

# REbejs

---

1. workshop (draft0)



# Pojetí workshopu

- 1× 14 dní
- Rychle a prakticky
- Teorie až později
- Podrobný slidy s klikacíma URL ke stažení na wiki
- Trochu ARM
- Crackme: jednoúčelový program pro reverzování, bez toho postrádá smysl
  
- Poděkování RubberDuckovi

# Podklady

- [wiki.base48.cz/REbejs](http://wiki.base48.cz/REbejs)
- [Nebojte se reverzního inženýrství I.](#)
- [Nebojte se reverzního inženýrství II.](#)
- [Nebojte se reverzního inženýrství III.](#)
- [x86asm.net/links](http://x86asm.net/links)
- [ref.x86asm.net/coder32.html](http://ref.x86asm.net/coder32.html)
- [ref.x86asm.net/coder64.html](http://ref.x86asm.net/coder64.html)

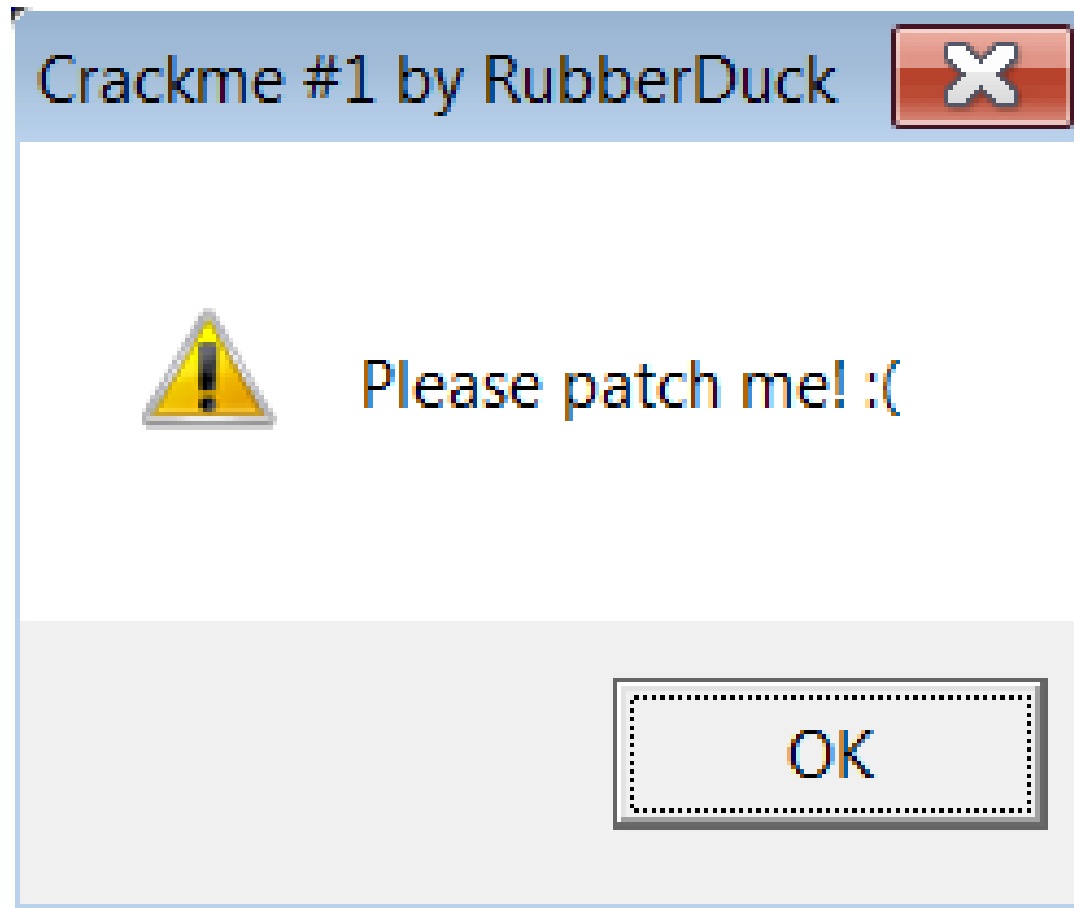
# 1. workshop - témata

- RubberDuckův seriál – první tři díly
- + opkódy instrukcí
- + x64
- + x64 verze crackme #1: [crackme1\\_x64.zip](#)

# 1. workshop - nástroje

- 32bitový debugger OllyDbg
  - [OllyDbg 2.01](#)
- 64bitový debugger x64dbg
  - [x64dbg snapshot\\_2016-01-21\\_02-44](#)

# Crackme #1 by RubberDuck



# OllyDbg – crackme1.exe

The screenshot shows the OllyDbg interface with the following components:

- Assembly View:**

```

00401000 $ B8 00000000 mov eax, 0
00401005 . 83F8 01      cmp eax, 1
00401008 . 74 0C       je short 00401016
0040100A . B8 14304000 mov eax, offset 00403014
0040100F . B9 30000000 mov ecx, 30
00401014 . EB 0A       jmp short 00401020
00401016 > B8 00304000 mov eax, offset 00403000
0040101B . B9 00000000 mov ecx, 0
00401020 > 51          push ecx
00401021 . 68 28304000 push offset 00403028
00401026 . 50          push eax
00401027 . 6A 00       push 0
00401029 . E8 0E000000 call <jmp.&user32.MessageBoxA>
0040102E . 6A 00       push 0
00401030 . E8 01000000 call <jmp.&kernel32.ExitProcess>
00401035 . CC         int3
00401036 $- FF25 00204000 jmp [<&kernel32.ExitProcess>]
0040103C $- FF25 08204000 jmp [<&user32.MessageBoxA>]
00401042 . 0000      add [eax], al
00401044 . 00       db 00
00401045 . 00       db 00
00401046 . 00       db 00
00401047 . 00       db 00
00401048 . 00       db 00
00401049 . 00       db 00

