龙鳞小队

Round

用GDA分析程序：

接受用户名和密码，用户名的Base64编码（其实不是）为c9m1bRmfY5Wk

密码经过round处理后会等于ointArray1，而round需要一个Box，用c9m1bRmfY5Wk来makeBox

```
int[] ointArray1 = new int[i1]{352,646,752,882,'A',0,'z',0,0,7,350,360};
```

那么我们主要分析三个函数encodeToBase64、Makebox、round

encodeToBase64

这个函数整体上还是Base64的逻辑，但是其中有所猫腻，这里只截取有关代码：

```
i5 = 2;
int i6 = 1;
if (i3 != i6) {
   i5 = (i3 == i5)? i6: i3;
}
if (i6 < i7) {
   i5 = i5 - 3;
   i5 = i5 * 6;
   i5 = i4 >> i5;
   i5 = i5 & 0x3f;
   str = str.append(MakePath.BASE64_CHARS[i5]);
```

发现在i3取0~3时，i5最后的取值却为-3、-1、-2、0，这说明他把对应64编码的第2、3个数字调换位置了

```
def decode(s):
    # 去除输入字符串中的填充字符'='
    padding = s.count("=")
    s = s.rstrip("=")

    # 初始化输出字节数组
    output = bytearray()

    # 按每4个字符一组处理
    for i in range(0, len(s), 4):
        # 将Base64字符转换为对应的6位整数
        i1 = BASE64_CHARS.index(s[i + 0])
        i2 = BASE64_CHARS.index(s[i + 1]) if padding < 3 or i + 1 < len(s) else 0
        i3 = BASE64_CHARS.index(s[i + 2]) if padding < 2 or i + 2 < len(s) else 0
        i4 = BASE64_CHARS.index(s[i + 3]) if padding < 1 or i + 3 < len(s) else 0
        print(i1, i2, i3, i4)
        # 合并成一个24位的整数
        combined = (i1 << 18) | (i3 << 12) | (i2 << 6) | i4
        # 分割成3个8位的字节
        b1 = (combined >> 16) & 0xFF
```

```
        b2 = (combined >> 8) & 0xFF
        b3 = combined & 0xFF
        # 添加到输出字节数组
        output.append(b1)
        if padding < 2 or i < len(s):
            output.append(b2)
        if padding < 1 or i < len(s):
            output.append(b3)

    # 返回解码后的字节序列
    return bytes(output)
```

我们只要在解码时把i2和i3换下位置即可，得到用户名：round_and

Makebox

其实这个函数挺简单的，但是有个点我有点困惑，我常用python，听说java不能用负索引，但在代码中显然会取负的，最后试的要直接变正数，所以 i1 - 1023 我改成了 1023 - i1

```
for (int i1 = i; i1 < 1024; i1 = i1 + 1) {
  i2 = i1 - 1023;
  ointArray[i2] = i1;
}
```

python代码：

```
def Makebox(p0):
    ointArray = [0] * 1024
    for i in range(1024):
        i2 = 1023 - i
        ointArray[i2] = i
    for i in range(1024):
        ointArray[i] ^= ord(p0[i % len(p0)])
    return ointArray
```

Round

最后是这个大头，简单分析代码，程序的主要变量是num、rip，根据p1（即输入的密码）先判断做哪种操作，而这些操作会更新num、rip，每轮的num都会存储起来，最后与ointArray1校对

下面是我用python简化的版本：

```

class Round:
    def __init__(self):
        super().__init__()

    def shl(self, p0, p1):
        p0 = (p0 >> 3) % 1024
        return Round.Result(p0, ((p1 + p0) % 1024))

    def shr(self, p0, p1):
        p0 = (p0 << 3) % 1024
        return Round.Result(p0, ((p1 + p0) % 1024))
```

```
    def add(self, p0, p1, p2):
        p1 = (p1 + p0[p2]) % 1024
        return Round.Result(p1, (p2 + p1) % 1024)

    def sub(self, p0, p1, p2):
        p1 = (((p1 - p0[p2]) % 1024) + 1024) % 1024
        return Round.Result(p1, (p2 + p1) % 1024)

    def xor(self, p0, p1, p2):
        i = (p0[p2] ^ p1) % 1024
        return Round.Result(i, (p2 + i) % 1024)

    def round(self, p0, p1):
        assert len(p1) == 12
        result = None
        ointArray = [0] * 12
        i2 = 33
        for i in range(12):
            c = ord(p1[i])
            for _ in range(32):
                match (p0[i2] ^ c) % 5:
                    case 0:
                        result = self.add(p0, c, i2)
                    case 1:
                        result = self.sub(p0, c, i2)
                    case 2:
                        result = self.xor(p0, c, i2)
                    case 3:
                        result = self.shl(c, i2)
                    case 4:
                        result = self.shr(c, i2)
                    case _:
                        Round.Result(c, i2)
                c = result.getNum()
                i2 = result.getRip()
            ointArray[i] = c
        return ointArray

    class Result:
        def __init__(self, num, rip):
            self.num = num
            self.rip = rip

        def getNum(self):
            return self.num

        def getRip(self):
            return self.rip
```

那我们考虑是否可以逆转程序，显然我们不知道rip最后的值，因此难以实施。

但是我们发现函数的特点：生成的ointArray已知长度为12，且每个元素只与p1的对应位置之前的元素有关。

因此可以采用逐个爆破的方法在合理时间内推算答案，每个字符的可能为大小写字母和下划线，直接爆破会得到_round_AB_

```
def Crack(encoded_box):
    ointArray1 = [352, 646, 752, 882, ord("A"), 0, ord("z"), 0, 0, 7, 350, 360]
    ps = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_"
    password = "000000000000"
    assert len(password) == 12
    for i in range(12):
        for p in ps:
            password = password[:i] + p + password[i + 1 :]
            result = Round().round(encoded_box, password)
            if result[i] == ointArray1[i]:
                if func(password):
                    continue
                break
        else:
            print("Failed to crack")
            break
    print(password)
    return password
```

因此我们再写一个函数，过滤掉不合适的答案

```
def func(password):
    match password[7]:
        case "A":
            return True
        case "C":
            return True
        case "E":
            return True
        case "M":
            return True
        case "P":
            return True
        case "b":
            return True
        case "n":
            return True
        case "w":
            if password[8] in ["J", "P", "Q", "T", "U", "Z"]:
                return True
    # if i == 8 and p in ["B", "I", "N", "T", "W", "d"]:
    #     continue
    # if i == 9 and p in ["g"]:
    #     continue
    return False
```

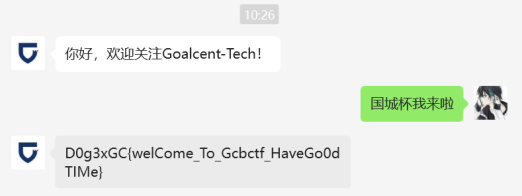
最后算出_rounD_we_go

flag:D0g3xGC{round_and_rounD_we_go}

调查问卷

D0g3xGC{Thanks_for_your_participation}

我是真签到



D0g3xGC{welCome_To_Gcbctf_HaveGo0dTIMe}