Security and Privacy in the Wild HW 1

Chloe Eghtebas

September 2015

Worked on assignment with Nwamika Nzeocha and Omri Sass.

1 Name three incentives an attacker might have for paying \$325 (or more) for a high-profile Twitter account password. At least one of these should make money for the attacker

List of hacker's incentives:

1. For the reputation of breaching a high profile account
2. Reuse of password to the high profile persons other accounts
3. Fishing attacks could be deployed on the high profile persons network to get other peoples information and perform scams. This option is money making because the attacker could sell the new information as well as directly scam people out for their bank information somehow using the reputation of the high profile person.

2 Bad Security Questions for Authentication

2.1 For an online service whose users are U.S. inhabitants, give an estimate of the average number of randomly selected accounts an attacker would have to try to answer the question "What's your mother's maiden name?" correctly. Assume one guess per account.

In order to find the number of accounts we must try in order to guess the security questions correctly, we want to guess the most probable answer to minimize the amount of times we have to try. From this link:

http://www.census.gov/topics/population/genealogy/data/2000_surnames.html

We see that the most common surname is Smith. The technical documentation pdf found on the link above says "The most common last name reported was SMITH, held by about 2.3 million people, or about .9 percent of the population." This can be found under section 3.1 in the second paragraph of the technical document.

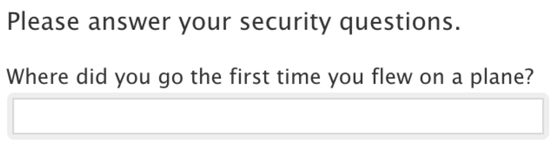
So, to find out how many accounts we must try in oder to suscessfully guess Smith as the answer to the security question, we can calcluate the following: Out of the total population, only 0.9% have the surname Smith. Lets say we have a total population of 100 people then there is the probability that 0.9 of them would have the surname Smith. In order to make sure that at least one person had the surname Smith, then we would need:

$$\frac{100}{\frac{\text{occurrences of smith}}{\text{total population}}} = \frac{100}{0.9} \approx 111.11 \text{ people} \tag{1}$$

Or 111.11 accounts to try before sucessfully getting into one. Since 111.11 accounts is not a possible number of accounts to guess the security question on, we can round up and say we need to try 112 accounts before correctly answering this bad security question.

2.2 Find the weakest password-recovery question you can online that wasn't discussed in class. Give an estimate of the guessing probability and explain how you arrived at it.

Just for this problem, my teammates and I started "forgetting" our passwords to our various accounts to see what recovery questions we were asked. The question shown in Figure 1 is the one we decided to go with.


Please answer your security questions.

Where did you go the first time you flew on a plane?

Figure 1: Apple password recovery question.

We did some further research and estimated the number of flights per year. From this super credible source:

http://www.garfors.com/2014/06/100000-flights-day.html

We estimated around 100,000*365 = 36,500,000 flights per year world wide. To be able to guess this security question, we want to find the most common answer. We thought that the most common answer would be the location with the most flight traffic which we found to be in London. From this source:

http://www.heathrow.com/company/company-news-and-information/company-information/facts-and-figures

We can see that Heathrow airport alone has 470,695 flights annually (this number was for 2014). So we can calculate the guessing probability as below:

$$P_{Guess} = \frac{470,695 \text{ Heathrow flights annually}}{36,500,000 \text{ total flights annually}} * 100 \approx 1.29\% \tag{2}$$

Or 1.29% chance that someones first flight is to London Heathrow airport. However, London has multiple airports nearby as shown in Figure 2.

