hackerearth

Penetration Testing Report

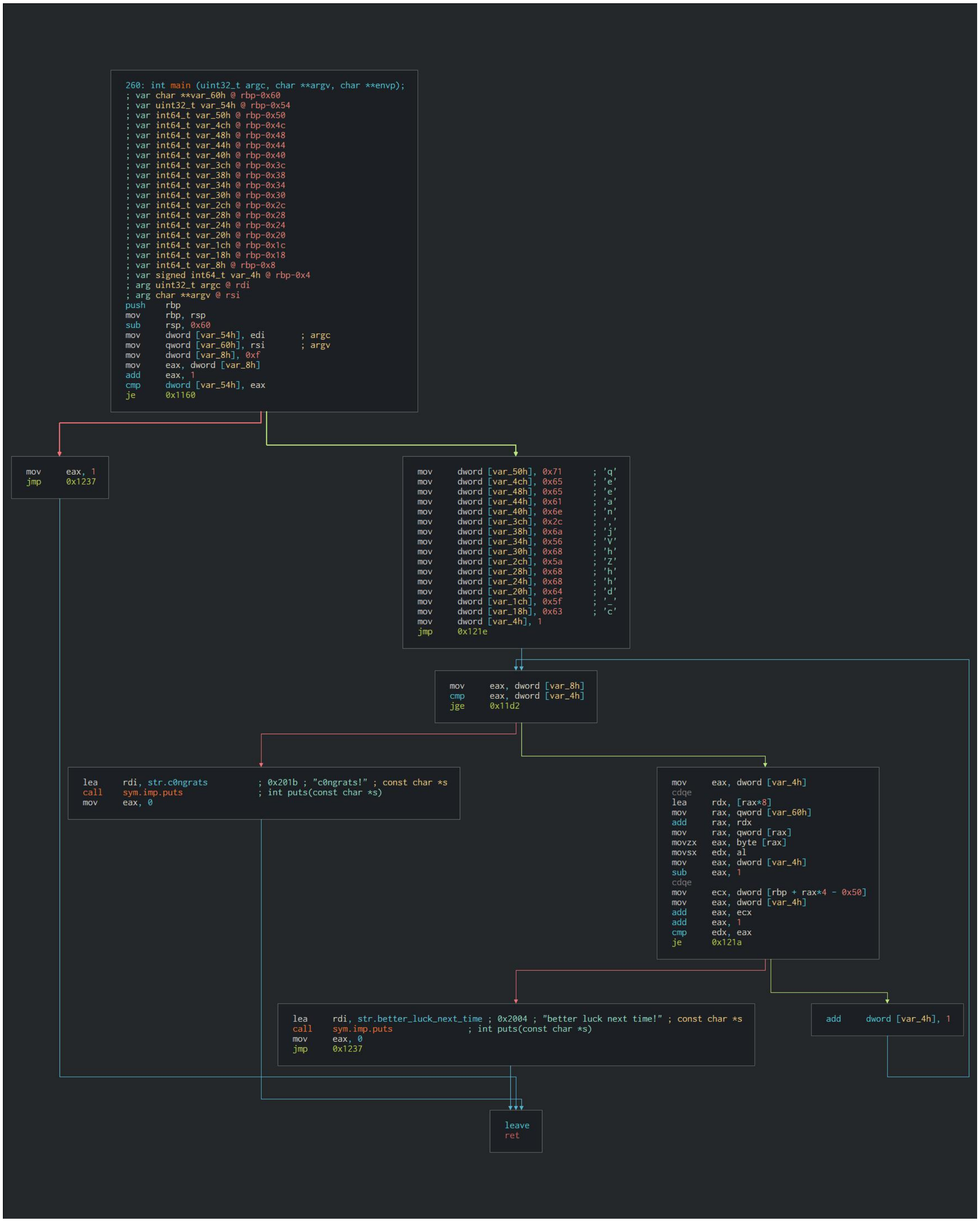
For

“Shifter2”

