Tobias Gurtzick Follow
Tobias is a passionate Technology, OSS and DevOps Evangelist. He engineers/architect solutions and landscapes. He has been programming for more than a decade.
Nov 29, 2017 · 10 min read

Your passwords are at risk! Why SSL is not enough and how to solve it.

When we talk about passwords this topic ends up always with people that are concerned about their privacy, even if they're non professionals in IT or IT-Security. All in all, this is one of the examples where everyone is aware of the dangers resulting from the risk of loosing their password to a third party or even by giving someone they know access.

Given this awareness makes the following even more frightning: Most of you already have too many third party people knowing your passwords and you do not even know about it.

. . .

Uhhhm SSL...?

But wait, didn't everyone told you as long as there is that green lock on your browser, or simply spoken you're browsing a SSL encrypted website, everything is fine? Ok, ok… . You're not that wrong, let us step back for a moment.

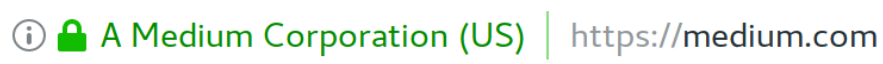
So is SSL broken yet?—No not **yet**.

Then what is actually wrong with it? To be correct nothing, but what **companies** and especially **enterprises** make out of it.

To know if you're affected, you would need to answer the following questions, which I separated into two kinds of question, those easy for everyone to answer and those that requires a bit more knowledge:

Easy Questions first:

- In your company/enterpise, are you restricted to browse through a proxy?
- Did you ever noticed that on google or another highly trusted site you **do not** see the usual green lock alltogether with the companies name printed in green? See below for a screenshot, how this **should** look like in your chrome or firefox browser (firefox is the second screen).



Advanced Questions:

- Do you have non standard root certificates installed into your System?
- Did you ever visited a website, whichs SSL certificate was signed by one of the "CA"s of this non standard root certificates that are not internal?

If you answered any of the **easy questions** with **yes**, **you might** be affected. Should you answered every **advanced question** with **yes**, **you** actually **are**! To note: The intentions of your company/enterprise are mostly not evil, they do not even

thought about this being a problem. Instead they actually have a valid interest to inspect traffic, which however should not be the solution to the problem. So I **don't** call out to **blame** them, except maybe for breaking security and privacy in the wrong way. But this is hardly discussed topic already anyway. See the linked article, which goes into all of this in more depth.

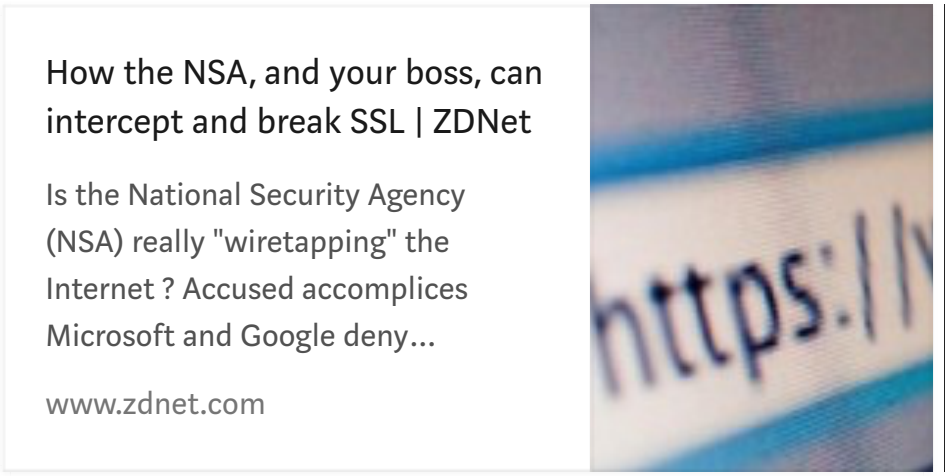
> Fighting Back Against SSL Inspection, or How SSL Should Work
>
> In the past, many websites used HTTPS in exceptional cases, such as the transmission of the password from a login page...
>
> securityevaluators.com

But why is this so important now?

