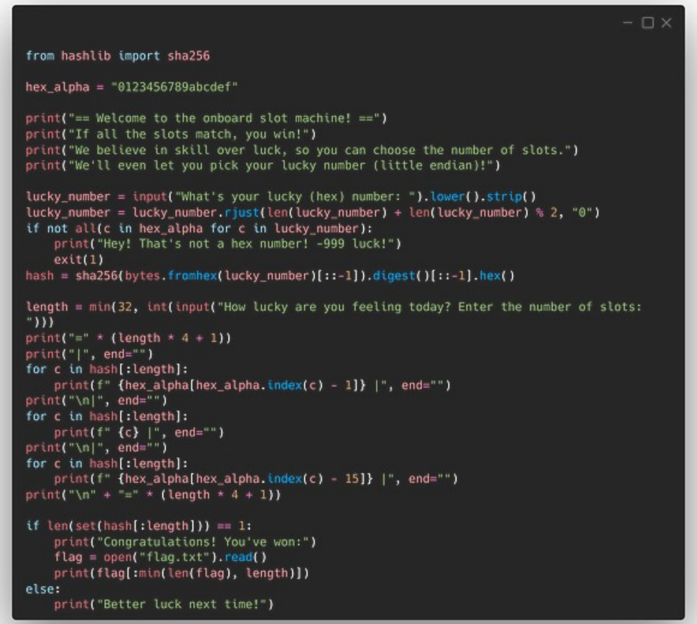
[UIUCTF 2024] Slot Machine WriteUp - MEHLOUL MOHAMED - Medium

MEHLOUL MOHAMED

`lsblk`

In this post, I aim to explain my solution for the “Slot Machine” challenge from UIUCTF 2024’s “Miscellaneous” category.

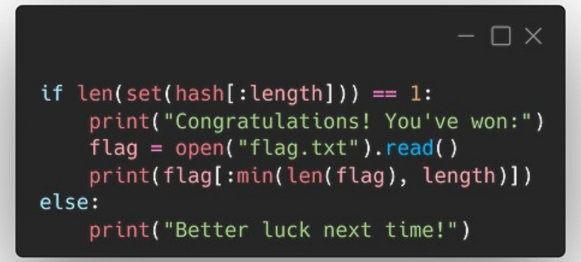
We’re given the following python script:



```python
from hashlib import sha256

hex_alpha = "0123456789abcdef"

print("== Welcome to the onboard slot machine! ==")
print("If all the slots match, you win!")
print("We believe in skill over luck, so you can choose the number of slots.")
print("We'll even let you pick your lucky number (little endian)!")

lucky_number = input("What's your lucky (hex) number: ").lower().strip()
lucky_number = lucky_number.rjust(len(lucky_number) + len(lucky_number) % 2, "0")
if not all(c in hex_alpha for c in lucky_number):
    print("Hey! That's not a hex number! -999 luck!")
    exit(1)
hash = sha256(bytes.fromhex(lucky_number)[::-1]).digest()[::-1].hex()

length = min(32, int(input("How lucky are you feeling today? Enter the number of slots:
")))
print("=" * (length * 4 + 1))
print("|", end="")
for c in hash[:length]:
    print(f" {hex_alpha[hex_alpha.index(c) - 1]} |", end="")
print("\n|", end="")
for c in hash[:length]:
    print(f" {c} |", end="")
print("\n|", end="")
for c in hash[:length]:
    print(f" {hex_alpha[hex_alpha.index(c) - 15]} |", end="")
print("\n" + "=" * (length * 4 + 1))

if len(set(hash[:length])) == 1:
    print("Congratulations! You've won:")
    flag = open("flag.txt").read()
    print(flag[:min(len(flag), length)])
else:
    print("Better luck next time!")
```

The script asks us for a “lucky number” in little-endian hexadecimal. Then it proceeds to ask for a number of slots.

Reading through the code shows us our winning condition:



```python
if len(set(hash[:length])) == 1:
    print("Congratulations! You've won:")
    flag = open("flag.txt").read()
    print(flag[:min(len(flag), length)])
else:
    print("Better luck next time!")
```

The `hash` variable represents the hash of our “lucky number”, while the `length` variable represents the number of slots we chose.

Note: The endianness of the “lucky number” will be swapped before calculating the hash (we are instructed to input a little-endian hex number). The endianness of the generated hash will also be swapped. More on that later.

I don't get it what does this mean?

Great question! Essentially, the process involves checking the first [`length`] digits of `hash` , then using the `set()` function to create a set of those digits. After that, it verifies the size of the set. If the size is `1`, it congratulates us and grants us some characters from the flag. The number of characters given is the minimum between the length of the flag and the number of slots provided(i.e the value of `length`).

So what's the plan

I only have one idea, which is to find an input that produces a hash starting with as many consecutive repeated hex digits as possible. By doing this, if we set the number of slots to be equal to the number of those digits, the set size will be `1`.

Where would we find that? You can't just make up a hash and find the input that generates it!

