**Assignment 1**

A bank requires for their customers to access their online banking accounts to provide as User Id (or user name) the last 8 digits of their bank card number, and a password with a length between 8 and 12 ASCII characters, including the following restrictions (posted on their website):

Passwords must have at least 8 characters long and at most 12 characters long, and must include at least one character from each of the following four character types:

* Upper case letters - A B C D E F
* Lower case letters - g h i j k l
* Numbers - 1 2 3 4 5 6 7 8 9 0
* Special characters - ! @ # $ % ^ & \* ( + ) = ~

The remaining characters of the password must be selected from the above character set (and can be from any of the character type).

For example, an acceptable password sample is gPanth2! While A@a#CDEF&\* will be considered unacceptable (with respect to the prescribed format).

The bank also requires that each password be changed at least once every five years.

**Q.1** Assume that 1,000,000 passwords can be tested per second, calculate the probability

that a hacker can guess a password in the timeframe between two consecutive changes.

**Soln.**

We know,

The probability ‘P’ for a hacker to guess a password in a specific time period ‘T’

It is defined by Eq. (i)

**…….. (i)**

Where,

**G** – The number of guesses

**N’** – the total number of possible passwords

To compute **N’**, we need to define the following events:

**X** – d-digit password from ASCII characters

**U** – d-digit password contains upper case characters from A, B, …F (Total-06 Characters)

**L** – d-digit password contains lower case characters from g, h,…l (Total-06 Characters)

**Y** – d-digit password contains numbers from 0, 1,…9 (Total-10 Characters)

**S** – d-digit password contains special characters from ! @ # $ % ^ & \* ( + ) = ~ (Total-13

Characters)

**U1** – d-digit password with no upper case characters from A, B,…F (Total-06 Characters)

**L1** – d-digit password with no lower case characters from g, h,…l (Total-06 Characters)

**Z** – d-digit password with no numbers from 0, 1,…9 (Total-10 Characters)

**S2** – d-digit password with no special characters from ! @ # $ % ^ & \* ( + ) = ~ (Total-13

Characters)

**|X|=d ……(ii)|S1∩Z|=d ……(x)**

**|U1|=d ……(iii)|U1∩S1|=d ……(xi)**

**|L1 |=d ……(iv)|L1∩S1|=d ……(xii)**

**|Z|= ……(v) |U1∩L1∩Z|=d ……(xiii)**

**|S1|=d ……(vi)|U1∩L1∩S1|=d ……(xiv)**

**|U1∩L1|=d ……(vii)|L1∩Z∩S1|=d……(xv)**

**|U1∩Z|=d ……(viii)|U1∩Z∩S1|=d ……(xvi)**

**|L1∩Z|=d…….(ix) |U1∩L1∩Z∩S1|=d….(xvii)**

The total number of possible characters can be calculated as:

N = |X|-|U1|-|L1|- Z|-|S1|+|U1∩L1|+|U1∩Z|+| L1∩Z|+| S1∩Z|+|U1∩S1|+|L1∩S1|+

|U1∩L1∩Z|+|U1∩L1∩S1|+|L1∩Z∩S1|+|U1∩Z∩S1|+|U1∩L1∩Z∩S1|

N =d -2\*(29)d-(25)d-(22)d+(23)+d+2\*(19)d+2\*(16)d+(12)d+(10)d-2\*(6)d-(9)d+0]

So, we got

**N=2698344020757897560**

We have given,

**G = 1,000,000 guesses**

T = 5 years = 5\*365.24\*24\*3600

**T = 1557784760 seconds**

and **Nu = 108**

Then,

Total numbers become N’ = N×108

**By using Eq..(i)**

**…….. (i)**

This results the total probability as:

**P = 5.773114×10-13**

In general, according to the size of the password we have the following corresponding probabilities as shown in a table below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Password Size** | 8 | 9 | 10 | 11 | 12 |
| **Probability** | 1.347…10-6 | 3.329..10-8 | 8.530…10-10 | 2.240..e-11 | 6.599…e-13 |

Attacker should start with 8 letters, then 9 and so on, so the chances increase. The probabilities for an attacker to find the combination of username and password is **5.773E-13.**

**Q.2** A hacker controls a network of compromised machines (botnet) that can be used to launch the attack. The network consists of 500,000 compromised machines (bots) located in different countries around the globe. Assume that the machines have approximately the same computing capability. The hacker uses a simple strategy consisting of slicing the username space in subsets of equal size, and assigning a subset to each of the bots to conduct the attack in parallel. Calculate the probability that a successful password guess can be obtained in the timeframe between two consecutive changes. Briefly comment the results.

