Salted Hashing of Passwords

During the course of Penetration testing of Web Applications at Cyber Security Division, NIC, several security vulnerabilities are identified. One of these vulnerabilities includes the finding that the credentials traveling in clear text can be sniffed from the network. The credentials can also be detected with the help of memory editing tools on shared systems which are used to access the authentication web pages. Considering the common nature of these problems and their solutions, throwing light on the underlying concepts is a must read for those targeting the problem while developing secure code in an effective manner. The following section sheds light on the solution to the problem.

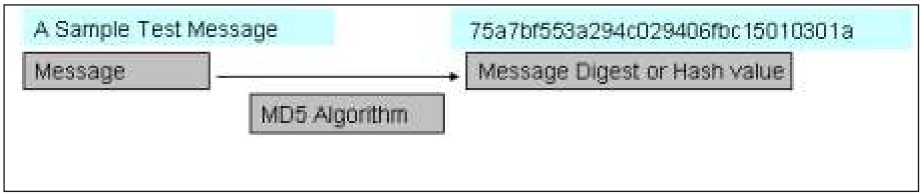
Snigdha Acharya
Technical Director
snigdha.acharya@nic.in

What is a Hash?

Hash algorithms map binary values of an input of arbitrary length to a binary value of a fixed length, known as hash values. A hash value is a unique and extremely compact numerical representation of a piece of data. If you hash a paragraph of plaintext and change even one letter of the paragraph, then a subsequent hash will produce a different value. It is computationally improbable to find two distinct inputs that hash to the same value. A hash value is also known as a Message digest. MD5, SHA1 etc. are Hash Algorithms. The following sections illustrate salted hashing with respect to MD5 algorithm. Other hash algorithms such as SHA-1 may be used alternately.

About MD5

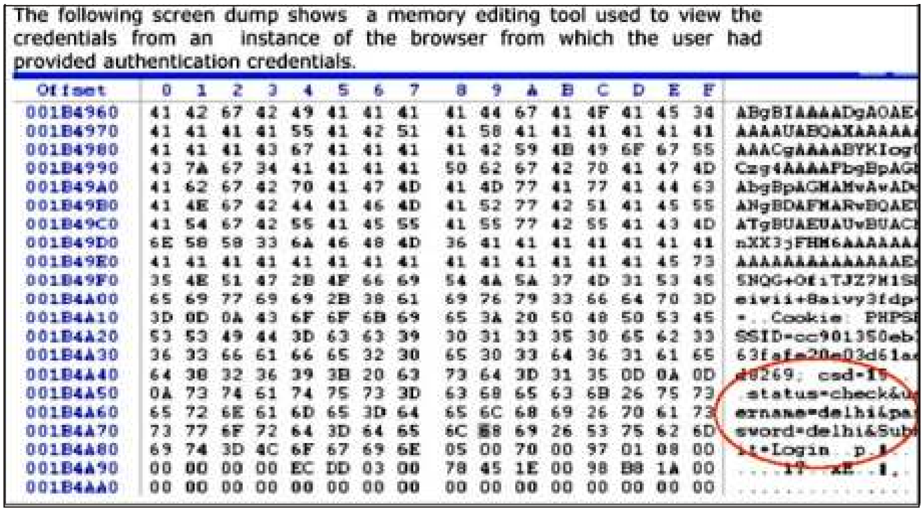
MD5 algorithm takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input. A SHA1 algorithm produces a 160 bit length hash of the input.


A Sample Test Message
75a7bf553a294c029406fbc15010301a
Message
Message Digest or Hash value
MD5 Algorithm

Application of MD5 to protect passwords

- **Problem:** When a site visitor submits his/her credentials on a login page it is submitted in clear text and this can be obtained by malicious users from a browser of a user even though he/she may have logged out.

But to make it possible, the hacker must have access to the user's system, which may be possible in the case of shared system in kiosks etc. Also, another precondition to this is that the browser must have not been closed.