S.NO.	Title	#
1.	Challenge Category	Reverse Engineering
2.	Challenge Related Files	shifter2.png
3.	File Link / Target IP	N/A

PROCEDURE

1. Analyzing the graph.



```
260: int main (uint32_t argc, char **argv, char **envp);
; var char **var_60h @ rbp-0x60
; var uint32_t var_54h @ rbp-0x54
; var int64_t var_50h @ rbp-0x50
; var int64_t var_4ch @ rbp-0x4c
; var int64_t var_48h @ rbp-0x48
; var int64_t var_44h @ rbp-0x44
; var int64_t var_40h @ rbp-0x40
; var int64_t var_3ch @ rbp-0x3c
; var int64_t var_38h @ rbp-0x38
; var int64_t var_34h @ rbp-0x34
; var int64_t var_30h @ rbp-0x30
; var int64_t var_2ch @ rbp-0x2c
; var int64_t var_28h @ rbp-0x28
; var int64_t var_24h @ rbp-0x24
; var int64_t var_20h @ rbp-0x20
; var int64_t var_1ch @ rbp-0x1c
; var int64_t var_18h @ rbp-0x18
; var int64_t var_8h @ rbp-0x8
; var signed int64_t var_4h @ rbp-0x4
; arg uint32_t argc @ rdi
; arg char **argv @ rsi
push    rbp
mov     rbp, rsp
sub     rsp, 0x60
mov     dword [var_54h], edi       ; argc
mov     qword [var_60h], rsi       ; argv
mov     dword [var_8h], 0xf
mov     eax, dword [var_8h]
add     eax, 1
cmp     dword [var_54h], eax
je      0x1160
```

```
mov     eax, 1
jmp     0x1237
```

```
mov     dword [var_50h], 0x71      ; 'q'
mov     dword [var_4ch], 0x65      ; 'e'
mov     dword [var_48h], 0x65      ; 'e'
mov     dword [var_44h], 0x61      ; 'a'
mov     dword [var_40h], 0x6e      ; 'n'
mov     dword [var_3ch], 0x2c      ; ','
mov     dword [var_38h], 0x6a      ; 'j'
mov     dword [var_34h], 0x56      ; 'V'
mov     dword [var_30h], 0x68      ; 'h'
mov     dword [var_2ch], 0x5a      ; 'Z'
mov     dword [var_28h], 0x68      ; 'h'
mov     dword [var_24h], 0x68      ; 'h'
mov     dword [var_20h], 0x64      ; 'd'
mov     dword [var_1ch], 0x5f      ; '_'
mov     dword [var_18h], 0x63      ; 'c'
mov     dword [var_4h], 1
jmp     0x121e
```

```
mov     eax, dword [var_8h]
cmp     eax, dword [var_4h]
jge     0x11d2
```

