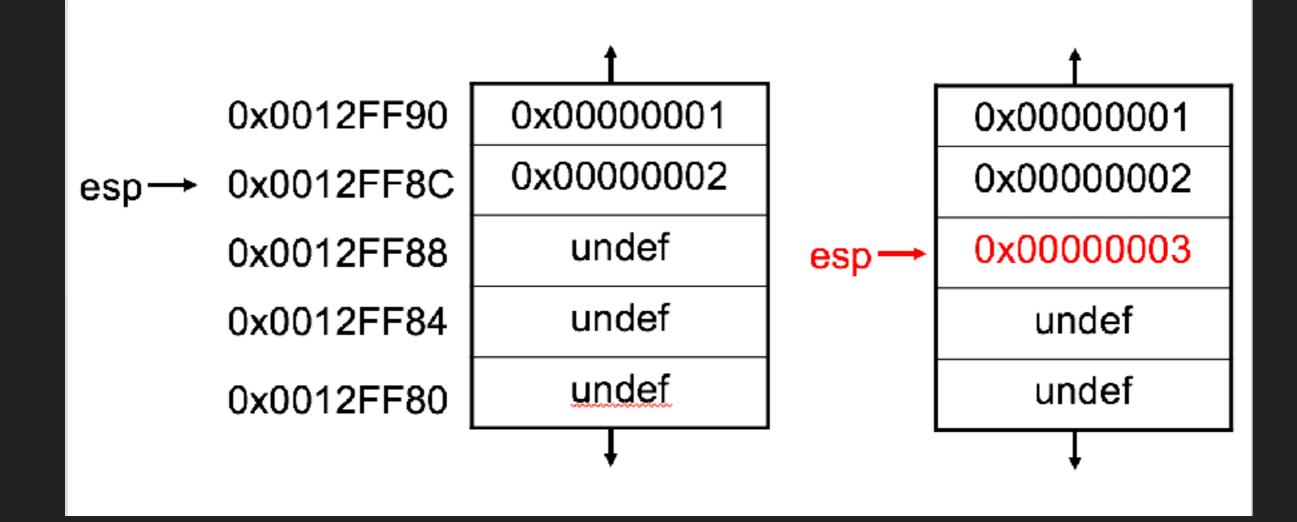
push eax

Registers After

eax	0x00000003
esp	0x0012FF88

Stack Before

Stack After



POP INSTRUCTION - 3RD

•Take a DWORD off the stack, put it in a register, and increment esp by 4

Registers Before

eax	0xFFFFFFF
esp	0x0012FF88

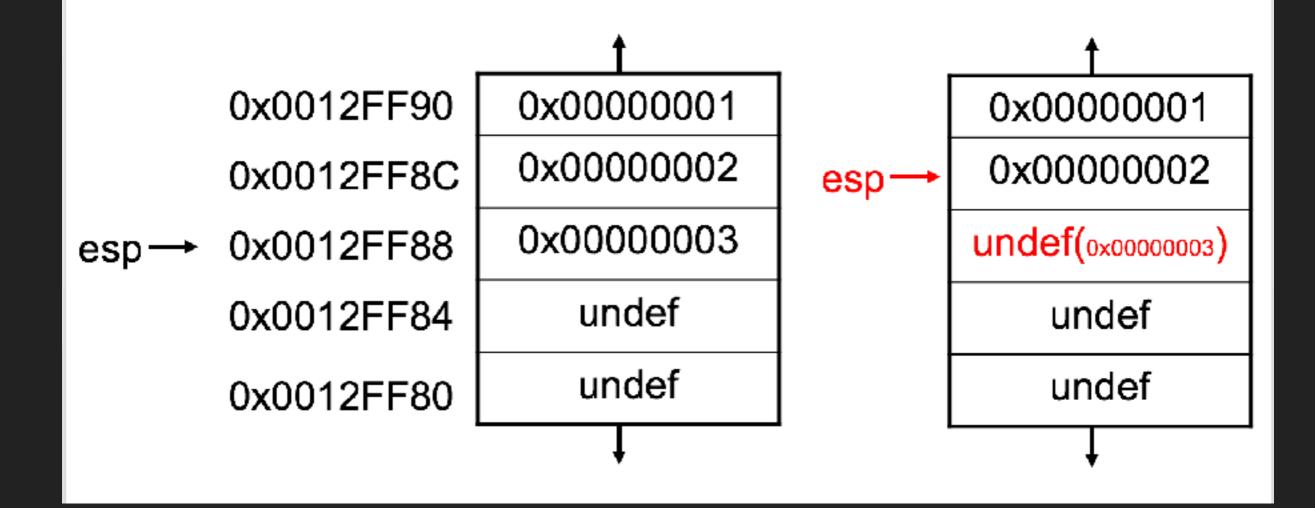
pop eax

Registers After

eax	0x00000003
esp	0x0012FF8C

Stack Before

Stack After



CALL - CALL PROCEDURE

- •CALL's job is to transfer control to a different function, in a way that control can later be resumed where it left off
- •First it pushes the address of the next instruction onto the stack
- -For use by RET for when the procedure is done
- Then it changes eip to the address given in the instruction
- Destination address can be specified in multiple ways
- –Absolute address
- -Relative address (relative to the end of the instruction)

RET – RETURN FROM PROCEDURE

- Two forms
- Pop the top of the stack into eip (remember pop increments stack pointer)
- •In this form, the instruction is just written as "ret"
- –Pop the top of the stack into eip and add a constant number of bytes to esp
- •In this form, the instruction is written as "ret 0x8", or "ret 0x20", etc

MOV - MOVE INSTRUCTION

- •Can move:
- -register to register
- -memory to register, register to memory
- -immediate to register, immediate to memory
- •Never memory to memory!
- Memory addresses are given in r/m32 form talked about later

EXAMPLE – 1

The stack frames in this example will be very simple.

Only saved frame pointer (ebp) and saved return addresses (eip)

```
sub:
//Example1 - using the stack
                                     00401000
                                               push
                                                            ebp
//to call subroutines
