International Journal of Computer Applications Technology and Research
Volume 6–Issue 4, 190-193, 2017, ISSN:-2319–8656

An Examination of the Bloom Filter and its Application in Preventing Weak Password Choices

Nancy Cheng
Department of Computer Science
Stanevagra University

Fabio Rocca
Department of Computer Science
Stanevagra University

Abstract: Choosing weak passwords is a common issue when a system uses username and password as credentials for authentication. In fact, weak passwords may lead to system compromising. Lots of approaches have been proposed to prevent user from selecting weak or guessable passwords. The common approach is to compare a selected password against a list of unacceptable passwords. In this paper we will explain space-efficient method, which is called Bloom Filter, of storing a dictionary passwords. The time complexity in this approach is constant time, no matter how many passwords we have in our dictionary.

Keywords: Weak passwords; Bloom filter; Authentication; Security; Web Security.

1. INTRODUCTION

Authentication is an absolutely essential element of having a secure environment in digital world. In this process the identity of the user (or in some cases, a machine) will be checked against the some credentials that he/she has provided before for a system[3,9, 12]. If they are matched the user will be authenticated and forwarded for the next step which is usually authorization. There are several approaches for authentication which are selected based on the system administrator policies and available infrastructures. One of the basic authentication mechanism is using user name and password to provide some fundamental security characteristics. This technique has been a part of security since long time ago[2,6,11]. However, now a days, systems administrators need to re-examine their password security policies to remain effective against modern programs and computers that can crack weak passwords in minutes. But it has always been a concern that how we can select a proper and strong password that cannot be predicted or cracked easily. Lots of approaches and techniques have been proposed in this area that try to generate a complicated combination of characters and numbers as password[14,15]. In the most cases it works but it is not always comfortable for the user to have or memorize a system generated password, instead if we could have a dictionary of weak passwords which are easy to recognize by attackers and check them against the entered password proposed by user we can easily decide the selected password by end user is strong enough or not. Having a dictionary of weak passwords to be checked against the entered password usually involves with reading the whole dictionary which needs extra facilities and hardware but there is a data structure model that can address this issue easily which is called Bloom Filter. Bloom filter is an efficient data structure in terms of space usage and lookup time that will be explained in details in the next section.

2. BLOOM FILTER

Bloom filter was introduced in “Space/time trade-offs in hash coding with allowable errors” paper by Burton H. Bloom[1]. In this paper he compared the space/time trade-offs of different types of hash-tables[12,14]. He found that a new type of hash-table, which is now known as a Bloom filter needed less time to reject elements that are not in the table and less space to store these elements[11]. Bloom filter as an efficient probabilistic data structure has been used in different areas such as checking for viruses in network packets, spam control in email, web caching, spell, password checking and etc [3,18]. It consists of a bit array of m bits which are all set to zero, adding the elements to the m bits of array can be done through the k different hash functions. Based on the output of each hash function the particular position in array will be changed to 1 and later on, the structure can be queried for the membership of elements. So the elements themselves are not stored in the Bloom filter, only their membership [5]. Figure 1 shows the

structure of bloom filter.

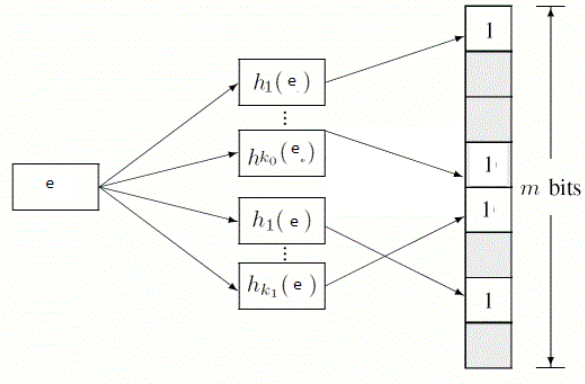

Figure 1: Bloom filter structure [5].

