Site-Specific Passwords

Alan H. Karp
Hewlett-Packard Laboratories
alan.karp@hp.com

Abstract

Most users have accounts on a large number of web sites. Today, they have a choice of one password for all sites or a different password for each site. Neither choice is attractive. This note describes a procedure that produces a different password for each site from a single password provided by the account holder.

Introduction

Many web sites require a user login, forcing users into an unfortunate dilemma [6]. They can either try to remember a different password for each site, or they can use the same password for all sites. If they elect the former, they most likely end up writing down passwords on paper. Even worse is writing these passwords in a file in their computers. Alternatively, users can ask the browser to remember the password. These last two approaches have the disadvantage of making login virtually impossible when using a different machine. They also make the password vulnerable to hackers.

If users elect to have a single password for all sites, they put themselves at risk. An unscrupulous web site owner could read passwords from users and attempt to use them at well-known banking sites. Passwords can also be stored unencrypted in some web sites where they are attractive targets for hackers.

Microsoft [3] is addressing this problem with Passport, a system that stores the required information at a central, secure site. Anyone with a Passport account who has access to the Internet can log in to a Passport compliant site. However, people have raised questions of security and trust implicit in this approach. Additionally, the Passport server is a single point of failure.

This note presents a stand-alone solution that requires the user to remember a single password and an easily remembered name for each web site requiring a login. The system combines these two elements to produce a site-specific password.

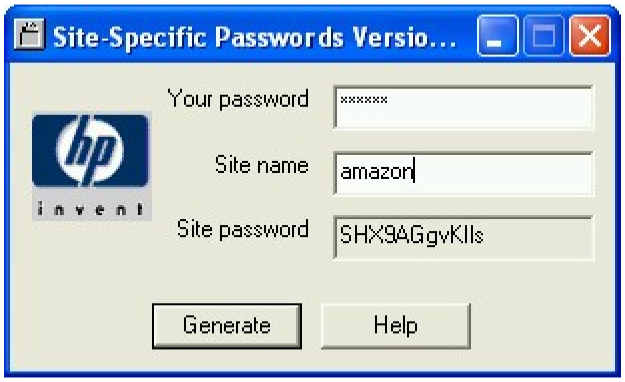
Generating the Password

We generate a site-specific password from a user password and an easy to remember name for the web site. These two strings are concatenated and hashed using the MD5 [2] algorithm. MD5 produces a binary output of 16 bytes, which we convert to ASCII by Base64 [1] encoding the MD5 result and truncating to 12 characters. Truncating protects the user password without overly compromising the security of the site password as explained below. A similar tool [4] has been distributed as freeware, but it has a number of flaws. Most serious is the use of a function that has not undergone scrutiny by the cryptographic community. Further, this function is not portable; different platforms generate different site passwords from the same inputs.

For our purposes, the MD5 algorithm has a number of advantages.

1. The algorithm cannot be inverted to discover the user password even if the site name is known.
2. The algorithm is a standard, meaning any implementation must produce the same output for a given input.
3. It is highly unlikely that two different inputs will produce the same output.

The figure shows a screen shot of a typical use. In this example, the unguessable password is **qwerty**.



Your password ******
Site name amazon
Site password SHX9AGgvKIls

The code contained in the Appendix is a Python [5] implementation of the program that produces this result. It can be used as a stand-alone application to generate passwords that can be typed manually into the password field of the web site. A better approach would be to build a browser plug-in that would pop up whenever a password field appeared in a web page. The output that today is displayed in the Site Password window could be written directly into the input area of the web page. In addition, the data on site names and password rules described in the next section can be attached to the window. Another idea is to store a one-way hash of the user password so the tool can detect when the user password was typed incorrectly.

Even better than improving this version of the code is putting it on a portable device. Porting to a PDA is straightforward, but more useful might be putting the tool on a cell phone. It might even be worth producing a specialized device small enough to carry on a key chain. The tool should also have an option to produce numeric values that can be used as PIN's for ATMs.

Practical Issues

It is still important to choose a good user password. The algorithm is well known, and the site names are easily guessable. If the user password can be guessed, then an attacker can generate the site password. However, the user needs only one such password, making the use of a good password less onerous.

