Chaos-Based Random Number Generator for Salt: Gacha

Rifkiansyah Meidian C. - 13511084
Informatics Major
School of Electrical Engineering and Informatics
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
rifkiansyahmeidian@gmail.com

***Abstract*—Password salts are commonly added to give more security to passwords, ensuring that they are harder to crack even if the password database were to be stolen away. However, salt are normally stored in a table, allowing whoever stole the password to crack faster with the salt. Chaos-based Random Number Generator, while chaotic, remain a pseudo random number generator due to its properties. This paper will introduce a method to create a salt that does not need to be stored in table, and can be used for dynamic salt changes if needed.**

***Keywords*—Chaos Theory, Salt, Hash.**

I. INTRODUCTION

Passwords are a key to an account security. As long as the password isn't known, the only one who could access an account would be the owner of the account. As an effect, passwords should be protected with layer upon layers of protection.

Protection of passwords can range from user's own protection by using a complex sequence of characters, secure connection when sending the table, to adding bits and hashing in order to make the password more secure.

Among those, there are one method known as *Salting*, a method that uses Salt which is extra bits added to the password before being hashed. Those extra bits are normally randomized and fixed, and normally stored in a table alongside the hashed password for authorization. New salt are generated for each new password.[1]

Usually, the salt used does not need to be changed. However, this poses a threat on the moment that an attacker managed to steal the salt table to get the password. While the salt prevents the attacker from stealing the password fairly quickly with the addition of salt, a strong enough computing ability would be able to steal the password faster still using the stolen salt table and the method known as the rainbow table[2].

Pseudo-Random Number Generator is an algorithm where a mathematical formula or precalculated table produces sequences of seemingly random number.[3]

Chaos Theory is a system where tiny, or even microscopic change in the initial conditions can result in significant changes in the behavior of the system[3]. While the result cannot be known, it is still considered pseudo-random number generator due to its ability to be reproduced using the exactly same initial condition and factors.

On this paper is the notion of using chaos-theory-based pseudo-random number generator to use for password salt with user ID and login time as the seed is proposed. The theory behind this is the thought that if the salt was stored on the table, then the attacker can use it at any time. However, by using pseudo-random number generator as salt, the need of storing salt diminished since the salt can just be checked by generating the salt the second time with the perfectly same seed which is user ID and login time. It also removes the salt from plain sight of the attackers.

The algorithm used for the pseudo-random number generator should satisfy the chaos theory, and a simple one will be used for this paper. Furthermore, the method that hashes the password can also be changed freely. However, this paper will use the SHA-1 hashing algorithm as an example.

II. THEORY BASIS

1. Random Number Generator

An Ideal Random Number Generator is a discrete memoryless information source that generates equiprobable symbols[4]. An RNG is made to generate sequence of independent and identically distributed random variables.

In practice, there are two types of random number generator. The first is Pseudo-random Number Generator, and the second is True Random Number Generator.

Pseudo-random Number Generator works by using a certain algorithm to generate a seemingly random number or sequence of numbers from predetermined seeds. Reusing those seed will result in the same number to appear when the algorithm is executed. Due to this, PRNG is efficient, deterministic and periodic, which made it useful for simulation and modeling purposes.

True Random Number Generator works by generating a totally random number without any seeds or inputs whatsoever. In theory, it introduces randomness from physical phenomena to use in a computer[5]. TRNG

generally inefficient due to the time required to produce number. It is also nondeterministic where none of the initial situation contributes to the result of the RNG.

2. Chaos Theory

A chaotic system is a system where tiny changes of the initial conditions of the system can result in dramatic changes of the overall behavior of the system.

Among mathematical equations that can create chaotic behavior, one such equation is the logistic equation:

$$X(n+1) = R\,X(n)\,(1 - X(n))$$

Where R is a parameter, X(n) is the variable at the n^{th} iteration value between 0 and 1, and n can be considered a running variable.[8]

The sensitivity of the above diagram is at best between 3.57 and 4, as shown by figure 2.1. Because of that, R is required to be between the two numbers.

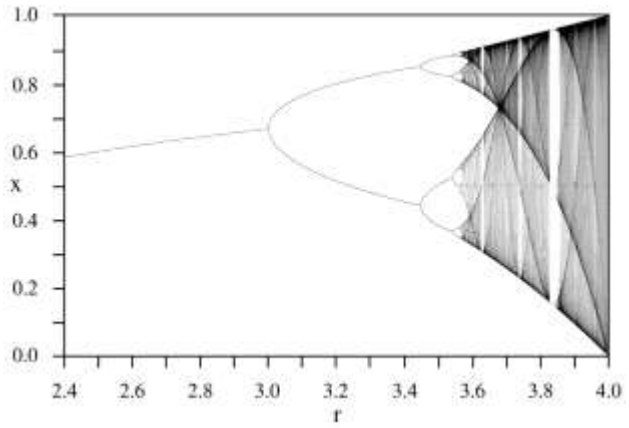

Figure 2.1 Bifurcation diagram of the logistic map.(en.wikipedia.org/wiki/Chaos_theory)

Mathematically, the condition for a system to be defined as chaotic is as follows:

1. Having a dense collection of points with periodic orbits,
2. Being sensitive to the initial condition, and
3. Being topologically transitive.[6]

Despite the seemingly random appearance of results, chaos remains deterministic due to the fact that if the initial condition is exactly the same, then the result will exactly be the same.

