

Re: Proposal for Lite Encryption for Login Form without SSL

Source: <http://coding.derkeiler.com/Archive/PHP/comp.lang.php/2007-10/msg00125.html>

- *From:* klenwell <klenwell@xxxxxxxx>
 - *Date:* Tue, 02 Oct 2007 04:35:29 -0000
-

On Oct 1, 6:38 pm, Jerry Stuckle <jstuck...@xxxxxxxxxxxxxxxx> wrote:

klenwell wrote:

On Oct 1, 2:40 am, "C. (<http://symcbean.blogspot.com/>)"
<colin.mckin...@xxxxxxxx> wrote:

On 1 Oct, 06:04, klenwell <klenw...@xxxxxxxx> wrote:

On Sep 30, 9:08 pm, Jerry Stuckle
<jstuck...@xxxxxxxxxxxxxxxx> wrote:

klenwell wrote:

Another
request for
comments
here.
I'd like to
accomplish
something
like the
scheme
outlined at
this page
here:
<http://tinyurl.com/3dtcdr>
In a
nutshell, the
form uses
javascript to
hash (md5)
the
password
field using a
random

Re: Proposal for Lite Encryption for Login Form without SSL

one-time
salt (nonce)

--

generated
by php and
pasted in
the form ---
that is then
posted with
the hashed
password
back to the
server. This
way the
password is
not sent to
the server
in plaintext.

In the
example
cited above,
however,
the
password is
stored
unhashed
back at the
server (i.e.,
in the
database)
and it's this
problem
that's been
tying me in
knots this
evening.

The most
obvious
way it
seems to me
to cut
through the
knot is to
simply copy
the
server-side
salt (sss)
used to hash
the pw in
the
database ---

Re: Proposal for Lite Encryption for Login Form without SSL

the salt is
constant —
within the
javascript
portion of
the form so
that that
client
would:

1. ssspw =
md5(sss +
pw)
2. nssspw =
md5 nonce
+ ssspw)
3. post (1)
nssspw, (2)
nonce, (3)
username
(in
plaintext)

Then on the
server side,
php would:

1. db_ ssspw
= fetch
hashed
password
from db
(which
should ==
ssspw)
using
username
2.
db_ nssspw
=
md5(POST[nonce]
+
db_ ssspw)
3. compare
POST[nssspw]
and
db_ nssspw

While this
does not
expose the
plaintext
password or
the hashed
password in

Re: Proposal for Lite Encryption for Login Form without SSL

the
database, it
does make
public the
server-side
salt
used to
generate the
hash
password
by pasting it
in plaintext
in the
javascript
-- though it
does not
post it from
the form
back to the
server.

So
questions:
1) Is
exposing
the
server-side
salt a
terminal
flaw in this
plan?

This would
be the
equivalent
to a public
key in
public key
encryption
systems,
no?

2) Does
exposing
the
server-side
salt render
hashing the
password in
the database
moot?

3) If this is
flawed, is it
still better

Re: Proposal for Lite Encryption for Login Form without SSL

than
nothing,
accepting as
given that
SSL has
been ruled
out? (The
article
above notes
that
Yahoo uses
a system
like this.)
4) Any
better ideas
for
accomplishing
the concept
outlined
here?
Before I run
this over to
the
cryptology
newsgroup,
I thought I'd
give
it an airing
here.
Thanks,
Tom

Why do you need to use the
same salt (or even the same
encryption
method) for the database?
Also, sending the password
over an unencrypted link
(even if the
password itself isn't
encrypted) doesn't really
give you anything. If I
want to hack into your
system, all I need to do is
watch the link for
the encrypted password
coming over it, and create
my own form (sans
javascript) to encrypt the
password on my end and
send it.

Re: Proposal for Lite Encryption for Login Form without SSL

--
=====
Remove the "x" from my
email address
Jerry Stuckle
JDS Computer Training
Corp.
jstuck...@xxxxxxxxxxxxxx
=====

Good points:

Why do you need to use the
same salt (or even the same
encryption
method) for the database?

