D. Hühnlein, H. Roßnagel, C. Schunck, M. Talamo (Eds.): Open Identity Summit 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 135

# Password Policy Markup Language

Moritz Horsch, Mario Schlipf, Stefen Haas, Johannes Braun, Johannes Buchmann[1]

**Abstract:** Password-based authentication is the most widely used authentication scheme for granting access to user accounts on the Internet. Despite this, there exists no standard implementation of passwords by services. They have different password requirements as well as interfaces and procedures for login, password change, and password reset. This situation is very challenging for users and often leads to the choice of weak passwords and prevents security-conscious behavior. Furthermore, it prevents the development of applications that provide a fully-fledged assistance for users in securely generating and managing passwords. In this paper, we present a solution that bridges the gap between the different password implementations on the service-side and applications assisting users with their passwords on the client-side. First, we introduce the Password Policy Markup Language (PPML). It enables a uniformly specified Password Policy Description (PPD) for a services. A PPD describes the password requirements as well as password interfaces and procedures of a service and can be processed by applications. It enables applications to automatically (1) generate passwords in accordance with the password requirements of a service, (2) perform logins, (3) change passwords, and (4) reset passwords. Second, we present a prototypical password manager which uses PPDs and is capable of generating and completely managing passwords on behalf of users.

**Keywords:** Passwords, Password Generation, Password Management

## 1 Introduction

Passwords are the dominating authentication means for granting access to user accounts on the Internet. The concept of password-based authentication is well-established and well-known by users. It is platform independent and can be used across many applications. From the perspective of services, password-based authentication can be implemented with little effort and has a negligible cost per user [Bo12b]. The key challenge for users is the creation of secure passwords and their proper management. Users must create strong and individual passwords for each of their accounts and need to memorize them for later use. Moreover, passwords should be changed on a regular basis. The possible oblivion of passwords require that users keep the recovery email address or phone number of the user accounts up-to-date. Doing all these tasks for the huge amount of passwords that users have today is practically impossible. This causes that users choose weak passwords and do not change them. However, this is crucial because passwords are the sole barrier that protects the multitude of personal data such as email and photos stored in the user accounts.

For several of these tasks, users can fall back on password generators and managers. They allow creating strong passwords and their secure storage and thus eliminate the burden of memorizing passwords. However, their assistance is nonsatisfying, because they still

[1] Technische Universität Darmstadt, Hochschulstraße 10, 64289 Darmstadt, {horsch | mschlipf | shaas | jbraun | buchmann}@cdc.informatik.tu-darmstadt.de

require a lot of manual user interactions. Users need to adapt generated passwords and the assistance for login and password change is error-prone or limited to a few services. The key problem is that password-based authentication is not standardized and services implement passwords in different ways. They have different password requirements as well as interfaces and procedures for login, password reset, and password change.

We address the issues induced by the different password implementations by services with the definition of the Password Policy Markup Language (PPML). PPML allows a structured description of password requirements as well as password interfaces and procedures of a service. Based on such a well-defined Password Policy Description (PPD), password creation and management can be automated and performed by applications without manual user interaction. More precise, optimal passwords are generated in accordance with the respective requirements and passwords can be changed or reset on behalf of users. This facilities a secure and ubiquitous user-side password management.

This paper is organized as follows. After a short presentation of related work in Section 2, we provide our main contributions:

First, in Section 3, we analyze the *password life cycle* and identify the different challenges for users. The password life cycle is described based on the work done in [SB14, Ch14]. It illustrates the different stages of a password and related tasks for users. The cycle starts with the generation of a password, continues with its management, and ends when the password cannot be used anymore. For each of the identified challenges we describe existing approaches and point out their drawbacks. We show, that many password related problems are unsolved so far.

Second, in Section 4, we specify PPML and the syntax of PPDs in detail and show how PPDs solve the identified open problems. We present how password requirements as well as password procedures and interfaces of a service are described in a PPD. We explain how the necessary information to enable an automatic password management are specified in a PPD. Moreover, we provide an exemplary PPD as well as PPDs for 5 existing services.

Third, Section 5 presents the implementation of our solution. First, we present a central service for making PPDs available to applications. Second, we provide a password manager that is capable of automatically creating passwords in accordance with service's password requirements and changing them, based on the information provided by the PPDs.

