Crypto

Due on Wednesday, February 15, 2016

SYSTEM SECURITY FRI

Abraham Adberstein, David Chavarria

Task I

The two encryption programs are available at the following URLs:

cs.utexas.edu/users/fri-security/enc-lab/p1/?q=Howdy!
cs.utexas.edu/users/fri-security/enc-lab/p2/?q=Howdy!

We made a python script to try different inputs and do not waste time.

Listing 1: Server Request Script

```
1
2   def makeRequest2(searchWord):
3     url = "http://www.cs.utexas.edu/users/fri-security/enc-lab/p1/?q="+searchWord
4     req = urllib2.Request(url)
5     res = urllib2.urlopen(req)
6     page = res.read()
7     result = page[6:len(page)-8].strip()
8     return result
```

Observations for first Encryption Algorithm:

Every two bits became 4 bits. If the number of bits is odd in the plaintext, the last bit encrypts to 4 bits. The following are some patterns we tested first.

plaintext: a : 022c
plaintext: b : 082c
plaintext: c : 0a2c
plaintext: d : 202c
plaintext: e : 222c
plaintext: f : 282c
plaintext: aa : 033c
plaintext: bb : 0c3c
plaintext: cc : 0f3c
plaintext: dd : 303c
plaintext: ee : 333c
plaintext: ab : 063c
plaintext: ac : 073c
plaintext: ae : 133c
plaintext: ba : 093c
plaintext: abcd : 063c 1a3c
plaintext: abcdedfg : 063c 1a3c 323c 3d3c
plaintext: efghi : 363c 6a3c 822c
plaintext: god : 7f3c 202c
plaintext: divine : 613c 693e b93c
plaintext: angel : 563c 3b3c a02c
plaintext: an : 563c
plaintext: ge : 3b3c
plaintext: l : a02c

After looking at the input, we began to notice patterns and we decided to make tables to help us identify those patterns. As a result, we came up with the following tables:

oc, od, 18, 19, 1c, 1d	24, 25, 30, 31, 34, 35	2c, 2d, 38, 39, 3c, 3d
0e, 0f, 1a, 1b, 1e, 1f	26, 27, 32, 33, 36, 37	2e, 2f, 3a, 3b, 3e, 3f

0c, 0e, 24, 26, 2c, 2e	18, 1a, 30, 32, 38, 3a	1c, 1e, 1f, 34, 36, 3c
0d, 0f, 25, 27, 2d, 2f	19, 1b, 33, 39, 3b, 3e	1d, 1f, 35, 37, 3d, 3f

The first table showed us that if the cipher text ended with any of those characters then the first two characters were in that specific location in the ASCII table. (Note: we divided the ASCII table into three columns and two rows and we omitted the first 31 characters of the ASCII table).

The second table showed us that if the cipher text ended with any of those characters then the last two characters were in that specific location in the ASCII table.

But after a while, we found out that there was an easier way to visualize what we did and we did that by passing in URL encoders (e.g. instead of passing in "@@" we passed in "%40%40"). From these URL encoders we were able to come up with the following mappings for the first and second encryption algorithms.

Mappings for first equation:	**Mappings for Second equation:**
0 -> 00	0 -> 80
1 -> 01	1 -> 81
2 -> 04	2 -> 84
3 -> 05	3 -> 85
4 -> 10	4 -> 90
5 -> 11	5 -> 91
6 -> 14	6 -> 94
7 -> 15	7 -> 95
8 -> 40	8 -> c0 ->120
9 -> 41	9 -> c1 -> 121
A -> 44	A -> c4 -> 124
B -> 45	B -> c5 -> 125
C -> 50	C -> d0 -> 130
D -> 51	D -> d1 -> 131
E -> 54	E -> d4 -> 134
F -> 55	F -> d5 -> 135
wxyz (2x + z) (2w + y)	wxyz (2x + z - 160) (2(w-80) + (y-80) + 1)

Another thing that we noticed was that the first and second encryption algorithms followed the same pattern; the only exception was that the second algorithm alternated between the first mappings and the second mappings.

Now, to explain what we found, I will begin by explaining the first equation that we came up with: (2x + z) (2w + y). Suppose we sent in "%wx%yz" to the algorithm. What the algorithm does is that it gets w, x, y, and z and it gets its value from the mapping table we provided and it passes it through the equation (2x + z) (2w + y). The equation then spits out two numbers (#)(#) that correspond to the cipher text. For example, if we send in "%46%43", we get w=10, x=14, y=10, and z=05, and therefore, the algorithm returns (33)(30). The actual cipher text is "2d30". (Note: 2d = 2*10 + 13 = 33).