It's a one-way hash, so the input strings
have to match. And I don't
think that's possible without sharing the db
salt with the client.
Two-way encrypt/decrypt might solve this,
but I don't know offhand a
library or function that's readily available for
both js and php.

Also, sending the password
over an unencrypted link
(even if the
password itself isn't
encrypted) doesn't really
give you anything. If I
want to hack into your
system, all I need to do is
watch the link for
the encrypted password
coming over it, and create
my own form (sans
javascript) to encrypt the
password on my end and
send it.

Yes, true, I've misapplied this. I don't think
you would even have to
hash the password. Just send those three
values: (1) nssspw, (2)
nonce, (3) username and they would work
every time.

Looking again at the website cited, what
should happen first is:

1. Server (PHP) generates challenge value

Re: Proposal for Lite Encryption for Login Form without SSL

(nonce) and includes it in form as hidden value and *saves it as a session value* so not passed back in open via form (i.e. SESSION[nonce]).

2. User submits form with : (1) username (in plaintext) (2) password (in plaintext) (3) nonce [this is all still private and it has not been posted yet]

3. The server-side salt (sss) is posted in the js section of form to hash password to match db.

Then before posting back to server, javascript kicks in client-side:

1. ssspw = md5(sss + pw)
2. nssspw = md5(nonce + ssspw)
3. POST (1) nssspw, (2) username (in plaintext)

Then on the server side, php would:

1. db_ ssspw = fetch hashed password from db (which should == ssspw) using username
2. db_ nssspw = md5(SESSION[nonce] + db_ ssspw)
3. compare POST[nssspw] and db_ nssspw
4. unset SESSION[nonce]

So then, if an eavesdropper sends back (1) nssspw, (2) username, request fails because there's no SESSION[nonce] value anymore.

Thanks for drawing attention to my error, Jerry. This still leaves open the questions:

- 1) Is exposing the server-side salt a serious issue?
- 2) Does exposing the server-side salt render hashing the password in the database moot?
- 3) Any better ideas for accomplishing the concept outlined here?

The article also makes the obvious point that javascript is not always enabled. I figure this could be addressed by including a note that recommends turning on javascript. I think it's fair to assume that anyone savvy enough to turn it off can turn it on long enough to log in. If not, info could just be submitted in plaintext (assuming the

Re: Proposal for Lite Encryption for Login Form without SSL

safety of the nation is not at risk if someone
does happen to be
eavesdropping.)
Tom

Hi Tom,

I've used a similar method before and believe its valid:

1) Is exposing the server-side salt a serious
issue?

It makes it possible to mount brute force attacks on the back
of being
able to sniff traffic. And if you have the same value for all
users
then the attacker only needs to crack one to get them all. I
use a
randomly generated value for all users, call it s_user and to a
two
stage submit – first the username gets sent to fetch s_user
and the
second salt (which MUST be single use and generated
server-side – so
stored in the session – call it s_s for salt(session)) then at
both
ends generate a comparison value:

client:
(send username u)

```
server:  
$s_u=retrieve_s_u($_REQUEST['u']);  
if (isnull($s_u)) {  
$s_u = generate_random_salt();}
```

```
$_SESSION['s_s']=generate_random_salt();  
// send back s_u, s_s
```

Re: Proposal for Lite Encryption for Login Form without SSL

client:

```
enc_pass=md5(password + s_u)
sendable_pas=md5(enc_pass + s_s)
submit(u, sendable_pass)
```

server:

```
$s_u=retrieve_s_u($_REQUEST['u']);
$s_s=$_SESSION['s_s']
$enc_password=retrieve_password($_REQUEST['u']);
$comparison=md5($enc_password . $s_s);
if (comparison==$_REQUEST['sendable_pass']) { // log em
in
```

2) Does exposing the server-side salt render hashing the password in the database moot?

Only if you can be 100.0000000000...% sure that there is no way to access the data – which you can't.

3) Any better ideas for accomplishing the concept outlined here?

