Unimod – NahamCon 2022

James Nelson

05/03/2022

The Unimod challenge gives you an output file that has a string of non-English characters as well as a Python script that was used for the encryption of the flag. The script first imports the "random" module, reads a file containing the flag to a variable, and then declares an empty string variable. A variable "k" is set to a random value between the range of 0 and 0xFFFD by using the random module's "randrange" method. In Python, hexadecimals can be used in place of integers; To use a hexadecimal value, prefix the hexadecimal with "0x". In the case of "0xFFFD", "FFFD" converts from hex to an integer value of 65,553, so the variable "k" is set to any number from 0 up to but not including 65,533 in this script.

```
1  import random
2
3  flag = open('flag.txt', 'r').read()
4  ct = ''
5
6  # random.randrange() returns a randomly selected element from the specified range
7  k = random.randrange(0,0xFFFD)
8
```

The main part of the Python script is within a single loop, which iterates over each character of the flag string. It appends each encrypted character onto the empty string that it initialized prior to this loop. It uses a function called "chr()", which returns the Unicode character of a numeric value given as an argument. In this case, an expression is passed as an argument to the "chr()" function. The expression within the "chr()" function's argument list uses a function called "ord()", which returns the integer value of a Unicode character that is passed as an argument to the function.

The “ord()” function is being passed the current flag string character throughout each iteration of the loop. The return value of the “ord()” function is the integer value of the current flag string character, the value is then added together with the random value that was assigned to the “k” variable. The modulo operator (the percentage sign “ % ”) is then used to return the remainder of the division between the result of the addition and the hexadecimal value of “FFFD” (which is 65,533). After the loop completes, the new encrypted string is written to an “out” file, which is what we need to decrypt to get the flag.

```
 9 for c in flag:
10     # chr() returns a unicode character from an integer
11     # ord() returns the unicode integer value from a given character
12     ct += chr((ord(c) + k) % 0xFFFD)
13
14
15 open('out', 'w').write(ct)
16
```

An important thing to note, is that if the result of the addition between the Unicode integer value and the random integer assigned to “k” is less than 65,533, then the modulo operator would do virtually nothing and the expression would just evaluate to the result of the addition before being converted back to a Unicode character. This would occur because the dividend is smaller than the divisor and the division would leave the entire dividend as a remainder. Another interesting feature, although not as important in this scenario, is that there is a possibility that this script could randomly assign a value of 0 to “k”, which means that if the Unicode integer values of the plaintext characters were less than 65,533 to begin with, then there would be no encryption performed and the plaintext would be written to the out file. We know this is not the case since we don’t have a flag in the out file, but it’s interesting. Another thing to consider is that the plaintext that was encrypted is most likely going to be alphanumeric characters along with some basic symbols, but probably nothing that doesn’t appear on a US

keyboard layout, and these Unicode characters range only from 0 to 127 in their integer values. Equally noteworthy would be the current integer values of the characters in the given “out” file, which have a maximum value of 39,262, a minimum value of 39,147, with a range of 115.

In one of my scripts, I get the information stated above, I first open and read the “out” file, then get the integer values of every character into an array. I sort the array, and then print to the screen the highest and lowest character values, as well as the range between them.

```
def readFile():
    ct = open('out', 'r').read()
    ctArr = []

    for x in ct:
        ctArr.append(ord(x))
    return ctArr

sortedArr = readFile()
sortedArr.sort()

print('Highest Value: {}\nLowest Value: {}\nRange: {}'.format(sortedArr[-1], sortedArr[0],
(int(sortedArr[-1]) - int(sortedArr[0]))))
```