There is, of course, the issue of remembering the site name. Using the domain name in the URL is one good scheme. Another would be to write down the site names used or put them in a file. They could even be stored in Passport. There is no security risk because knowing the site name provides no advantage in guessing either the user password or the site password.

Different web sites have different rules for passwords. For example, Schwab.com allows only alphanumeric characters in the password, but PayPal.com requires at least one non-alphanumeric character. Some sites allow six-character passwords, but others require at least eight.

When changing passwords on a web site, you want to be careful that you typed your user password correctly. The easiest way to check is to generate a site password, retype your user password, and generate the site password again. If it changes, you mistyped your user password.

The tool described here returns the leading 12 characters of the result, more than needed for any web site I have encountered. If a web site accepts only alphanumeric characters, and the generated site password has some non-alphanumeric characters, they can be skipped when being typed into the password field; enough characters should be left to serve as a good site password. If the web site requires special characters in the password, and the generated site password does not have any, the user will have to add one or more. In both cases additional information must be remembered, but there is little danger in recording it with the site names. The table gives some examples.

Site Name	**Location**	**Comments**
Schwab	www.schwab.com	Up to but not including /
PayPal	www.paypal.com	Append = to end
WAMU	www.wamu.com	Washington Mutual

A table like this is particularly important for sites that require periodic password changes. The rules can be quite strict. For example, some sites do not allow you to use any of your previous five passwords. Others require that the new and old passwords differ by a certain minimum number of characters. With this tool, you need only append an integer to the site name each time you need to change your password. For example, a user password of “qwerty” and a site name of “Amazon” result in a site password of “SHX9AGgvKIls”. Changing the site name to “Amazon1” results in a site password of “qZ1ZIQURZ5Ko”. These two passwords should pass any tests the web sites make.

One issue is the display of the site password; it is not obscured. Password fields display only asterisks because someone might be looking over your shoulder. We can’t do that with the generated site password because it must be copied to the web site’s password field. There is some benefit to this choice. Seeing the site password can help detect errors typing the user password or site name. You might not remember that your site password for Amazon is “SHX9AGgvKIls”, but the substring “SHX” sticks in your mind. Fortunately, the site passwords look random, making them hard to remember. Nevertheless, you should be aware of who is watching.

Note that there’s no reason why you must have only one user password. You could certainly have one for your sites that can cost you money and another for those that can’t. Should you inadvertently use the wrong one, say your money password, you’re not exposing any useful information. The web site will reject the computed site password, and you’ll just try your other user password. Actually, more often than not, you’ll notice that the site password looks wrong, and you’ll correct your mistake quickly.

Safety Considerations

Attacks against MD5 are normally attempts to find two inputs that produce the specified output. That is not the problem here. We are worried about someone determining the user password using the site password and knowledge of the site name. As far as we know, the only such attack is exhaustive search. However, the inputs are short, so such an attack may be feasible.

There are two things to be protected, the user password and the site password. The generated site password consists of what appears to be random, ASCII characters. Thus, the only attack is exhaustive search. Since each attempt involves a round trip to the web server, it is hard to try more than one password per second. Hence, even relatively short passwords will be safe from guessing. An 8-character site password provides more than enough protection.

We also need to worry about attacks that figure out the user password knowing the site password. We assume here that the site name is known. While not precisely true, we expect people to choose relatively obvious strings as site names. These names won’t be very long, perhaps 8 bytes if a shorthand is used or 16-20 if the location in the URL is used. In either case, it is important that user passwords be sufficiently long and unguessable. If the user password is a word found in the dictionary, it can be guessed in a few hours on a modern PC. Even if the password is not in the dictionary, it must still be long enough. If the user password consists of 4 or 5 bytes, an exhaustive search can find it in a day or so.

Since MD5 is a one-way hash function, we assume the only way to get the user password from the site name and site password is a brute force attack. It only takes a few hours on a modern PC to find a collision, a value for a user password that produces the same site password for the assumed site name. However, that isn't good enough. The attacker must find the exact string used to generate the site password. It would take centuries to guess a well-chosen, 8-byte user password.

