درس ۱۱

چگونه کراک کنیم، کراکینگ بعنوان یک هنر

(برای اولین قسمت این جلسه کراکینگ به درس ۱۰ مراجعه کنید).

در اینجا روشهای محافظت را برای اولین کد عددی («ثبت») ، با توضیحات من دنبال کنید:

شما باید کد زیر را مورد بررسی قرار دهید.

سپس، زمانیکه شما به روش خودتان کراک میکنید ، سعی کنید روتین‌های زیادی را شناسایی کنید که قبل از (محافظت واقعی) مربوطه، ورودی شما را دستکاری می‌کند. برای مثال، در این مورد، یک روتین، درستی اعداد و ارقام ورودی شما را کنترل می‌کند.

این حلقه ارقام مربوطه را کنترل می‌کند.

```
This_loop_checks_that_numbers_are_numbers:
1B0F:2B00 C45E06    LES    BX,[BP+06]  ; set/reset pointer
1B0F:2B03 03DF      ADD    BX,DI
1B0F:2B05 268A07    MOV    AL,ES:[BX]  ; get number
1B0F:2B08 8846FD    MOV    [BP-03],AL  ; store
1B0F:2B0B 807EFD30  CMP    BYTE PTR [BP-03],30
1B0F:2B0F 7C06      JL     2B17        ; less than zero?
1B0F:2B11 807EFD39  CMP    BYTE PTR [BP-03],39
1B0F:2B15 7E05      JLE    2B1C        ; between 0 & 9?
1B0F:2B17 B80100    MOV    AX,0001     ; no, set flag=1
1B0F:2B1A EB02      JMP    2B1E        ; keep flag
1B0F:2B1C 33C0      XOR    AX,AX       ; flag=0
1B0F:2B1E 0BC0      OR     AX,AX       ; is it zero?
1B0F:2B20 7507      JNZ    2B29        ; flag NO jumps away
1B0F:2B22 8A46FD    MOV    AL,[BP-03]  ; Ok, get number
1B0F:2B25 8842CC    MOV    [BP+SI-34],AL ; Ok, store number
1B0F:2B28 46        INC    SI          ; inc storespace
1B0F:2B29 47        INC    DI          ; inc counter
1B0F:2B2A C45E06    LES    BX,[BP+06]  ; reset pointer
1B0F:2B2D 03DF      ADD    BX,DI       ; point next number
1B0F:2B2F 26803F00  CMP    BYTE PTR ES:[BX],00 ; input end?
1B0F:2B33 75CB      JNZ    2B00        ; no:loop next num
```

اکنون شما بطور واضح می‌فهمید که رشته واقعی داخل حافظه [BP+SI-34] ذخیره می‌شود...

نقطه۱ توقف حافظه را برای بدست آوردن بلوک بعدی کدی که ورودی تغییر یافته را دستکاری می‌کند در این محل قرار دهید. توجه داشته باشید که چگونه این روتین. ورودی را نرمال‌سازی می‌کند. «-» را برمیدارد و ۱۰ رقم را کنار هم قرار میدهد.

ورودی استفاده کننده

```
          1  2  1  2  1  2  1  2  1  2 End
1E7F:92E2 31 32 31 32 31 32 31 32 31 32 00 45 AF 1F 70 9B
Stack ptr: 0 1 2 3 4 5 6 7 8 9 A B C D E F
```

اکنون اجازه دهید به روتین محافظت «واقعی» نگاهی بیندازیم. روتینی که این اعداد و ارقام را کنترل می‌کند و اگر آنها «درست» نباشند ارقام شما را حذف می‌کند. لطفاً به بلوک کد زیر توجه کنید:

کنترل کنید و ببینید که آیا مجموع ۹ رقم مساوی با باقیمانده عدد سوم هست یا نه:

ما داخل استاک(پشته) کار خواهیم کرد:

```
4B79  8CD0  MOV       AX,SS

٤:کدB79 8CD0       MOV    AX,SS ; we'll work inside the stack...
:4B7B 90         NOP
:4B7C 45         INC    BP
:4B7D 55         PUSH   BP    ; save real BP
:4B7E 8BEC       MOV    BP,SP ; BP = stackpointer
:4B80 1E         PUSH   DS    ; save real Datasegment
:4B81 8ED8       MOV    DS,AX ; Datasegment = stacksegment
:4B83 83EC04     SUB    SP,+04
:4B86 C45E06     LES    BX,[BP+06] ; BX points input_start
:4B89 268A07     MOV    AL,ES:[BX] ; load first number
:4B8C 98         CBW               ; care only for low
:4B8D C45E06     LES    BX,[BP+06] ; reset pointer
:4B90 50         PUSH   AX         ; save 1st number
:4B91 268A4701   MOV    AL,ES:[BX+01] ; load 2nd number
:4B95 98         CBW               ; only low
:4B96 8BD0       MOV    DX,AX      ; 2nd number in DX
:4B98 58         POP    AX         ; get 1st number
:4B99 03C2       ADD    AX,DX      ; sum with second
:4B9B C45E06     LES    BX,[BP+06] ; reset pointer
:4B9E 50         PUSH   AX         ; save sum
:4B9F 268A4707   MOV    AL,ES:[BX+07] ; load 8th number
:4BA3 98         CBW               ; only low
:4BA4 8BD0       MOV    DX,AX      ; 8th number in DX
:4BA6 58         POP    AX         ; old sum is back
```

```
:4BA7 03C2       ADD    AX,DX      ; sum 1+2+8
:4BA9 C45E06     LES    BX,[BP+06] ; reset pointer
:4BAC 50         PUSH   AX         ; save sum
:4BAD 268A4703   MOV    AL,ES:[BX+03] ; load 4rd number
:4BB1 98         CBW               ; only low
:4BB2 8BD0       MOV    DX,AX      ; #4 in DX
:4BB4 58         POP    AX         ; sum is back
:4BB5 03C2       ADD    AX,DX      ; sum 1+2+8+4
:4BB7 C45E06     LES    BX,[BP+06] ; reset pointer
:4BBA 50         PUSH   AX         ; save sum
:4BBB 268A4704   MOV    AL,ES:[BX+04] ; load 5th number
:4BBF 98         CBW               ; only low
:4BC0 8BD0       MOV    DX,AX      ; #5 in DX
:4BC2 58         POP    AX         ; sum is back
:4BC3 03C2       ADD    AX,DX      ; 1+2+8+4+5
:4BC5 C45E06     LES    BX,[BP+06] ; reset pointer
:4BC8 50         PUSH   AX         ; save sum
:4BC9 268A4705   MOV    AL,ES:[BX+05] ; load 6th number
:4BCD 98         CBW               ; only low
:4BCE 8BD0       MOV    DX,AX      ; #6 in DX
:4BD0 58         POP    AX         ; sum is back
:4BD1 03C2       ADD    AX,DX      ; 1+2+8+4+5+6
:4BD3 C45E06     LES    BX,[BP+06] ; reset pointer
:4BD6 50         PUSH   AX         ; save sum
:4BD7 268A4706   MOV    AL,ES:[BX+06] ; load 7th number
:4BDB 98         CBW               ; only low
:4BDC 8BD0       MOV    DX,AX      ; #7 in DX
:4BDE 58         POP    AX         ; sum is back
:4BDF 03C2       ADD    AX,DX      ; 1+2+8+4+5+6+7
:4BE1 C45E06     LES    BX,[BP+06] ; reset pointer
:4BE4 50         PUSH   AX         ; save sum
:4BE5 268A4708   MOV    AL,ES:[BX+08] ; load 9th number
:4BE9 98         CBW               ; only low
:4BEA 8BD0       MOV    DX,AX      ; #9 in DX
:4BEC 58         POP    AX         ; sum is back
:4BED 03C2       ADD    AX,DX      ; 1+2+8+4+5+6+7+9
:4BEF C45E06     LES    BX,[BP+06] ; reset pointer
:4BF2 50         PUSH   AX         ; save sum
:4BF3 268A4709   MOV    AL,ES:[BX+09] ; load 10th #
:4BF7 98         CBW               ; only low
:4BF8 8BD0       MOV    DX,AX      ; #10 in DX
:4BFA 58         POP    AX         ; sum is back
:4BFB 03C2       ADD    AX,DX      ; 1+2+8+4+5+6+7+9+10
:4BFD 0550FE     ADD    AX,FE50    ; clean sum to 0-51
:4C00 BB0A00     MOV    BX,000A    ; BX holds 10
:4C03 99         CWD               ; only AL
:4C04 F7FB       IDIV   BX         ; remainder in DX
