Password-manager friendly (PMF)

Frank Stajano, Max Spencer, Graeme Jenkinson

Note: These slides were originally presented by Max Spencer at Passwords 2014 (http://passwords14.item.ntnu.no/). The slide notes are not a script for, or transcript of, that presentation.

These slides cover two topics:

- Things password-managers get wrong and why implementing a password manager is hard.
- Our “password-manager friendly” or PMF semantic markup, which can be used to improve the effectiveness and reliability of password managers.
The accounts.google.com login form.

Ignoring issues of style, to humans websites seem to present a remarkably consistent user interface, for logging in with a username and password.

They all have:

- A box to type your username in
- A box to type your password in
- A sign in button to press

and almost always:

- A “stay signed in” checkbox
More login forms from a selection of popular websites. Clockwise from top left: facebook.com (Alexa rank as of 08/12/2014: 2), yahoo.com (4), taobao.com (8), baidu.com (5).

Other login forms may be styled or laid-out differently, but they all have the same set of controls. The two login forms in the bottom half of the slide are from the popular Chinese websites Baidu and TaoBao.
However, this apparent similarity hides a wide array of variations, and from the point of view of the author of a password manager deficiencies, in the implementations of these interfaces.

The slide shows Firefox’s built-in password manager offering to save an incorrect password. Upon submitting the incorrect password, the Facebook server returns a 200 OK response which is fine, 200 OK just means a page was successfully served. The human user knows there was a problems because of the big red box on the screen after the HTML document in the response has been rendered, but the password manager is not, or cannot be programmed to reliably detect such signals.
Internet Explorer makes the same mistake.
Interestingly Chromium and Google Chrome don't offer to save an incorrect Facebook password.
Looking at the source code reveals the following logic:

If the HTML document in the response from the server contains a form with the same action as the one which was submitted, then assume the submission failed.

This seems like a reasonable heuristic and it will probably work a lot of the time. It works for facebook because a login form appears right below the red error message.

However, we could envisage false positive cases where it would not find a form with exactly the same action and wrongly offer to save an incorrect password, or much worse false negative cases where it would fail to save a correct password. For example consider a change password form which when submitted successfully still returns the user to the same page. In this case the above heuristic would indicate an error when in fact there was none and the password manager will not update its record of the password.

Furthermore, even if this heuristic is in face near-perfect, it’s still a waste of effort for every password manager to have to replicate and maintain the same code.
The slide shows a filled-in Amazon registration form. When submitted Firefox correctly offers to save the password, but unfortunately it gets saved without the associated username (“example@example2.com”).
When viewing the login form page, the password manager is able to automatically fill the password field, but the user must still enter the username manually.

Presumably this problem occurs because the password manager cannot reliably detect which input in the registration forms contains the username. This is the kind of ambiguity that our proposed PMF markup can remove.
In this case Chrome does not behave correctly either. When the registration form is submitted, like firefox it saves the password without an associated username. When the login form is submitted it saves an entirely new record.

Saving passwords without usernames, possibly with duplicates, seems likely to make password changes and resets difficult. Multiple accounts for the same site are likely to cause even more problems.
VimeUhOh.

There are a range of other problems caused by the use of JavaScript. For example, Vimeo (pictured) have a script which dynamically adds a hidden form field containing a Cross-Site Request Forgery (CSRF) token which means no user or software agent can log into Vimeo with JavaScript disabled.

The same is true of other sites for a range of reasons. For example baidu.com which we saw earlier dynamically adds its entire login form to the HTML Document Object Model (DOM) using JavaScript.

Password managers aside, these idioms reduce accessibility for users who can’t, or don’t want to run scripts and are considered bad practice. Our PMF markup can’t solve these problems, but we wish to highlight the fact that using JavaScript in these ways may reduce the reliability of password managers or other software agents acting on the users behalf.

After all, it’s in the best interests of site operators to make logging in both easy and secure!
PMF semantic markup

Note: See http://pmfriendly.org/specification.html for a complete specification of the PMF markup.

Our proposed password-manager friendly or PMF semantic markup removes the ambiguity that password managers struggle with.
The PMF markup takes the form of semantic class names which mark the purpose or meaning of certain HTML elements.

A simple example of a semantic class name, “warning”, is shown in slide.

Semantic class names were chosen because:

- They are simple and easy for website authors to add to new and existing sites.
- Their use is supported by the W3C.
- They are interoperable with other standards and web frameworks. Standards may specify values for other element attributes such as `name`, or their values may be automatically generated by web frameworks. But any HTML element may have any number of class names.
Microformats

HTML marked-up with h-card semantic class names.

