A0217609B

Assignment 1 Writeup

Easy Challenges

E.0 Sanity Check

Go to last page of the pdf file

E.1 Rivest–Shamir–Adleman

We have c,p,q and e so we need d and n in order to get back m. First we derive phi as (p – 1)*(q – 1). Then using formula: $d = e^{-1} \bmod \Phi(n)$, we get d. I used a script taken from stackoverflow to calculate d. n is derived using p*q. To get back m, we follow $m = c^d \bmod n$ and since the values are large, we need an optimum algorithm. However, m is just another long number and needs to be decrypted. To decrypt m character by character, I iteratively mod m by 256 and then divide m by 256 until m is 0. Then I reversed the output in order to get the flag. I found out the way through this helpful link about the RSA algorithm.

Script used:

```
c= … #abbreviated for brevity
p= …
q= …
e= 65537
phi = (p - 1) * (q - 1)
def inverse(a, n): #extended culidean algo
    t, newt = 0, 1
    r, newr = n, a
    while newr:
        quotient = r // newr  # floor division
        t, newt = newt, t - quotient * newt
        r, newr = newr, r - quotient * newr
    if r > 1:
        return None  # there's no solution
    if t < 0:
        t = t + n
```

```
    return t
d = inverse(e,phi)
n = p * q
def pow_h(base, power, modulus):
    a = 1
    while power:
        power, d = power // 2, power % 2
        if d:
            a = a * base % modulus
        base = base * base % modulus
    return a
c = pow_h(c, d, n)
def decrypt(c):
    s = ""
    while (c > 0):
        s = chr(c%256)+s
        c=c//256
    return s
print(decrypt(c))
```

Flag: CS2107{pl3ase_$3cure_ur_p_q_RS@}

E.2 xor_secure

We have retrieved the 4 keys used in encrypting as well as the cipher. Using the property of xor where if c = m xor k then m = c xor k, we get back m using m = c xor k1 xor k2 xor k3 xor k4.

Script used:

```
key1 = …
key2 = …
key3 = …
key4 = …
enc_msg = …
def encrypt(msg, key):
    return bytes(a ^ b for a, b in zip(msg, key))
```

```
def xor_secure(keys, msg):
    result = msg
    for key in keys:
        result = encrypt(result, key)
    return result
enc_msg = xor_secure([key1,key2,key3,key4],enc_msg)
msg = enc_msg.decode()
print(msg)
```

Flag: CS2107{M4St3R_0f_aNc13nT_x0R_t3CHn1qu3s!!!!}

E.3 Hash Browns

We have retrieved a wall of hashed values using SHA1. On inspection, I realized that some of the hashes were repeated so I guessed that it must be alphanumeric characters being hashed. Using a script to apply SHA1 hash function on every punctuation and alphanumeric character to obtain their hashes and then comparing them with the encrypted text, I manage to find the flag.

Script used:

```
def getAll():
    for char in range(ord('a'), ord('z') + 1):
        s = str(chr(char))
        print(s)
        hsh = hashlib.sha1(s.encode())
        print(hsh.hexdigest())
    for char in range(ord('0'),ord('9') + 1):
        s = str(chr(char))
        print(s)
        hsh = hashlib.sha1(s.encode())
        print(hsh.hexdigest())
    for char in range(ord('A'),ord('Z') + 1):
        s = str(chr(char))
        print(s)
        hsh = hashlib.sha1(s.encode())
        print(hsh.hexdigest())
getAll()
```

```
punctuation = list(string.punctuation)
for p in punctuation:
    print(p)
    hsh = hashlib.sha1(p.encode())
    print(hsh.hexdigest())
```

Flag: CS2107{hash_br0wn$_cr@ck3rs}

E.4 Caesar with a capital C

This can be easily done using an already available website designed to crack Caesar ciphers. Simply shift each character back by 5.

Flag: CS2107{345y_p345y_l3m0n_5qu33zy}

Medium Challenges

M.2 Baby Shark

Using wireshark's export objects dialog box, I manage to get a bunch of different files sent over http. The first part of the flag is found in the wallpaper. Second part is found in the excel file. Third part is in the config html file. Fourth and final part is in the pdf file obtained by highlighting the invisible text.

Flag: CS2107{b@by_sh@aRk_d00_dOo_143192}

M.4 Salad

The code was encrypted using complex procedure that applies different sets of shifts on groups of characters based on their ascii values. I wrote a script to reverse the shifts.

Script used:

```
cipher = '*s<;:{7M>?T=RY:FYS"B?TI{"{I:NY=NJ:&Y"rYS>L6D~~9'
def shift(c,key):
    return c+key
def dec(text):
    aabb = ""
    for c in text:
        ac = ord(c)
        print(ac)
        if ac >= 85 and ac <= 90:
            ac = shift(ac,6)
```

```
            aabb += chr(ac)
            continue
        if ac >= 105 and ac <= 122:
            ac = shift(ac,-32)
            aabb += chr(ac)
            continue
        if ac >= 40 and ac <= 47:
            ac = shift(ac,25)
            aabb += chr(ac)
            continue

        if ac >= 91 and ac <= 104:
            ac = shift(ac,-57)
            aabb += chr(ac)
            continue

        if ac >= 123 and ac <= 125:
            ac = shift(ac,-68)
            aabb += chr(ac)
            continue
        if ac >= 58 and ac <= 64:
            ac = shift(ac,-10)
            aabb += chr(ac)
            continue
        if ac>=48 and ac<=54:
            ac = shift(ac,10)
            aabb += chr(ac)
            continue

        if ac >= 55 and ac < 58:
            ac = shift(ac,68)
            aabb += chr(ac)
```

```
            continue


        if ac >= 34 and ac <= 39:
            ac = shift(ac,83)
            aabb += chr(ac)
            continue

        if ac >= 34 and ac <= 84:
            ac = shift(ac,32)
            aabb += chr(ac)
            continue

        if ac == 126 or ac == 33:
            aabb += chr(33)
    return aabb


print(dec(cipher))
```

