International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 10, ISSN No. 2455-2143, Pages 51-54
Published Online August - September 2016 in IJEAST (http://www.ijeast.com)

PASSWORD HASHING

Pankit Arora
Department of IT
Amity University, Noida, India

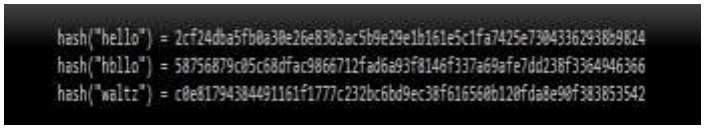
Akshath Dhar
Department of IT
Amity University, Noida, India

***Abstract*— Passwords play a critical role in online authentication. Unfortunately, passwords suffer from two seemingly intractable problems: password cracking and password theft. In this paper, we propose PasswordAgent, a new password hashing mechanism that utilizes both a salt repository and a browser plug-in to secure web logins with strong passwords. Password hashing is a technique that allows users to remember simple low-entropy passwords and have them hashed to create high-entropy secure passwords. PasswordAgent generates strong passwords by enhancing the hash function with a large random salt. With the support of a salt repository, it gains a much stronger security guarantee than existing mechanisms. PasswordAgent is less vulnerable to offline attacks, and it provides stronger protection against password theft. Moreover, PasswordAgent offers some usability advantages over existing hash-based mechanisms, while maintaining users' familiar password entry paradigm. We build a prototype of PasswordAgent and conduct usability experiments.**

Keywords— **Passwords, Password Hashing, Account Verification**

I. INTRODUCTION

HASHMETHODS



Hash methods are one-way features. Any quantity of information can be modified into an irreversible fixed-length "fingerprint" using these features. They also have the feature that if the feedback changes by even a small bit, the causing hash is absolutely distinct. This is excellent for defending security passwords, because we want to shop security passwords in a type that defends them even if the security password information file itself is affected, but simultaneously, we need to be able to confirm that a customer's security password is appropriate.

ACCOUNT VERIFICATION PROCESS

The common work-flow for account verification and signing up in a hash-based account program is as follows:

i. An account is created by the user.
ii. The database stores their password after hashing. The hard drive never stores the plain-text (unencrypted) password.
iii. The hash of the password they entered is checked against the hash of their real password, whenever the user attempts to login.
iv. Access is granted to the user if the hashes match. Else, the user is asked to enter valid login credentials.
v. Steps iii and iv repeat whenever someone tries to login to their account.

In phase 4, never tell the customer if it was the login name or security password they got incorrect. Always show a general concept like "Invalid login name or security password." for avoiding attackors from enumerating legitimate usernames without understanding their security passwords.

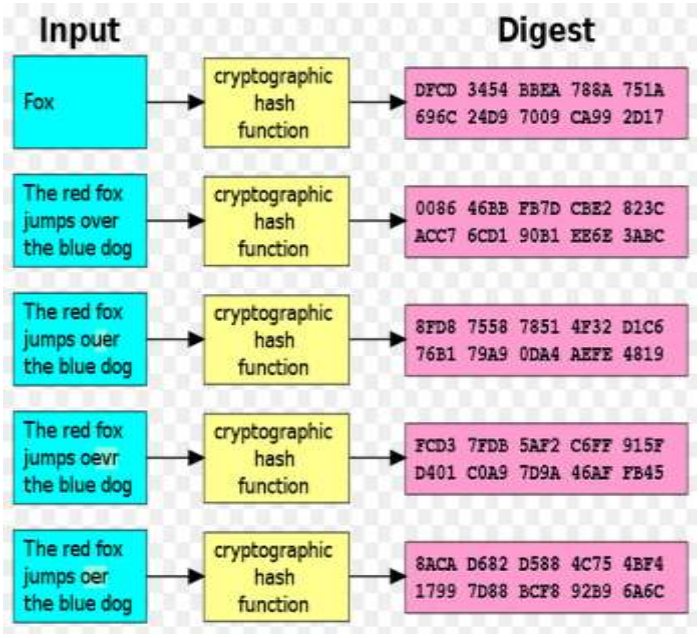


Fig 1.2 Cryptographic hash Functions

It should be pointed out that the hash-functions used to secure passwords are not the identical to the hash-functions you could possibly have seen in a D.S course. Only cryptographic hash-functions could be made use of to apply password-hashing.

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 10, ISSN No. 2455-2143, Pages 51-54
Published Online August - September 2016 in IJEAST (http://www.ijeast.com)

"Hash-functions like SHA256, SHA512, RipeMD, and WHIRLPOOL are cryptographic hash features."