```
- Registers (FPU):**

```

EAX 76B63378 kernel32.BaseThreadInitThunk
ECX 00000000
EDX 00401000 crackme1.<ModuleEntryPoint>
EBX 7EFDE000
ESP 0018FF8C
EBP 0018FF94
ESI 00000000
EDI 00000000
EIP 00401000 crackme1.<ModuleEntryPoint>
C 0 ES 002B 32bit 0(FFFFFFFF)
P 1 CS 0023 32bit 0(FFFFFFFF)
A 0 SS 002B 32bit 0(FFFFFFFF)
Z 1 DS 002B 32bit 0(FFFFFFFF)
S 0 FS 0053 32bit 7EFDD000(FFF)
T 0 GS 002B 32bit 0(FFFFFFFF)
D 0
O 0 LastErr 00000000 ERROR_SUCCESS
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0

```
- Hex Dump:**

Address	Hex dump	ASCII (ANSI - Ce)
00403000	43 6F 6E 67 72 61 74 20 72 65 76 65 72 73 65 72	Congrat reverser
00403010	20 3A 29 00 50 6C 65 61 73 65 20 70 61 74 63 68	:)aPlease patch
00403020	20 6D 65 21 20 3A 28 00 43 72 61 63 6B 6D 65 20	me! :(aCrackme
00403030	23 31 20 62 79 20 52 75 62 62 65 72 44 75 63 6B	#1 by RubberDuck
00403040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	aaaaaaaaaaaaaaaa
00403050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	aaaaaaaaaaaaaaaa
00403060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	aaaaaaaaaaaaaaaa
00403070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	aaaaaaaaaaaaaaaa
00403080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	aaaaaaaaaaaaaaaa
00403090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	aaaaaaaaaaaaaaaa
004030A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	aaaaaaaaaaaaaaaa
004030B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	aaaaaaaaaaaaaaaa
004030C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	aaaaaaaaaaaaaaaa
004030D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	aaaaaaaaaaaaaaaa
004030E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	aaaaaaaaaaaaaaaa
- Registers (FPU) (continued):**

```

0018FF8C 76B6338A 03Au RETURN to kernel32.76B6338A
0018FF90 7EFDE000 a0r~
0018FF94 0018FFD4 da1a
0018FF98 776097F2 .S`w RETURN to ntdll.776097F2
0018FF9C 7EFDE000 a0r~
0018FFA0 769B432B +C1v
0018FFA4 00000000 aaaa
0018FFA8 00000000 aaaa
0018FFAC 7EFDE000 a0r~
0018FFB0 00000000 aaaa
0018FFB4 00000000 aaaa
0018FFB8 00000000 aaaa
0018FFBC 0018FFA0 da1a
0018FFC0 00000000 aaaa
0018FFC4 FFFFFFFF aaaa End of SEH chain
0018FFC8 77643885 ũ8dw SE handler

```

# crackme1.exe – 1. instrukce MOV

- `00401000 B8 00000000 MOV EAX, 0`
- 1. sloupec – *virtuální adresa* `00401000`
- 2. sloupec – operační kód – *opcode* `B8 00000000`
- 3. sloupec – instrukce `MOV EAX, 0`
- OllyDbg disassembler: Intel syntax, MASM příchuť
- OllyDbg AT&T syntaxe:  
`MOVL $0, %EAX`



# 1. instrukce podrobně

- **B8 00000000 MOV EAX, 0**
- *Mnemonic MOV*
- Cílový 32bitový *operand EAX*
- Zdrojový 32bitový operand **0x0**
- Jednobaýový primární opkód **0xB8**
- [ref.x86asm.net/coder32.html#xB8](http://ref.x86asm.net/coder32.html#xB8)
- Skupina opkódů **B8+r MOV r16/32, imm16/32**
- 32bitová konstanta **0x00000000**
- Intel jí říká *immediate value*

# IA-32 general-purpose registers

- Osm 32bitových všeobecných registrů, kód 0-7 (3 bity)

EAX ECX EDX EBX ESP EBP ESI EDI  
0 1 2 3 4 5 6 7

## General-Purpose Registers

31	16	15	8	7	0	16-bit	32-bit
	AH		AL			AX	EAX
	BH		BL			BX	EBX
	CH		CL			CX	ECX
	DH		DL			DX	EDX
			BP				EBP
			SI				ESI
			DI				EDI
			SP				ESP

# General-purpose registers - příklady

- EAX = 44332211 – 32 bitů, velikost DWORD
  - AX: 2211 – 16 bitů, velikost WORD
    - AL: 11 – dolních 8 bitů, velikost BYTE
    - AH: 22 – horních 8 bitů, velikost BYTE
- EBP = 40302010
  - BP: 2010
- Horní WORD nemá název a nejde ho adresovat
- Pozn. 4 bitům (půlbajt) se říká *nibble*