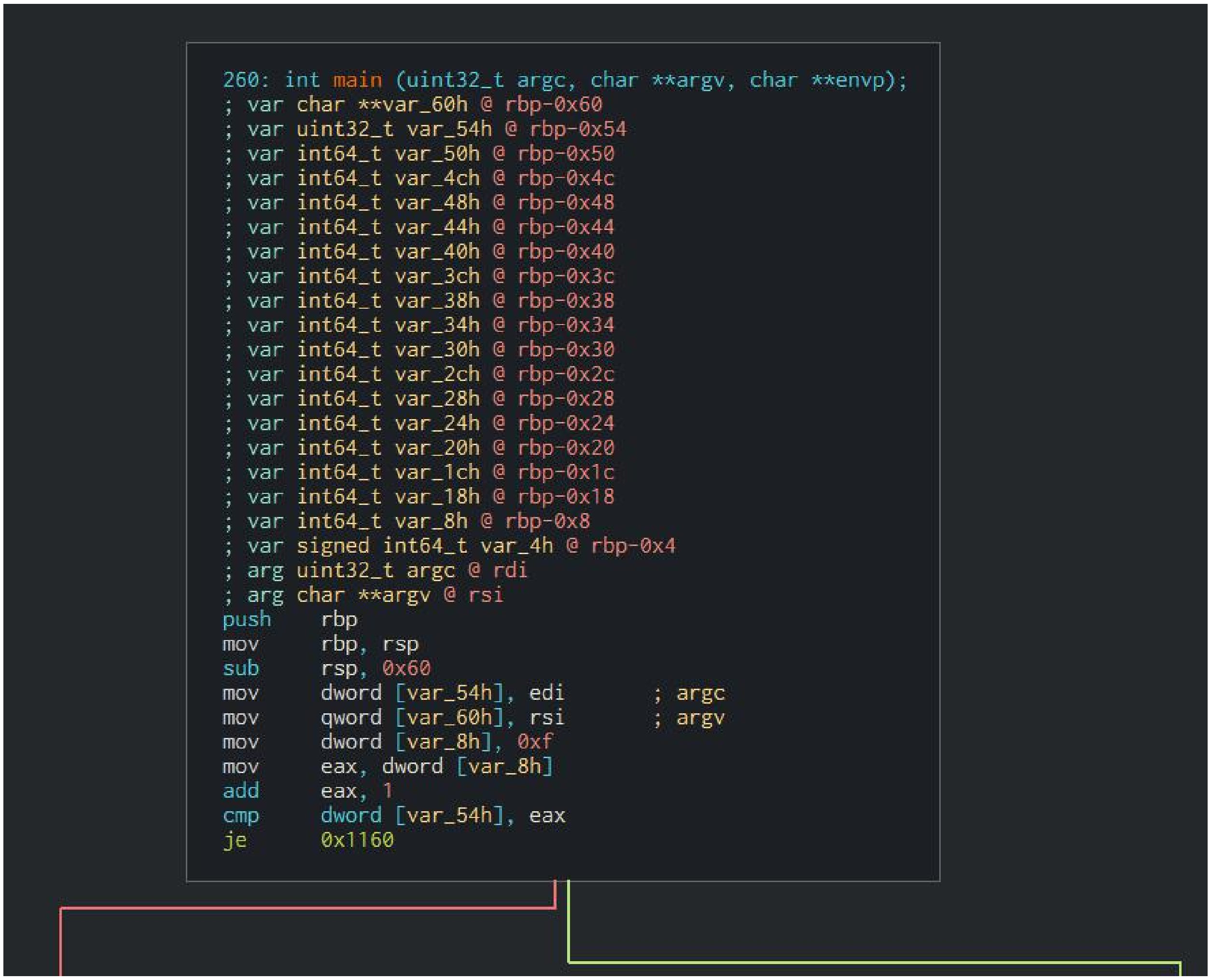
```
lea     rdi, str.c0ngrats            ; 0x201b ; "c0ngrats!" ; const char *s
call    sym.imp.puts                 ; int puts(const char *s)
mov     eax, 0
```

```
mov     eax, dword [var_4h]
cdqe
lea     rdx, [rax*8]
mov     rax, qword [var_60h]
add     rax, rdx
mov     rax, qword [rax]
movzx   eax, byte [rax]
movsx   edx, al
mov     eax, dword [var_4h]
sub     eax, 1
cdqe
mov     ecx, dword [rbp + rax*4 - 0x50]
mov     eax, dword [var_4h]
add     eax, ecx
add     eax, 1
cmp     edx, eax
je      0x121a
```

```
lea     rdi, str.better_luck_next_time ; 0x2004 ; "better luck next time!" ; const char *s
call    sym.imp.puts                 ; int puts(const char *s)
mov     eax, 0
jmp     0x1237
```

