

REbejs

2. workshop (draft0)



Opakování 1. workshopu

- Virtuální adresa
- Op kód instrukce, primární op kód, REX prefixy
- Instrukce, mnemonic, operandy
- Všeobecné registry, EIP, RIP, EFlags
- Volání funkce
- Základy OllyDbg, x64dbg
- Základní instrukce
- Hlavní volací konvence

Podklady

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

Domácí úkoly

- Cracknutí crackme1 podle návodu
- Cracknutí jiným způsobem
- Změna CMP na SUB a její důsledky
- Přepsání crackme1 do vyššího jazyka
- Nejkratší kód na vyčerpání paměti zásobníku

2. workshop - témata

- Crackme #2 by RubberDuck
- + opkódy instrukcí
- + x64
- + x64 verze crackme #2: [crackme2_x64.zip](#)
- Paměťový model aplikace na Windows
- Módy, v kterých procesor může běžet (TODO SMM rootkit zjistit, co a jak)
- Ochrana paměti, virtuální paměť
- Stack, heap, entry point, kód, data, [TEB](#) (TIB)
- Binární formát instrukce
- Orientace v manuálech procesorů
- Vyhledávání na [MSDN](#)

1. workshop - nástroje

- 32bitový debugger OllyDbg
 - [OllyDbg 2.01](#)
- 64bitový debugger x64dbg
 - [x64dbg snapshot_2016-01-21_02-44](#)

Crackme #2 by RubberDuck

```
C:\>crackme_2.exe  
Crackme #2 by RubberDuck  
Insert password: dunno  
No. This isn't good password.. :(  
C:\>
```

OllyDbg – crackme2.exe

OllyDbg - crackme_2.exe - [CPU - main thread, module crackme_2]

File View Debug Trace Plugins Options Windows Help

LEMTWTCR...K B M H

| | | | | |
|----------|---------------|-------------------------------------|--|------------------------------|
| 00401000 | 6A F5 | push -0B | StdHandle = STD_OUTPUT_HANDLE KERNEL32.GetStdHandle | Registers (FPU) |
| 00401002 | E8 A3000000 | call < jmp.&kernel32.GetStdHandle> | | EAX A2F00EB0 |
| 00401007 | A3 95304000 | mov [403095],eax | | ECX 00401000 crackme_2.<Mod |
| 0040100C | 6A F6 | push -0A | StdHandle = STD_INPUT_HANDLE KERNEL32.GetStdHandle | EDX 00401000 crackme_2.<Mod |
| 0040100E | E8 97000000 | call < jmp.&kernel32.GetStdHandle> | | EBX 7FFDE000 |
| 00401013 | A3 99304000 | mov [403099],eax | | ESP 0019FF84 |
| 00401018 | 6A 00 | push 0 | Reserved = 0 | EBP 0019FF94 |
| 0040101A | 68 8D304000 | push offset 0040308D | pWritten = crackme_2.40308D -> € | ESI 00401000 crackme_2.<Mod |
| 0040101F | 68 2B000000 | push 2B | Count = 43. | EDI 00401000 crackme_2.<Mod |
| 00401024 | 68 00304000 | push offset 00403000 | Buffer = "Crackme #2 by RubberDuck" Insert | EIP 00401000 crackme_2.<Mod |
| 00401029 | FF35 95304000 | push dword ptr [403095] | hConsole = NULL | |
| 0040102F | E8 82000000 | call < jmp.&kernel32.WriteConsoleA> | KERNEL32.WriteConsoleA | C 0 ES 002B 32bit 0(FFFFFFF) |
| 00401034 | 6A 00 | push 0 | | P 1 CS 0023 32bit 0(FFFFFFF) |
| 00401036 | 68 91304000 | push offset 00403091 | Jump to KERNEL32.ReadConsoleA | A 0 SS 002B 32bit 0(FFFFFFF) |
| 0040103B | 68 14000000 | push 14 | String2 = "Bad password bad day" | Z 1 DS 002B 32bit 0(FFFFFFF) |
| 00401040 | 68 77304000 | push offset 00403077 | String1 | S 0 FS 0053 32bit 7FFDD000 |
| 00401045 | FF35 99304000 | push dword ptr [403099] | KERNEL32.lstrcp | T 0 GS 002B 32bit 0(FFFFFFF) |
| 0040104B | E8 60000000 | call < jmp.&kernel32.ReadConsoleA> | | D 0 |
| 00401050 | 68 62304000 | push offset 00403062 | | 0 0 LastErr 00000032 ERROR |
| 00401055 | 68 77304000 | push offset 00403077 | | EFL 00000246 (NO,NB,E,BE,NS) |
| 0040105A | E8 5D000000 | call < jmp.&kernel32.lstrcpA> | | ST0 empty 0.0 |
| 0040105F | 85C0 | test eax,eax | Reserved = 0 | ST1 empty 0.0 |
| 00401061 | 74 1E | jz short 00401081 | pWritten = crackme_2.40308D -> € | ST2 empty 0.0 |
| 00401063 | 6A 00 | push 0 | | ST3 empty 0.0 |
| 00401065 | 68 8D304000 | push offset 0040308D | | ST4 empty 0.0 |