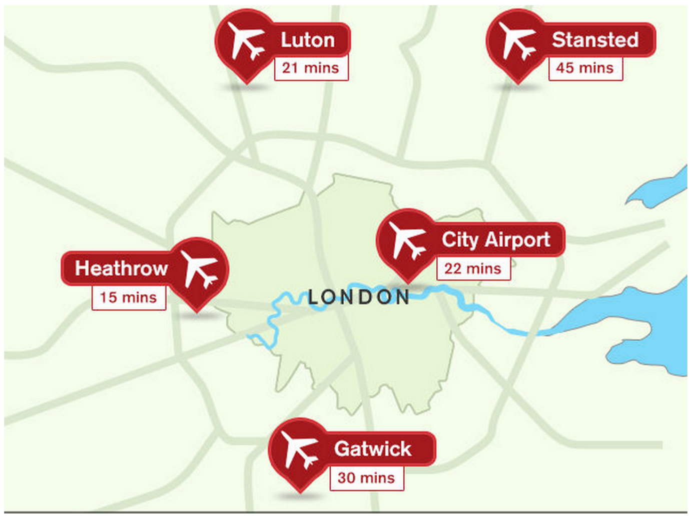


Figure 2: Airports in and neighboring London. Source: http://www.visitlondon.com/traveller-information/travel-to-london/airport/london-airport-map

So, depending on how someone answers this ambiguous security question (whether the person counts connecting flight destination or the city, county, or any other form of identifying a place as the destination) we will have a change in our guessing probability. An increase in probability if we count the air traffic in the neighboring airports, and a decrease in probability based on the variance of how some people may interpret and answer the question.

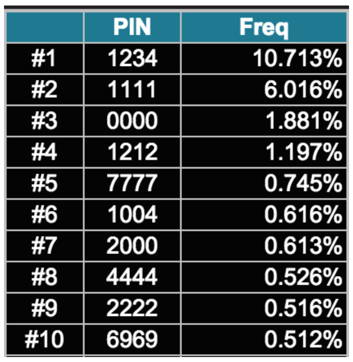
3 PINs and Needles (or passwords in a Haystack)

3.1 Suppose an attacker is given a single encrypted PIN randomly selected from the spreadsheet. How many decryption attempts, expressed as a decimal number, would the attacker need to perform to learn the PIN with probability 1?

There are 10,000 possibilities a 4 digit pin can presumably be. There are 2^{256} possible combinations for guessing the key using AES-256. The number of decryption attempts needed is $2^{256} * 10,000 \approx 1.158 * 10^{81}$.

3.2 Now, to show a different attack that efficiently obtains information about customer PINs, answer the following question. What PIN belongs to the customer Ebenezer Scrooge?

Simply using excel's COUNTIF function, we see that Ebenezer Scrooge's hashed password comes up 184 in the RNBPINs dataset. So, out of 10,000 passwords, 184 of them will be the same as Scrooge's password or $P = \frac{10,000}{184} * 100 = 1.84\%$.



	PIN	Freq
#1	1234	10.713%
#2	1111	6.016%
#3	0000	1.881%
#4	1212	1.197%
#5	7777	0.745%
#6	1004	0.616%
#7	2000	0.613%
#8	4444	0.526%
#9	2222	0.516%
#10	6969	0.512%

Figure 3: Table of commonly used pins with their associated frequesncy of use. Source: http://www.datagenetics.com/blog/september32012/

Looking at Figure 3, we see that the most likely pin associated with the password frquency of 1.84 %, which is the frequency 1.881 %, is the pin 0000. The 3rd most commonly used pin!

4 Meddling with Medallions

4.1 What is the maximum number of attempts needed to crack the hash of a medallion number (assuming no precomputation and no database of issued medallions)?

To calculate the maximum number of attempts an attacker must perform we must find the number of possible medallions.

The three authorized medallion formats are: DLDD, LLDDD, and LLLDDD. We know that there can be up to 10 possible combinations for each D and 26 possible combinations for each L. Thus the sum of all these possible combinations is the max number of tries to exhaustively break the hash of a medallion number.

$$\begin{aligned} DLDD + LLDDD + LLLDDD &= 26 * 10^3 + 26^2 * 10^3 + 26^3 * 10^3 && (3)\\ &= 26,000 + 676,000 + 17,576,000 && (4)\\ &= 18,278,000 && (5) \end{aligned}$$

4.2 What is the security goal of the NYC Department of Transportation in this setting, i.e., why did they hash medallion numbers?

From medallion numbers, you can track the amount of money each cab driver makes and where they live. Both of these things can be sensitive information given the circumstance and could lead to the unveiling of other potentially personal information depending on how far the attacker wants to go.