Because your company infiltrated your privacy! This injected root certificates effectively enables them, to **fake** a valid certificate and **fool** you believing this would be actually a secure connection. Lucky enough after they spy on you, they mostly reencrypt the outgoing traffic again before sending it. So at least no one outside of the company network should be able to eavesdrop for your passwords, unless… they hacked them. However, everyone in IT with enough access, can get knowledge of your password. And even worse, in the case of a security breach, the attackers will too! So your company not just spy at you, they also put you at risk. If you followed the media, than you know that almost every company is getting hacked once in a while. And even worse, it is very probable that your company was already hacked or is currently actively being spied, but either does not know they're yet or they keep it secret

from everyone that they have been hacked, which sadly seems to be the case for way to many security breaches.

If you want to know a bit more about the topic, you may want to read about this here:



So should companies be allowed at all to peak into your traffic? Should they be allowed at all to intercept SSL traffic? This is a question you should think of yourself and try to find an answer.

My *personal opinion* on this is:

> *NO, they definitely should be even regulated to NEVER peak at your traffic, regardless of their interest in preventing corporate espionage.*

But this goes a bit too off topic, maybe we can start a discussion to this or I write an article where I go in depth why I think like that and how to tackle the problem instead.

. . .

How may I protect myself? — Spoiler, not the solution.

So how can you protect yourself? — Never use your “casual” passwords for your company accounts and never login into google with your private account or other pages that are not job related, when browsing through this intercepting proxy. And as a real security measure: Do use multi factor authentication (MFA) wherever possible, at least 2 factor though (2FA).

Does this sound bad to you? In fact, it is, except for the MFA part!

However, it is recommended anyway to never use your private passwords for company use, to protect you **and** the company. You may never want to browse websites unrelated to your work requiring credentials, if you do not want to loose them to the wrong persons.

. . .

Solving the problem!

But this recommendation does not sound like a solution! — Yes, you’re completely right! However I intended it to be as little technical as possible, to enable everyone to understand the problem. From here on I’m going to cover all the technical topics, so here is the

> *TL;DR; for non technical users, unless you want to really understand this stuff — Authentication as a*

concept itself needs to be fixed, to keep your credentials safe from third parties.

To everyone *TL;DR;* just jump now to the last part of this article.

Existing solutions

First of all, there is a solution, that is not quite practical as you can't really carry them around safely. I'm talking about Client Certificates, however, they're **currently** way to hard to use, especially for non technical users and you always need to have them with you whenever you want to access any of your websites, i.e. google, for example at a friend, *note that it is already a bad idea to enter your password, key or whatever else on any device other than yours*. And carrying with you means: Install them from i.e. an USB Stick, to the machine, use them, deinstall them from the machine again.

. . .

A new Solution to the problem, *without changing anything for the user,* is to have a secure authtentication solution instead. This means simply deviating away from the current, send username and password and we're all good.

A part of the idea is not new, one of the first times I have seen this in action is on the popular bulletin board software vBB. They do not send, *if javascript is enabled though*, your password, but the hash of your password to the server. This eliminates the breach of the password, if anyone should for any reason be able to inspect your traffic. However, this is still not enough and

broken. Why? Because the attacker still can login, **everywhere** where you used that password and **where** a website accepts the **same kind of hash** of your password.

Before we go fix this problem, let's go back and review our existing Systems.

What we currently have is a database storing your passwords. The following measures are taken to protect this passwords in this database:

- To protect this passwords from security breaches, they're stored hashed.
- To protect this hashed passwords from rainbow table attacks they're salted.
- To protect this hashes from being brute forced, you choose anything stronger than md5. For example sha512 and you probably also hash multiple rounds. Or you rely on bcrypt or something like PBKDF2.

So we're not touching any of these, but this includes the information, that we need to know for both sides:

- Which algorithm is being used in which way and with which parameters?
- What is the users salt?

With that information we now can go ahead and let the user hash his password instead of the server. This solves the problem of the password on the wire, the salt is now possibly known by a third party, but this is no problem since a salt is not really a secret information.

This being solved, the attacker still can login with the hash that

you used to login to this website. And additionally he is still able to run brute force attacks against your password. Let us first solve the problem of reusability.