Stack [0019FF80]=0
Imm=FFFFFFF5 (decimal -11.)

| Address | Hex dump | ASCII (ANSI - st | 0019FF84 | 74A53744 | D7at | RETURN to KERNEL32.Base |
|----------|---|------------------|----------|----------|------|-------------------------|
| 00403000 | 43 72 61 63 68 6D 65 20 23 32 20 62 79 20 52 75 | Crackme #2 by Ru | 0019FF88 | 7FFDE000 | a0rA | |
| 00403010 | 62 62 65 72 44 75 63 68 0A 0D 49 6E 73 65 72 74 | bberDuck Insert | 0019FF8C | 74A53720 | 7at | KERNEL32.BaseThreadInit |
| 00403020 | 20 70 61 73 73 77 6F 72 64 3A 00 47 6F 6F 64 20 | password:aGood | 0019FF90 | A2F00EB0 |]-6 | |
| 00403030 | 6A 6F 62 20 63 72 61 63 6B 65 72 21 20 3A 29 00 | job cracker! :)a | 0019FF94 | 0019FFDC | ala | |
| 00403040 | 4E 6F 2E 20 54 68 69 73 20 69 73 6E 27 74 20 67 | No. This isn't g | 0019FF98 | 77149E54 | TxW | RETURN to ntdll.77149E5 |
| 00403050 | 6F 6F 64 20 70 61 73 73 77 6F 72 64 2E 2E 20 3A | ood password.. : | 0019FF9C | 7FFDE000 | a0rA | |
| 00403060 | 28 00 42 61 64 20 70 61 73 73 77 6F 72 64 20 62 | (aBad password b | 0019FFA0 | 94D0BECD | =zd0 | |
| 00403070 | 61 64 20 64 61 79 00 00 00 00 00 00 00 00 00 | ad dayaaaaaaaaa | 0019FFA4 | 00000000 | aaaa | |

Module <Mod_5405> (anonymous)

Paused

crackme2.exe – 1. instrukce PUSH

- 00401000 6A F5 PUSH -0B
- ref.x86asm.net/coder32.html#x6A
- 6A PUSH (sign extended) imm8
- imm8: 8bitová immediate value
- Takže jinak (velikost položky na zásobníku je DWORD):
- PUSH *FFFFFFFF5*
- Je to první parametr volání WinAPI funkce GetStdHandle()

GetStdHandle()

- [MSDN](#): Retrieves a handle to the specified standard device (standard input, standard output, or standard error).

```
HANDLE WINAPI GetStdHandle (  
    __In__ DWORD nStdHandle  
);
```

- Potřebujeme návratovou hodnotu HANDLE
- `winnt.h` (Microsoft SDKs):
- `typedef void *HANDLE;`
- Volací konvence `stdcall`: návratová hodnota je v EAX, ta je uložena pomocí následující instrukce na později

crackme2.exe – 3. instrukce MOV

- 00401007 A3 95304000 MOV [403095], EAX
- ref.x86asm.net/coder32.html#xA3
- A3 MOV moffs16/32, eAX
- Opkódy A0–A3
- speciální forma MOV mezi AL/AX/EAX a paměti
- Registr eAX (tzn. AX nebo EAX):
- *accumulator*, historicky optimalizovaný pro častý přístup
- Spousta opkódů podporujících accumulator, umožňujících kratší kódování instrukcí (viz třeba 0x04, 0x05, 0x0C, ...)
- Dědictví historie, dnes nemá podstatný význam
- *moffs* - *memory offset*: konstantní hodnota adresuje paměť

Crackme2.exe – 3. instrukce MOV

- **MOV [403095], EAX**
- 0x403095 = 4206741 = nezarovnaná paměť při ukládání DWORD hodnoty
- Architektura x86 nevynucuje zarovnání paměti na určitou hodnotu
- Připomenutí: 64-bit Windows vynucuje zarovnání zásobníku na 16 bytů

crackme2.exe – 10. a 11. instr. PUSH

- 00401024 **68** 00304000

PUSH OFFSET 00403000

- 00401029 **FF35** 95304000

PUSH DWORD PTR [403095]