# Crackme1.exe – 2. instrukce CMP

- `00401005 83F8 01 CMP EAX, 1`
- Mnemonic `CMP`
- Cílový 32bitový operand `EAX`
- Zdrojový 32bitový operand `0x1`
- Jednobaýový primární opkód `0x83`
- [ref.x86asm.net/coder32.html#x83](http://ref.x86asm.net/coder32.html#x83)
- Skupina opkódů `83`
- *ModR/M byte* `F8 = 11111000``bin`
  - zatím to víc neřešíme
- 8bitová přímá (immediate) hodnota `0x01`

# Skupina opkódů 0x83

- Osm aritmetických a logických instrukcí
- Operandy:  $r/m16/32$ ,  $imm8$
- Viz taky opkódy 80, 81

		83	0				L ADD	$r/m16/32$	$imm8$
		83	1 03+				L OR	$r/m16/32$	$imm8$
		83	2				L ADC	$r/m16/32$	$imm8$
		83	3				L SBB	$r/m16/32$	$imm8$
		83	4 03+				L AND	$r/m16/32$	$imm8$
		83	5				L SUB	$r/m16/32$	$imm8$
		83	6 03+				L XOR	$r/m16/32$	$imm8$
		83	7				CMP	$r/m16/32$	$imm8$

## 2. instrukce podrobně

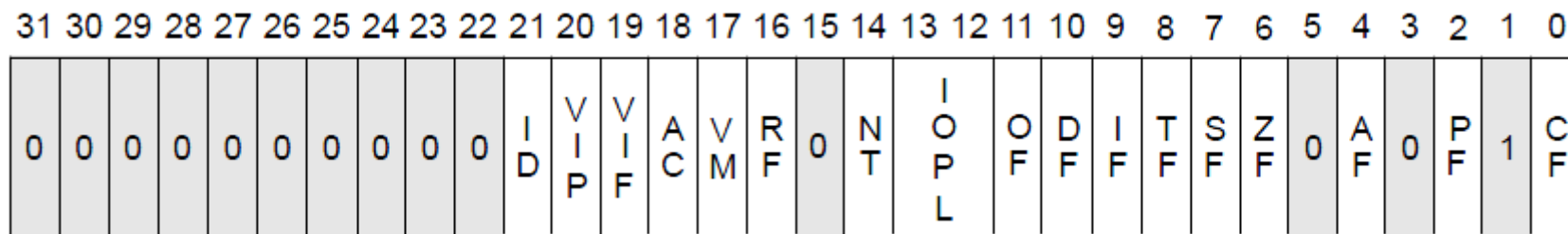
- CMP: *compare two operands*
- CMP nemění žádný operand
- Operace CMP = operace SUB
- SUB: *subtract two operands*
- SUB EAX, 1: EAX = EAX - 1
- CMP nastavuje šest stavových příznaků (status flags):  
CF, PF, AF, ZF, SF, OF
- Viz instrukční reference:

CMP	r/m16/32	imm8					o..szapc
-----	----------	------	--	--	--	--	----------

- Příznaky jsou uloženy v registru EFlags

# Registr EFlags, operace CMP

- Šest stavových příznaků v bitech 0, 2, 4, 6, 7 a 11
- Hromada jiných, ty teď neřešíme
- CMP nastaví ZF (Zero Flag), pokud mají operandy stejnou hodnotu
- Stejně jako SUB nastaví ZF, pokud je výsledek odečítání nula



# crackme1.exe – 3. instrukce JE

- 00401008 74 0C JE SHORT 00401016
- Jednobažtový primární opkód 0x74
- [ref.x86asm.net/coder32.html#x74](http://ref.x86asm.net/coder32.html#x74)
- Mnemonic JE: Jump if Equal
- Mnemonic JZ: Jump if Zero
- SHORT znamená skok v rozmezí -128 a +127 bajtů
- Skupina opkódů 70-7F: 16 podmíněných skoků *Jcc*
- 8-bit *relative offset* 0x0C
- Relativní offset se přičítá k registru EIP



# 3. instrukce JE a reg. EIP podrobně

- EIP: Instruction Pointer Register
- EIP obsahuje virtuální adresu následující instrukce, tzn. v době spouštění instrukce JE je EIP 40100A

- Operace:

```
if (ZF == 1)
```

```
    EIP = EIP + sign extended rel8
```

- Kalkulace návěští (*label*) skoku: adresa instrukce JE je 401008, je dlouhá 2 bajty:  
 $401008 + 2 + 0C = 401016$

# crackme1.exe – 4. instrukce MOV

- `0040100A B8 14304000`  
`MOV EAX, OFFSET 00403014`
- Viz 1. instrukce MOV
- **OFFSET**: MASM příchuť, důsledek analýzy – OllyDbg rozeznal, že jde o adresu, ne o nahodilou hodnotu
- Na adrese 403014 je řetězec "Please patch me! :(")

# crackme1.exe – 5. instrukce MOV

- `0040100F B9 30000000 MOV ECX, 30`
- Viz 1. instrukce MOV
- Připomenutí: opkód `B9 = B8+r`, kód ECX je 1

# crackme1.exe – 6. instrukce JMP

- `00401014 EB 0A JMP SHORT 00401020`
- [ref.x86asm.net/coder32.html#xEB](http://ref.x86asm.net/coder32.html#xEB): `JMP rel8`
- Mnemonic JMP: Unconditional Jump
- `SHORT` znamená skok v rozmezí -128 a +127 bajtů
- 8-bit relative offset `0x0A`
- Relativní offset se přičítá k registru `EIP`
- Kalkulace návěští skoku:  $401014 + 2 + 0A = 401020$