For the coming request after applying k hash functions, the bloom filter will be checked If one or more of mapped bits are still 0, the element is certainly not in the set. If all bits are 1, the element was probably in the set, although there is a small probability that the tested bits were set to 1 due to the addition of different elements. Then we have a false positive. There is a trade-off between the probability of false positives and the size of the Bloom filter (m), number of hash functions (k) and number of items in the set (n)[5]. The false-positive probability can be calculated from m and k in the following way. The probability p of one of the m bits still being zero after the addition of n elements is[5]

$$p = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}.$$

And the probability of a false positive f is then equal to the probability that all the k bits that we test are equal to 1, which is equal to

$$f = (1-p)^k = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k.$$

From simple calculations it follows that for a given m and n, the value of k that minimizes the false- positive probability f is equal to:

$$k = \frac{m}{n} \ln 2$$

Which gives the probability of

$$f = \left(\frac{1}{2^{\ln 2}}\right)^{m/n} \approx 0.62^{m/n}$$

There are some main parameters that affects the accuracy in bloom filters, table 1 shows the key components in bloom filters [5].

Table 1: Key Bloom Filter Parameters

Parameter	Outcome
Number of hash functions (k)	More computation, lower false positive rate
Size of filter (m)	More space is needed, lower false positive rate
Number of elements in the set (n)	Higher false positive rate

3. EXPERIMENT AND EVALUATION

In the current scenario the dataset of popular weak password is needed, so the dataset of weak passwords with 300,000 passwords was selected. For this project two approaches experimental and theoretical will be calculated and compare together to have a good inside about the implemented version of bloom filter besides having some optimal values for the other parameters. The key practice in this project is to calculate the lowest false positive rate for the implementation through the adjusting the key parameters. To address that, the list of 30,000 strong password was used to calculate the false positive rate from empirical point of view.

The first experiment is to calculate the false positive rate based on different hash functions to have a basic idea about the some optimal numbers of hash functions. In bloom filter using more hash functions may be considered as a way to reach the lowest positive rate, it is true but at some points adding more hash functions to the system does not work and from those points the false positive rate again starts to grow. In practice, hash functions yielding sufficiently uniformly distributed outputs, such as MD5 ,CRC32 ,SHA1, SHA2, murmur, FNV which are useful for most probabilistic filter purposes like bloom filter [5].

For the first experiment the m/n = 6 was selected and different hash functions were applied. Figure 2 shows the details about two approaches

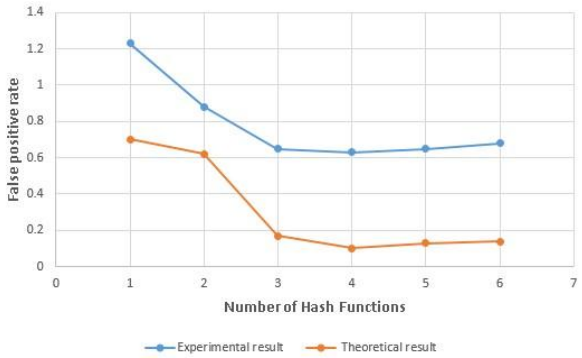

Figure 2: False positive rate through different number of hash functions

As figure 2 depicts the experimental result has a little higher error in comparison with the theoretical one but almost both are following the same trend. From the theoretical result point of view, K=4 plays an optimal solution for the given data and the experimental one confirms this result. Table 2 shows the exact result for both cases.

Table 2: False positive rate through different number of hash functions

K	Experimental result	Theoretical result
1	1.23	0.7
2	0.88	0.62
3	0.65	0.17
4	0.63	0.1
5	0.65	0.13
6	0.68	0.14

For the next step the false positive rate is calculated based on the different bloom filter sizes (m). Figures 3, 4 and 5 shows the different values for false positive rates while we are applying different number of hash functions