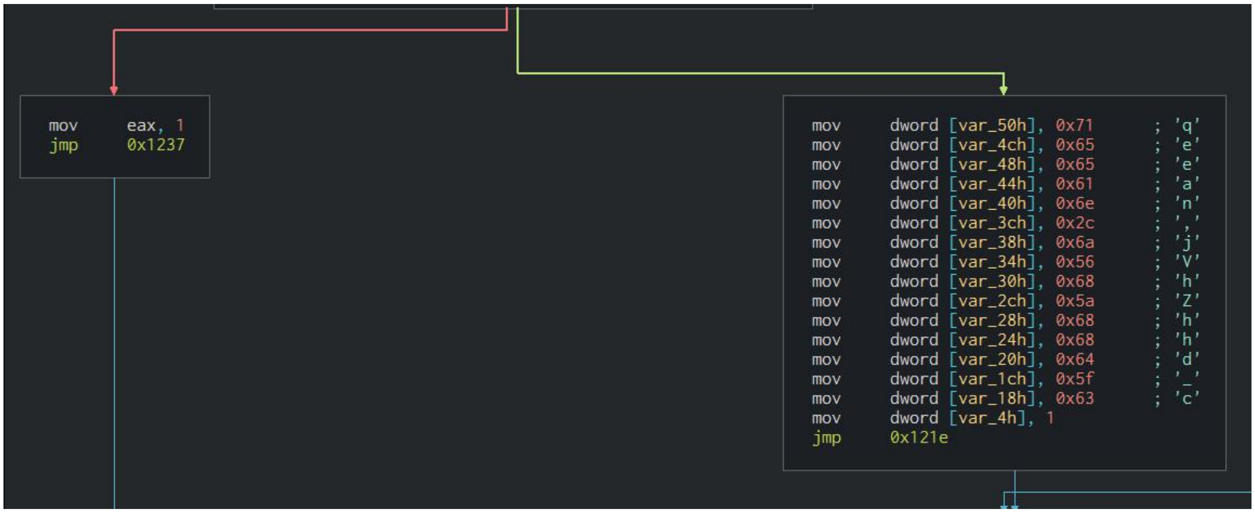
```
add     dword [var_4h], 1
```

```
leave
ret
```



```
260: int main (uint32_t argc, char **argv, char **envp);
; var char **var_60h @ rbp-0x60
; var uint32_t var_54h @ rbp-0x54
; var int64_t var_50h @ rbp-0x50
; var int64_t var_4ch @ rbp-0x4c
; var int64_t var_48h @ rbp-0x48
; var int64_t var_44h @ rbp-0x44
; var int64_t var_40h @ rbp-0x40
; var int64_t var_3ch @ rbp-0x3c
; var int64_t var_38h @ rbp-0x38
; var int64_t var_34h @ rbp-0x34
; var int64_t var_30h @ rbp-0x30
; var int64_t var_2ch @ rbp-0x2c
; var int64_t var_28h @ rbp-0x28
; var int64_t var_24h @ rbp-0x24
; var int64_t var_20h @ rbp-0x20
; var int64_t var_1ch @ rbp-0x1c
; var int64_t var_18h @ rbp-0x18
; var int64_t var_8h @ rbp-0x8
; var signed int64_t var_4h @ rbp-0x4
; arg uint32_t argc @ rdi
; arg char **argv @ rsi
push    rbp
mov     rbp, rsp
sub     rsp, 0x60
mov     dword [var_54h], edi       ; argc
mov     qword [var_60h], rsi       ; argv
mov     dword [var_8h], 0xf
mov     eax, dword [var_8h]
add     eax, 1
cmp     dword [var_54h], eax
je      0x1160
```