The tool described here is even safer to use because we don't use the entire MD5 result as the site password. If we use only 8 bytes of the result as the site password, a large number of strings will produce these bytes. Each of these strings must be used to generate a site password for the site to be attacked, and that site password must be sent to the web server to see if it is correct. Hence, the attack takes an impractical length of time. However, if it does succeed, all the user's site passwords can be generated. That's why picking a good user password is important.

What about those who insist on using weak user passwords? We could write them off, and tell them they're taking a chance, or we could help them. One bit of help is to remind them to change their user passwords periodically. Of course, when they do, all their generated site passwords will change. That's such a nuisance that they won't bother. What they could do instead is create a strong password, perhaps using this tool, and encrypt it with their weak passwords. Now, when they want to change their weak passwords, they just decrypt the strong one with the old weak password and encrypt it with the new weak one. If their weak passwords aren't too weak, and if they are changed often enough, the encrypted strong password can be stored in a public place.

There is a flaw in the operation of the tool that needs to be fixed before it is widely distributed; the user password and site name are concatenated. That means that a user password of "qwert" and a site name of "yahoo" produce the same site password as a user password of "qwerty" and a site name of "ahoo". This accidental collision should never be a problem in practice, but there's no need for it. All we need to do is add a padding character, even a blank, between the user password and the site name before hashing. This change hasn't been made in the current tool because the users don't want to change all their site passwords.

Rejected Ideas

It is often important to know what was rejected and why. This information makes it less likely that mistakes will be repeated. More importantly, it may trigger new ideas that weren't considered.

The basic idea used here comes from the way Unix handles login passwords. When a new password is created, the OS kernel adds a few characters, referred to as *salt*, to the password, hashes the result and stores it in the publicly readable password file. (Some systems use a level of indirection to make breaking passwords more difficult.) However, we can’t use this approach directly for web logins.

We rejected the idea of doing the hashing on the web site because that would require transmitting the user password, exactly what we are trying to avoid. We also rejected doing the hash in the browser with a fixed salt, because doing so would result in the same site password for every site.

One idea was to have the web site transmit its salt to the browser. However, this choice would involve changing the protocol. In the long run, this approach might be viable, but taking such a stance is not likely to help get early acceptance. In addition, there’s nothing to prevent a malicious site sending Schwab.com’s salt and capturing the site password.

We thought about automatically using the location field in the URL as the site name. This approach has two problems. First of all, if the URL changed at all, the user would not be able to log on. Secondly, the only way to change a site password would be to change the user password. Doing so would require the user to change site passwords on every site. One idea that appears viable is to use the URL as the site name but give the user an optional field to include in the hash. We didn’t feel this approach was worth the effort.

One idea was to put the result of the hash directly into the password field of the web page. However, different web sites have different rules for what constitutes a legal password. As noted, Schwab.com does not allow special characters, while Paypal.com requires them. Editing the password field is difficult because it only displays asterisks.

We could make the tool widely available by having it run on a server. While not as dangerous as sending user passwords to a wide variety of web servers, we prefer to keep the user password on the machine where it is entered.

The tool currently runs as a stand-alone application. We could write an applet, but it might be too easy to create a Trojan horse that could capture user passwords. We thought of a browser plug in, but they are started by the web page. We must assume that the web site is unaware of our tool. A browser helper might be the best approach, but you might need two browser instances since the first is blocked waiting for the site password.

Availability

The tool is available from http://www.hpl.hp.com/research/downloads/. There is a patent pending, so be sure you understand the terms of use described at the bottom of that page.

Acknowledgements

I’d like to thank Jaap Suermondt, George Forman, Alan Haggard, and Evan Kirshenbaum for suggesting improvements to the basic tool. Ren Wu produced a version in C for

Windows, Kevin Smathers for Linux, Bill Serra for the HP Jornada, and John Schettino a Java applet and versions for Palm Pilot and Nokia EPOC cell phones.

