

## Hash salting -- digression

**Source:** <http://www.derkeiler.com/Mailing-Lists/securityfocus/secprog/2003-12/0027.html>

---

**From:** Kenneth Buchanan (*K.Buchanan\_at\_Kastenchase.com*)

**Date:** 12/18/03

To: 'Eric Knight' <eric@swordsoft.com>, secprog@securityfocus.com

Date: Thu, 18 Dec 2003 14:50:43 -0500

Thanks, Eric. There are many discussion points in this post but I'm going to seize on one, because I would like to know if anyone has access to in-depth research on it:

> *A single second as deemed "adequate"*.

Something that goes hand-in-hand with a salt is an iteration count. Recently I have been doing some password security investigations and I tried to approach the question of the actual value of an iteration count.

To quote PKCS #5:

"For the methods in this document, a minimum of 1000 iterations is recommended. This will increase the cost of exhaustive search for passwords significantly, without a noticeable impact in the cost of deriving individual keys."

Iteration counts seem to be valuable in two ways: they greatly increase the difficulty and library size of precomputed hashes (just as a salt does), and also increase the cost of brute force even when the attacker has access to the hash+salt+IC.

At first glance, this seems very effective. If an IC causes the hash computation time (on a modern desktop) to grow to 1 or 2 seconds, it makes the cracker's job exceedingly difficult. It seems that with a good (and enforced) password policy, and an adequate random salt, the threat of password compromise could become limited to attackers with access to vast computing resources.

\*\*\*But is that true?\*\*\*

While I know many systems employ iteration counts, I have been unable to find any good analysis on their value in this regard. There exists the possibility that a hash function could be algorithmically 'shortcutted', allowing for much faster computation of digest values.

This seems unlikely, but I am certainly no cryptanalyst. If anyone is aware of any work that has been done in this area, I'd appreciate a point in the

right direction.

Cheers,  
Ken Buchanan

-----Original Message-----

From: Eric Knight [mailto:eric@swordsoft.com]  
Sent: Wednesday, December 17, 2003 7:24 PM  
To: secprog@securityfocus.com  
Cc: CraigSecurity@blazemail.com  
Subject: Re: Values to use for a salt?

Craig,

Yes, the salt is meant to be a deterrent to guessing passwords, the purpose is exactly to make using a dictionary attack computationally unfeasible. However, once a salt is computed and the hashing algorithm is all that stands between the attacker and the defender, the process speeds up entirely.

The crypto-logic, based on Dr. Robert Morris Sr.'s approach, was that the password would require 1 second of computational time on a PDP-8 computer. These days, 1 second of computational time on desktop PC could do tens-of-thousands of these. A single second as deemed "adequate". DES was used as the hash function, and it wouldn't be useful entirely by itself if it didn't have the one second computation, so the salt was added to the hash to slow it down.

The salt also served the purpose of disguising the similar passwords in the database when two users pick the same one. As a second purpose, the salt provides better protection against casual observation than it does against computational attacks against it. To ensure this, salts are computed differently than the hash. If you were to, for example, use the password key to create a salt to hide the hash, you are created a two-part hash instead of a salt.

The magic formula should be something along the lines of:  $1 \text{ sec} = T(S(\text{SaltValue}) + H(\text{HashValue}))$  [where T is a time measurement, H is the hash function, S is the salt function].

"Same Salt" attack is where the salt is calculated once, and the hash is calculated multiple times. That is  $1 \text{ sec} = T(S(\text{SaltValue}) + H(\text{HashValue}))$  for the first try, then its  $1 \text{ sec} - T(H(\text{HashValue})) = T(S(\text{SaltValue}))$

"Similar Salts" attack is where multiple entries with the same salt value so logically, its the computation of  $O(\sqrt{N})$  (from the sort) added into the "Same Salt" attack, subtracting each instance of a similar salt from the overall required list. Although this can create a speed decrease in cases of no repetitive salts, in most cases it yields at least a few.

## SecurityFocus SecProg: Hash salting -- digression

There is strength inherent in using salts, but they have weaknesses as well. The optimum use of salt+hash for passwords would have very little weight on the Time factor on the salt, and very heavy weight on T for the hash.

As far as crypto is concerned, its hard to explain that the strength of any crypto is dependant on the time it takes to break, and as we've expanded the keys to sizes that are impossible for anyone but a computer to remember, the hash itself is going to have to become more time consuming in order to have validation of strength.

After having "broken" nearly a quarter-million passwords in the course of my career pen-testing and research, its a semi-dead subject to me. As an industry, we really need to be moving onward to solving other surrounding issues to better enforce the password collections. The onion effect of packing passwords in passwords, encrypting and hiding password files, and so forth and so on is only as reasonable as the question of who knows the process and how difficult is it to figure out.

This brings about the question of creating custom crypto with unpublished implementations. I think of it was "security with inherent weakness", which is a broad category that all crypto falls into. I use it all the time, but that's mostly because I can control its implementation better and I can change the standards at a whim. Obviously, this can't be done well with ITAR restrictions and allowing people to program to your code, but the "game" of crypto legalities is to hide and uncover secrets, and its neither a blackhat nor a whitehat exclusive.

Well, I'm going to cut off here because otherwise I'd write another 100 page book, there is a lot of depth to authentication components compared to simple password hiding. In the future, especially with keys reliably generated from biometric data, that shouldn't benefit much at all from salts.

Take it easy,

Eric Knight