:4C06 C45E06     LES    BX,[BP+06] ; reset pointer
:4C09 268A4702   MOV    AL,ES:[BX+02] ; load now # 3
:4C0D 98         CBW               ; only low
:4C0E 05D0FF     ADD    AX,FFD0    ; clean # 3 to 0-9
:4C11 3BD0       CMP    DX,AX  ; remainder = pampered #3?
:4C13 7407       JZ     4C1C       ; yes, go on good guy
:4C15 33D2       XOR    DX,DX  ; no! beggar off! Zero DX
:4C17 33C0       XOR    AX,AX  ;     and FLAG_AX = FALSE
:4C19 E91701     JMP    4D33       ; go to EXIT
let's_go_on_if_first_check_passed:
:4C1C C45E06     LES    BX,[BP+06] ; reset pointer
:4C1F 268A4701   MOV    AL,ES:[BX+01] ; now load #2 anew
:4C23 98         CBW               ; only low
:4C24 05D7FF     ADD    AX,FFD7    ; pamper adding +3
```

```
:4C27 A38D5E     MOV    [5E8D],AX  ; save SEC_+3
:4C2A 3D0900     CMP    AX,0009    ; was it < 9? (no A-F)
:4C2D 7E05       JLE    4C34       ; ok, no 0xletter
:4C2F 832E8D5E0A SUB    WORD PTR [5E8D],+0A ; 0-5 if A-F
:4C34 C45E06     LES    BX,[BP+06] ; reset pointer
:4C37 268A07     MOV    AL,ES:[BX] ; load 1st input number
:4C3A 98         CBW               ; only low
:4C3B 05C9FF     ADD    AX,FFC9    ; pamper adding +7
:4C3E A38F5E     MOV    [5E8F],AX  ; save it in FIR_+7
:4C41 0BC0       OR     AX,AX      ; if #1 > 7
:4C43 7D05       JGE    4C4A       ; no need to add 0xA
:4C45 83068F5E0A ADD    WORD PTR [5E8F],+0A ; FIR_+7 + 0xA
now_we_have_the_sliders_let's_prepare_for_loop:
:4C4A C45E0E     LES    BX,[BP+0E] ; Set pointer to E
:4C4D 26C747020000 MOV  WORD PTR ES:[BX+02],0000 ; 0 flag
:4C53 26C7070000   MOV  WORD PTR ES:[BX],0000    ; 0 flag
:4C58 C706975E0900 MOV  WORD PTR [5E97],0009     ; counter=9
:4C5E E99500     JMP    4CF6       ; Jmp check_counter
loop_8_times:
:4C61 C45E06     LES    BX,[BP+06] ; reset pointer
:4C64 031E975E   ADD    BX,[5E97]  ; add running counter
:4C68 268A07     MOV    AL,ES:[BX] ; load # counter+1
:4C6B 98         CBW               ; only low
:4C6C 50         PUSH   AX         ; save 10th number
:4C6D A18D5E     MOV    AX,[5E8D]  ; ld SEC_+3 down_slider
:4C70 BA0A00     MOV    DX,000A    ; BX holds 0xA
:4C73 F7EA       IMUL   DX         ; SEC_+3 * 0xA
:4C75 03068F5E   ADD    AX,[5E8F]  ; plus FIR_+7 up_slider
:4C79 BAA71E     MOV    DX,1EA7    ; fixed segment
:4C7C 8BD8       MOV    BX,AX ; BX = Lkup_val=(SEC_+3*10+FIR_+7)
:4C7E 8EC2       MOV    ES,DX      ; ES = 1EA7
:4C80 268A870000 MOV    AL,ES:[BX+0000] ; ld 1EA7:[Lkup_val]
:4C85 98         CBW               ; only low: KEY_PAR
:4C86 8BD0       MOV    DX,AX      ; save KEY_PAR in DX
:4C88 58         POP    AX         ; repops 10th number
:4C89 03C2       ADD    AX,DX      ; RE_SULT=KEY_PAR+#10
:4C8B 05D0FF     ADD    AX,FFD0    ; polish RE_SULT
:4C8E 99         CWD               ; only low: RE_SULT
:4C8F 8956FC     MOV    [BP-04],DX ; save here KEY_PAR [9548]
:4C92 8946FA     MOV    [BP-06],AX ; save here RE_SULT [9546]
:4C95 0BD2       OR     DX,DX      ; KEY_PAR < 0?
:4C97 7C0F       JL     4CA8       ; yes: KEY_PAR < 0
:4C99 7F05       JG     4CA0       ; no: KEY_PAR > 0
:4C9B 3D0900     CMP    AX,0009    ; KEY_PAR = 0
:4C9E 7608       JBE    4CA8 ; no pampering if RE_SULT < 9
:4CA0 836EFA0A   SUB    WORD PTR [BP-06],+0A ; else pamper
:4CA4 835EFC00   SBB    WORD PTR [BP-04],+00 ; and SBB [9548]
:4CA8 C45E0E     LES    BX,[BP+0E] ; reset pointer to E
:4CAB 268B4F02   MOV    CX,ES:[BX+02] ; charge CX [958C]
:4CAF 268B1F     MOV    BX,ES:[BX] ; charge BX slider [958A]
:4CB2 33D2       XOR    DX,DX      ; clear DX to zero
:4CB4 B80A00     MOV    AX,000A    ; 10 in AX
:4CB7 9A930D2720 CALL   2027:0D93  ; call following RO_routine
```