You're right, but the cryptocurrency community has been calculating hashes for years. Surely they have something that could help us.
A quick search for "smallest sha256 hash" points us to the Bitcoin block
`756951` which has a hash of
`0000000000000000000000005d6f06154c8685146aa7bc3dc9843876c9cefd0f`
A whopping 24 zeros!

Again, don't get too excited. This is only a hash. How will you find the input that generated it?

Well, I did some digging and found out how Bitcoin block hashes are calculated. It seems to work like this:
It is the double `sha256` (`sha256` applied two times) of the following:

version || previous block hash || MerkleRoot ||timestamp || Bits || Nonce

Each value is a little-endian hexadecimal number.
Note: The `||` simply means concatenation.

Where are we going to find all of these?

It's public information. For the block we are interested in, the values are:

Version: `0x20400000`
Previous block hash: `0x0000000000000000000050da0da9451c2e1306db4ddb5acc965fc1016678d9154`
MerkleRoot: `0x62c46f1efadf6e39b7463e5362bb552cba98f74a80a58378ff5194c7b058005a`

Timestamp: `1664846893`

Bits: `0x1708f9ae`

Nonce: `3240300428`

But some of these values aren't in little-endian representation, and some of them aren't even in hex (timestamp and nonce).

Let's fix that:

Version: `0x00004020`

Previous Block Hash: `0x54918d671610fc65c9acb5ddb46d30e1c25194daa00d05000000000000000000`

MerkleRoot: `0x5a0058b0c79451ff7883a5804af798ba2c55bb62533e46b7396edffa1e6fc462`

Timestamp: `0x2d8c3b63`

Bits: `0xaef90817`

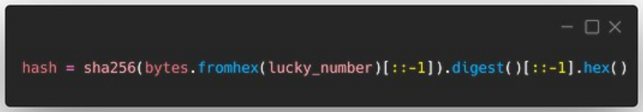
Nonce: `0x8c0f23c1`

Now let's put it all together:

```
0000402054918d671610fc65c9acb5ddb46d30e1c25194daa00d050000000000000000005a0058b0c79451ff7883a5804af798ba2c55bb62533e46b7396edffa1e6fc4622d8c3b63aef908178c0f23c1
```

I hashed this twice and it gave me the hash we're looking for, but it's inverted?

No, it isn't inverted. It's just that the endianness is swapped. We don't need to worry about that since the challenge script swapping the endianness.



```
hash = sha256(bytes.fromhex(lucky_number)[::-1]).digest()[::-1].hex()
```

Note the [::-1]

It's swapping the endianness of the number we give it and then hashing it. then swapping the endianness of the generated hash

So we need to hash that big hex number we got from concatinating all those values. Convert the hash to little-endian and give it to the script, right?

Exactly, you're a pretty fast learner imaginary reader!

Thanks!

So the sha256 hash of:

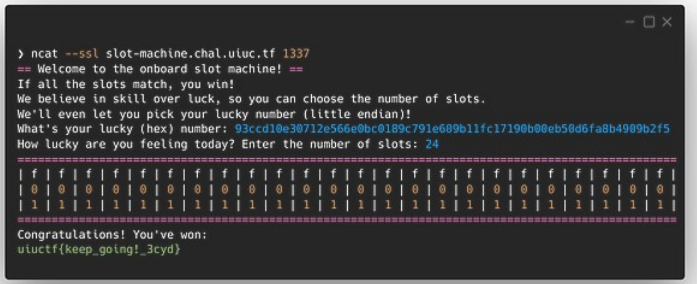
```
0000402054918d671610fc65c9acb5ddb46d30e1c25194daa00d050000000000000000005a0058b0c79451ff7883a5804af798ba2c55bb62533e46b7396edffa1e6fc4622d8c3b63aef908178c0f23c1
```

is:

```
f5b209498bfad650eb000b1917fc119b601e799c18c00b6e562e71300ed1cc93
```

And in little-endian it would be:
```
93ccd10e30712e566e0bc0189c791e609b11fc17190b00eb50d6fa8b4909b2f5
```

Now we simply input it into the challenge as our "lucky number," choosing 24 as the number of slots (because its hash starts with 24 zeros) and see if that's enough to get the flag.



```
❯ ncat --ssl slot-machine.chal.uiuc.tf 1337
== Welcome to the onboard slot machine! ==
If all the slots match, you win!
We believe in skill over luck, so you can choose the number of slots.
We'll even let you pick your lucky number (little endian)!
What's your lucky (hex) number: 93ccd10e30712e566e0bc0189c791e609b11fc17190b00eb50d6fa8b4909b2f5
How lucky are you feeling today? Enter the number of slots: 24
=================================================================================================
| f | f | f | f | f | f | f | f | f | f | f | f | f | f | f | f | f | f | f | f | f | f | f | f |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
=================================================================================================
Congratulations! You've won:
uiuctf{keep_going!_3cyd}
```

And would you look at that? Here’s our flag!

GG!

Hope you enjoyed this write-up and learned something new.