The following screen dump shows a memory editing tool used to view the credentials from an instance of the browser from which the user had provided authentication credentials.

```
Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
001B4960  41 42 67 42 49 41 41 41  41 44 67 41 4F 41 45 34  ABgBIAAAADgAOAE
001B4970  41 41 41 41 55 41 42 51  41 58 41 41 41 41 41 41  AAAAUABQAXAAAAAA
001B4980  41 41 41 43 67 41 41 41  41 42 59 4B 49 6F 67 55  AAACgAAAABYKIog
001B4990  43 7A 67 34 41 41 41 41  50 62 67 42 70 41 47 4D  Czg4AAAAPbgBpAG
001B49A0  41 62 67 42 70 41 47 4D  41 4D 77 41 77 41 44 63  AbgBpAGMAMwAwAD
001B49B0  41 4E 67 42 44 41 46 4D  41 52 77 42 51 41 45 55  ANgBDAFMARwBQAE
001B49C0  41 54 67 42 55 41 45 55  41 55 77 42 55 41 43 4D  ATgBUAEUAUwBUAC
001B49D0  6E 58 58 33 6A 46 48 4D  36 41 41 41 41 41 41 41  nXX3jFHM6AAAAAA
001B49E0  41 41 41 41 41 41 41 41  41 41 41 41 41 41 45 73  AAAAAAAAAAAAAAE
001B49F0  35 4E 51 47 2B 4F 66 69  54 4A 5A 37 4D 31 53 45  5NQG+OfiTJZ7M1S
001B4A00  65 69 77 69 69 2B 38 61  69 76 79 33 66 64 70 3D  eiwii+8aivy3fdp
001B4A10  3D 0D 0A 43 6F 6F 6B 69  65 3A 20 50 48 50 53 45  =..Cookie: PHPS
001B4A20  53 53 49 44 3D 63 63 39  30 31 33 35 30 65 62 33  SSID=cc901350eb
001B4A30  36 33 66 61 66 65 32 30  65 30 33 64 36 31 61 65  63fafe20e03d61a
001B4A40  64 38 32 36 39 3B 20 63  73 64 3D 31 35 0D 0A 0D  d8269; csd=15
001B4A50  0A 73 74 61 74 75 73 3D  63 68 65 63 6B 26 75 73  .status=check&u
001B4A60  65 72 6E 61 6D 65 3D 64  65 6C 68 69 26 70 61 73  ername=delhi&pa
001B4A70  73 77 6F 72 64 3D 64 65  6C 68 69 26 53 75 62 6D  sword=delhi&Sub
001B4A80  69 74 3D 4C 6F 67 69 6E  05 00 70 00 97 01 08 00  it=Login..p.
001B4A90  00 00 00 00 EC DD 03 00  78 45 1E 00 98 B8 1A 00  .....xE...
001B4AA0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

Similarly, this can be illustrated with the help of Network sniffing tools when credentials are traveling in clear text.

- **Solution:** The problem outlined above can be solved with hashing. A hash of the password can be sent from the client browser to the server application. It is not possible to extract the clear text password from the network or from the browser memory as only the hashed form of the password can be obtained.

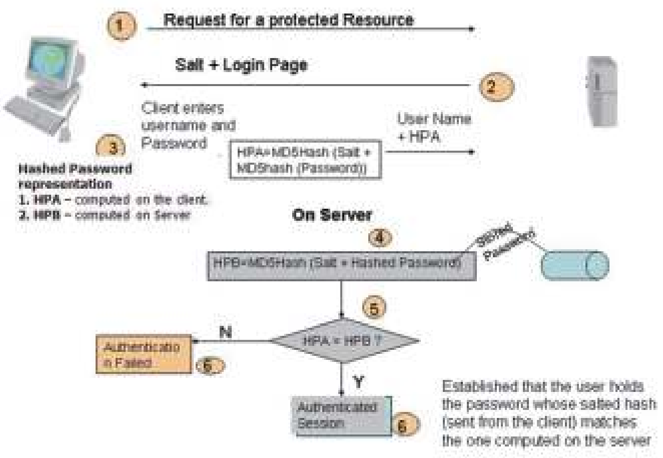
Threat of Hash Replay

However, a hashed password submitted from the client can be sniffed while in transit from the network or obtained from a shared system used to browse the web site with the help of tools. This hashed password can then be replayed or pasted while submitting to the server and access gained as seen from tests in the lab.

Solution: Salted Hashed Password

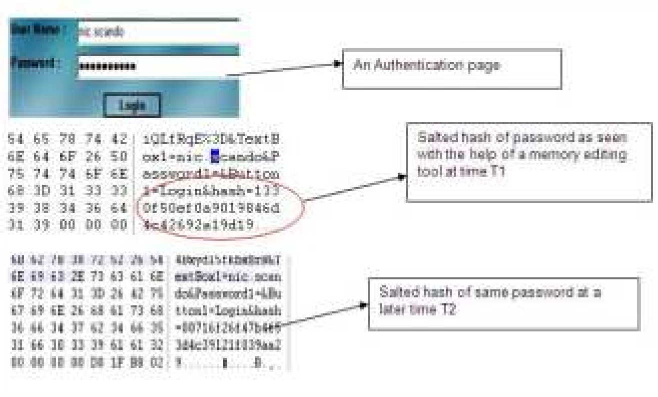
Salted MD5 hash of the password can be submitted to avoid the replay attack. In this case the password will vary every time the salted MD5 password is submitted to the server. Since the salt is a random number and changes every time, the salted hashed password also changes every time.

The pre-requisite to this is that the backend database stores a hash of the password. When a client requests for the login page, the server generates a random number or the salt, and sends it to the client along with the page. A JavaScript code on the client computes the MD5 hash of the password entered by the user. It then concatenates the salt to the hash and re-computes the MD5 hash. This result is then sent to the server. The server picks the hash of the password from its database, concatenates the salt and computes the MD5 hash. If the user entered the correct password, these two hashes should match. The server compares the two and if they match, the user is authenticated. This session persists or is valid till the user logs out or the session times out due to inactivity.


1
Request for a protected Resource
Salt + Login Page
2
Client enters username and Password
3
HPA=MD5Hash (Salt + MD5hash (Password))
User Name + HPA
Hashed Password representation
1. HPA – computed on the client.
2. HPB – computed on Server
On Server
4
HPB=MD5Hash (Salt + Hashed Password)
Stored Password
5
HPA = HPB ?
N
Authenticatio n Failed
6
Y
Authenticated Session
6
Established that the user holds the password whose salted hash (sent from the client) matches the one computed on the server

Step by step illustration of the salted MD5

Verification of a salted hash implementation in an application



An Authentication page