This is the only routine called from our protection, inside the loop (therefore 8 times), disassembly from WCB. Examining this code please remember that we entered here with following configuration: DX=0, AX=0xA, CX=[958C] and BX=[958A]...

```
 1.0D93  56      push   si     ; save si
 1.0D94  96      xchg   ax, si ; ax=si, si=0xA
 1.0D95  92      xchg   ax, dx ; dx=0xA ax=dx
```

```
 1.0D96  85C0    test   ax, ax ; TEST this zero
 1.0D98  7402    je     0D9C   ; zero only 1st time
 1.0D9A  F7E3    mul    bx     ; BX slider! 0/9/5E/3B2...
 1.0D9C >E305    jcxz   0DA3   ; cx=0? don't multiply!
 1.0D9E  91      xchg   ax, cx ; cx !=0? cx = ax & ax = cx
 1.0D9F  F7E6    mul    si     ;     ax*0xA in ax
 1.0DA1  03C1    add    ax, cx ; ax=  ax*0xA+cx = M_ULT
 1.0DA3 >96      xchg   ax, si ; ax=0xA; si evtl. holds M_ULT
 1.0DA4  F7E3    mul    bx     ; ax= bx*0xA
 1.0DA6  03D6    add    dx, si ; dx= dx_add
 1.0DA8  5E      pop    si     ; restore si
 1.0DA9  CB      retf          ; back to caller with two
                                 parameters: DX and AX
Back_to_main_protection_loop_from_RO_routine:
:4CBC C45E0E     LES    BX,[BP+0E] ; reset pointer
:4CBF 26895702   MOV    ES:[BX+02],DX ; save R_DX par  [958C]
:4CC3 268907     MOV    ES:[BX],AX ; save R_AX par     [958A]
:4CC6 0346FA     ADD    AX,[BP-06] ; add to AX RE_SULT [9546]
:4CC9 1356FC     ADC    DX,[BP-04] ; add to DX KEY_PAR [9548]
:4CCC C45E0E     LES    BX,[BP+0E] ; reset pointer
:4CCF 26895702   MOV    ES:[BX+02],DX ; save R_DX+KEY_PAR [958C]
:4CD3 268907     MOV    ES:[BX],AX ; save R_AX+RE_SULT    [958A]
:4CD6 FF0E8D5E   DEC    WORD PTR [5E8D] ; down_slide SEC_+3
:4CDA 7D05       JGE    4CE1       ; no need to add
:4CDC 83068D5E0A ADD    WORD PTR [5E8D],+0A  ; pamper adding 10
:4CE1 FF068F5E   INC    WORD PTR [5E8F] ; up_slide FIR_+7
:4CE5 A18F5E     MOV    AX,[5E8F]  ; save upslided FIR_+7 in AX
:4CE8 3D0900     CMP    AX,0009    ; is it over 9?
:4CEB 7E05       JLE    4CF2       ; no, go on
:4CED 832E8F5E0A SUB    WORD PTR [5E8F],+0A ; yes, pamper -10
:4CF2 FF0E975E   DEC    WORD PTR [5E97]  ; decrease loop counter
check_loop_counter:
:4CF6 833E975E03 CMP    WORD PTR [5E97],+03  ; counter = 3?
:4CFB 7C03       JL     4D00       ; finish if counter under 3
:4CFD E961FF     JMP    4C61       ; not yet, loop_next_count
loop_is_ended:
:4D00 C45E06     LES    BX,[BP+06] ; reset pointer to input
:4D03 268A4701   MOV    AL,ES:[BX+01] ; load 2nd number (2)
:4D07 98         CBW               ; only low
:4D08 05D0FF     ADD    AX,FFD0    ; clean it
:4D0B BA0A00     MOV    DX,000A    ; DX = 10
:4D0E F7EA       IMUL   DX         ; AX = SEC_*10 = 14
:4D10 C45E06     LES    BX,[BP+06] ; reset pointer
:4D13 50         PUSH   AX         ; save SEC_*10
:4D14 268A07     MOV    AL,ES:[BX] ; load 1st number (1)
:4D17 98         CBW               ; only low
:4D18 8BD0       MOV    DX,AX      ; save in DX
:4D1A 58         POP    AX         ; get SEC_*10
:4D1B 03C2       ADD    AX,DX      ; sum SEC_*10+1st number
:4D1D 05D0FF     ADD    AX,FFD0    ; clean it
:4D20 99         CWD               ; only low
:4D21 C45E0A     LES    BX,[BP+0A] ; get pointer    to   [9582]
:4D24 26895702   MOV    ES:[BX+02],DX ; save 1st (1) in  [9584]
:4D28 268907     MOV    ES:[BX],AX ; save FINAL_SUM (15) [9582]
:4D2B 33D2       XOR    DX,DX      ; DX = 0
:4D2D B80100     MOV    AX,0001    ; FLAG TRUE !
:4D30 E9E6FE     JMP    4C19       ; OK, you_are_a_nice_guy
EXIT:
:4D33 59         POP    CX         ; pop everything and
:4D34 59         POP    CX         ;  return with flag
:4D35 1F         POP    DS         ;  AX=TRUE if RegNum OK
```

```
:4D36 5D          POP     BP          ;  with 1st # in      [9584]
:4D37 4D          DEC     BP          ;  with FINAL_SUM in [9582]
:4D38 CB        RETF
```

