

Re: Hashed password secure?

Source: <http://www.derkeiler.com/Newsgroups/sci.crypt/2004-06/2141.html>

From: Matthijs Hebly (heeb_at_iname.com)

Date: 06/29/04

Date: Tue, 29 Jun 2004 17:21:23 GMT

Bill Unruh wrote:

> *The stupidly written BSD md5 based unix password function simply runs the
> hash 1000 times to try to slow it down. Your suggestion would run it 65000
> times.*

Correction: my suggestyion would run it

**Random(SomeNumberDependentOnCPUSpeed)* times. Which, IMHO, makes it (almost) impossible for some attacker to create a dictionary of hashes before the actual attack, because for every password in the dictionary there would have to be e.g. 65536 entries in the attackers table, for each salt one entry, per password. Building such a dictionary would take 65536 times as long as building a dictionary of normal hashes, and the dictionary would be 65536 times as big. (Of course, 65536 is just a number. As machines get faster, the salt will get more bits, so it always takes approx. the same amount of time, let's say a second (for arguments sake. And the necessary tables will grow larger as a consequence).*

> *Also in your case if someway was found to reduce the number of salts needed
> to be searched, the attacker would suddenly have a much faster way of
> attacking the passwords.
> Ie, why not just do something like
> pi=sha1(password+pi-1+salt)
> for i from 1 to 65000.*

Where is the Random(...) here? It's essential to my argument that the number of iterations or tries before success is random (even if you continue after success, to make sure a login always takes approx. as long)! I don't understand your argument, BTW. How **reduce** the number of salts? The one-way Hash(pw+salt) function makes this infeasible. Am I wrong?

> *Because the salt is for preventing lookup tables, not for slowing down. The
> slowing down is done by just rerunning the hash multiple times-- ie making
> the slowdown an inherent part of the process rather than based on a secret.*
My method (using a random salt, not storing it) is certainly faster when storing the hash, because Hash(...) is just executed once, not an iteration of Hash(...) Random(...) times. Checking will take a second.

Matthijs.