# crackme1.exe – 9. instrukce PUSH

- 00401020 51 PUSH ECX
- [ref.x86asm.net/coder32.html#x50](http://ref.x86asm.net/coder32.html#x50): 50+r PUSH r16/32
- Skupina opkódů 50-57
- Uloží hodnotu operandu na zásobník ve dvou krocích:  
 $ESP = ESP - 4$   
 $*ESP = ECX$
- Zásobník je zatím prostě nějaká paměť, víc to neřešíme
- OllyDbg: zásobník je vidět vpravo dole

# Parametry WinAPI MessageBoxA()

- Prototyp funkce z [MSDN](#):

```
int WINAPI MessageBox(  
    _In_opt_ HWND      hWnd,  
    _In_opt_ LPCTSTR   lpText,  
    _In_opt_ LPCTSTR   lpCaption,  
    _In_     UINT       uType  
);
```

# Parametry WinAPI MessageBoxA()

- Odpovídající instrukce PUSH

```
51          PUSH ECX          ; uType
68 28304000 PUSH OFFSET 403028   ; lpCaption
50          PUSH EAX         ; lpText
6A 00      PUSH 0           ; hWnd
```

# Volací konvence stdcall

- Konvence je vidět v prototypu:
- `int WINAPI MessageBox () ;`
- WinDef.h (Windows SDKs):
- `#define WINAPI __stdcall`
- [MSDN stdcall calling convention](#)
  - Parametry jsou uloženy zprava doleva
  - Zásobník čistí volaná funkce
  - Návrátová hodnota je v registru EAX
  - Registry EAX, ECX a EDX může volaná funkce změnit, ostatní musí zachovat
- stdcall používá většina WinAPI funkcí



# Další formy instrukce PUSH

- *PUSH immediate value*: buď 32bitová hodnota, nebo znaménkově rozšířený `imm8` na 32 bitů

```
68 28304000 PUSH OFFSET 403028
```

```
6A 00          PUSH 0
```

# crackme1.exe – 13. instrukce CALL

- 00401029 E8 0E000000  
CALL <jmp.&user32.MessageBoxA>
- Trochu jinak:
- CALL 0040103C
- Jednobytový primární opkód E8 CALL rel16/32
- Operace:  
PUSH EIP  
JMP 0040103C
- Tento CALL volá MessageBoxA () nepřímo přes JMP umístěný na konci kódu, je to záležitost importování funkcí a assembleru MASM, toto teď neřešíme

# Volání funkce: pár CALL - RET

```
CALL func
```

```
...
```

```
func:
```

```
...
```

```
RET ; taky „RETN“
```

- Operace CALL:
  - $ESP = ESP - 4$
  - $*ESP = EIP$
  - JMP func
- Operace RET:
  - $EIP = *ESP$
  - $ESP = ESP + 4$

# OllyDbg: základní akce

- Step into (klávesa F7): dojde k přechodu na návěští CALL
- Step over (klávesa F8): krokování pokračuje až po návratu z CALL
- Run (klávesa F9): spustí debugovaný program, tzn. nekrokuje ho
- Restart (Ctrl+F2): zabije aktuální sešn a načte debugovaný program znovu; dobrá věc, pokud krokováním dojdou donikam a potřebuju začít znovu

# Disassembling MessageBoxA()

```
8BFF          mov edi, edi
55           push ebp
8BEC          mov ebp, esp
6A 00         push 0
FF75 14       push dword ptr [ebp+14]
FF75 10       push dword ptr [ebp+10]
FF75 0C       push dword ptr [ebp+0C]
FF75 08       push dword ptr [ebp+8]
E8 A0FFFFFF   call MessageBoxExA
5D           pop  ebp
C2 1000      retn 10
```

# MessageBoxA(): RETN 10

- `C2 1000 RETN 10`
- `0x10 = 16`
- `16/4 = 4` DWORD parametry
- Pseudooperace:

`ESP = ESP + 16`

`EIP = *ESP`

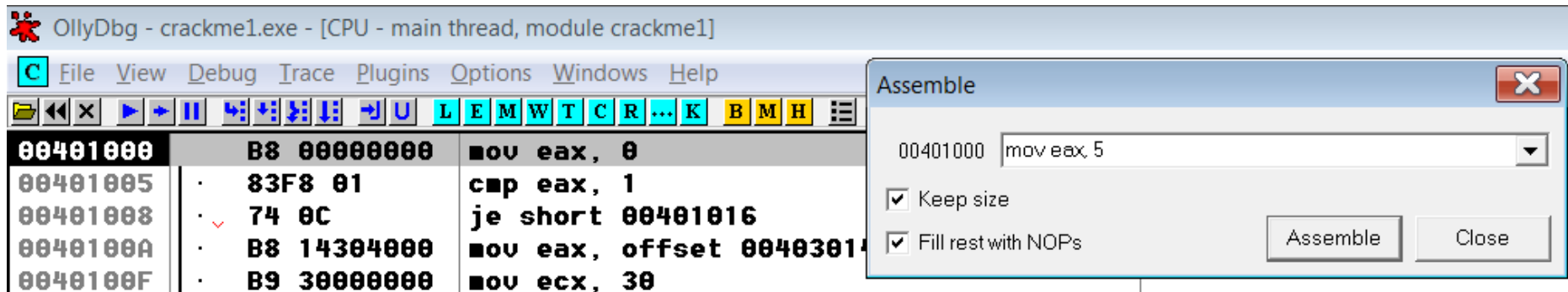
`ESP = ESP + 4`

# crackme1.exe – 15. instrukce CALL

- 00401030 E8 01000000  
CALL <jmp.&kernel32.ExitProcess>
- Trochu jinak:
- CALL 00401036
- Viz předchozí instrukce CALL
- Jde o ukončení procesu: funkce `ExitProcess()` v knihovně `kernel32.dll`
- I když nikdy nedojde k návratu, stejně se používá CALL a ne JMP