اجازه دهید که قبلی را ترجمه کنیم. قبل از هر چیز اشـاره‌گرهـا، در خـط 4B86 مـا اولیـن لیسـت طولانی ptrهای استاک را داریم:

LES BX, [BP+06]

این اشاره گر استاک به ابتدای رشته ورودی اشاره می‌کند که ابتدا از «-» پاک شـده و اکنـون دارای طول ۱۰ بایت می‌باشد. در ابتدا، قبل از حلقه اصلی، ۹ رقم از ۱۰ رقم ما اضافه میشوند همه به غـیر از سومین رقم.

توجه داشته باشید که محافظت در 3# جهش داشته است(و بیرون از خط، 8# را اضافـه می‌کنـد) . بقیه در یک خط راست قرار میگیرند.

اکنون در خط 4BFD ما اولین دستورالعمل (cleaning) تمیز کردن را داریم. میبینید اعدادی کـه در مبنای شانزده هستند با کدهای 0X30 تا 0X39 نشان داده میشـوند. اگر شـما FE50 را بـه مینیمـم مقداری که با جمع کردن ۹ عدد بدست می‌آورید اضافه کنید، عدد صفـر را بدسـت خواهیـد آورد. ماکزیممی که شما میتوانید با جمع کردن ۹ عدد بدست آورید به اضافه OX51, FE50 خواهد بود. بنابراین، به جای عددی بین 0X201, 0X160 ما یـک عـدد جـادویی بیـن 0X0 و 0X51 خواهیـم داشت. محافظت به این نتیجه کمک می‌کند و تنها آخرین کد : ۹-۰ را نگه میدارد . سپس این عـدد از طریق OXA تقسیم می‌شود و چه اتفاقی می‌افتد؟ DX باقیمانده آن را بدست می‌آورد.

اگر ما هگز کدهای (12-1212-1212) را جمع ببندیم به 0X1BE دست خواهیم یافت، (ما بـه غـیر از آن اعداد، ۹ تا را باهم جمع خواهیم کرد. سـومین «۱» یعنـی «۳۱»، جـزو ایـن عملیـات ریـاضی نیست) OX1BE، پس از تمیز شدن E را خواهد داد. بنابراین، (0XE/0XA=1) ما بـا بـاقیمانده ٤، یک را بدست خواهیم آورد.

ممکن است مشاهده کنید که از تمام جوابهای ممکن تنها مجموعه‌هایی که با E,D,C,B,A یا F بـه اتمام میرسند جواب ۱ را میدهند. مجموعه‌های به اتمام رسـیده بـا 0,1,2,3,4,5,6,7,8,9 جـواب ۰ را میدهنـد. شـانس بدسـت آوردن ٤، ۳، ۲، ۱، ۰ بیشـتر از شـانس بدسـت آوردن ۹، ۸، ۷، ٦ و ٥ می‌باشد. ما تنها ناظر هستیم.. هنوز هم نمیدانیم که این امر نقشی دارد یا نه، اکنون این باقیمـانده در 4C11 با سومین عدد بدست آمده از 0X30-0X39 تا ۹-۰ مقایسه می‌شود. ایـن تنـها روش کنـترل ثبت ورودی است. اگر عدد سوم شما با باقیمانده مجموع تمام ۹ رقم دیگر ورودی مطـابقت نکنـد. شما فوراً با FLAG AX=FALSE از موضوع خارج میشوید.