I wont do much explaining for the second encryption algorithm because it is basically the first one but with an exception. Unlike the first one, this one alternates between the first equation/mappings and the second equation/mappings. For example, suppose we send in "%46%43%50%00%46%43%50%00", the output will be "2d30 8023 2d30 8023" where the first four characters correspond to the first equation/mappings and the next four characters correspond to the second equation/mappings. To solve the second equation in this scenario we do as follows:
1) w=91, x=80, y=80, z=80
2) (2(80) + 80 - 160) (2(91-80) + (80-80) + 1) = (80) (23)
3) (80) (23) = "8023"

Listing 2: Encryption Schema Script.

```
def equations(word):
   ascii_map = {
     "0" : 0,
     "1" : 1,
     "2" : 4,
     "3" : 5,
     "4" : 10,
     "5" : 11,
     "6" : 14,
     "7" : 15,
     "8" : 40,
     "9" : 41,
     "A" : 44,
     "B" : 45,
     "C" : 50,
     "D" : 51,
     "E" : 54,
     "F" : 55
   }

   characters = list(word)
   w = ascii_map[word[0]]
   x = ascii_map[word[1]]
   y = ascii_map[word[2]]
   z = ascii_map[word[3]]

   eq1 = 2*x + z
   eq2 = 2*w + y
   server_result = makeRequest(word)
   return  str(eq1) + "" + str(eq2)
```

Task II

Listing 3: Encryption Algorithm. encrypter.py

```
1
2  from datetime import datetime
3  import getopt, sys, string, time, math
4
5  def encrypt(word):
6
7      hour = datetime.now().hour
8      minutes = datetime.now().minute
9      seconds = datetime.now().second
10     rotations = ""
11     ascii_dictionary = {}
12     to_ascii_dictionary = {}
13     counter = 1
14     encrypted_word = ""
15
16     for c in (chr(i) for i in range(32,127)):
17         ascii_dictionary[c] = counter
18         to_ascii_dictionary[counter] = c
19         counter = counter + 1
20
21     word_characters = list(word)
22
23     for c in word_characters:
24         num = ascii_dictionary[c] + pow(hour + minutes + seconds, 2)
25         value = num % 95
26         rotations =  num / 95
27         encrypted_word += to_ascii_dictionary[int(value)]
28
29     if seconds < 10:
30         seconds = "0"+str(seconds)
31
32     if minutes < 10 :
33         minutes = "0" + str(minutes)
34
35     if hour < 10:
36         hour = "0" + str(hour)
37
38     if rotations < 10:
39         rotations = "0" + str(rotations)
40
41     encrypted_word += str(rotations) + str(hour) + str(minutes) + str(seconds)
42     return encrypted_word
43
44 def main():
45     opts, args = getopt.getopt(sys.argv[1:], 'w:')
46     word = opts[0][1]
47     print encrypt(word)
48
49 if __name__ == "__main__":
50     main()
```

Listing 4: Decryption Algorithm. decrypter.py

```
1
2  import getopt, sys
3
4  def decrypt(word):
5
6      word_length = len(word)
7
8      #get the time
9      hours = int(word[word_length-6:word_length-4])
10     minutes = int(word[word_length-4:word_length-2])
11     seconds = int(word[word_length-2:word_length])
12
13     letters = list(word[0: word_length-8])
14     rotations = int(word[word_length-8:word_length-6])
15
16     plaintext = ""
17     ascii_dictionary = {}
18     to_ascii_dictionary = {}
19     counter = 0
20
21     for c in (chr(i) for i in range(32,127)):
22         ascii_dictionary[c] = counter
23         to_ascii_dictionary[counter] = c
24         counter += 1
25
26     for i in range(len(letters)):
27         num  = rotations * 95
28         cValue = ascii_dictionary[letters[i]]
29         fValue = num + cValue - pow(hours + minutes + seconds, 2)
30         fChar = to_ascii_dictionary[fValue]
31         plaintext += fChar
32
33     return plaintext
34
35
36 def main():
37     opts, args = getopt.getopt(sys.argv[1:], "w:")
38     word = opts[0][1]
39
40     print "Ciphertext: " + word + " Plaintext: " + decrypt(word)
41
42 if __name__ == "__main__":
43     main()
```

Usage:

Encryption: python encrypter.py -w <plaintext>

Decryption: python decrypter.py -w <plaintext>

Input:

sabrina
david
abraham
ashay

Output:

’tu|"t34000453
;8M@;06000518
ghxgngs02000509
?QF?W14000531