Solving this needs us to provide some information that we can use to bind a login to a specific authorization request. A randomly generated challenge for example, that our user will use to hash the password in a second round. This makes the authorization attempt usable just this single time, unless you're provided the same challenge twice.

So currently we have the following delivered to the Server:

```
Hash(challenge||Hash(salt||password))
```

We still are reusable, to solve this, we include the current time as well. This date might be as well provided by the Server to avoid time desync scenarios.

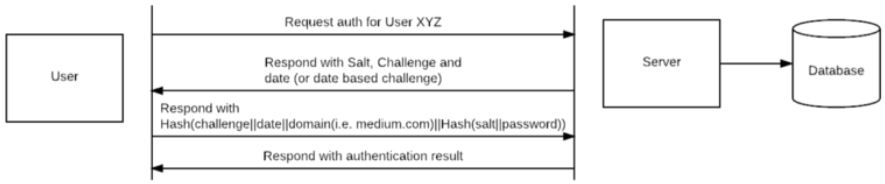
```
Hash(challenge||date||Hash(salt||password
))
```

With this date included, the server can reject any authentication attempt, that has been too far in the past. And as a last protection level, we also add the domain of the service to the hashing function.

```
Hash(challenge||date||domain(i.e.
medium.com)||Hash(salt||password))
```

This finally protects against replay attacks unless, you're getting the very same challenge from the same site at the same time. Which is unlikely enough to not further improve here.

So finally we end up with the following:



. . .

Improving further

This as an authentication strategy wont reveal the password on the wire anymore, but of course require javascript to operate, at least when using a browser.

We can of course still improve on this, one thing we will never fix is that an attacker might be still able to brute force the password. But at least his game has gotten way more difficult and if you have chosen a secure password, it might be even near to "impossible", if that is fair enough.

I wont go to deep into further improvements here, instead I will provide you with

- An example implementation
- Some loose thoughts

I will start with some loose thoughts what could be improved, regardless if it makes sense or not.

One thing that could be added is encrypting the whole authentication attempt, which would probably open the same problems we have with SSL currently. The only thing that is possible without exchanging a key, would be to encrypt with the users Hash(salt||password), but this would be meaningless, as it would work just in the direction of the server where it would probably even downgrade the cost to brute force and in the case of the client anyone could request an attackable object for any user for that particular server. Which would be a horror scenario.

Another idea could be to integrate authentication more deeply into the browser. So to have a browser provided native implementation of a secure authentication just like this one. This is not to far off, as current browser also already support client certificates, which are still not as user friendly as traditional passwords paired with MFA.

And lastly, although a bit off topic:

A way to sign the javascript code and allow the browser to only execute the code if it was validly signed. Here we may end up again with the same problems we already have with SSL but this would be still benfitial. I could think of an implementation with PGP signing where the public key is stored in a trusted resource, which as stated above, again results in the problem we have with SSL: trust.

. . .

Here comes all the magic

I published an actually usable module which acts as a PoC. You can find the PoC in the examples folder.

wzrdtales/secure-passwords

secure-passwords - An authtentication strategy that finally stops exposing users passwords on the wire.

github.c

. . .

A good step forward, but...

We still have some way to go, first of all, everyone that got a login, so everyone except OAuth2 Users, would need to fix their authentication first to get us to a point where at least credentials remain secret again.

So as a summary, there are still more problems, with the exact same problem, but in way more delicate scenarios. I.e. a scenario where you actually are even forced to use data you should not want to use in such an environment. To call them out: Payment Data. Many companies do not provide their employees proper payment equipment (not everyone needs it, I'm just referencing those that do here), such as credit cards. That leads to a quite common practice, that you pay first and

the company pays you back after you've entered all the bills to their accounting. So you're actually forced to use your payment details, in a non trusted environment, unless you do half your work at home on your own equipment instead. This is a problem as you're not just revealing your credit card number, but all details you need to pay online as well. Just praise that no one infiltrated the company at that moment and dumps your credit card data, to sell them online afterwards.

So all in all, cutting privacy in the way that companies practice today, is way more dangerous to everyone involved, than probably many thought before. To work on all this problems, we will need solutions that solve the original problem that lead to actually cutting privacy. May be, some day, we finally get there, making this half baked privacy cutting solutions a thing of the past again.