برای بکارگیری محافظت اکنون باید رشته ورودی خود را تغییر دهید. رشته جدیـد ورودی از ایـن به بعد 12-1212-1242 خواهد بود و ما عدد سوم را بـه 4 تغییـر خواهیـم داد. اکنـون محافظـت قسمت ریاضی آن را آغاز می‌کند.

محافظت، دومین رقم ورودی شما را در SEC+3 نگه می‌دارد و در صورتیکه این رقم بزرگتـر از ۹ باشد به آن کمک می‌کند). بنابراین در اینجا شما تطابق ذیل را خواهید داشت:

9=6, 8=5, 7=4, 6=3, 5=2, 4=1, 3=0, 2=9, 1=8, 0=7

دومین عدد ورودی شما به اضافه 3+ خواهد شد. این، مقدار SEC+3 می‌باشد. در C آن شبیه ایـن موارد است:

اگر (Regstromg(2) کمتر از ۷ باشد) Regstring(2)+3=Regstring(2)

دیگر (Regstring(2)-(0)+(3)=Regstring(2)

محافظت، اولین عدد شما را در FIR+7 با پارامتر پاک کننده متفاوت FFC9 نگه میدارد. در مرحلـه بعدی OXA اضافه می‌شود اگر آن 7/8/9 نباشد بنابراین شما تطابق ذیل را خواهید داشت.

8=1, 7=0, 6=9, 5=8, 4=7, 3=6, 2=5, 1=4, 0=3, 9=2 این،مقدار FIR+7 می‌باشد.

در C آن شبیه این مورد خواهد بود.

اگر Regstring(1) کمتر از ۳ باشد Regstring(1)+7=Regstring(1)

دیگر Regstring(1)=7+(10-)Regstring(1) بنابراین، حفاظت تغییر یافتـــه و در [5E80] ,[5E8F] دو عدد ۱ و ۲ ذخیره می‌شود.

در Regstring مـا : 12-1212-1242 دو عـدد اولـی «۱۲» بصـورت «۹٤» ذخـیره می‌شـود. اینـــها بصورت پارامترهای اسلایدر در داخل حلقه اصلـی بصورتیکـه خواهیـد دیـد مـورد اسـتفاده قـرار خواهند گرفت.

اکنون محافظت حلقه اصلی را با شروع از آخرین رقم آغاز می‌کند. حلقه تنها اعـــداد از ۱۰ تـا ۳ را بکار میبرد. این حلقه‌ایست که بدون بکارگیری اولین و دومین عدد پایان می‌یابد. در این حلقـه چـه اتفاقی می‌افتد؟ ... محافظت حلقه را با جایگذاری عدد از Regstring آغاز می‌کند.

این مجموعه بصورت «اشاره گر جستجو» برای پیدا کردن پـارامتری در داخـل جـدول پارامترهـای حافظه استفاده می‌شود و اعداد بین ۹ و ۰ هستند. اجازه دهید این مقدار را lkup val بنامیم.

محافظت، داده‌ها را در 1EA7 جستجو می‌کند. در مورد ما:

[Lkup _ val=9 * 0XA+4; 0X5A+4=0X5E

بنابراین در خط AL-4C80 بایت را در 1EA7:005E قرار خواهد داد. در مورد مـا KEY_PAR در 1EA7:005E آن عدد «۷» است و به 0X32=2 اضافه می‌شود و ۹ بدست می‌اید.

اجازه دهید ثـــابت کنیـم کـه KEY-PAR می‌توانـد بدسـت آیـد. مـاکزیمم 0X63 و مینیمـم 0X0 می‌باشد.