- ref.x86asm.net/coder32.html#xFF

- Skupina různých instrukcí

- **FF** /6 **PUSH r/m16/32**

- *ModR/M byte 0x35*

- **DWORD PTR**: označení velikosti odkazované hodnoty

crackme2.exe – porovnání hesel

- `WriteConsoleA()` : vypíše výzvu k zadání hesla
- `ReadConsoleA()` : přečte zadané heslo
- `lstrcmp()` : WinAPI funkce porovnávající řetězce, podobná funkci ze standardní C knihovny

```
int WINAPI lstrcmp(  
    _In_ LPCTSTR lpString1,  
    _In_ LPCTSTR lpString2  
);
```

- „If the strings are equal, the return value is zero.“

crackme2.exe – vyhodnocení hesla

- **85C0 TEST EAX, EAX**
- Zvláštní případ na vyhodnocení nulové hodnoty v registru
- Opkódy 84 a 85: TEST r/m, r
- ModR/M byte C0
- TEST: logical compare
- TEST nemění žádný operand
- Operace TEST = operace AND
- TEST nastavuje šest stavových příznaků, viz:

| | | | | | | | | | | | | | | | | | | |
|----|---|--|--|--|------|----------|--------|--|--|--|--|--|--|--|--|--|--|-----------|
| 85 | r | | | | TEST | r/m16/32 | r16/32 | | | | | | | | | | | o...szapc |
|----|---|--|--|--|------|----------|--------|--|--|--|--|--|--|--|--|--|--|-----------|

- Smysl dávají tři: SF, ZF a PF
- $0 \text{ AND } 0 = 0 \rightarrow$ nastaví se ZF

crackme1.exe – vyhodnocení CMP

- crackme1:

```
83F8 01  CMP EAX, 1
```

```
74 0C    JE SHORT 00401016
```

- crackme2:

```
85C0    TEST EAX, EAX
```

```
74 1E    JZ SHORT 00401081
```

- TEST EAX, EAX má kratší opkód než CMP EAX, 0
- Zajímavost: OllyDbg disassembler vyhodnotil, že vhodnější je mnemonic JZ

Shrnutí instrukcí

- MOV
- CMP, SUB
- **TEST, AND**
- JE, Jcc
- JMP
- PUSH
- CALL
- **Adresace paměti pomocí [moffs] a [disp32]**
- INT3
- NOP
- ADD [EAX], AL

Crackme #2 (x64) by RubberDuck

```
C:\>crackme2_x64.exe
Crackme #2 (x64) by RubberDuck
Insert password: rebejs!
No. This isn't good password.. :(
C:\>
```