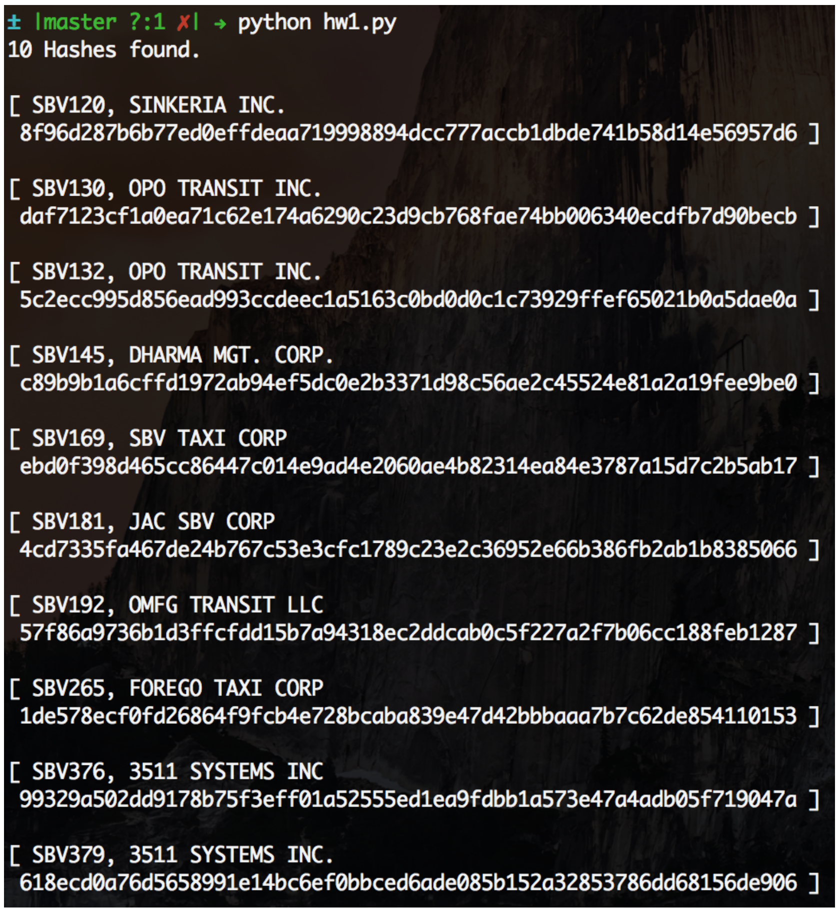
4.3 On the course site (file medallionhashes.txt), you'll find a list of SHA-256 hashes for some LLLDDD-format medallions that are active at the time of this exercise. Crack them! Also indicate who owns the corresponding taxis.

In order to do crack these hashes, I wrote a python script. You can view this script along with the data here on my github or see below:

```
import hashlib
import csv

data_file = open("taxi.csv","rb")
data = csv.reader(data_file,delimiter=",")
medallions = []
```

```
names = []
index = 0

# hashes to be found
find_these_hashes= [
"618ecd0a76d5658991e14bc6ef0bbced6ade085b152a32853786dd68156de906",
"99329a502dd9178b75f3eff01a52555ed1ea9fdbb1a573e47a4adb05f719047a",
"ebd0f398d465cc86447c014e9ad4e2060ae4b82314ea84e3787a15d7c2b5ab17",
"c89b9b1a6cffd1972ab94ef5dc0e2b3371d98c56ae2c45524e81a2a19fee9be0",
"1de578ecf0fd26864f9fcb4e728bcaba839e47d42bbbaaa7b7c62de854110153",
"4cd7335fa467de24b767c53e3cfc1789c23e2c36952e66b386fb2ab1b8385066",
"8f96d287b6b77ed0effdeaa719998894dcc777accb1dbde741b58d14e56957d6",
"daf7123cf1a0ea71c62e174a6290c23d9cb768fae74bb006340ecdfb7d90becb",
"57f86a9736b1d3ffcfdd15b7a94318ec2ddcab0c5f227a2f7b06cc188feb1287",
"5c2ecc995d856ead993ccdeec1a5163c0bd0d0c1c73929ffef65021b0a5dae0a"]

# putting data from file into arrays
for row in data:
        if index!=0:
                medallions.append(row[0])
                names.append(row[1])
        index+=1

# found hashes array
found= []

# append each computed medallion hash in the
# find_these_hashes array to found array
for med in medallions:
        hashed_med = hashlib.sha256(med).hexdigest()
        ind = medallions.index(med)
        if hashed_med in find_these_hashes:
                found.append((med, names[ind], hashed_med))

# making sure all ten hashes are found
print  "{0} Hashes found. \n".format(len(found))

# print hash results
for m, n, h1 in found:
        print "[ {0}, {1} \n {2} ]\n".format( m, n, h1)
```