بنابراین KEY_PARهای ممکن در حافظه بین 1EA7 ,1EA7:0063 هستند، اجازه دهید به جدول تناسب در حافظه نگاهی بیندازیم جاییکه این KEY_PAR ذخیره میشوند.

```
1EA7:0000 01 03 03 01 09 02 03 00-09 00 04 03 08 07 04 04
1EA7:0010 05 02 09 00 02 04 01 05-06 06 03 02 00 08 05 06
1EA7:0020 08 09 05 00 04 06 07 07-02 00 08 00 06 02 04 07
1EA7:0030 04 04 09 05 09 06 00 06-08 07 00 03 05 09 00 08
1EA7:0040 03 07 07 06 08 09 01 05-07 04 06 01 04 02 07 01
1EA7:0050 03 01 08 01 05 03 03 01-02 08 02 01 06 05 07 02
1EA7:0060 05 09 09 08 02 09 03 00-00 04 05 01 01 03 08 06
1EA7:0070 01 01 09 00 02 05 05 05-01 07 01 05 08 07 01 09
1EA7:0080 08 07 07 04 04 08 03 00-06 01 09 08 08 04 09 09
1EA7:0090 00 07 05 02 03 01 03 08-06 05 07 06 03 07 06 07
1EA7:00A0 04 02 02 05 02 04 06 02-06 09 09 01 05 02 03 04
1EA7:00B0 04 00 03 05 00 03 08 07-06 04 08 08 02 00 03 06
1EA7:00C0 09 00 00 06 09 04 07 02-00 01 01 01 01 00 01 FF
1EA7:00D0 00 FF FF FF FF 00 FF 01-00 00 00 00 00 00 00 00
```

یک جدول جالب، جاییکه تمام تطابقها بین ۰ و ۹ ، □هستند و آیا عددی سری در اینجا بدست می آوریم؟

اما بجای بایتهای 0-0X63 در اینجا DOUBLE داریم: بایتهای 0-0XC8 (توالی ۰۱ از CA شروع می‌شود) بعدها خواهیم دید که □ چقدر مهم است، در یک لحظه درک خواهید کرد چیزی باید با جدول پیش رود.

همانطور که گ‌فتم حاصل KEY_PAR به اضافه عدد ورودی با یک FFDB صیقل داده می‌شود. بنابراین حاصل، عدد ورودی به اضافه KEY_PAR خواهد بود که در مثال ما ۹ می‌باشد. اکنون 0X در [9548],RE_SULT در [9546] ذخیره خواهد شد.

اکنون محافظت، برای سیر R0 آماده می‌شود، اشاره گر را مجدداً راه اندازی می‌کند و CX و BX را از [958C],[958A] شارژ می‌کند و AX را با 0XA شارژ کرده و DX را مساوی صفر قرار میدهد. در این روش،عملیات های مختلفی روی AX و DX انجام می‌شود و نتایج در مکانهای فوق‌الذکر [958A],[958C] ذخیره می‌شود.

اکنون KEY_PAR و RE_SULT به ترتیب به DX و AX اضافه میشوند. ما مجدداً، RO را بدست آورده و یکبار دیگر آن را در دو مکان آخر ذخیره می‌کنیم: AX+RE_SULT در ADX_KEY_PAR, [958A] در [958C]

اکنون ۱ را از مقـدار موجـود در SEC+3 کـم می‌کنیـم. آن یـک پـارامتر اسـلایدری اسـت کـه در محافظتهای پیچیده برای بوجود آوردن تأثیری اتفاقی در نـــاظر اسـتفاده می‌شـود. در طـرف دیـگر، مقدار FIR_+7 به اضافه ۱ می‌شود و از ٤ به ٥ میرسد.

اکنون حفاظت عدد بعدی ورودی شما را برای حلقه بکار میبرد . در مورد مثـال مـا، ایـن حلقـه از پیکربندی حفاظتی ذیل را پارامترهای اسلایدینگ استفاده می‌کند:

```
Input #  pamp_2nd   pamp_1st   Lookup value  KEY_PAR  # RE_SULT
# 10 = 2, SEC_+3= 9, FIR_+7= 4, Lkup_val = 0x5E, KEY=7 +2 = 9
# 9  = 1, SEC_+3= 8, FIR_+7= 5, Lkup_val = 0x55, KEY=3 +1 = 4
# 8  = 2, SEC_+3= 7, FIR_+7= 6, Lkup_val = 0x4C, KEY=4 +2 = 6
# 7  = 1, SEC_+3= 6, FIR_+7= 7, Lkup_val = 0x43, KEY=7 +1 = 7
# 6  = 2, SEC_+3= 5, FIR_+7= 8, Lkup_val = 0x3A, KEY=0 +2 = 2
# 5  = 1, SEC_+3= 4, FIR_+7= 9, Lkup_val = 0x31, KEY=4 +1 = 5
# 4  = 2, SEC_+3= 3, FIR_+7= 0, Lkup_val = 0x1E, KEY=5 +2 = 7
# 3  = 4, SEC_+3= 2, FIR_+7= 1, Lkup_val = 0x15, KEY=2 +4 = 5
```