It only protects the login credentials, and as others have pointed out it doesn't protect against attacks outside the operational domain (but neither does SSL). You could address the former by using a javascript implementation of an asymmetric algorithm (I believe there's at least one RSA implementation, and there are raw RSA implementations in PHP, but life's just too short to see if they work together – SSL's much easier). Alternatively you could just keep the password client side and use it as a key for symmetric encryption – but you need to use separate frames/windows to retain state client side – which is tricky.

Re: Proposal for Lite Encryption for Login Form without SSL

I'd be willing to bet though that most users would have an expectation that this is more secure than using a self-signed SSL certificate and getting warnings about untrusted content popping up on their browser!

Jerry wrote:

Why do you need to use the same salt (or even the same encryption method) for the database?

You need to use the same salt putting passwords into the database as on the first iteration of the client side hash so the results are consistent. You don't have to use the same hash for both iterations – but why make life more complicated than it need be? Or did I miss something?

(PS SHA1 is generally considered more secure than md5() and in PHP it runs faster too)

HTH

C.

Thanks for the detailed response, C. The user-specific hash is a good idea. How do you execute the two stages? Does the user make two submissions? Some kind of AJAX implementation?

One thing I haven't done is explore the login mechanism on the popular open source PHP packages. Has anyone looked at these and seen anything that they think is worthy of emulation?

Re: Proposal for Lite Encryption for Login Form without SSL

Jerry, you note: if someone can see the traffic going one way, they can see it going the other. How serious a concern is this out in the wild -- esp. for popular sites of general interest but no specific financial or security significance?

It depends. If you're concerned about it enough that you want to encrypt your password when it's being sent, then you should be concerned about both directions being intercepted. It's quite easy to do, and it doesn't take much of a filter to see both sides of the conversation.

I see that facebook, for instance, doesn't seem to use SSL for ordinary users, so presumably uses something like this or perhaps even sends them plaintext. Is packet sniffing a significant vector of attack? Any data or references on the subject?

They probably just send in plain text.

Packet sniffing can be done anywhere between the client and the server. However, since every packet can theoretically take a different route, the most common sniffers are at the ends of the link.

The server end is probably pretty safe -- if you don't trust your host, you have a problem. And they can get to your data on the server, anyway.

The client side is a lot less secure. Someone using a wireless card, for instance, can be intercepted. Encryption might help, if it uses one of the latest encryption schemes. But most public hot spots run unencrypted.

Additionally, cable modems can be insecure. It would be highly unusual for one to be the only user on a cable distribution line. More likely there are dozens (if not hundreds). And with the right software, any person on the link can see all data on the link.

I've read a bit on website and network security and am familiar with many of the various hypothetical attacks. But most real-world instances I'm familiar with involved very high-stakes military or financial events. The famous website exploits I've read about seem to involve things like the js-exploit on myspace that allowed that guy to add himself to everybody's friend-list but didn't involve exploiting the login system or XSS vulnerabilities (which wouldn't be applicable here) or turned out to be cases of social engineering.

Of course, those are the ones which get the news. You don't hear about

Re: Proposal for Lite Encryption for Login Form without SSL

Mom's Homemade Cookie Shop's website getting broken into. It just isn't news.

Anyway, just curious.

Tom

The question here is – why do you want to encrypt the password? Is there any reason you need it encrypted? What's on there that someone would want?

And in that case you probably should be using SSL.

--

=====

Remove the "x" from my email address

Jerry Stuckle

JDS Computer Training Corp.

jstuck...@xxxxxxxxxxxxxxx

=====

The question here is – why do you want to encrypt the password? Is there any reason you need it encrypted? What's on there that someone would want?

A couple reasons:

(1) This is being developed as part of a framework that I plan to reuse on many of my projects. The goal is to incorporate this enhancement more or less transparently. So if it offers even only marginally better security, I figure it's worth the trouble.

(2) As a learning exercise.

Plus, I guess I'm just nervous by nature. :)

Tom

.