3. Salt

Salt are fixed bits generated to be added to a password before hashing to increase security of the password. Normally, Salt are generated with true random generator since there is no need for any deterministic attribute of the Salt. It is then stored in a salt table along with the hashed password which had been concatenated with the Salt. Salt makes the password hashes significantly harder to crack[7].

III. DESIGN

For the proposed random number generator, User ID will be used along with the timestamp of the last time user login as a seed for the algorithm. The time when user registers will be treated as a login time for the first time. The mathematical equation that will be used for the random number generator is the logistic equation on part II.

Since x is required to be between 0 and 1, adjustments will be made for the User ID and timestamp to fit on the equation.

As an example, the following user ID and timestamp is used.

ID: 1086
Time: 15/05/2016, 18:13

Thus, the adjustment will be as follows:

Seed I(from ID): 0.1086
Seed II(from timestamp): 0.15052016
Iteration(from timestamp): 1813

Since the iteration requires to be 1000 or above, the number of iteration will be 1000+n, where n is the number of iteration from the timestamp. For the case above, the total iteration will be 2813.

Also, because the iteration requires R to be between 3.57 and 4 to be chaotic, alterations need to be made for the equation to fit. The process for the variables above are as following:

```
real seedI = 0.1086;
real seedII = 0.15052016
real n = 3.57;
int iterate = 2813;

bifurcation(y,r) {
return r*y*(1-y);
}

n = n+seedII;

for( int q=0; q<iterate; q++){
        bifurcation(seedI,n);
```

In theory, Seed II will not go over 0.43 due to the properties of the timestamp, eliminating the need to taking care of numbers outside 3.57 and 4.

Next, the resulting number will be concatenated with a password then hashed with a hashing algorithm.

IV. ANALYSIS

The result of chaos equation above is as follows:

```
run:
0.3601688234297664
0.8573836098968505
0.4549338775839053
0.922573829528849
0.26576180971753044
0.7259942892987862
0.7401103557306111
0.7156309547282197
0.7571380981423822
0.6841292413794445
0.8039910962999041
0.5863145894470815
0.9024113896323481
0.32764788132691325
0.8196110480583991
0.5500743590278666
0.9208010536020325
0.27132441406354535
0.7355746517163457
0.7236582240077845
0.7440184556970166
```

Figure 4.1, Several Results of Logistic Equation.

```
run:
0.3611368838297664
0.8606945502016545
0.44728728351996855
0.9222643029820033
0.26745165369040774
0.7308882349310034
0.733758334274207
0.7287833804354367
0.7373677687112941
0.7224396354533733
0.7480461728008101
0.7031026848382855
0.7787434698404024
0.6427763756684743
0.8565830379249271
0.4582889441446173
0.9261396355857667
0.25518627251646847
0.7090459357116522
0.7696055508261842
0.6614691502251142
```

Figure 4.2, Several Results of second run.

The 2813rd result is 0.806397969308579, which will be concatenated with the password and hashed.

As an example, the password used is admin, which concatenated resulting in: admin0.806397969308579. The resulting SHA-1 hash is

6cdf5bc241aa6d9c6dd5b9884a136f79e42475b9

A possible defect in this system is when the local time is tampered with, which can cause failure in authorization when the user logs in next time. A possible solution is to use system time synchronized to the internet so the actual time used is always the same time.

A second run is done with the same parameters, except that the date is 16/05/2016, which translates into the seed 0.16052016. The 2813rd result is 0.7473897118290384, which concatenated with the password 'admin' results in SHA-1 hash of:

8886accdee538621e10e50b33d2ba2a1b805789b

Another run with the parameters of the first run results in the same number of 0.806397969308579. This makes it viable for the user authorization to work by using the data from last login time. Also, with the second run it is proven that changing 1 number makes the result differ greatly, which offers protection that is further extended by hashing.

REFERENCES

[1] https://en.wikipedia.org/wiki/Salt_(cryptography), Salt(cryptography). Visited on 04/05/2016

[2] http://goodmath.scientopia.org/2013/03/02/passwords-hashing-and-salt/, Passwords, Hashing, and Salt. Visited on 05/05/2016

[3] https://www.random.org/randomness/, Pseudo-Random Number Generator. Visited on 05/05/2016

[4] Stojanovki, Toni and Kocarev, Ljupco. *Chaos-Based Random Number Generators—Part I: Analysis*. IEEE Transactions on Circuits and Systems, 2001.

[5] https://www.random.org/randomness/, *True Random Number Generators(TRNGs)*, last visited 15/05/2016

[6] http://mathworld.wolfram.com/Chaos.html, *Chaos*. Visited on 16/05/2016

[7] http://php.net/manual/en/faq.passwords.php#faq.passwords.salt, *What is a salt?*. Visited on 17/05/2016

[8] https://universe-review.ca/R01-09-chaos.htm, *Bifurcation*. Visited on 17/05/2016

[9] http://introcs.cs.princeton.edu/java/94diffeq/, *Numerical Solutions to Differential Equations*, Creative Exercises. Visited on 18/05/2016

Bandung, 18 Mei 2016

Rifkiansyah Meidian Cahyaatmaja , 13511084