# OllyDbg assembler

- Assembluj MOV EAX, 5 na adrese 401000 (= EIP)



OllyDbg - crackme1.exe - [CPU - main thread, module crackme1]

File View Debug Trace Plugins Options Windows Help

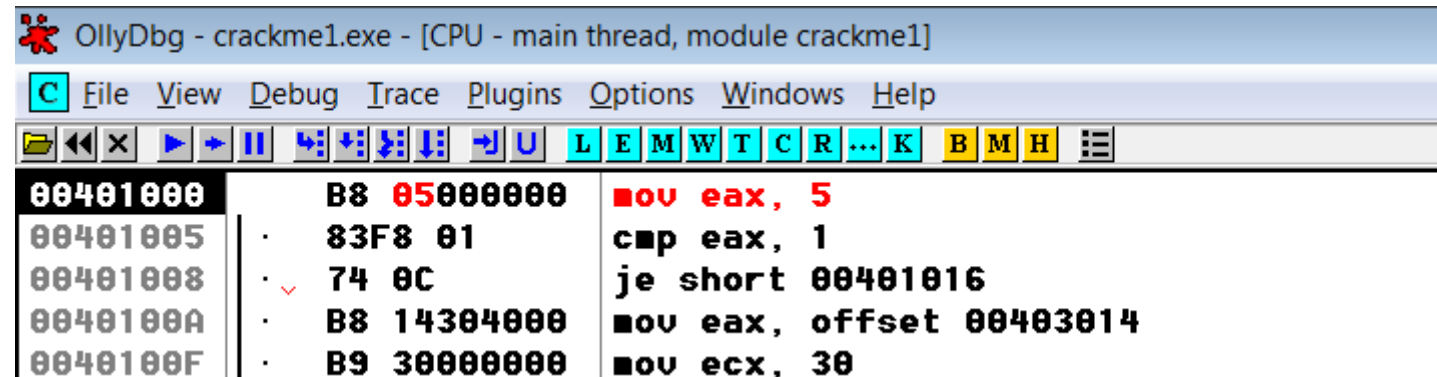
00401000 B8 00000000 mov eax, 0  
00401005 . 83F8 01 cmp eax, 1  
00401008 . 74 0C je short 00401016  
0040100A . B8 14304000 mov eax, offset 00403014  
0040100F . B9 30000000 mov ecx, 30

Assemble

00401000 mov eax, 5

Keep size  
 Fill rest with NOPs

Assemble Close



OllyDbg - crackme1.exe - [CPU - main thread, module crackme1]

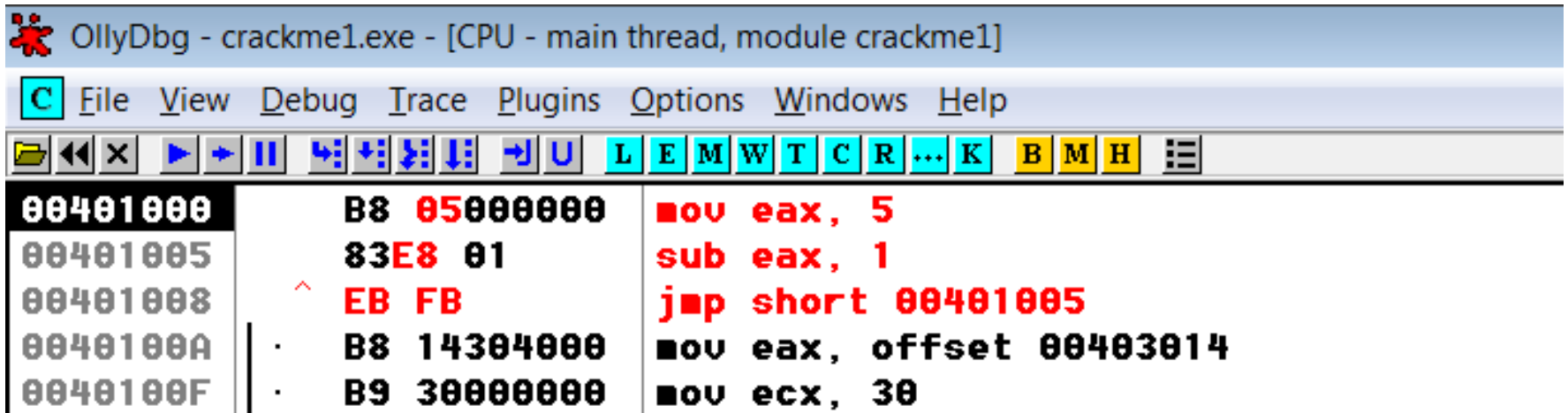
File View Debug Trace Plugins Options Windows Help

