Replacing plain text password for app

Asked 16 years, 3 months ago Modified 12 years, 3 months ago

Viewed 1k times

6

We are currently storing plain text passwords for a web app that we have.

I keep advocating moving to a password hash but another developer said that this would be less secure -- more passwords could match the hash and a dictionary/hash attack would be faster.

Is there any truth to this argument?

passwords

Share

Improve this question

Follow

edited Sep 12, 2008 at 14:14

Community Bot

1 ● 1

asked Sep 12, 2008 at 13:50

Jack Bolding

3,829 ● 3 ● 42 ● 45

14 Answers Sorted by: Highest score (default)

15

Absolutely none. But it doesn't matter. I've posted a similar response before:

It's unfortunate, but people, even programmers, are just too emotional to be easily be swayed by argument. Once he's invested in his position (and, if you're posting here, he is) you're not likely to convince him with facts alone. What you need to do is switch the burden of proof. You need to get him out looking for data that he hopes will convince you, and in so doing learn the truth. Unfortunately, he has the benefit of the status quo, so you've got a tough road there.

Share Improve this answer Follow

edited Sep 12, 2008 at 15:02

answered Sep 12, 2008 at 13:54

Joel Coehoorn

415k ● 114 ● 577 ● 813

good catch there--it really is an emotional response from the OP's coworker. Addressing the warm fuzzies will probably be more productive that hard logic and metrics. Maybe push the hashing algo choice onto the colleague so they have more buy in to the proper solution? – Stu Thompson Sep 12, 2008 at 14:02

You have a very good point here. I did not mention in the question, but this was his code -- he implmenented it this way. Its now my code so I could change it unilaterly but I would prefer to allow him to save face/agree in case I need further help understanding the code... – Jack Bolding Sep 12, 2008 at 14:15

6

From Wikipedia

> Some computer systems store user passwords, against which to compare user log on attempts, as cleartext. If an attacker gains access to such an internal password store, all passwords and so all user accounts will be compromised. If some users employ the same password for accounts on different systems, those will be compromised as well.
>
> More secure systems store each password in a cryptographically protected form, so access to the actual password will still be difficult for a snooper who gains internal access to the system, while validation of user access attempts remains possible.
>
> A common approache stores only a "hashed" form of the plaintext password. When a user types in a password on such a system, the password handling software runs through a cryptographic hash algorithm, and if the hash

> value generated from the user's entry matches the hash stored in the password database, the user is permitted access. The hash value is created by applying a cryptographic hash function to a string consisting of the submitted password and, usually, another value known as a salt. The salt prevents attackers from building a list of hash values for common passwords. MD5 and SHA1 are frequently used cryptographic hash functions.

There is much more that you can read on the subject on that page. In my opinion, and in everything I've read and worked with, hashing is a better scenario unless you use a very small (< 256 bit) algorithm.

Share Improve this answer

Follow

answered Sep 12, 2008 at 13:56

palehorse

27.4k ● 4 ● 42 ● 49

5

There is absolutely no excuse to keeping plain text passwords on the web app. Use a standard hashing algorithm (SHA-1, not MD5!) with a salt value, so that rainbow attacks are impossible.

Share Improve this answer

Follow

edited Sep 20, 2012 at 12:34

answered Sep 12, 2008 at 13:53

qbeuek
4,206 ● 4 ● 21 ● 17

md5 is completely inappropriate for password hashes
– Joel Coehoorn Jul 6, 2012 at 0:10

3

I don't understand how your other developer things 'more passwords could match the hash'.

There is argument to a 'hash attack would be faster', but only if you're not salting the passwords as they're hashed. Normally, hashing functions allow you to provide a salt which makes the use of known hash table a waste of time.

Personally, I'd say 'no'. Based on the above, as well as the fact that if you do somehow get clear-text expose, a salted, hashed value is of little value to someone trying to get in. Hashing also provides the benefit of making all passwords 'look' the same length.

ie, if hashing any string always results in a 20 character hash, then if you have only the hash to look at, you can't tell whether the original password was eight characters or sixteen for example.

Share Improve this answer
Follow

answered Sep 12, 2008 at 13:55

Peter Bernier
8,058 ● 6 ● 40 ● 53

3

I encountered this exact same issue in my workplace. What I did to convince him that hashing was more secure was to write a SQL injection that returned the list of users and passwords from the public section of our site. It was escalated right away as a major security issue :)

To prevent against dictionary/hash attacks be sure to hash against a token that's unique to each user and static (username/join date/userguid works well)

Share Improve this answer

Follow

answered Sep 12, 2008 at 13:59

rpetrich

32.3k ● 7 ● 67 ● 89

3

If you do not salt your Password, you're suspect to Rainbow Table attacks (precompiled Dictionaries that have valid inputs for a given hash)

The other developer should stop talking about security if you're storing passwords in plaintext and start reading about security.

Collisions are possible, but not a big problem for password apps usually (they are mainly a problem in areas where hashes are used as a way to verify the integrity of files).

So: Salt your passwords (by adding the Salt to the right side of the password*) and use a good hashing

algorhithm like SHA-1 or preferably SHA-256 or SHA-512.

PS: A bit more detail about Hashes here.