It's basic to think that all you have to do is run the protection password through a cryptographic hash operate & users' protection passwords will be protected. This is far from the fact. There are many methods to restore protection passwords from basically hashes very easily. There are several easy-to-implement methods that create these "attacks" much less efficient. To encourage the need for these methods, consider this very web page. On the first web page, you can publish a record of hashes to be damaged, and get outcomes in less than a second. Clearly, basically hashing the protection password does not fulfill our needs for protection.

II. METHODOLOGY

2.1 HASH CRACKING

2.1.1 Dictionary and Brute Force Attacks

A dictionary strike uses a computer file that contains words, terms & other post that can be used as a security password. Every single term in the computer file is hashed and then compared to security password hash. If they coordinate, that term is the security password. Further handling is often used to dictionary information, such as changing terms with their "leet speak" counterparts ("hello" becomes "h3110"), to make them more effective.

A brute-force enemy tries each possible mixture of figures up to a given length. These strikes are very computationally expensive, and are usually the least effective with regards to hashes damaged per processer time, but they will always gradually look for the security password. Security passwords should be lengthy enough that searching through all possible personality post to discover it will take a lengthy time to be beneficial.

2.1.2 Lookup Tables

Lookup-Tables are an efficient means for breaking many hashes of the similar kind very easily. The common concept is to pre-compute the hashes of the security passwords in a security password vocabulary and shop them, and their corresponding security password, in a search desk information framework. A good execution of a search desk can procedure thousands of hash queries per second, even when they contain many immeasurable hashes.

- The attacker doesn't have to pre-compute a lookUp table for applying a dictionary or brute-force attack.
- First of all, a lookUp desk is designed by the enemy which charts the record of the customers having that hash to each security password hash including in the data source. The enemy then hashes each security password think and uses the search desk to get a record of customers whose security password was the assailant's think. This strike is especially efficient because it is typical for many customers to have the same security password.

2.1.3

- ***Rainbow Tables :*** *"Rainbow platforms are a time-memory trade-off strategy. They are like search platforms, except that they compromise hash breaking rate to make the search platforms more compact." Because they are more compact, the alternatives to more hashes can be saved in the same amount of area, making them more efficient. Spectrum platforms that can break any md5 hash of a security password up to 8 figures are available more time.*

2.2 ADDING SALT

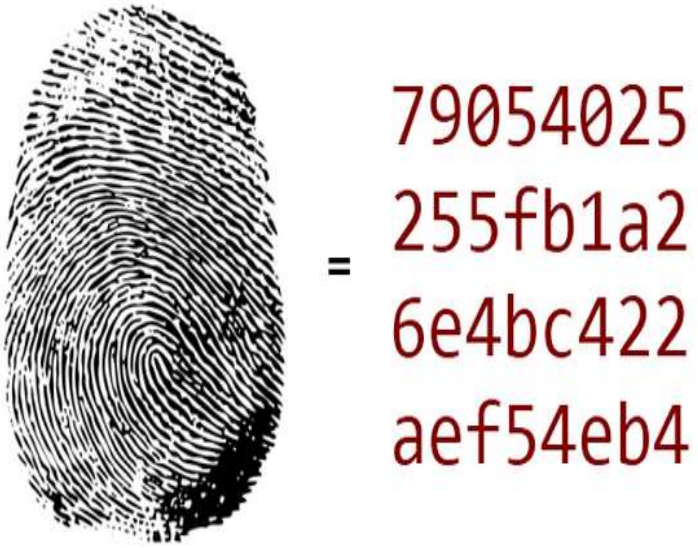


Fig 2.3 Adding Salt

Lookup-tables & rainbow-tables perform only when each security password is hashed in the identical way.two customers will have identical hashes if they have identical security passwords. So the hashes need to be randomized to avoid this strike.

This can be done by appending or prepending a unique sequence, known as a salt, to the security password before hashing. As proven in the example above, this creates the same security password hash into a absolutely different sequence whenever. To examine if a security password is appropriate, we need the salt, so it is usually saved in the customer consideration data source along with the hash, or as aspect of the hash sequence itself.

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 10, ISSN No. 2455-2143, Pages 51-54
Published Online August - September 2016 in IJEAST (http://www.ijeast.com)

"The salt does not need to be key. Just by randomizing the hashes, search platforms, opposite search platforms, and spectrum platforms become worthless."

2.3 SALT IMPLEMENTATION ERRORS

*1) **Salt Reuse***

Using same salt in each hash is the most typical error. This causes ineffectively as the two customers will have the same hash for same security passwords. An enemy can still use a reverse-lookUp-Table strike to run a dictionary strike on every hash simultaneously.

"A new unique salt must be produced every time a customer makes an consideration or changes their security password."

*2) **Short Salt***

An attackor can quickly develop a search desk for any possible brief salt. For eg.," if the sodium has only 3 ASCII figures, therr are only "95x95x95 = 857,375 possible salts". This might seem to be a lot, but if each search desk contains only 1MB of the most typical security passwords, jointly they will be only 837GB."