References

1. Base64, http://www.ietf.org/rfc/rfc1341.txt
2. MD5, http://www.ietf.org/rfc/rfc1321.txt
3. Microsoft Passport, http://www.passport.com
4. Yan, Q., SuperPassword, http://www.thinkingsoft.com
5. Python, http://www.python.org
6. Reagan, K., “Report: Passwords Not Good Enough for E-Shoppers”, E-commerce Times, February 7, 2002, http://www.ecommercetimes.com/perl/story/16212.html

Appendix

This section contains the Python source code that implements the algorithm described in the body of this note. The author has only a passing familiarity with Tk, the tool used to generate the user interface. The result needs to be made prettier, the enter key should be useable in addition to the button, and the cursor should be placed in the password field when the window appears. We could also add the table of site names described earlier to the tool.

```
# Hewlett-Packard, © Hewlett-Packard, 2002
#
# A little program to hash a user password with a site-specific string
# The result is a random looking string that can be used as a password
# on that site.  This tool lets you use the same password string for
# all sites, but assures that the password sent to the site is
# different from that of any other site.
#
# Alan Karp, alan.karp@hp.com, 7 February 2002

version = "Version 0.2"
from Tkinter import *

# Generate Help window

def gethelp():
  top = Toplevel()
  h = Frame(top)
  h.pack()
  m = Message(h,relief=SUNKEN, text = """
                   Site-specific Password Generator
                   (Patent Pending, Hewlett-Packard)

    Combines a name for a web site and your password to produce
    a password unique to the site.  This tool lets you use a single
    password for all your web logins without worrying that the owner
    of one site will use your password on another one, perhaps
    draining your bank account.

    There are two input fields.

       Your password
             A string that you can remember.  It should still be
             hard to guess because this tool uses a well-known
             algorithm, and you use well-known site names.  Case
             sensitive.

       Site name
             Your name for the site.  For example, Schwab, or
             Amazon.  Case insensitive.

    Click on "Generate Password" to get a site-specific password.
    The result is 12 ASCII characters.  Some sites do not accept
    characters other than number or letters.  In this case, just
    don't use other characters when you enter your site-specific
    password in their login windows.
    """)
  m.pack()
  m.config(width=500)

  c = Button(h,text='Close',command=top.destroy)
  c.pack()

# Generate Error window

def error():
  top = Toplevel()
  h = Frame(top)
  h.pack()
  m = Message(h,relief=SUNKEN, text = "You forgot to enter a
password.", width=150)
  m.pack()
  c = Button(h,text='Close',command=top.destroy)
  c.pack()

# Routine that does the actual work

import md5, base64
def genpw():

# Use MD5 hash to produce an unguessable string from the inputs

  m = md5.new()
  location = site.get()
  m.update(location.lower())
  password = pw.get()
  if len(password) == 0:
    error()
  m.update(password)

# Convert to ascii characters

  password = (base64.encodestring(m.digest()))[0:12]
  pwout.delete(0,END)
  pwout.insert(0,password)

# Main routine

# Header

root = Tk()
Label(root, text=("Site-specific Password "+version)).pack()

# Password entry field

f2 = Frame(root)
f2.pack()
Label(f2, text="Your password", relief=RIDGE, width=20, font =
'Courier').pack(side=LEFT)
pw = Entry(f2, show="*", relief=SUNKEN, font = 'Courier')
pw.pack(side=RIGHT, expand=YES, fill=X)

# Site entry field

f1 = Frame(root)
f1.pack()
Label(f1, text="Site name", relief=RIDGE, width=20, font =
'Courier').pack(side=LEFT)
site = Entry(f1, relief=SUNKEN, font = 'Courier')
site.pack(side=LEFT, expand=YES, fill=X)


# Output field

f3 = Frame(root)
f3.pack()
Label(f3, text="Site password", relief=RIDGE, width=20, font =
'Courier').pack(side=LEFT)
pwout = Entry(f3, font = 'Courier', relief=RIDGE)
pwout.pack(side=LEFT)

# Generate Buttons

f4 = Frame(root)
f4.pack()
Button(f4, text='Generate Password', command=genpw).pack(side=LEFT)
Button(f4, text='Help', command=gethelp).pack(side=RIGHT)
Button(f4, text='Close', command=root.destroy).pack(side=LEFT)

# Run it

root.mainloop()
```