                                                            ebp.esp
                                     00401001
                                               mov
//New instructions:
                                                            eax,0BEEFh
                                     00401003
                                               mov
//push, pop, call, ret, mov
                                     00401008
                                                            ebp
                                               pop
                                     00401009
                                               ret
int sub(){
                                     main:
   return 0xbeef;
                                               push
                                     00401010
                                                            ebp
                                     00401011
                                                            ebp.esp
                                               mov
int main(){
                                                            sub (401000h)
                                     00401013
                                               call
                                     00401018
                                                            eax,0F00Dh
   sub();
                                               mov
                                     0040101D
                                                            ebp
                                               pop
   return 0xf00d;
                                     0040101E
                                               ret
```

EIP = 00401010, but no instruction yet executed

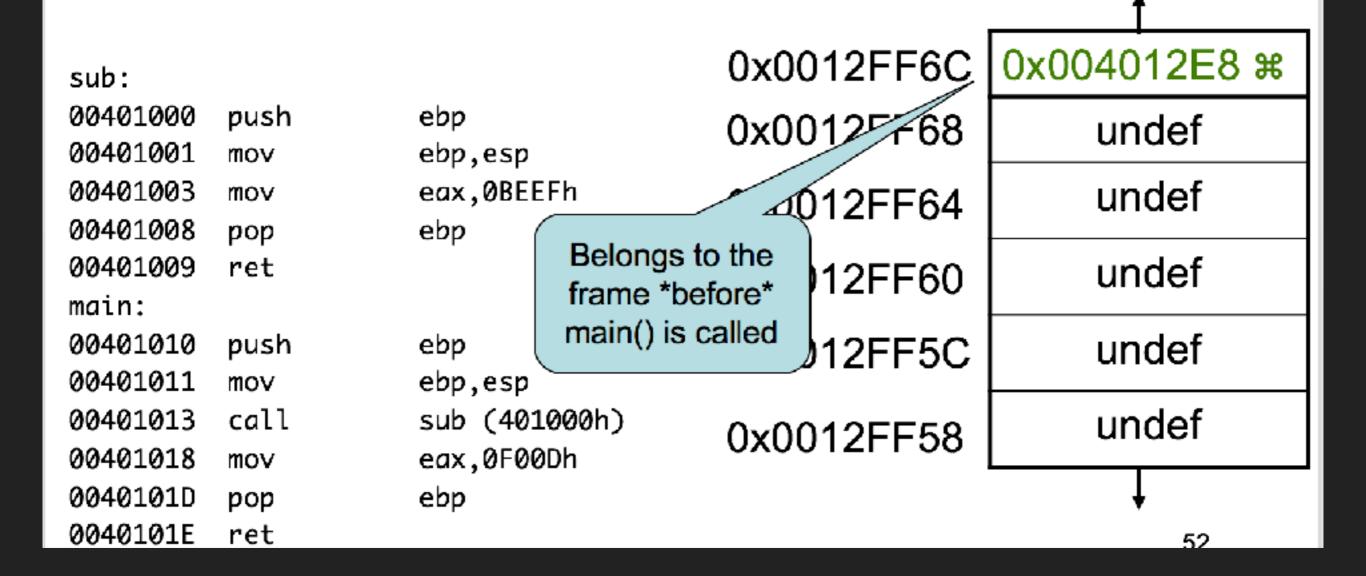
eax	0x003435C0 ∺
ebp	0x0012FFB8 ∺
esp	0x0012FF6C ∺