Finally, Section 6 concludes the paper and presents future work.

## 2 Related Work

There are multiple studies such as [Fu11, Fu07, WW15, FH10] which analyze password policies of services. All point out the enormous diversity of password requirements and password management implemented by the services. Shay et al. [SBB07] present a language and simulation model for password policies. Their language is based on the generic authentication policy language AuthSL [Sq07], but it is not capable of expressing password requirements of real services. The simulation model is later enhanced by technical

and human factors [SB09], but it still focuses on simulating policies instead of expressing policies of existing services. Stobert and Biddle [SB14] conducted a series of interviews to analyze how users cope with the challenge of creating and managing passwords. Based on the answers of the participants they identify a password life cycle which follows the password behavior of the users. Choong [Ch14] develop a cognitive-behavioral framework which presents the cognitive process and user behavior of managing passwords in order to provide guidance on future research directions. Stajano et al. [St14] propose semantic annotations for password-related HTML forms to improve the effectiveness of password managers. They introduce a set of class names which should be used by web developers for the `class` attribute of HTML elements. Password managers can detect these class names and interpret the meaning of the HTML form. Whereas this approach requires that every service updates its website, our approach is completely independent from the services.

## 3 Password Life Cycle

In this section, we describe the password life cycle and show the numerous challenging duties and tasks of users with regard to passwords.

The password life cycle is a progression of stages through which each password passes (cf. Figure 1). The circles depict the states of a password from the perspective of a user. The arrows represent tasks for the user. The dashed arrow illustrates the possible oblivion or expiration of a password. The cycle begins with the creation of a password. This task is repeated until the password fulfils the individual password requirements of a service. Then the user needs to assign the password to a service and memorize it. For each login or password change the password must be recalled by the user. The change of a password leads back to the creation and finally to the memorization of the new password. In case that the user forgets the password or it expires he or she cannot use it anymore. The user can reset the password which requires the creation and memorization of a new password.

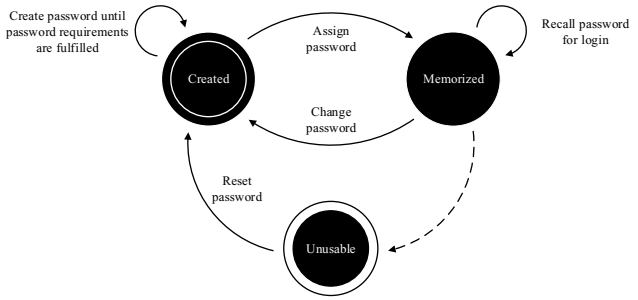


Figure 1: Password Life Cycle.

We use the term *password creation* to refer to the part of the password life cycle which addresses the creation of a password. And we use the term *password management* to refer to the part of assigning, memorizing, recalling, changing, and resetting a password.

In the following, we take a closer look at the different stages and transactions of the cycle. We show how users cope with the various tasks in practice and point out security and

usability problems. Moreover, we analyze existing proposals to mitigate these problems. We show that password generators and managers are the best approaches today. However, they are not solving the problems in a satisfactory manner, because they still require that users need to do many tasks manually. For an overview of approaches to replace passwords we refer to Bonneau et al. [Bo12b].

### 3.1 Password Creation

The creation of passwords that resist brute-force [Ke12, We09], dictionary [Bo12a, We10], and social engineering [Ca13] attacks is difficult. Users tend to choose passwords that are simple and easy to remember, but at high risk of compromise [ASL97, BK95, DMR10, FH07, ZH99]. However, strong passwords are essential for the user accounts' security.

Services implement password requirements and password meters to enforce or rather encourage users to select strong passwords. Password requirements are fixed rules with respect to the password length and the allowed and/or required characters. They do not cover the issue of common passwords like *123456*, where the password on the one hand might fulfill the requirements, while on the other hand is easy to guess.

Password meters assess the strength of passwords. They provide dynamic feedback to users by labeling passwords as weak, medium, or strong. On the one hand studies have shown that password meters lead towards more secure passwords [Ur12, Eg13]. On the other hand, in practice they are highly inconsistent in assessing the password strength. Even obviously weak passwords like *password1* are rated as strong which makes users believe that they have chosen a strong password, but actually they have not [dCdCM14].