Flag: CS2107{m45t3r_0f_sub5ti7u7i0n_3nj0y_uR_s4l@d!!}

Hard Challenge

H.2 Secure Password

Given a html file, we can use inspect tools to retrieve the backend verification javascript function. From the javascript:

```
let attempts = 3;
function secureHash() {
  const passwordInput =
document.getElementById("passwordInput").value;
  const expectedString = "e|278Fx7|!VeX7_!V!|@8SR";
  const magic = [
    BigInt("0x1fa9787f52d6819dac3e51c96c9850ac9a68a000"),
    BigInt("0x551e7b2ade66a9cd21538d24f8232eb9e3c6a00"),
    BigInt("0x685130edf575c5fd89b4ea52d8ce440fb75d40"),
```

```
  BigInt("0x4d2b06845e7f210fd15f3697fe234c69919a0"),
  BigInt("0x267227d769f1422427c2f550f7852c59bfec"),
  BigInt("0xd9fd323c23dd5a26579cb53a8a42996b38"),
  BigInt("0x388a9fbf545b3b1a5e4b80376e94de767"),
  BigInt("0xadef7b085371d7244d43d0011e7c6d5"),
  BigInt("0x18cbc26aefc3b3b1ef4588ce4acc6b"),
  BigInt("0x296e5ed6f99d55e5efb08eb856e9"),
  BigInt("0x314ef6584d10a8c5226f105685"),
  BigInt("0x2798a7a450463592994fc72f"),
  BigInt("0x133caaa3da819c1ca0087d"),
  BigInt("0x445974d799d8bcf9c3b"),
];
let magic2 = BigInt("0x2971713e56d0006e6a0b48126ca34000");
let calculatedString = "";
let oneChar = 0;
let result = BigInt(0);
let nresult = BigInt(0);
for (let i = 0; i < passwordInput.length; i++) {
  result = BigInt(0);
  oneChar = -passwordInput.charCodeAt(i);
  for (let j = 0; j < magic.length; j++) {
    result *= BigInt(oneChar);
    result += magic[magic.length - 1 - j];
  }
  nresult = result % magic2;
  result = Number(-result / magic2);
  result += (888 - result) * (result > 127);
  result += (888 - result) * !(nresult == 0);
  result += (888 - result) * (result < 33);
  calculatedString += String.fromCharCode(result);
}
```

```
  if (calculatedString === expectedString) {
    alert("Yay you got the password. Submit the password as the flag
:D");
  } else {
    attempts--;
    if (attempts === 0) {
      document.getElementById("passwordInput").disabled = true;
      document.querySelector("button").disabled = true;
      alert("Too many attempts (≧ω≦*). Go try other challenges.");
    } else {
      alert(`Try harder. You have ${attempts} attempts left.`);
    }
  }
}
```

It is difficult to reverse engineer the decryption function from the encryption function like in Salad. Still,we know that the password must equal the length of the encrypted password and that it is made out of ascii characters. Using a brute force approach to try every single possible passwords with the same length for all ascii characters, we can get the flag.

Script used:

```
ascii_table = [chr(i) for i in range(128)] #brute force all possible
passwords using all ascii characters
def breakPW():
    password_input = ascii_table
    expected_string = "e|278Fx7|!VeX7_!V!|@8SR"
    # Retrieve big integer values by going to
    # to a javascript playground and printing them out
    # https://playcode.io/new
    magic = [
        180758036634373586855986153129327037440435200000,
        303714979546645605479342826765247319825792640000,
        232635024566280605144825935146492857706239520 0,
        107556589139054689493151371480217131922561440,
        334911201362030401938262572173412064727857 2,
```

```
    741778291394062241074382244326495534067776,
    120250471667593836985674735265058900567l,
    144499936010007312185241547504494691 41,
    128747857113173407665651901872655467,
    840323243106107751155149189961449,
    390661760670649286847795562662 9,
    122544637738669183090965727 19,
    232560803150345442563216 61,
    20173193776555036220475,
]
magic2 = 55087374223494444286125519719270400000
calculated_string = ''
one_char = 0
result = 0
nresult = 0
for i in range(len(expected_string)):
    for j in range(len(ascii_table)):
        result = 0
        nresult = 0
        one_char = -ord(ascii_table[j])
        for k in range(len(magic)):
            result *= one_char
            result += magic[len(magic) - 1 - k]
        nresult = result % magic2
        result = int(-result / magic2)
        result += (888 - result) * (result > 127)
        result += (888 - result) * (nresult != 0)
        result += (888 - result) * (result < 33)
        if (expected_string[i] == chr(result)):
            calculated_string += ascii_table[j]
            break
print(calculated_string)
```

```
# Call the function to simulate the behavior
breakPW()
```

Flag: CS2107{1S_4Ct1y_4_Sb0x}