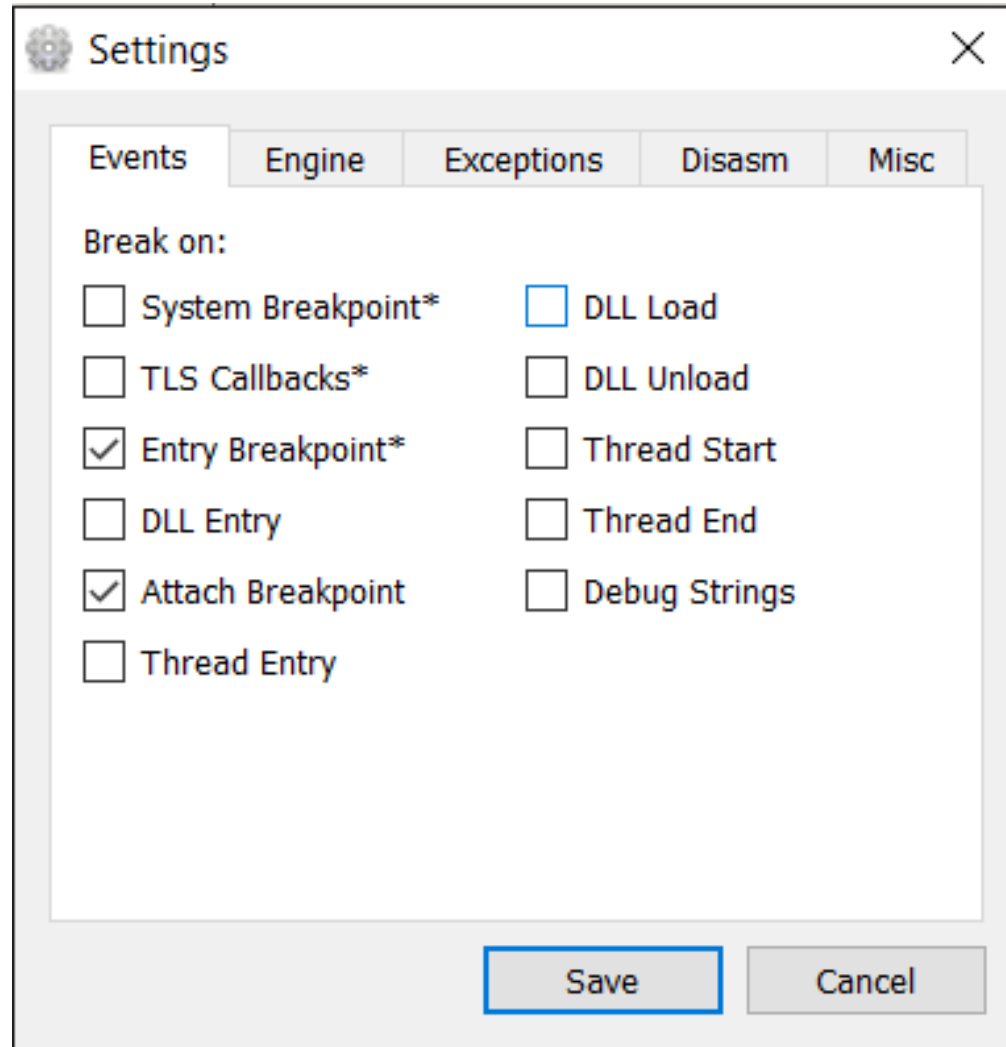
x64dbg – crackme2_x64.exe

x64dbg - File: crackme2_x64.exe - PID: 1780 - Module: crackme2_x64.exe - Thread: 23B4

File View Debug Plugins Options Help Jan 21 2016

The screenshot displays the x64dbg debugger interface. The main window shows assembly code for the function `crackme2_x64.7FF605493098`. The code includes instructions such as `sub rsp,8`, `mov ecx,FFFFFFFF5`, `call <crackme2_x64.GetStdHandle>`, `mov qword ptr ds:[7FF605493098],rax`, `add rsp,20`, `mov qword ptr ds:[7FF6054930A3],rax`, `sub rsp,30`, `mov rcx,qword ptr ds:[7FF605493098]`, `movabs rdx,crackme2_x64.7FF605493000`, `mov r8d,31`, `movabs r9,crackme2_x64.7FF605493093`, `mov qword ptr ss:[rsp+20],0`, `call <crackme2_x64.writeConsoleA>`, `add rsp,30`, `sub rsp,30`, `mov rcx,qword ptr ds:[7FF6054930A3]`, `movabs rdx,crackme2_x64.7FF60549307D`, `mov r8d,14`, `movabs r9,crackme2_x64.7FF605493097`, `mov qword ptr ss:[rsp+20],0`, `call <crackme2_x64.ReadConsoleA>`, `add rsp,30`, `sub rsp,20`, `movabs rdx,crackme2_x64.7FF60549307D`, `movabs rdx,crackme2_x64.7FF605493068`, `call <crackme2_x64.lstrcmp>`, `add rsp,20`, `test eax,ecx`, `je crackme2_x64.7FF605491102`, `sub rsp,30`, `mov rcx,qword ptr ds:[7FF605493098]`, `movabs rdx,crackme2_x64.7FF605493046`, `mov r8d,22`, `movabs r9,crackme2_x64.7FF605493093`, and `mov qword ptr ss:[rsp+20],0`. The registers window shows `RAX: 0000000000000000`, `RBX: 00007FF605491000`, `RCX: 00000007C1B32000`, `RDX: 00007FF605491000`, `RBP: 0000000000000000`, `RSP: 00000007C1CFFE58`, `RSI: 00000007C1B32000`, `RDI: 00000007C1B32000`, `R8: 00000007C1B32000`, `R9: 0000000000000000`, `R10: 0000000000000000`, `R11: 0000000000000000`, `R12: 0000000000000000`, `R13: 0000000000000000`, `R14: 0000000000000000`, `R15: 0000000000000000`, and `RIP: 00007FF605491000`. The memory dump window shows the current instruction's memory dump, including the address `00007FF605491000` and the instruction `sub rsp,8`. The command window shows `INT3 breakpoint "entry breakpoint" at 00007FF605491000`.

x64dbg – nastavení eventů



Domácí úkoly

1. Crackněte crackme2 (x86 i x64), jakobyste neznali heslo