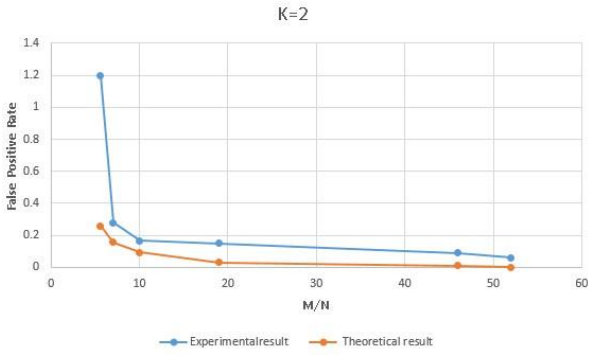

Figure 3: False positive rate with different size of bloom filter (K=2)

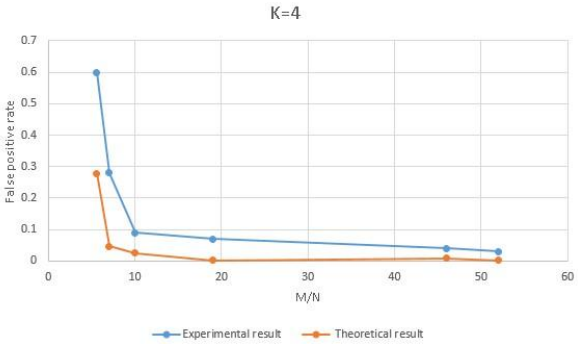

Figure 4: False positive rate with different size of bloom filter (K=4).

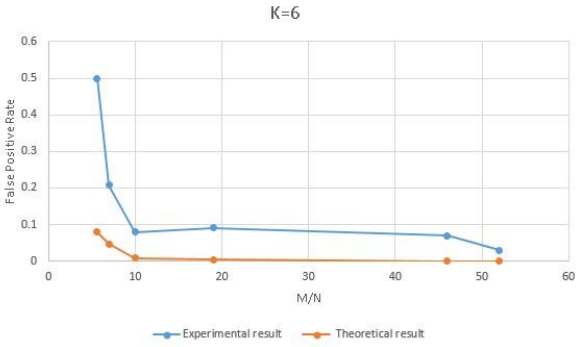

Figure 5: False positive rate with different size of bloom filter (K=6).

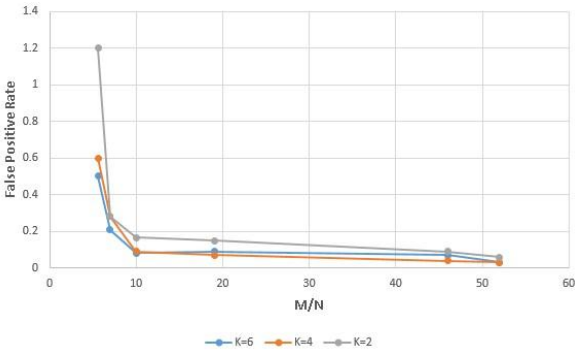

Figure 6: False positive rate with different size

of bloom filter and different number of hash functions. As the figure 6 shows the lowest false positive rate can be achieved with a bloom filter with m/n=50. Although, it can be considered as an optimal solution for this scenario but if we had some restrictions in regards of required space, we can consider the m/n=10 with number of hash functions equal to 4. In fact, it depends on the domain and how accurate we expect to receive the outcome from the bloom filter. In the current case as we do not have that much sensitivities we can take K=4 with m/n=10 as an optimal solution.

4. CONCLUSION

Bloom filter is a compact data structures for probabilistic representation of a set in order to support membership queries[7,8]. It has a strong space advantage over other data structures for representing sets and it also comes with lookup time efficiency in comparison with other approaches, but as may be expected, there is always a tradeoff and for this case the false positive is a tradeoff for space and time efficiency. Based on the domain and acceptable false positive rate, the bloom filter can be adjusted to achieve the optimal lookup time beside space efficiency. In this paper, based on the input data and admissible false positive error rate, 4 independent hash functions with m/n =10 were selected as an optimal solution.

REFERENCES

[1] Spafford, Eugene H. "Preventing weak password choices." (1991).