```
54 65 78 74 42 | iQLtRqEx3D&TextB
6E 64 6F 26 50 | ox1=nic scando&P
75 74 74 6F 6E | assword1=&Button
68 3D 31 33 33 | 1=Login&hash=133
39 38 34 36 64 | 0f50ef0a9019846d
31 39 00 00 00 | 4c42692e19d19
```

Salted hash of password as seen with the help of a memory editing tool at time T1

```
48 62 78 38 72 52 26 54 | 4Abyd15tXbm8rWkT
6E 69 63 2E 73 63 61 6E | extBox1=nic.scan
6F 72 64 31 3D 26 42 75 | do&Password1=&Bu
67 69 6E 26 68 61 73 68 | tton1=Login&hash
36 66 34 37 62 34 66 35 | =8071612647b4f5
31 66 38 33 39 61 61 32 | 3d4c391218839aa2
00 00 00 00 D8 1F B8 02 | ........
```

Salted hash of same password at a later time T2

It can be seen from the above sample snapshots of an application with salted hash implementation in the authentication module, the salted hash passwords are different at the two instances of authentication and hence cannot be used in replay attacks.

Effective implementation of the above steps in the code logic is an adequate defense against the credentials leakage problem encountered in web applications accessed from shared client systems used for browsing authenticated sessions as well as from the network.

For further information, contact:

Snigdha Acharya
Technical Director
Cyber Security Division
snigdha.acharya@nic.in