What is the best way to send web form authentication data over HTTP?

Asked 15 years, 8 months ago Modified 8 years, 5 months ago

Viewed 2k times

6

A company I know is in discussions to firm up its password security policy across all its web application products.

Right now they are sending username / password authentication in POST forms over HTTP, and thus, they are being sent plaintext.

The simplest solution to the problem is simply to require HTTPS for logon across all our applications, right?

Well, there's some internal discussion about instead doing some kind of roll-our-own client-side encryption of passwords (password + salt, etc.).

Is there an accepted HTTP-only solution?

Opinions are like... well, everyone has an opinion, so I'm looking for credible security literature that can support your recommendation. Don't just google and send me to a blog post... I've already done that and further.

I have found OWASP's recommendations: http://www.owasp.org/index.php/Top_10_2007-

A7#Protection

As well as Microsoft's: http://msdn.microsoft.com/en-us/library/aa302420.aspx

EDIT: Giving your recommendation for using SSL isn't enough. I need some kind of supporting documentation. I KNOW that rolling our own client side encryption is bad. I need to be able to credibly sell that to co-workers and management.

Also, HTTP Digest has been mentioned. Seems nice, but Digest is ONLY for HTTP authentication, and not for data sent over POST.

security http https owasp

Share

Improve this question

Follow

edited Jul 24, 2016 at 7:53

Antti Haapala -- Слава Україні

134k ● 23 ● 289 ● 342

asked Apr 3, 2009 at 19:30

danieltalsky

7,900 ● 5 ● 41 ● 63

10 Answers

Sorted by: Highest score (default)

I highly **recommend against** going with your own solution (in security sensitive environments). Go with

12

SSL. It's a proven technology and it's easy to implement.

Rolling your own security solutions can be really dangerous and even if it's implemented properly (0.000001% chance), it **will** be expensive.

Share Improve this answer

Follow

answered Apr 3, 2009 at 19:32

Mehrdad Afshari

422k ● 92 ● 859 ● 794

I agree, a server certificate is a lot more cheaper and easier to maintain that any other solution out there and keeps your app a lot easier – Eugenio Miró Apr 3, 2009 at 19:38

Thanks for the recommendation, but what I DON'T need is someone's recommendation. I really need credible supporting documentation. – danieltalsky Apr 3, 2009 at 19:40

What do you exactly want? Documentation to crack your solution or to ensure SSL security? – Mehrdad Afshari Apr 3, 2009 at 19:42

No, but I explicitly said in my message that a personal recommendation wasn't going to do it. I can't believe this has 5 upvotes already. Plus, the questions were: Is there any better solution than SSL? AND Is there a decent HTTP method? *I* know rolling our own is bad... – danieltalsky Apr 3, 2009 at 19:45

This is a community QnA. Useful answers are usually much more generic than what the OP wants. BTW, I think *no there isn't* a better method (which is basically what my answer says) is a valid answer to your question. – Mehrdad Afshari Apr 3, 2009 at 19:51

5

If the data itself isn't too sensitive, but the passwords are, I'd suggest using HTTP digest authentication (which is quite a different beast from HTTP basic authentication). It's is quite secure over straight HTTP, and not at all difficult to implement on the server. Nothing is sent over the wire that could reveal what the password is, just information that allows the client to demonstrate to the server that they have the correct password.

If you want a decent explanation of how to implement HTTP digest authentication in your application, Paul James has an excellent article on it.

The only real problem with HTTP authentication is in the browsers themselves: the UI is terrible, but that can be overcome with some Javascript.

You can store the passwords securely by storing the A1 hash.

UPDATE: As mentioned in other answers, while the server can avoid MITM attacks by not accepting Basic auth, the client is still vulnerable because it will.

UPDATE: You can't secure data over POST unless you either (a) do encryption in JavaScript on the client or, (b) do everything over SSL.

You *can*, with a little bit of magic, use HTTP auth with forms. The article I linked to above is called 'HTTP Auth with HTML forms', after all. But that won't be done over POST.

If you *really* need it do use POST, use SSL and salt the passwords in your database.

If you want avoid CSRF, I recommend using formkeys, though the idea goes by many different names, I picked up that name from hacking Slashcode for my own use a few years back, which is where I came across it first.

Share Improve this answer Follow

edited Apr 3, 2009 at 20:02

answered Apr 3, 2009 at 19:38

Keith Gaughan

22.6k ● 3 ● 34 ● 31

It was my impression that HTTP digest authentication was for HTTP authentication, and not for authentication information sent as POST data. – danieltalsky Apr 3, 2009 at 19:41

MITM can change Digest to Basic, stripping out all security without raising suspition :(– Kornel Apr 3, 2009 at 19:43

There's no way to do it using POSTed data. However, one of the articles I pointed to mentions how to use as combination of JS and a form to do more or less what you're looking for. – Keith Gaughan Apr 3, 2009 at 19:43

porneL: Very true. You can avoid it on the Server by not accepting Basic auth, but the client's another story. – Keith Gaughan Apr 3, 2009 at 19:45

+1 to Mehrdad's answer. Going ahead with home-grown cryptography is risky.

Speaking from experience here, after having seen vulnerabilities in home-grown client-side encryption solutions. Most of the vulnerabilities are due to the same reason - the data transfer was protected by encryption with a secret key, but the key itself was exchanged in an insecure manner.