00401000 B8 05000000 **mov eax, 5**  
00401005 . 83F8 01 cmp eax, 1  
00401008 . 74 0C je short 00401016  
0040100A . B8 14304000 mov eax, offset 00403014  
0040100F . B9 30000000 mov ecx, 30



# OllyDbg assembler

- Např. nekonečná smyčka – instrukce MOV, SUB a JMP:
  1. Assembluj MOV EAX, 5 na adrese 401000 (= EIP)
  2. Assembluj SUB EAX, 1 na adrese 401005
  3. Assembluj JMP 401005 na adrese 401008
- Změny se obarví červeně, super věc na učení se opkódů



```
OllyDbg - crackme1.exe - [CPU - main thread, module crackme1]
File View Debug Trace Plugins Options Windows Help
[Icons] [L] [E] [M] [W] [T] [C] [R] [K] [B] [M] [H] [Menu]
00401000 B8 05000000 mov eax, 5
00401005 83E8 01 sub eax, 1
00401008 ^ EB FB jmp short 00401005
0040100A . B8 14304000 mov eax, offset 00403014
0040100F . B9 30000000 mov ecx, 30
```

# Instrukce INT3; NOP; ADD [EAX], AL

- CC INT3
- Výplň kódu (i dat) na místech, kam se nemá dostat řízení
- Způsobí ladící výjimku; tu teď neřešíme
  
- 90 NOP
- No OPeration
- Výplň pro zarovnání kódu
  
- 0000 ADD [EAX], AL
- Podezřelá instrukce
- Inicializační hodnota při alokaci paměti, nejde o kód

# Shrnutí pojmů (1)

- Virtuální adresa
- Opkód, primární opkód
- Mnemonic instrukce
- Operand instrukce
- Intel syntaxe vs. AT&T syntaxe
- Osm všeobecných registrů
- Velikosti: DWORD, WORD, BYTE
- Orientace v primárních opkódech
- Registr EFlags, šest stavových příznaků, příznak ZF
- Registr EIP, kalkulace návěští skoku
- Volací konvence stdcall

# Shrnutí pojmů (2)

- Volání funkce: CALL – RET
- Odstraňování parametrů ze zásobníku pomocí RET
- OllyDbg assembler