Furthermore, semantic class names have been used effectively by other microformats such as h-card (http://microformats.org/wiki/h-card) which is for marking-up personal information.
Forms

<form action="/login" method="POST" class="pmf-login">

- pmf-register
- pmf-change-password
- pmf-reset-password

There is a different PMF class name for each of four different types of password form.
Username inputs

<input type="text" name="user" class="pmf-username" />

Any input element where the user would enter their username is marked with the pmf-username class.
This prevents this problem of passwords getting saved without associated usernames upon registration.
The fields marked with the red arrows should be marked with the `pmf-username` class.
A password form which relates to a specific account and username, such as a password-reset form, must contain a hidden-type input marked with the `pmf-username` class and with its `value` attribute set to the username of the relevant account. This allows the password manager to update its records when passwords are changed, even when it has multiple records for the same site.
Passwords

<form action="/change" method="post" class="pmf-change-password">
  Current password:
  <input type="password" name="current" class="pmf-password" />

  New password:
  <input type="password" name="new" class="pmf-new-password" />
  <input type="password" name="confirm" class="pmf-new-password" />
</form>

There is only one password type for HTML input elements, but there are two PMF semantic class names to distinguish inputs for existing passwords and inputs for new passwords.
Stay signed in checkboxes

Stay signed in?
<input type="checkbox"
    name="persist"
    class="pmf-stay-signed-in" />

A checkbox-type input element which the user would check to stay signed in for a longer amount of time should have the `pmf-stay-signed-in` class. This allows a password manager or other software agent to apply a global staying signed in policy on the users behalf.
Errors

<div class="pmf-error">
  Incorrect username or password
</div>

All error messages related to the submission of a password form, within a HTML document, must have the pmf-error class.
Marking errors unambiguously in this way makes it trivial to avoid simple mistakes such as offering to save incorrect passwords.
We know from password leaks that many users optimize their passwords for memorability rather than security and some websites attempt to force their users to select passwords that are harder to guess using password policies. We don’t want to get side-tracked into a debate on the effectiveness of these policies, but they are widespread.

Therefore we have defined a simple specification for machine-readable password policies to allow password managers to generate strong, compliant passwords for their users.
PMF password policies are specified as JSON objects with four keys:

- **minLen** is the minimum number of characters the password must contain.
- **maxLen** is the maximum number of characters allowed. We don’t think imposing a maximum length is sensible, but here we are just trying to make the most common password policies expressible.
- **mustHave** is a list of sets of characters which the password must contain.
- **mayHave** is a set of characters which the password is restricted to.

Example policy

```json
{
    minLen: 8,
    maxLen: 16,
    mustHave: ["lower", "digit"],
    mayHave: ["ascii"]
}
```
Embedding policies

The website author embeds these password policies directly into the HTML of the page, as the value of a hidden form field.

The hidden form field is marked with the `pmf-policy` semantic class to make it easy for a password manager to find. It also has no name attribute so that it won’t be submitted back to the server when the form is submitted.

Of course alongside this machine-readable version of the policy there should be a human-readable one to accommodate those users without pmf-enabled password managers. In fact, from this machine readable description of the policy, a tool could generate the required localized human-readable version.
Summary

- Password managers currently have to rely on heuristics.
- We propose simple semantic markup to make websites password-manager friendly.
- Make your websites PMF!

Thanks for listening

Max Spencer (max.spencer@cl.cam.ac.uk)
Extra slides

More detail on password policies
For a more concrete example, let's take the policy of amazon.com. Amazon don't have their password policy on their account creation page, but here's an approximation we found with a bit of testing:

- You can't use fewer than six characters or more than 128.
- You aren't required to include any specific classes of characters.
- It seems to accept any ASCII character and in fact, any printable character up to unicode number 255.

This slide shows some notation for defining arbitrary character classes:

- There are some predefined classes of characters like "ascii", that's all of the printable ASCII characters from number 32 (space) to number 127 (tilde).
- You can also include arbitrary other characters either by entering directly, or using the JSON representation of their unicode code point ("u" followed by four hexadecimal digits).
- The "..." item is a shorthand we define for ranging between the characters on either side.

We wanted website operators to be able to express more complex policies where passwords of different lengths are subject to different requirements. In the above example:

- Passwords between 8 and 11 characters in length require lower and upper case characters, numbers and symbols.
- 12-15 require lower, upper and number.
- 16-19 require lower and upper.
- Those of 20 characters can consist of whatever characters you like.
This is expressed in our policy language (still just JSON) as an array of sub-policies and a valid password is one which matches at least one of the sub-policies.