*i'm a bit unsure whether or not the Salt should to to the beginning or to the end of the string. The problem is that if you have a collisions (two inputs with the same hash), adding the Salt to the "wrong" side will not change the resulting hash. In any way, you won't have big problems with Rainbow Tables, only with collisions

Share Improve this answer Follow

edited May 23, 2017 at 12:13

Community Bot

1 ● 1

answered Sep 12, 2008 at 13:55

Michael Stum

181k ● 119 ● 407 ● 541

3

There is an old saying about programmers pretending to be cryptographers :)

Jeff Atwood has a good post on the subject: You're Probably Storing Passwords Incorrectly

To reply more extensively, I agree with all of the above, the hash makes it easier *in theory* to get the user's password since multiple passwords match the same hash. However, this is much less likely to happen than someone getting access to your database.

Share Improve this answer

Follow

answered Sep 12, 2008 at 15:19

Luk

5,511 ● 4 ● 42 ● 55

2

There is truth in that if you hash something, yes, there will be collisions so it would be possible for two different passwords to unlock the same account.

From a practical standpoint though, that's a poor argument - A good hashing function (md5 or sha1 would be fine) can pretty much guarantee that for all meaningfully strings, especially short ones, there will be no collisions. Even if there were, having two passwords match for one account isn't a huge problem - If someone is in a position to randomly guess passwords fast enough that they are likely to be able to get in, you've got bigger problems.

I would argue that storing the passwords in plain text represents a much greater security risk than hash collisions in the password matching.

Share Improve this answer

Follow

answered Sep 12, 2008 at 13:58

Matt Sheppard

118k ● 46 ● 113 ● 134

@SLaks: md5, broken as it is, is still better than storing plain text passwords. – Graeme Perrow Dec 6, 2009 at 15:08

I'm not a security expert but I have a feeling that if plain text were more secure, hashing wouldnt exist in the first place.

1

Share Improve this answer

Follow

answered Sep 12, 2008 at 13:52

DaveK

4,597 ● 3 ● 34 ● 33

1 -1: Hashing is used for much more than passwords. And just because something exists doesn't mean it's of any value (though in this case it is). A real explanation would have added value. – user14070 Feb 3, 2009 at 9:19

In theory, yes. Passwords can be longer (more information) than a hash, so there is a possibility of hash collisions. However, most attacks are dictionary-based, and the probability of collisions is infinitely smaller than a successful direct match.

1

Share Improve this answer

Follow

answered Sep 12, 2008 at 13:56

Matthias Winkelmann

16.4k ● 8 ● 68 ● 77

It depends on what you're defending against. If it's an attacker pulling down your database (or tricking your application into displaying the database), then plaintext passwords are useless. There are many attacks that rely on convincing the application to disgorge it's private data-

1

SQL injection, session hijack, etc. It's often better not to keep the data at all, but to keep the hashed version so bad guys can't easily use it.

As your co-worker suggests, this can be trivially defeated by running the same hash algorithm against a dictionary and using rainbow tables to pull the info out. The usual solution is to use a secret salt plus additional user information to make the hashed results unique- something like:

```
String
hashedPass=CryptUtils.MD5("alsdl;ksahglhkjfsdkjhkjhk
+ user.getCreateDate().toString() +
user.getPassword);
```

As long as your salt is secret, or your attacker doesn't know the precise creation date of the user's record, a dictionary attack will fail- even in the event that they are able to pull down the password field.

Share Improve this answer

Follow

answered Sep 12, 2008 at 13:59

Tim Howland

7,970 ● 4 ● 29 ● 46

What? If attacker has your "users" table he will have the users' CreationDate. If CreationDate is not in the "users" table how will the application process logins? Salting with data unique to each user is good idea but its also a given that attacker will have access to it. The point of this is that attacker would have to create separate rainbow tables for each and every user. And this is where slow hashing algos come in. – Shinhan Aug 4, 2009 at 6:41

2 The salt need not be secret, and it shouldn't be the same for every user. Then, a rainbow table can be generated with that salt. Instead, each user should have a randomly generated salt. It need not be so secret. You can publish it if you wanted. The point is an entire rainbow attack would only be effective against max one user at a time. – ZaBlanc Oct 12, 2009 at 3:47

1

Nothing is less secure than storing plain-text passwords. If you're using a decent hashing algorithm (at least SHA-256, but even SHA-1 is better than nothing) then yes, collisions are possible, but it doesn't matter because given a hash, it's impossible* to calculate what strings hash to it. If you hash the username WITH the password, then that possibility goes out the window as well.

* - technically not impossible, but "computationally infeasible"

If the username is "graeme" and the password is "stackoverflow", then create a string "graeme-stackoverflow-1234" where 1234 is a random number, then hash it and store "*hashoutput*1234" in the database. When it comes to validating a password, take the username, the supplied password and the number from the end of the stored value (the hash has a fixed length so you can always do this) and hash them together, and compare it with the hash part of the stored value.

Share Improve this answer Follow

answered Sep 12, 2008 at 14:27

Graeme Perrow
57.2k ● 24 ● 86 ● 125

0

> more passwords could match the hash and a dictionary/hash attack would be faster.

Yes and no. Use a modern hashing algorithm, like an SHA variant, and that argument gets very, very week. Do you really need to be worried if that brute force attack is going to take only 352 years instead of 467 years? (Anecdotal joke there.) The value to be gained (not having the password stored in plain text on the system) far outstrips your colleague's concern.

Share Improve this answer

Follow

answered Sep 12, 2008 at 13:56

Stu Thompson
38.9k ● 19 ● 111 ● 156

0

Hope you forgive me for plugging a solution I wrote on this, using client side JavaScript to hash the password before it's transmitted: http://blog.asgeirnilsen.com/2005/11/password-authentication-without.html

Share Improve this answer

Follow

answered Sep 17, 2008 at 14:41

Asgeir S. Nilsen
1,137 ● 9 ● 13

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.