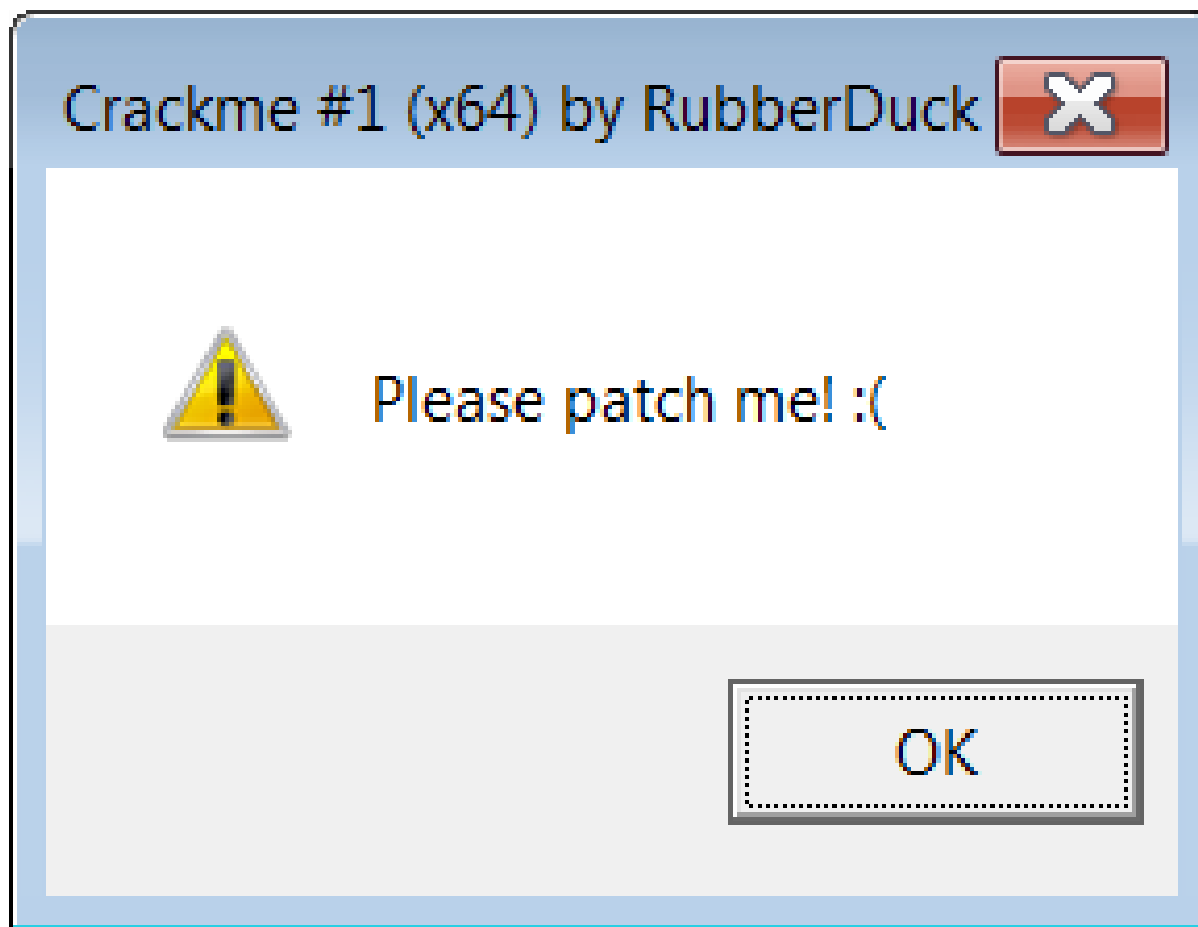
# Shrnutí instrukcí

- MOV
- CMP, SUB
- JE, Jcc
- JMP
- PUSH
- CALL
- INT3
- NOP
- ADD [EAX], AL

# Problémy s označením architektury

- Hromada zjednodušujících názvů pro komplikovanou architekturu:
  - Intel
    - IA-32 architecture
    - Intel64 architecture (nový „**64-bit mode**“)
  - AMD64 architecture
    - x86 architecture, „legacy mode“
    - 64-bit x86 architecture (nový „Long Mode“)
  - Microsoft
    - x86
    - x64

# Crackme #1 (x64) by RubberDuck



# x64dbg – crackme1\_x64.exe

The screenshot displays the x64dbg debugger interface for the file crackme1\_x64.exe. The main window shows assembly code with the following instructions:

```
sub rsp, 8
mov eax, 0
cmp eax, 1
je crackme1_x64.13FC0101F
movabs rax, crackme1_x64.13FC03014
mov ebx, 30
jmp crackme1_x64.13FC0102E
movabs rax, crackme1_x64.13FC03000
mov ebx, 0
sub rsp, 20
mov rcx, 0
mov rdx, rax
movabs r8, crackme1_x64.13FC03028
mov r9, rbx
call <crackme1_x64.MessageBoxA>
add rsp, 20
sub rsp, 20
mov ecx, 0
call <crackme1_x64.RtlExitUserProcess>
jmp qword ptr ds:[<&RtlExitUserProcess>]
jmp qword ptr ds:[<&MessageBoxA>]
add byte ptr ds:[rax], al
add byte ptr ds:[rax], al
add byte ptr ds:[rax], al
add byte ptr ds:[rax], al
add byte ptr ds:[rax], al
add byte ptr ds:[rax], al
add byte ptr ds:[rax], al
add byte ptr ds:[rax], al
add byte ptr ds:[rax], al
add byte ptr ds:[rax], al
```

The registers window shows the following values:

RAX	0000000076C959D0	<kernel32.BaseThreadIn
RBX	0000000000000000	
RCX	000007FFFFFFD3000	
RDX	000000013FC01000	<crackme1_x64.EntryPoi
RBP	0000000000000000	
RSP	000000000024FE98	
RSI	0000000000000000	
RDI	0000000000000000	
R8	000007FFFFFFD3000	
R9	000000013FC01000	<crackme1_x64.EntryPoi
R10	0000000000000000	
R11	0000000000000000	
R12	0000000000000000	
R13	0000000000000000	
R14	0000000000000000	
R15	0000000000000000	
RIP	000000013FC01000	<crackme1_x64.EntryPoi

The registers window also shows RFLAGS: 0000000000000244, ZF: 1, PF: 1, AF: 0, OF: 0, SF: 0, DF: 0, CF: 0, TF: 0, IF: 1, and LastError: 00000000 (ERROR\_SUCCESS).

The memory dump window shows the following hex and ASCII data:

Address	Hex	ASCII
0000000076DA1000	48 8B 40 48 48 39 41 10 0F 84 25 D3 05 00 E8 6D	H.@HH9A...%0..em
0000000076DA1010	4C 02 00 84 C0 0F 85 20 D3 05 00 48 8B 68 01	L...A. O..H..h.
0000000076DA1020	00 00 E8 D9 CA 04 00 41 BA 01 00 00 00 41 8B D2	..èÙÊ..A°.A.O
0000000076DA1030	48 8B CE E8 E8 13 02 00 E9 7A 02 03 00 48 89 5F	H.Ièè...éz...H..
0000000076DA1040	10 48 89 3B 44 89 6F 30 E9 62 CB 00 00 33 C0 48	.H.;D.o0èBÈ..3AH
0000000076DA1050	83 C4 28 C3 48 8B 5C 24 08 33 C0 C3 0F B7 47 E6	.A(AH.\\$.3AA..GA
0000000076DA1060	0F B6 C0 66 44 0F B7 0C 4E 66 41 83 F9 61 0F 82	..l..D..NFA.üa..
0000000076DA1070	D4 3B 06 00 66 41 83 F9 7A 0F 87 9C 3B 06 00 66	Ö;..f.ü.ü...;..f
0000000076DA1080	44 03 CD 41 0F B7 C1 0F B6 0C 18 41 88 4B F3 E9	V..L..eê..H..Y..
0000000076DA1090	04 00 00 4C 8D 2D 65 BF 0F 00 48 85 FF 0F 84	.V...L..eê..H..Y..
0000000076DA10A0	43 06 01 00 E9 F0 41 06 00 90 90 90 90 90 90 90	C...èDA...e...A
0000000076DA10B0	41 0B 01 EB 0B 90 90 90 90 90 90 90 90 90 90 90	A...è...A...e...A

The command window shows the command: `INT3 breakpoint "entry breakpoint" at 000000013FC01000!`