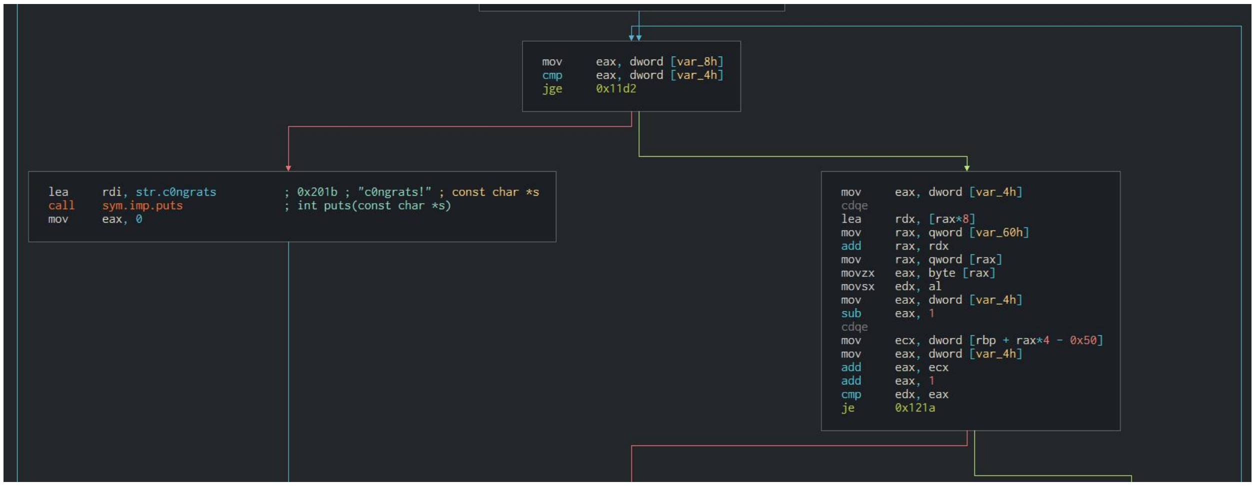
This section basically check if the number of arguments passed is equal to 16 or not.



```
mov     eax, 1
jmp     0x1237
```

```
mov     dword [var_50h], 0x71       ; 'q'
mov     dword [var_4ch], 0x65       ; 'e'
mov     dword [var_48h], 0x65       ; 'e'
mov     dword [var_44h], 0x61       ; 'a'
mov     dword [var_40h], 0x6e       ; 'n'
mov     dword [var_3ch], 0x2c       ; ','
mov     dword [var_38h], 0x6a       ; 'j'
mov     dword [var_34h], 0x56       ; 'V'
mov     dword [var_30h], 0x68       ; 'h'
mov     dword [var_2ch], 0x5a       ; 'Z'
mov     dword [var_28h], 0x68       ; 'h'
mov     dword [var_24h], 0x68       ; 'h'
mov     dword [var_20h], 0x64       ; 'd'
mov     dword [var_1ch], 0x5f       ; '_'
mov     dword [var_18h], 0x63       ; 'c'
mov     dword [var_4h], 1
jmp     0x121e
```

If the number of arguments is equal to 16 then 15 unique values are being stored at 15 different address locations with an offset of 4 bytes. So it could be an integer array with hexadecimal values - 0x71, 0x65, 0x65, 0x61, 0x6e, 0x2c, 0x6a, 0x56, 0x68, 0x5a, 0x68, 0x68, 0x64, 0x5f, 0x63. Also a counter variable is being initialized with integer value 1.