This is why, the login name must not be applied as a salt. They might be exclusive to a particular support, but they are foreseeable and often recycled for records on other solutions.

"To create it difficult for an enemy to create a search desk for every possible salt, the salt must be lengthy. A excellent principle is to use a salt that is the same dimension as the outcome of the hash operate. For example, the outcome of SHA256 is 256 pieces (32 bytes), so the salt should be at least 32 exclusive bytes."

2.4 HASH COLLISIONS

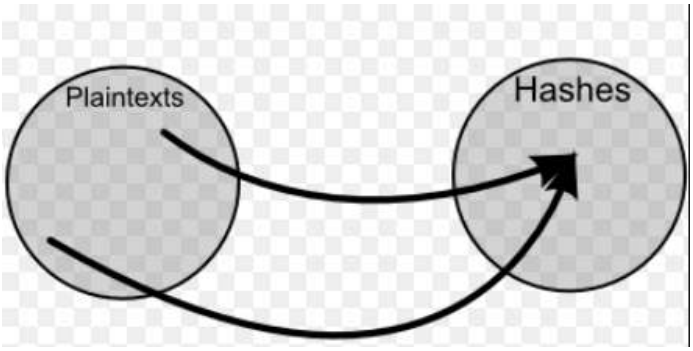


Fig 2.4 Hash Collisions

There shall be some information hashing into the same sequence as hash-functions map irrelavent quantities of information to fixed-length post. Cryptographic hash features create these crashes very hard to discover. Every now and then, cryptographers discover "attacks" on hash features that create discovering collisions simpler. A latest eg. is the MD5 hash operate, for which collisions have actually been discovered.

Collision strikes are a indication that it can be more likely for a sequence except the customer's security password to have the identical hash. However, discovering crashes in even a poor hash operate like MD5 needs a lot of devoted processing energy, so it is very unlikely that these crashes will occur "by accident" in exercise. Nevertheless, it's a smart concept "to use a more protected hash operate like SHA256, SHA512, RipeMD, or WHIRLPOOL if possible."

2.5 PROPER HASHING TECHNIQUES

Platform	CSPRNG
PHP	mcrypt_create_iv, openssl_random_pseudo_bytes
Java	java.security.SecureRandom
Dot NET (C#, VB)	System.Security.Cryptography.RNGCryptoServiceProvider
Ruby	SecureRandom
Python	os.urandom
Perl	Math::Random::Secure
C/C++ (Windows API)	CryptGenRandom
Any language on GNU/Linux or Unix	Read from /dev/random or /dev/urandom

Fig 2.5 Software Requirements

The salt needs to be exclusive per-user per-password. Every time a customer makes an consideration or changes their security password, the security password should be hashed using a new exclusive salt. Never recycling a salt. The salt also needs to be lengthy, so that there are many possible salt. As a concept, make your salt is at least provided that the hash function's outcome. The salt should be saved in the customer consideration desk plus the hash.

TO STORE A PASSWORD

i. Create a lengthy unique salt using a CSPRNG.

ii. Prepend the salt to the security password and hash it with a conventional cryptgraphic hash operate such as SHA256.

iii. Conserve both the salt and the hash in the customer's data source history.

TO VALIDATE A PASSWORD

i. Access the customer's salt and hash from the data source.

ii. Prepend the salt to the given security password and hash it using the same hash operate.

iii. Evaluate the hash of the given security password with the hash from the data source. If they coordinate, the

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 10, ISSN No. 2455-2143, Pages 51-54
Published Online August - September 2016 in IJEAST (http://www.ijeast.com)

security password is appropriate. Otherwise, the security password is wrong.

