

Re: Importance of salt

Source: <http://www.derkeiler.com/Newsgroups/microsoft.public.dotnet.security/2005-09/0172.html>

From: William Stacey [MVP] (staceyw_at_mvps.org)

Date: 09/16/05

Date: Thu, 15 Sep 2005 20:36:17 -0400

V, that is all correct. That is the problem with using one-way hash function for sending passwords. The strength is still based on the strength of the password. If the password is easy, it will be crack fast via a dictionary attack. Once it is captured, they can process offline for as long as it takes. The other issue is your basically creating a password equivalent. Depending on the what is going on, I can do reply attacks or maybe even other stuff. What transport tech are you using? WSE, sockets, Web services, etc.? If you want real security, I would take the next step into real encryption and key exchange protocol of some kind. WSE, for example, has a few options built in. I have some other stuff too, but need to know more about your app as far as what the transport is.

--

William Stacey [MVP]
<vlal0d@gmail.com> wrote in message
news:1126792852.969379.163050@z14g2000cwz.googlegroups.com...
> Hello,
>
> Maybe I wasn't clear enough. Let me try to explain a bit more... I
> mentioned salting because I am referring to a key generation based on a
> password (PasswordDeriveBytes class). So, users use plain passwords to
> generate a key which is then used for encryption. The salt is used on
> that password, which should help prevent the attacker trying to get the
> key by using the dictionary.
>
> Now, I understand how this could be helpfull if I use a static salt and
> then just transmit the message. The attacker really couldn't use his
> dictionary, because he doesn't know the salt. But, what if the attacker
> has access to the application that encrypts the messages? He can then
> see the static/hardcoded salt and simply use it with his dictionary
> when trying out the passwords.
>
> If I want to avoid static salt, I have to generate it somehow, and then
> store it so I can use it when decrypting, right? If I store it in a
> message, again, whats the point? The attacker can again retrieve the
> salt and use it with his dictionary.
>
> I repeat, the assumption is that we are using an publicly available
> software for encryption, so the attacker can decompile it or do
> whatever he wants with it.
>
> As for the iteration count... if I understood you correctly, all this
> does is increase the time that it takes to generate the key from the
> password? In other words, if I use 10 iterations to generate the

microsoft.public.dotnet.security: Re: Importance of salt

> key/hash for the password, that key can ONLY be generated with the same
> password and exactly 10 iterations? The attacker would have to run 10
> iterations for each word in his dictionary to get the key, assuming he
> even knows the correct iteration count used.
>
> Thanks again for your help,
>
> V.
>
>
>
>
> Dominick Baier [DevelopMentor] wrote:
>> Hello vla10d@gmail.com,
>>
>> first of all - you are mixing some terms here
>>
>> Encryption - the term salt isn't commonly used here, you may think of an
>> IV (initialization vector) which is used to start a feedback chain when
>> using
>> CBC.
>>
>> but i think you really mean hashing (e.g. for passwords) -
>>
>> salted hashes are : H(salt+password)
>>
>> reasons for salting
>>
>> a) you are not leaking information, e.g. if alice and bob have the same
>> password
>> - the resulting hash would be the same - not with salted hashes
>> b) there a tables of pre-computed hashes, so e.g. you encounter a hashed
>> password of "HJK)((bnnmm" - all you have to do is, look up that table for
>> the hash and retrieve the clear text value. If you use salted hashes, you
>> cannot use pre-computed tables, but have to calculate the hash on each
>> try.
>> this takes time.
>>
>> By using PasswordDeriveBytes with a high iteration count, you even raise
>> the bar
>>
>> a) the attacked does not know the iteration count from looking at the
>> hash
>> b) it takes even longer now to mount brute force/dictionary attacks - say
>> a simple hash needs 1 ms - and a iterated hash 1 s to calculate - this
>> makes
>> password guessing really infeasible
>>
>> this all depends of course on the password complexity and the computing
>> power
>> the attackers has at his disposal.
>>
>> you are basically buying time.
>>
>> That said - go for salted, iterated hashes by using PasswordDeriveBytes -
>> or even better the new .NET 2.0 Rfc2898DeriveBytes class.
>>
>>
>> -----
>> Dominick Baier - DevelopMentor
>> <http://www.leastprivilege.com>
>>

microsoft.public.dotnet.security: Re: Importance of salt

```
>> > Hello,  
>> >  
>> > I have one question regarding the importance of salt in encryption.  
>> >  
>> > As I understand, the salt is used to prevent dictionary attacks. Also,  
>> > it is recommended that the salt isn't always the same, and that it  
>> > should be randomly generated for each message. This random salt should  
>> > then be stored in the encrypted message, as a prefix for example, so  
>> > that it could be retrieved during the decryption.  
>> >  
>> > Now, I don't understand how this helps with dictionary attacks? For  
>> > example, if the attacker knows that the first 8 bytes for example are  
>> > salt, can't he simply modify his attacking program to include that  
>> > salt for each word he retrieves from the dictionary? The assumption  
>> > here is that the attacker gets access to the original encryption  
>> > software as well as the message.  
>> >  
>> > Secondly, can someone explain how do the increased iterations in  
>> > PasswordDeriveBytes help?  
>> >  
>> > Thanks for your help,  
>> >  
>> > V.  
>> >  
>
```