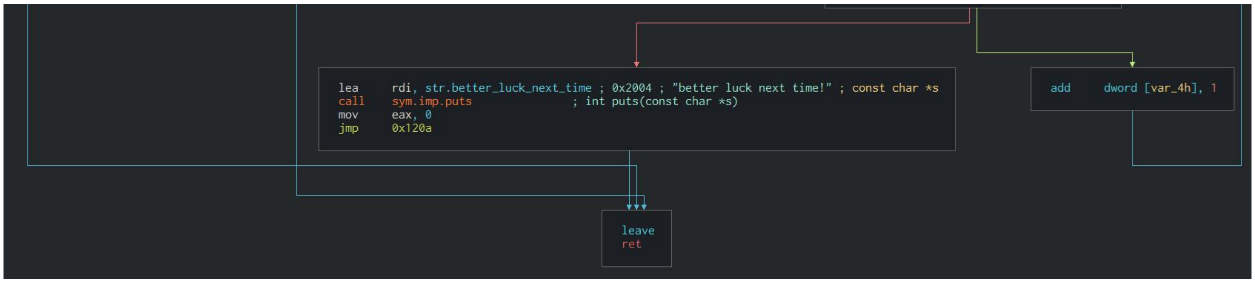


```
mov     eax, dword [var_8h]
cmp     eax, dword [var_4h]
jge     0x11d2
```

```
lea     rdi, str.c0ngrats           ; 0x201b ; "c0ngrats!" ; const char *s
call    sym.imp.puts                ; int puts(const char *s)
mov     eax, 0
```

```
mov     eax, dword [var_4h]
cdqe
lea     rdx, [rax*8]
mov     rax, qword [var_60h]
add     rax, rdx
mov     rax, qword [rax]
movzx   eax, byte [rax]
movsx   edx, al
mov     eax, dword [var_4h]
sub     eax, 1
cdqe
mov     ecx, dword [rbp + rax*4 - 0x50]
mov     eax, dword [var_4h]
add     eax, ecx
add     eax, 1
cmp     edx, eax
je      0x121a
```

The counter variable is now being checked if it is less than 16. If it is not, then “c0ngrats!” message is printed otherwise each 15 argument is being checked against the hex values stored in the array which has been increased by one more than the number of bytes stored in counter & then the counter is being increased by one. This suggests that a loop is being run which starts iterating arguments passed which are further checked as stated above.



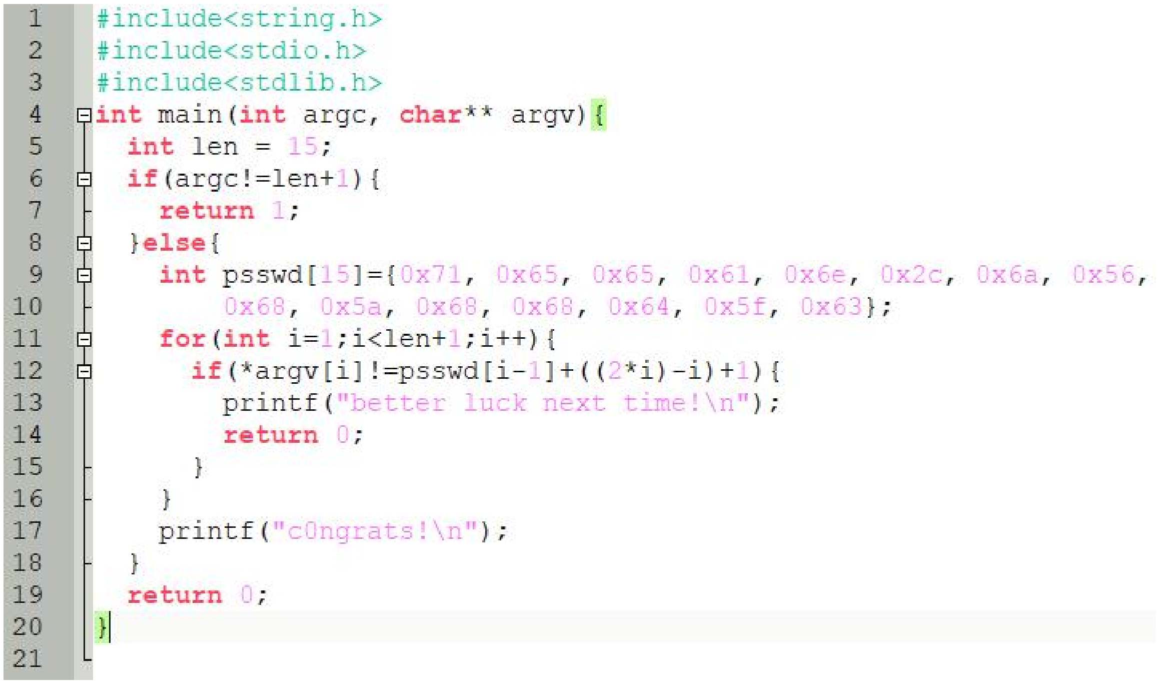
```
lea     rdi, str.better_luck_next_time ; 0x2004 ; "better luck next time!" ; const char *s
call    sym.imp.puts                   ; int puts(const char *s)
mov     eax, 0
jmp     0x120a
```

```
add     dword [var_4h], 1
```

```
leave
ret
```