توجه داشته باشـید کـه چگونـه ورودی منظـم 21212124 ورودی نـامنظم 94672575 را میدهـد. ممکن است از خودتان بپرسید معنی این کار چیست. این RE_SULT ها(نتایج) برای چه منظـوری استفاده میشوند؟ آنها برای اسلاید کردن پارامتر دیگری مورد استفاده قـرار میگیرنـد، ایـن یکـی در داخل روتین.... این چیزیست که برای AX و DX در داخل مسیر اتفاق می‌افتد:

```
:4CBF 26895702   MOV    ES:[BX+02],DX ; save R_DX par  [958C]
:4CC3 268907     MOV    ES:[BX],AX ; save R_AX par     [958A]
:4CC6 0346FA     ADD    AX,[BP-06] ; add to AX RE_SULT [9546]
:4CC9 1356FC     ADC    DX,[BP-04] ; add to DX KEY_PAR [9548]
:4CCC C45E0E     LES    BX,[BP+0E] ; reset pointer to E
:4CCF 26895702   MOV    ES:[BX+02],DX ; save R_DX+KEY_PAR [958C]
:4CD3 268907     MOV    ES:[BX],AX ; save R_AX+RE_SULT    [958A]
```

	:4CC6	:4CC9	:4CCF Odd_DX	:4CD3 slider_sum
RE_SULT	[958A]	[958C]	[958C]	[958A]
0	0	0	0	0
9	5A	0	0	9
4	3AC	0	0	5E
6	24F4	0	0	3B2
7	71CE	1	1	24FB
2	7220	4	E	71D0
5	7572	4	90	7225
				7579

حال انتهای حلقه‌ها اعداد ورودی را از دهم تا سوم بکار میبرند. محافظت دومین عدد را بـارگذاری کرده و آن را در ۱۰ ضرب می‌کند که در مثال ما 0XA=14 *2 می‌باشد . محـافظت اولیـن عـدد را جایگذاری کرده و آن را به ضرب اضافه می‌کند که در مثال ما 1+0X14=0X15 می‌باشد.

اکنون هر چیزی می‌تواند به FFD0 اضافه شـــود. اشـاره گر در انتـهای عـدد ورودی قـرار خواهـد گرفت. DX که با CDW صفر می‌شود بصـورت پـارامتری در [9584] ذخـیره می‌شـود و مجمـوع بدست آمده در [9582] ذخیره می‌شود.

FLAG در مکان درستی قرار گرفته و این روتین پایان می‌یابد. هیچ پـــارامتری انتقـال داده نشـده و تنها چیز جالب چیزیست که در آدرسهای [9582],[9584],[958A],[958C] اتفـــاق می‌افتـد یعنـی FINALSUM، Slider sum، Odd dx

در درس بعدی هرچیزی را کراک خواهیم کرد. اما به شما توصیـه‌ای هـم میکنـم، در مـورد کـراک کردن شما دوست دارید روش خود را از سر بگیرید: و خواهید دید که چگونـه طرحـهای اسـتفاده شده برای (ثبت) سومین عدد، شباهتها و تفاوتهایی را با طرحهایی که مـا دراینجـا بـرای عـدد اول مطالعه کرده‌ایم نشان میدهند.

رشته ورودی 34-3434-3434-3434-3434 بـــرای ثبـت عـدد در رشـته 1415933848415474 31 تغییر خواهد یافت اما آن بکار نخواهد آمد زیرا دوازدهمین رقم «۱» با باقیمـــانده محاسـبه شـده در این کنترل از طریق آدرسهای قبلی کنترلهای دیگر تطبیق نخواهد کرد.

در اینجا چیزهای پیچیده‌تری وجود دارد زیرا هر تغییر ناچیزی در رشته ورودی شما رشته دیـگر را کاملاً تغییر میدهد.

بنابراین شما بـاید 34-3434-3434-3434-3434 را بـه 96-3434-3434-3434-7434 تغیـیر دهیـد.

رشته 2197029609744980 56 که این ورودی به ما میدهد از این صافی حفاظتی عبور خواهد کرد و ما تنها قادر خواهیم بود قدم به قدم پیش رفته و سرانجام کل محافظت را بکار گیریم.