TLS/SSL solves the problem of not just secure key exchange, but also of secure data transfer.

TLS/SSL protects your sensitive information by using assymmetric key cryptography to exchange the symmetric key used to encrypt information back and forth, once the TLS/SSL session has been established. The symmetric key is generated at runtime, and is a shared secret between the client and the server; it is this key that is used to encrypt all other traffic once the session is ready for transfer of data.

The server's public and private keys are used only to exchange this shared secret. The only known way to compromise the shared key is to compromise the server's private key (so that the secret key can then be decrypted). In reality, it requires negligence or malice on the part of the server administrator.

If you roll your own encryption solution, you must exchange the symmetric key, in a secure manner. The

easiest way to do so is to use TLS/SSL; the harder way is to implement your own assymmetric key exchange cryptography solution.

Share Improve this answer

Follow

edited May 23, 2017 at 12:06

Community Bot

1 ● 1

answered Apr 4, 2009 at 15:26

Vineet Reynolds

76.7k ● 17 ● 153 ● 177

2

You'll spend more time and money rolling your own solution and not be assured that it's secure. Industry standard SSL is easy to implement and far more secure out of the box than you can afford to make your own solution.

time == money

Buy a certificate and spend your time working on your application instead of a secure login form.

Share Improve this answer

Follow

answered Apr 3, 2009 at 19:54

Chris Nava

6,802 ● 3 ● 27 ● 31

2

Client-side encryption is helpless against most MITM attacks. Attacker can simply remove your script.

It will only protect against passive sniffing. If that is good enough for you, you can use:

- Hashing implemented in Javascript. It's easy to implement challenge-response for initial login, but don't forget that for attacker session cookie is almost as good as password (you'd have to limit login to single IP and/or use script to generate one-time cookie for every request, which I imagine would be difficult and brittle solution).
- HTTP Digest authentication. More secure, as it can use one-time hashes, mutual authentication, etc., but standard UI is absolutely repulsive.

...just use SSL.

Share Improve this answer Follow

edited Apr 3, 2009 at 20:38

answered Apr 3, 2009 at 19:40

Kornel

99.9k ● 38 ● 232 ● 318

Why is it hard to protect against replays? If you're hashing a password based on a salt from the server, said salt can be expired on the server after first use. Still not protected from MitM, but replay shouldn't be an issue. – Chris K Apr 3, 2009 at 20:10

It's OK if you only want to keep password safe, but if you want to secure whatever the password is securing, you have to protect session cookie as well as password itself.

Otherwise attacker can sniff session cookie and use that to perform malicious actions. – Kornel Apr 3, 2009 at 20:40

2

HTTP only solutions will always be susceptible to man-in-the-middle attacks.

EDIT: A network trace would be an easy way to prove that HTTP isn't secure.

Share Improve this answer

Follow

edited Apr 3, 2009 at 22:52

answered Apr 3, 2009 at 19:39

David

34.5k ● 3 ● 64 ● 80

If you're using HTTP auth, this can be avoided somewhat by not accepting Basic auth. – Keith Gaughan Apr 3, 2009 at 19:41

1

Okay, here's the only answer you need: Your BANK only supports login/auth via SSL. If there was a better way to do it, they would.

Share Improve this answer

Follow

answered Apr 3, 2009 at 20:12

Chris K

12.3k ● 7 ● 40 ● 65

> Well, there's some internal discussion about instead doing some kind of roll-our-own client-side encryption of passwords (password + salt, etc.).

The first thing we need to address is data in transit vs. data at rest. From the OP it sounds like the big concern is username/password in the clear over the Internet. So, you really care about data in transit. HTTPS via SSL is a tested method of doing this (and it's a pretty easy fix!). Now, if you really want to roll your own for data at rest (in a db, on a file server, etc.), that's a different beast, but the existing methods for data encryption are going to be better than a roll your own.

Relying on the client side for your security is bad. Period. In a corporate environment, yes, you can somewhat rely on a standard setup. But, ultimately, users are dumb and you may run into someone disabling javascript or whatever client based solution you roll out, so they end up sending the username/password in plain text anyway (even if your servers dont' know what to do with it because it's not in the expected format.

Roll-your-own is not going to be subject to the scrutiny that an existing solution has. You could end up adding additional security issues into the mix because of the code.

Follow

answered Apr 3, 2009 at 20:56

community wiki
Lo123

1

We just put out a web/mobile app to do some of this stuff. It creates random URLs for logins, using HTTPS and a hashed/AES encryption method for the database storage. Theres a simple JSON API to use it without our UI, heres our writeup, give it a look..
http://blog.primestudiosllc.com/security/send-time-limited-secure-logins-with-timebomb-it

Share Improve this answer

Follow

answered Aug 28, 2010 at 18:24

Prime Studios
61 ● 2 ● 5

0

you mentioned doing your own encryption plus salt... I suggest using javascript md5 (it's also used by yahoo on their non ssl pages, or so he claims)...

And you double hash the password... some may argue that double hashing it would make it prone to collision attacks. I would have to disagree because people have been able to do till now md5 signature collisions only on files, where large amount of data is md5'ed...

If it's a big corporate website (and there would be reasons to break in) there is no excuse not using SSL.

Share Improve this answer

Follow

answered Apr 6, 2009 at 13:49

dblackshell