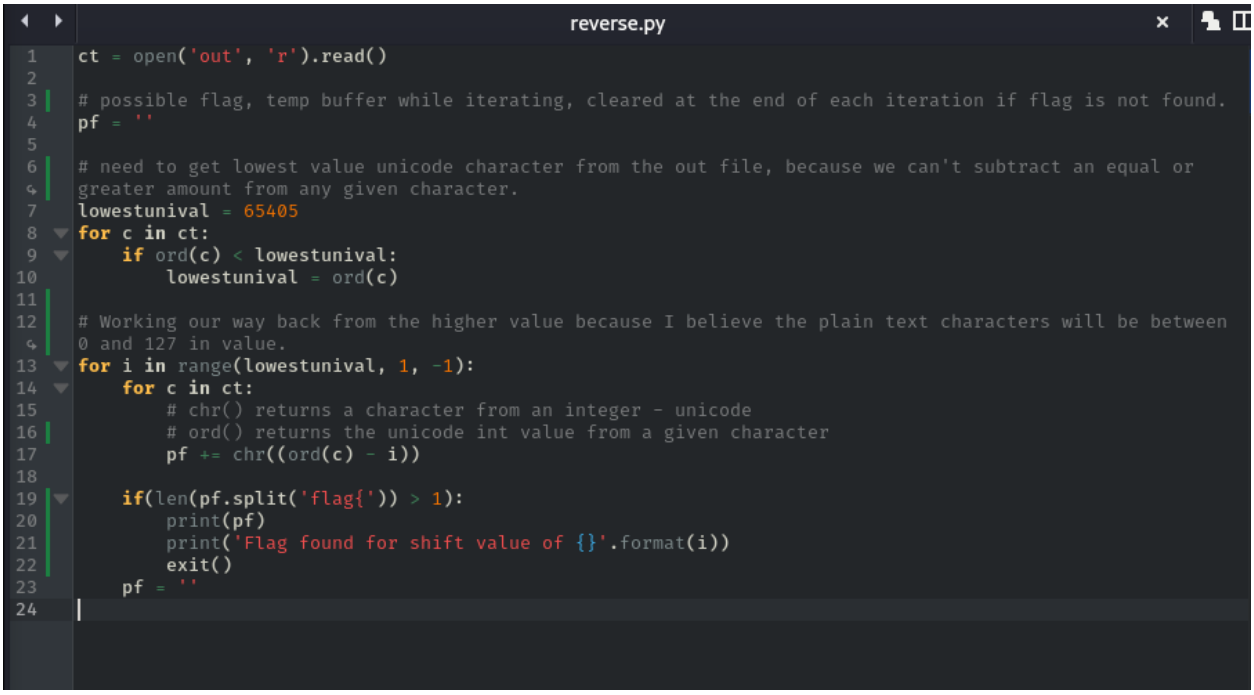
The output looks like this:

```
└─$ python3 getCharIntVal.py
Highest Value: 39262
Lowest Value: 39147
Range: 115
```

There are a variety of options to deal with breaking this encryption, with the simplest being that you accept the fact that 39,262, or even 65,533, is not a huge number to brute force the encryption and set a stop when you find a string that has the word “flag” in it. With the information noted above, you could even infer that the first character should be an “f”, and since it should have a Unicode integer value of 102 and the first character of the encrypted string has a value of 39,239, the shift value should be equal to 39,137 for all of the characters in the string. If you were in a situation where you weren’t able to infer these things, but you suspect characters

within a specific keyboard layout (which in our case would be US), then you could limit the scope of your brute force iterations to the minimum character value of the ciphertext minus the minimum character value of the suspected character range and run it up until the maximum value of the suspected character range.

Here is a script that first runs through the encrypted characters to find the lowest value between them all. It then runs through a loop which will begin by shifting all the characters by the lowest value within the encryption, and then each iteration shifts by one less until it reaches 1 (should have used 0 here but it doesn't matter). Each iteration prepares a string, which is then checked to see if a resulting array from a string split using 'flag{' as a delimiter has a length of more than one; If there are more than 1 elements then the substring 'flag{' is present in our prepared string. From here we print the flag to the screen, as well as the shift value that generated the result, and then exit the script.



The other way was built onto the script that gave me the range between highest and lowest characters earlier. In the rest of this script, after printing the range and the other info to the screen, I read the "out" file again to another, unsorted, array.

```
def readFile():
    ct = open('out', 'r').read()
    ctArr = []

    for x in ct:
        ctArr.append(ord(x))
    return ctArr

sortedArr = readFile()
sortedArr.sort()

print('Highest Value: {}\nLowest Value: {}\nRange: {}'.format(sortedArr[-1], sortedArr[0],
(int(sortedArr[-1]) - int(sortedArr[0]))))


unsortedArr = readFile()

print("\nOut File Encrypted Characters:\n", unsortedArr)
```

I print the array to the screen:

```
Out File Encrypted Characters:
 [39239, 39245, 39234, 39240, 39260, 39189, 39238, 39191, 39193, 39237, 39186, 39191, 39234, 39191, 39186, 39235, 39236,
 39187, 39238, 39234, 39192, 39187, 39237, 39190, 39239, 39194, 39192, 39186, 39188, 39189, 39189, 39238, 39193, 39189,
39239, 39186, 39186, 39262, 39147]
```

Seeing that the first value is 39,239, and inferring that this character should be 'f' with a value of 102, I add to the script to shift all elements of this array by 39,137. To do this, I initialize an empty array, then iterate over the array of the encrypted character values and append the Unicode character of the shifted value to the empty array. After the loop finishes, I join all the elements of the plaintext array to a "flag" string and then print the flag to the screen.

```
pt = []

for x in unsortedArr:
    pt.append(chr(x-39137))

print('\nOut File Characters after shifting left by 39137\n', pt)

flag = ''.join(pt)

print('\nFlag:\n', flag)
```

All of the code, the scripts I built and used as well as the original files given during the challenge, can be found on GitHub at:

https://github.com/j4655/Write-Ups/tree/main/NahamCon2022_Unimod

I had some trouble running these scripts in a PowerShell terminal when writing this document, it seems that the terminal window wasn't capable of using Unicode characters (or at least printing them), so I used my Linux instance to run them while writing this.