Key:

executed instruction,

modified value

第 start value



eax	0x003435C0 ∺
ebp	0x0012FFB8
esp	0x0012FF68 ™

c		h	•
-	ч	•	٠

00401000 push ebp ebp,esp 00401001 mov

00401003 eax,0BEEFh mov

ebp

ebp

ebp,esp

X

sub (401000h)

eax,0F00Dh

00401008 ebp pop

00401009 ret

main:

00401010 push

00401011 mov

00401013 call

00401018 mov

0040101D pop

0040101E ret

Key:

executed instruction,

modified value

start value

0x0012FF68

0x0012FF64

0x0012FF60

0x0012FF5C

0x0012FF58

0x0012FF6C | 0x004012E8 #

0x0012FFB8 m

undef

undef

undef

undef

eax	0x003435C0 ж
ebp	0x0012FF68
esp	0x0012FF64 ነው

0040101E

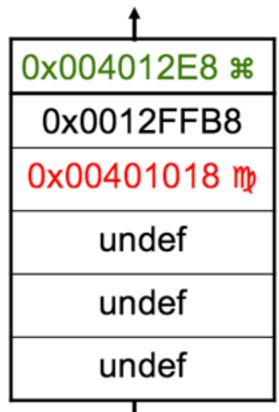
ret

sub: 00401000 00401001 00401003 00401008 00401009 main:	push mov mov pop ret	ebp ebp,esp eax,0BEEFh ebp	0x0012FF6C 0x0012FF68 0x0012FF64 0x0012FF60
00401010 00401011	push mov	ebp ebp,esp	0x0012FF5C
00401013 00401018 0040101D	call mov pop	sub (401000h) ⊠ eax,0F00Dh ebp	0x0012FF58

Key:

modified value

% start value



eax	0x003435C0 ж
ebp	0x0012FF68
esp	0x0012FF60 ነው

00401018

0040101D pop

0040101E ret

mov

sub:			0x0012FF6C
00401000	push	ebp ⊠	0x0012FF68
00401001	mov	ebp,esp	0,000121100
00401003	mov	eax,0BEEFh	0x0012FF64
00401008	pop	ebp	0,00121104
00401009	ret		0x0012FF60
main:			07.0012.100
00401010	push	ebp	0x0012FF5C
00401011	mov	ebp,esp	
00401013	call	sub (401000h)	0.0012EE58

eax,0F00Dh

ebp

Key:

X



executed instruction,

modified value

start value

0x0012FF58

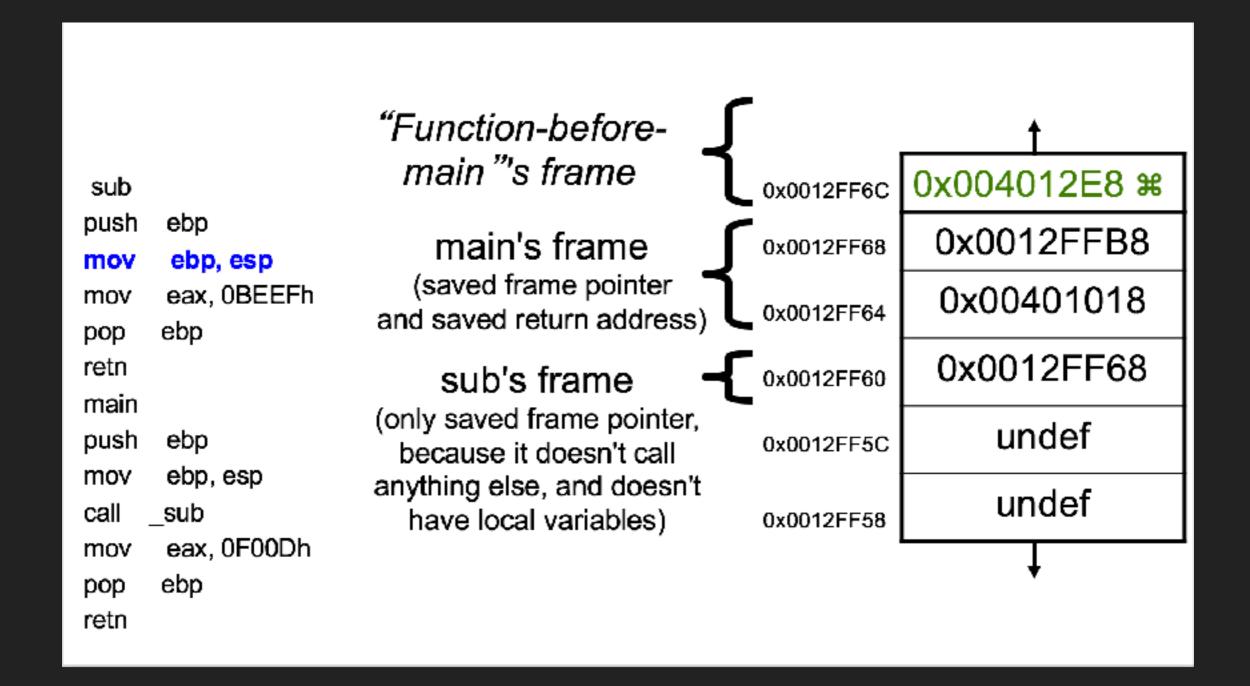
eax	0x003435C0 ж
ebp	0x0012FF60 ነው
esp	0x0012FF60

0040101E ret

sub:			0x0012FF6C
00401000 00401001	push	ebp	0x0012FF68
00401001	mov	ebp,esp ⊠ eax,0BEEFh	0x0012FF64
00401008 00401009	pop ret	ebp	
main:	100		0x0012FF60
00401010 00401011	push mov	ebp ebp,esp	0x0012FF5C
00401013	call	sub (401000h)	0x0012FF58
00401018 0040101D	mov pop	eax,0F00Dh ebp	37.30

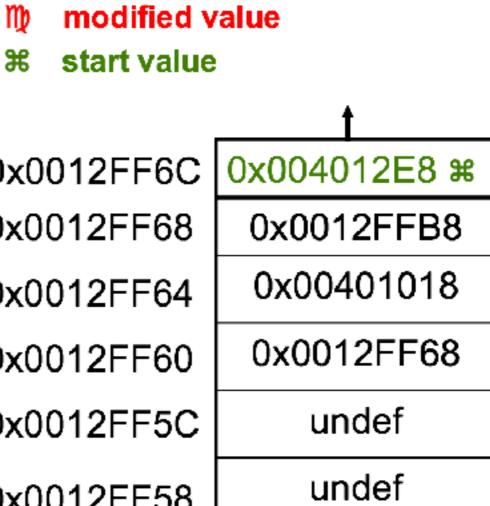
modified value	
≋ start value	
x0012FF6C	0x004012E8 #
x0012FF68	0x0012FFB8
x0012FF64	0x00401018
x0012FF60	0x0012FF68
x0012FF5C	undef
x0012FF58	undef

Key:



eax	0x0000BEEF
ebp	0x0012FF60
esp	0x0012FF60

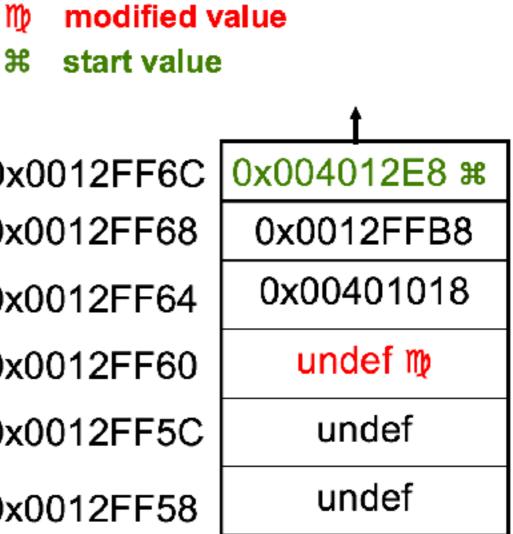
sub: 00401000 00401001 00401003	push mov mov	ebp ebp,esp eax,0BEEFh ⊠	0x0012FF6C 0x0012FF68
00401008 00401009 main: 00401010	pop ret push mov	ebp ebp ebp,esp	0x0012FF64 0x0012FF60
00401013 00401018 0040101D 0040101E	call mov pop ret	sub (401000h) eax,0F00Dh ebp	0x0012FF5C 0x0012FF58



Key:

eax	0x0000BEEF
ebp	0x0012FF68 m)
esp	0x0012FF64 m

sub:			0x0012
00401000	push	ebp	0x0012
00401001	mov	ebp,esp	
00401003	mov	eax,0BEEFh	0x0012
00401008	pop	ebp ⊠	
00401009	ret	COP ES	
main: 00401010	push	ebp	0x0012
00401011	mo∨	ebp,esp	0x0012
00401013	call	sub (401000h)	
00401018	mov	eax,0F00Dh	0004
0040101D	pop	ebp	
0040101E	ret		0x0012



executed instruction,

Key:

X

eax	0x0000BEEF
ebp	0x0012FF68
esp	0x0012FF68 m/

sub:			0x0012FF6C
00401000 00401001	push mov	ebp ebp,esp	0x0012FF68
00401003	mov	eax,0BEEFh	
00401008	рор	ebp	0x0012FF64
00401009 main:	ret 🗵		0x0012FF60
00401010	push	ebp	0,00121100
00401011 00401013	mov call	ebp,esp sub (401000h)	0x0012FF5C
00401018	mov	eax,0F00Dh	
0040101D	рор	ebp	0x0012FF58
0040101E	ret		

modified value	
ಕ start value	
k0012FF6C	0x004012E8
k0012FF68	0x0012FFB8
0012FF64	undef 🅦
(0012FF60	undef
(0012FF5C	undef
(0012FF58	undef

executed instruction,

Key:

eax	0x0000F00D тр
ebp	0x0012FF68
esp	0x0012FF68

sub:		
00401000	push	ebp
00401001	mov	ebp,esp
00401003	mov	eax,0BEEFh
00401008	рор	ebp
00401009	ret	•
main:		
00401010	push	ebp
00401011	mov	ebp,esp
00401013	call	sub (401000h)
00401018	mov	eax,0F00Dh ☒
0040101D	рор	ebp
0040101E	ret	

executed instruction, modified value start value		
0x0012FF6C	0x004012E8 #	
0x0012FF68	0x0012FFB8	
0x0012FF64	undef	
0x0012FF60	undef	
0x0012FF5C	undef	
0x0012FF58	undef	

Key:

eax	0x0000F00D
ebp	0x0012FFB8 m₂
esp	0x0012FF6C ₩

sub:		
00401000	push	ebp
00401001	mov	ebp,esp
00401003	mov	eax,0BEEFh
00401008	рор	ebp
00401009	ret	
main:		
00401010	push	ebp
00401011	mov	ebp,esp
00401013	call	sub (401000h)
00401018	mov	eax,0F00Dh
0040101D	рор	ebp 🗵
0040101E	ret	

Key:

図 executed instruction,

modified value

第 start value

	<u>_</u>
0x0012FF6C	0x004012E8
0x0012FF68	undef 🅦
0x0012FF64	undef
0x0012FF60	undef
0x0012FF5C	undef
0x0012FF58	undef

eax	0x0000F00D
ebp	0x0012FFB8
esp	0x0012FF70 m₂

Key:

modified value

% start value

sub:				0x0012FF6C	
00401 00401		push mov	ebp ebp,esp	0x0012FF68	
00401 00401		mov pop	eax,0BEEFh ebp	0x0012FF64	
00401 main:		ret		0x0012FF60	
00401 00401	011	push mov	ebp ebp,esp	0x0012FF5C	
00401 00401	.018	call mov	sub (401000h) eax,0F00Dh		
00401 00401		pop ret ⊠	ebp	0x0012FF58	_

undef m
undef

THAT'S ALL FOR TODAY

QUESTIONS ???