[2] Broder, Andrei, and Michael Mitzenmacher. "Network applications of bloom filters: A survey." *Internet mathematics* 1, no. 4 (2004): 485-509.

[3] Ganesan, Ravi, and Christopher I. Davies. "Method and system for proactive password validation." U.S. Patent 5,394,471, issued February 28, 1995.

[4] Mitzenmacher, Michael. "Distributed, compressed Bloom filter Web cache server." U.S. Patent 6,920,477, issued July 19, 2005.

[5] Porat, Ely. "An optimal Bloom filter replacement based on matrix solving." In *International Computer Science Symposium in Russia*, pp. 263-273. Springer Berlin Heidelberg, 2009.

[6] Pouriyeh, Seyed Amin, and Mahmood Doroodchi. "Secure SMS Banking Based On Web Services." In SWWS, pp. 79-83. 2009.

[7] Weirich, Dirk, and Martina Angela Sasse. "Pretty good persuasion: a first step towards effective password security in the real world." In *Proceedings of the 2001 workshop on New security paradigms*, pp. 137-143. ACM, 2001.

[8] Hao, Fang, Murali Kodialam, and T. V. Lakshman. "Building high accuracy bloom filters using partitioned hashing." In *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, no. 1, pp. 277-288. ACM, 2007.

[9] Shanmugasundaram, Kulesh, Hervé Brönnimann, and Nasir Memon. "Payload attribution via hierarchical bloom filters." In *Proceedings of the 11th ACM conference on Computer and communications security*, pp. 31-41. ACM, 2004.

[10] Kirsch, Adam, and Michael Mitzenmacher. "Distance-sensitive bloom filters." In *2006 Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pp. 41-50. Society for Industrial and Applied Mathematics, 2006.

[11] Pouriyeh, Seyed Amin, Mahmood Doroodchi, and M. R. Rezaeinejad. "Secure Mobile Approaches Using Web Services." In SWWS, pp. 75-78. 2010.

[12] Zhong, Ming, Pin Lu, Kai Shen, and Joel Seiferas. "Optimizing data popularity conscious bloom filters." In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pp. 355-364. ACM, 2008.

[13] Mitzenmacher, Michael. "Bloom filters." In *Encyclopedia of Database Systems*, pp. 252-255. Springer US, 2009.

[14] Allahyari, Mehdi, Krys J. Kochut, and Maciej Janik. "Ontology-based text classification into dynamically defined topics." In Semantic Computing (ICSC), 2014 IEEE International Conference on, pp. 273-278. IEEE, 2014.

[15] Tarkoma, Sasu, Christian Esteve Rothenberg, and Eemil Lagerspetz. "Theory and practice of bloom filters for distributed systems." *IEEE Communications Surveys and Tutorials* 14, no. 1 (2012): 131-155.

[16] Melicher, William, Blase Ur, Sean M. Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. "Fast, lean and accurate: Modeling password guessability using neural networks." In *Proceedings of USENIX Security*. 2016.

[17] Laufer, Rafael P., Pedro B. Velloso, and O. C. M. B. Duarte. "Generalized bloom filters." *Electrical Engineering Program, COPPE/UFRJ, Tech. Rep. GTA-05-43* (2005).

[18] Allahyari, Mehdi, and Krys Kochut. "Automatic topic labeling using ontology-based topic models." In *Machine Learning and Applications (ICMLA), 2015 IEEE 14th International Conference on*, pp. 259-264. IEEE, 2015.

[19] Morris, Robert, and Ken Thompson. "Password security: A case history." *Communications of the ACM* 22, no. 11 (1979): 594-597.

[20] Kaufman, Charlie, Radia Perlman, and Mike Speciner. *Network security: private communication in a public world*. Prentice Hall Press, 2002.

[21] Rosenberg, Jothy, and David Remy. *Securing Web Services with WS-Security: Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption*. Pearson Higher Education, 2004.

[22] Rubin, Aviel D., Daniel Geer, and Marcus J. Ranum. *Web security sourcebook*. John Wiley & Sons, Inc., 1997.