If any of the argument comparison comes out to be false then “better luck next time” is being printed which is then being followed by termination of the program & if wrong number of argument is being supplied, then the program just terminated.

2. Implementing the algorithm in C.



```
#include<string.h>
#include<stdio.h>
#include<stdlib.h>
int main(int argc, char** argv){
  int len = 15;
  if(argc!=len+1){
    return 1;
  }else{
    int psswd[15]={0x71, 0x65, 0x65, 0x61, 0x6e, 0x2c, 0x6a, 0x56,
        0x68, 0x5a, 0x68, 0x68, 0x64, 0x5f, 0x63};
    for(int i=1;i<len+1;i++){
      if(*argv[i]!=psswd[i-1]+((2*i)-i)+1){
        printf("better luck next time!\n");
        return 0;
      }
    }
    printf("c0ngrats!\n");
  }
  return 0;
}
```

3. Reversing the algorithm & finding out the hex values against which the arguments will be further compared.

0x71 + 0x2 = 0x73

0x65 + 0x3 = 0x68

0x65 + 0x4 = 0x69

0x61 + 0x5 = 0x66

0x6e + 0x6 = 0x74

0x2c + 0x7 = 0x33

0x6a + 0x8 = 0x72

0x56 + 0x9 = 0x5f

0x68 + 0xa = 0x72

0x5a + 0xb = 0x65

0x68 + 0xc = 0x74

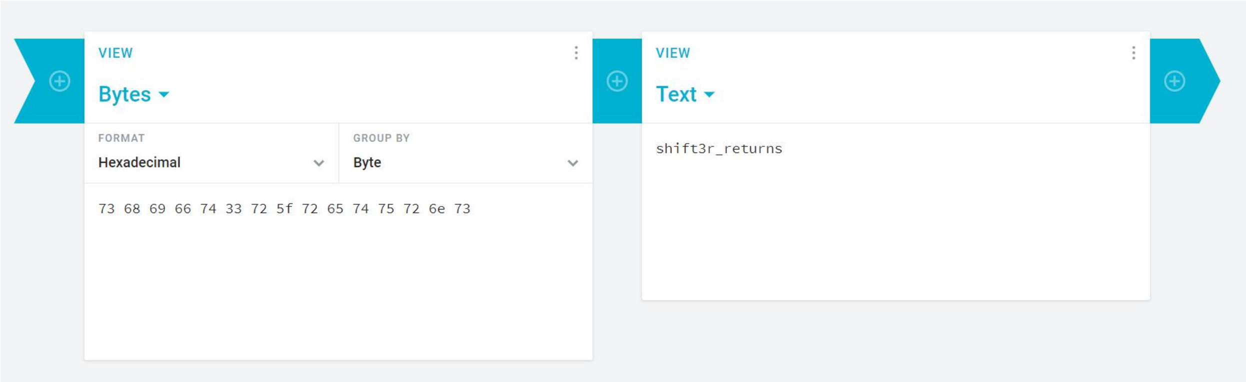
0x68 + 0xd = 0x75

0x64 + 0xe = 0x72

0x5f + 0xf   = 0x6e

0x63+ 0x10 = 0x73

4. Converting the hexadecimal values to text.



FORMAT Hexadecimal

GROUP BY Byte

```
73 68 69 66 74 33 72 5f 72 65 74 75 72 6e 73
```

```
shift3r_returns
```

Flags:

S.No.	**Flag - No.**	**Flag**
1.	Flag 1	HE{shift3r_returns}