Figure 4 shows the output of the script when run.



```
± |master ?:1 ✗| → python hw1.py
10 Hashes found.

[ SBV120, SINKERIA INC.
 8f96d287b6b77ed0effdeaa719998894dcc777accb1dbde741b58d14e56957d6 ]

[ SBV130, OPO TRANSIT INC.
 daf7123cf1a0ea71c62e174a6290c23d9cb768fae74bb006340ecdfb7d90becb ]

[ SBV132, OPO TRANSIT INC.
 5c2ecc995d856ead993ccdeec1a5163c0bd0d0c1c73929ffef65021b0a5dae0a ]

[ SBV145, DHARMA MGT. CORP.
 c89b9b1a6cffd1972ab94ef5dc0e2b3371d98c56ae2c45524e81a2a19fee9be0 ]

[ SBV169, SBV TAXI CORP
 ebd0f398d465cc86447c014e9ad4e2060ae4b82314ea84e3787a15d7c2b5ab17 ]

[ SBV181, JAC SBV CORP
 4cd7335fa467de24b767c53e3cfc1789c23e2c36952e66b386fb2ab1b8385066 ]

[ SBV192, OMFG TRANSIT LLC
 57f86a9736b1d3ffcfdd15b7a94318ec2ddcab0c5f227a2f7b06cc188feb1287 ]

[ SBV265, FOREGO TAXI CORP
 1de578ecf0fd26864f9fcb4e728bcaba839e47d42bbbaaa7b7c62de854110153 ]

[ SBV376, 3511 SYSTEMS INC
 99329a502dd9178b75f3eff01a52555ed1ea9fdbb1a573e47a4adb05f719047a ]

[ SBV379, 3511 SYSTEMS INC.
 618ecd0a76d5658991e14bc6ef0bbced6ade085b152a32853786dd68156de906 ]
```

Figure 4: Broken SHA256 results output.

4.4 Suppose that VIN numbers and licensee names were concatenated with medallion numbers prior to hashing active medallion numbers. Would this fix the problem and achieve the desired security goal? Why or why not?

NO! This doesn't change and the same thing as before would just happen. Why? Looking at Kerckhoffs's principle which states "A cryptosystem should be secure even if everything about the system, except the key, is public knowledge." So, even if the way the data is hashed with the concatenated VIN numbers, the attackers would be able to see that and reconstruct the hashes the same way they did before especially with unlimited access to the data since it is a public record.

4.5 Design a cryptographic scheme (a mechanism) using SHA-256 that would permit NYC to generate a unique pseudonym for each taxi medallion and meet the desired security goal.

Maybe by just salting the hash with a random number would add enough complexity to just increase the difficulty to break the hash using a frequency count. Also, since MD5 has collisions with enough data, they could have used another hashing algorithm like SHA256.

5 Gambling

5.1 Otis's friend, Mississippi Mabel, the famous sharpshooter, shot down his idea. She said his protocol won't work because the server can cheat without being detected. What is the security goal of the client? [2 points] How can the server defeat this security goal? [2 points] Is there any way in Otis's protocol for the client to detect server cheating?