**Soln.**

According to the statement in question, there is a reduction in user space by 500,000.

It means,

Nu = 108/500,000

**Nu = 200**

Then, total numbers become

**N’ = N\*200**

This results the total probability as:

**P = 2.9227232001492×10-7**

Then, according to the size of the password we should have the following corresponding probabilities as shown in a table below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Password Size** | 8 | 9 | 10 | 11 | 12 |
| **Probability** | .676932212711 | 0.001565422 | 0.000417228 | 1.110..×105 | 3.014..×107 |

In this case the probabilities improved to **14.59 %** as compared to previous case.

* **In order to strengthen the above password scheme, the bank investigates the following two different solutions:**
* Using an exponential back off scheme, i.e., introduces a delay of xn between consecutive failed authentications. The back off scheme begins when a user attempts to authenticate and fails. The system waits x0=1 second before re-prompting for the name and authentication data. If the User fails again, the system re-prompts after x1=x seconds. After n failures, the system waits xn-1 seconds.
* Using One-Time Password (OTP) tokens. A standard token displays a variable password consisting of 6 digits. For the sake of simplicity, consider that the OTP tokens are event-based.

**Q.3** Calculate the probability of successfully breaching an online account for each of the above

Options (in the time period between 2 consecutive password changes) using the botnet, i.e.:

1. Standard password scheme with exponential back off using x=1.1 second for the base delay.

**Soln.**

Let consider the use of the botnet to find the number of attempts per bot are allowed, during the time period of 5-years.

It would be clear from the table shown below, it has value of ‘X’ = 1.1, no. of attempts are described by ‘n’ and time delay to be calculated should be less than or almost equal to **T = 1557784760 seconds.**

|  |  |  |
| --- | --- | --- |
| **X** | **No. of Attempts (n)** | **Time Delay** |
| 1.1 | 1 | 1.10=1 |
| 1.1 | 2 | (1.1)1 + 1 = 2.1 |
| 1.1 | 3 | (1.1)2 + 2.1 = 3.31 |
| 1.1 | 4 | (1.1)3 + 3.31 = 4.61 |
| .. | .. | .. |
| .. | .. | .. |
| 1.1 | 172 | (1.1)171 + 119,724,153 = 131,695,669 |
| 1.1 | 173 | (1.1)172 + 131,695,669 = 144,864,336  **(Approx.. equal to T)** |
| 1.1 | 174 | (1.1)173 + 144,864,336 = 159,349,870 |

Now,

We have,

**n = 173**

**N = 2698344020757897560 …as calculated in first question**

Then,

**P = n/N**

P = 6.4×10-12

During the 5-year period, only 173 attempts per bot are allowed When considering the use of the botnet,. This gives the probability of **6.4E-12**.

1. Multifactor authentication scheme combining standard password and OTP, assuming that the test frequency for (OTP, fixed password) pairs remains virtually the same as that of single password (i.e. 1,000,000 guesses/second).

**Soln..**

Here mentioned OTP is 6 character long i.e. 106, it means m = (N+106) \* Nu

m = 269833921755900000000L

Then,

Using Eq…(i)

=5.848765540110199×10-13

and for the different sizes of password shown in a table below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Password Size** | 8 | 9 | 10 | 11 | 12 |
| **Probability** | 11.344..×10-6 | 3.233..×10-11 | 8.452..×10-11 | 2.239..×10-11 | 5.997..×10-13 |

Hence, the probability of cracking the password is **5.847E-13**

**Q.4** Discuss the benefits and limitations of each solutions and indicate (in your opinion) the best

option.

**Soln.**

The [exponential backoff](http://en.wikipedia.org/wiki/Exponential_backoff) algorithm prevents the very least slow down dictionary attacks. It increases the time between subsequent login attempts exponentially. Under this scenario, a normal user wouldn’t be able to type or navigate faster than the minimum lockout period and probably has a very low likelihood of ever hitting the limit. In contrast, if someone was to make a number of repetitive requests in a small timeframe, the time he would be locked out would rise exponentially. Well, the negative side of this scheme is that, it is effective only for long passwords.

On the other hand, OTPs are descriptions of two-factor authentication. Two-factor authentication is a type of layered security where it is not likely that both layers would be hindered by somebody using only one type of attack. There is an administrative cost of handling these elements; but, for the exponential backoff algorithm, these elements are not necessary.

From my perspective, exponential backoff algorithm is more efficient approach in the field of network security.