

Re: Securing hashing algorithm

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2004-11/5415.html>

From: Wm. Scott Miller (*Scott.Miller_at_spam.killer.wvinsurance.gov*)

Date: 11/24/04

Date: Wed, 24 Nov 2004 10:15:47 -0500

I've noticed by searching the groups that you are a strong supporter of the "keyed" approach. So, what are the advantages and disadvantages of using keyed vs non-keyed hashes?

Thanks,
Scott

"William Stacey [MVP]" <staceywREMOVE@mvps.org> wrote in message news:%238pbc8e0EHA.3468@TK2MSFTNGP14.phx.gbl...

- > *When using one way hashing like this, you always have one issue – securing the password/key passed to the keyed hash. I assume your using a "keyed" hash like HMACSHA1 or a keyed MD5 to hash your secret. Some options are:*
- > *1) string encrypted in your program using obfuscator (better then nothing, but probably not great.)*
- > *2) Protected via ACLs in registry, file or smartcard (or others).*
- > *3) Use DPAPI to set and get your password in the OS. See*
- >

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/secmod/html/secmod21.asp>

- >
- > *Of the three, DPAPI is probably the best. It is still not perfect as a user logged in using same user account as app used can decrypt the data. If you use the machine store, any user could decrypt the data unless you use additional entropy. So possible a slightly better approach may be to use machine store so any user on local machine could use your program and keep your "entropy" encrypted in your program (via obfuscator encryption.) So not a hacker needs to hack your assembly to figure out general idea of what your doing, somehow decrypt your entropy (i.e. second pw) and then figure out how to get data from DPAPI and what store your used, etc. As you can see, you could keep jumping through hoops hiding data until your blue and still not get 100% protection.*
- > *Possibly a better approach is to use RSA private/public key and client only knows public key. That way, you have nothing you need to hide. You still need to protect your client apps public key so it can't be replaced by a*

> *hackers public key, etc.*
>
> *Naturally, none of this matters if your code is plain .net as hacker can
use
> ildasm and ilasm to round trip your code and do remove your protections
and
> public key removing the strong name checking at assembly load time. Only
> thing that helps this is obfuscating using a good one (I use and like
> Xenocode which also prevents ildasm on your assemblies). Getting to a
point
> where the only option for the hacker is to hack your code is probably the
> best effort and should be the goal I would think.*
>
> --
> *William Stacey, MVP*
> <http://mvp.support.microsoft.com>
>
> *"Wm. Scott Miller" <Scott.Miller@spam.killer.wvinsurance.gov> wrote in
> message news:OPhhtfa0EHA.3744@TK2MSFTNGP10.phx.gbl...*
> *Hello all!*
>>
>> *We are building applications here and have hashing algorithms to secure
>> secrets (e.g passwords) by producing one way hashes. Now, I've read
alot
>> and I've followed most of the advice that made sense. One comment I've
> seen
>> alot about is "securing the hashing routine" but no-one explains how to
>> accomplish this. So how do I secure my hashing routine? Do I use code
>> access security, role based security, ACLs, etc or combination? And if
>> combination what combination is the best? The routines will be used by
> two
>> "applications." A ASP.NET and a Windows application. It already has a
>> strong name and is installed in the GAC. How do I prevent it from being
> run
>> by any code besides our two applications? Should it be installed in the
>> GAC? And if not, how to I guarantee the two applications are using the
> same
>> version?*
>>
>> *Thanks for your help,*
>> *Scott*
>>
>>
>