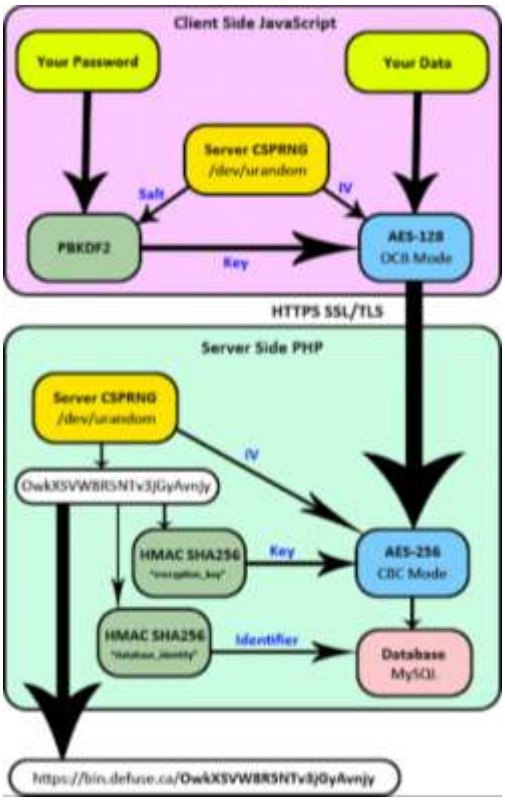


Fig 2.6 DFD

III. CONCLUSION

3.1 JAVA CODEOUTPUT

```
1000:a2800f90f88fda1c8214ac23b431ba1b0da911a3725b60b0:c43480864e720d3fba6b4361d6d9c6
a7fbad0629d23281fd
1000:325c8694e8777c7e8161b55b35e248e9f23bb59a0d444bb2:d387b6ae038dc530f9a36cbbdb7c7a
886756be46a9d56e44
1000:f6a371ae494bdb780661279b32ad2528f9a37d69695d6eb7:0f566346c8a04b618faec8b03db486
da43096956ed1e2f74
1000:8bc114538d0abf7a756e387ccb5f8c956d3a18c38658f140:175d8ebc2222d2658302888c694893
c097c3044d4faf11ee
```

Fig 3.1 Code1

```
1000:de1b390b783894767546bb57c8f64f8ffb714ba78045c49f:8067e1c81441250d5e3379af3d2aed
788c59f48eed70e1ac
1000:3317c166595b05cf9ee9c71b6759ae45b74487d5747f4da7:a8808c51eb5782b3dead138b87cd7b
d3e179f948e77a4940
1000:b85ee7132be537717059de4ce07a75517c482448666f4ec5:7bcc52ff8d3ae6eae4f213117eef05
a06f509419cfec7c5f
1000:a70cac84dc0e090ca41ec04b8899bdd52bb615cf7a9f1b62:b543b3f528363969e40600ce34f975
7d3f78fe6add57fcbc
```

Fig 3.2 Code2

```
7d3f78fe6add57fcbc
1000:5011b70726beca92daa9fb1f42f8dc211aebcb401b4c6b37:a50cc6e565f656f6a95349322293fd
eb15b83bc3dd734d0e
1000:4d27cfe924ad5a18fd78145d758e133c6eb350352ac0d069:f1f0ef71cb848b4ba092f4d4895b06
45b76df146868a350e
Running tests...
TESTS PASSED!
```

Fig 3.3 Code3

The report explains that how password hashing works exactly. All the concepts and topics have been included in the project report. Passwords should be hashed with either PBKDF2, bcrypt or scrypt, MD-5 and SHA-3 should never be used for password hashing and SHA-1/2(password+salt) are a big no-no as well. Currently the most vetted hashing algorithm providing most security is bcrypt. PBKDF2 isn't bad either, but if you can use bcrypt you should. Scrypt, while still considered very secure, hasn't been around for a long time, so it doesn't get recommended a lot, but it seems it will become the successor of bcrypt, once it has been around a bit longer. Note that while there are some caveats and password bruteforcing strategies for PBKDF2 and bcrypt, they are still considered unfeasable for strong passwords (passwords longer than 8 characters, containing numbers, letters, signs and at least one capital letter).

IV. REFERENCE

[1] "https://www.google.co.in/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0CAYQjB1qFQoTCPLb0Zjw3MYCFUHnpgodiisAkw&url=http%3A%2F%2Fomaristraker.blogspot.com%2F&ei=Bi-mVfLAPMHOmwWK14CYCQ&bvm=bv.97653015,d.ddGY&psig=AFQjCNFxSDotyF6fXyPl65nbpbXPTqO-sQ&ust=1437040739403191"

[2] "security.stackexchange.com/questions/.../how-to-securely-**hash-password**..."

[3] "www.codeproject.com"

[4] "https://en.wikipedia.org/wiki/Cryptographic_**hash**_function"

[5] "https://www.google.co.in/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0CAYQjB1qFQoTCM_YkLjx3MYCFaI0pgodx3cAfA&url=http%3A%2F%2Fonewebsql.com%2Fblog%2Fhow-to-store-passwords&ei=VTCmVY_aHKLpmAXH74HgBw&bvm=bv.97653015,d.dGY&psig=AFQjCNFxSDotyF6fXyPl65nbpbXPTqO-sQ&ust=1437040739403191"

[6] https://www.google.co.in/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0CAYQjB1qFQoTCJytltHw3MYCFePYpgodLXYISg&url=http%3A%2F%2Fblog.codinghorror.com%2Fspeed-hashing%2F&ei=fS-mVdz6IeOxmwWt7KHQBA&bvm=bv.97653015,d.dGY&psig=AFQjCNFxSDotyF6fXyPl65nbpbXPTqO-sQ&ust=1437040739403191

[7] https://media.blackhat.com/us-13/US-13-Aumasson-Password-Hashing-the-Future-is-Now-WP.pdf

[8] https://www.usenix.org/legacy/event/lisa09/tech/full_papers/strahs.pdf