# crackme1\_x64: 1. instrukce SUB

- `000000013FC0100 48 83 EC 08 SUB RSP, 8`
- Vypadá jako alokace 8 bajtů na zásobníku
- Ve skutečnosti zarovnání zásobníku na 16 bajtů, vynucené volací konvencí (viz následující slajdy)
  - Komplikace při programování v assembleru
- Jednobažtový primární operační znak `0x83`
- [ref.x86asm.net/coder64.html#x83](http://ref.x86asm.net/coder64.html#x83)
- Nová věc: prefix `48 REX.W`
- [ref.x86asm.net/coder64.html#x48](http://ref.x86asm.net/coder64.html#x48)
- Nejčastější REX.W zvětší velikost operandu na 64 bitů
- REX prefixy na x86 neexistují

# 64-bit registers

- Šestnáct 64bitových všeobecných registrů, kód 0-15 (4 bity)

RAX	RCX	RDX	RBX	RSP	RBP	RSI	RDI
-----	-----	-----	-----	-----	-----	-----	-----

0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---

R8	R9	R10	R11	R12	R13	R14	R15
----	----	-----	-----	-----	-----	-----	-----

8	9	10	11	12	13	14	15
---	---	----	----	----	----	----	----

- 64bitový instrukční ukazatel RIP

# 64-bit general-purpose registers

- RAX = 8877665544332211, velikost QWORD
  - EAX: 44332211
    - AX: 2211
      - AL: 11
      - AH: 22
- RSP – ESP – SP – SPL
- RBP – EBP – BP – BPL
- RSI – ESI – SI – SIL
- RDI – EDI – DI – DIL
- R15 - R15D - R15W - R15B
  
- Horní DWORD nejde adresovat a nemá název

# crackme1\_x64: 5. instrukce MOVABS

- 000000013FC0100 48 B8 1430C03F01000000  
MOVABS RAX, 13FC03014
- Jednobajtový primární opkód 0xB8
- Prefix 48 REX.W
- Mnemonic MOVABS neexistuje, jde o MOV
  - lidová tvořivost, odpovídá to správnějšímu OFFSET v OllyDbg

# crackme1\_x64: volání MessageBoxA

- Předávání parametrů je zhruba vidět:

```
SUB RSP, 20
MOV RCX, 0 ; hWnd
MOV RDX, RAX ; lpText
MOVABS R8, 13FC03028 ; lpCaption
MOV R9, RBX ; uType
CALL <crackme1_x64.MessageBoxA>
ADD RSP, 20
```

- První 4 parametry jdou vždycky do registrů RCX, RDX, R8 a R9
- Ale co ten SUB/ADD RSP, 20?

# Volací konvence fastcall (1)

- Pozor, nejde o x86 fastcall, i když je podobný
- Ještě jednou, konvence bývá vidět v prototypu:
  - `int WINAPI MessageBox () ;`
- WinDef.h (Windows SDKs): `#define WINAPI`
  - tentokrát je to „prázdná“ hodnota, všechno je fastcall (s výjimkami)
- [MSDN x64 fastcall calling convention](#)
  - První 4 parametry jdou vždycky do registrů RCX, RDX, R8 a R9, ostatní na zásobník
  - Zásobník čistí volaná funkce
  - Návratová hodnota je v registru RAX
  - Registry RAX, RCX, RDX, R8, R9, R10 a R11 může volaná funkce změnit, ostatní musí zachovat

# Volací konvence fastcall (2)

- První 4 parametry jsou v registrech, ale na zásobníku musí být tak jako tak alokovaný prostor, další jdou na zásobník jako u stdcall
- Alokace a uvolnění 4 parametrů pomocí  
`SUB/ADD RSP, 20`
- $0x20 = 32 = 4 * 8$

# fastcall a zarovnání zásobníku

- MSDN: [The stack will always be maintained 16-byte aligned](#)
- Vstupní bod programu je vlastně vstupní bod funkce `main()`
- Volání funkce zruší zarovnání na 16 bytů:
  1. Před provedením `CALL` je zásobník zarovnaný na 16 bytů
  2. `CALL` provede `PUSH RIP` (8 bytů)
  3. Na vstupu do funkce je potřeba zásobník znovu zarovnat pomocí `SUB RSP, 8`



# crackme1\_x64: volání ExitProcess

- ExitProcess() má jenom jeden parametr, ale na zásobníku je potřeba vždycky alokovat 4 parametry:

```
SUB RSP, 20 ; 4*QWORD
```

```
MOV ECX, 0
```

```
CALL crackme1_x64.RtlExitUserProcess
```

# Shrnutí pojmů (x64)

- 16 64bitových všeobecných registrů
- 64bitová virtuální paměť, registr RIP
- Velikost QWORD
- Volací konvence fastcall
- Zarovnání zásobníku
- Prefixy REX

# Domácí úkoly

1. Crackněte crackme1 (x86 i x64) podle návodu v článku [Nebojte se reverzního inženýrství I.](#)
2. Crackněte crackme1 ještě jinak než podle návodu
3. Pokud v crackme1 změníte instrukci CMP na SUB, jaký vliv to bude mít na funkčnost programu?
4. Přepište crackme1 do vyššího jazyka podle vašeho výběru
5. Vytvořte přímo v OllyDbg assembleru co nejkratší kód, který způsobí vyčerpání paměti zásobníku