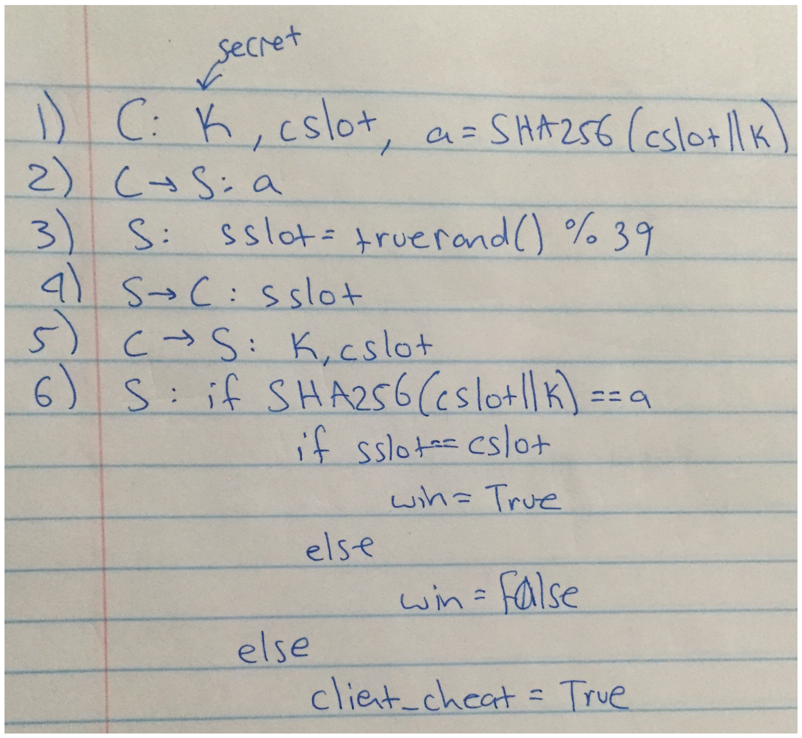
1. The security goal of the client is to be able to gamble without being cheated.
2. The server can easily change the sslot value as many times without evidence to prevent the client from winning or loosing unfairly.
3. It is possible to check whether or not the server is cheating by recording the values given to the client over a large period of time to see if there is a uniformly random output of numbers distributed across 39 slots.

5.2 Why doesn't Mabel's protocol work, i.e., how can a server ensure that it never awards a win to a client but cheated = false?

Mabel's protocol still doesn't work because the server can still reproduce a sslot after checking the client cslot. There is still no way for the client to be sure that the server didn't regenerate the random value or cheat.

5.3 Fix Mabel's protocol so that it does that does work, i.e., it allows a client to detect unfair play immediately.

Figure 5 shows a picture of our proposed protocol. This design was inspired by the commitment scheme.



```
      secret
1) C: K, cslot, a = SHA256(cslot||K)
2) C→S: a
3) S:  sslot = truerand() % 39
4) S→C : sslot
5) C→S: K, cslot
6) S : if SHA256(cslot||K) == a
               if sslot == cslot
                     win = True
               else
                     win = False
           else
               client_cheat = True
```

Figure 5: Fixed Protocol.

1. The client has private key and pick a value she/he wishes to gamble on. The client then proceeds to hash their gamble choice, cslot, usign the key and SHA256.

2. The client sends the hashed gamble choice over to the server.

3. The server ”rolls” and produces a valid result, sslot.

4. The server sends over the sslot value being assured that the client can no longer cheat due to being committed to the values she/he chose.

5. The client must comply (or not) and send over the original key and cslot value.

6. The server computes a hash on the recently provided key and cslot value and checks it against the previously shared hashed value from before. If the values don’t match, then it is clear that the client has cheated. If they do match, the Server continues to check if the roll result and the guess are equivalent, thus allowing the client to win, or not, in which the client looses.

6 Give a possible design for RuriMAC.

Given a (message, MAC) pair, the user must be able to determine the key in order to successfully forge the message.

A lazy design for this RuriMAC system could then be to just make the key public but that doesn't comply with the eavesdropper also not being able to forge the message if an invalid (message, MAC) pair is received.

Ran out of time to implement my own or document the rest of my thoughts.