In the past it was generally acknowledged that users should create passwords based on mnemonic phrases (so called passphrases), because they are easy to remember but hard to guess by current cracking tools [Ya04]. However, the recently published tool Phraser [SS16] uses common sentences to crack passphrases consisting of up to 20 characters.

Another approach are password generators. They create random and strong passwords based on predefined password generation rules. These rules can be modified with respect to the length and the allowed character sets for the generated passwords. However, in practice such passwords are often rejected by services because they do not comply with the various password requirements of services. For instance, the generated passwords are too short, too long, or do not contain a special character. One possible solution would be that all services agree on a single set of requirements [Al12, St14]. Yet, this seams unrealistic with respect to the multitude of existing different password implementations and security needs. Also, a common set of generation rules that fit to the password requirements of all services is not realizable due to the enormous diversity of requirements. Wang et al. [WW15] mention that such a set does not even exist for a set of 50 services. Currently, the only solution for users is to look up the password requirements of each service manually and configure the password generator accordingly. This is error-prone, very inconvenient for users, and not infrequently prevents users from employing password generators at all.

### 3.2 Password Memorization

Memorizing strong and individual passwords for many user accounts is practically impossible. Users are reusing passwords to cope with this issue. However, this bears the risk that adversaries get access to multiple accounts by just obtaining a single password. To prevent this, users must use individual passwords for their accounts. One possible solution for the problem of memorizing numerous passwords is to store them. This also solves the problem of memorizing which password is assigned to which service as well as recalling passwords even if they are rarely used. However, storing passwords demands an protection mechanism (usually another password) to protect them from unauthorized access. Moreover, the stored passwords need to be available on all devices of the users, so that they are able to access their accounts at anytime and from anywhere. This becomes even more challenging in case that the passwords are changed on a regular basis.

In comparison to diaries or text files, password managers are the best solution for storing passwords. They securely store them in a database and protect them by a *master password*. Furthermore, the database can be stored online in order to share it between arbitrary devices. Password managers can also automatically fill out login forms to make the use of passwords for users very convenient. However, this requires complex heuristics to detect login forms. Such heuristics are error-prone and can even be exploited by adversaries to extract passwords from the password manager without any user interaction [Si14].

### 3.3 Password Change

Regularly changing passwords of all user accounts is a very time-consuming task. Therefore, users barely change their passwords [Ha15, THB15], even after security breaches or exceptional events like the Heartbleed bug [Co14, Da14, Pe14]. However, changing passwords is crucial to invalidate former possibly compromised passwords.

The password managers LastPass [La16] and Dashlane [Da16] provide means for changing passwords automatically. However, both support only popular services and do not allow to add further services. LastPass is implemented as a browser extension and performs the password change in a browser tab. This is error-prone, because any interaction or close of the tab causes a failure of the password change. Furthermore, this solution makes it practically impossible to change a huge number of passwords at the same time. In comparison, Dashlane sends the user's username as well as his or her old and new password to a Dashlane server, which performs the password change on behalf of the user. As a consequence, the company behind Dashlane gets to know the passwords of the users!

### 3.4 Password Reset

Memorized or stored passwords might be forgotten or lost. Users need to answer security questions or need to prove access to a recovery phone number or email address to get access to their accounts again. However, memorizing and keeping all recovery information up-to-date is practically impossible. Today, there exists no solution for this problem.

# 4 Password Policy Markup Language

In Section 3, it was shown that password generators and managers are the best approach for users to obtain and use secure passwords, but they still require many manual user interactions. Users need to adapt generated passwords and the assistance in automatic login and password change are error-prone or limited to a few services. The root of these problems are the different password implementations of services, which prevent a fully automated generation and management of passwords.

In this Section we present the Password Policy Markup Language (PPML) which solves this problem. PPML enables the definition of Password Policy Descriptions (PPDs) for Internet services. A PPD is a standardized description of the password requirements as well as the password interfaces and procedures of a service. The objective of a PPD is twofold. First, it provides all required information to enable applications to completely automate the password life cycle. More precise, a PPD facilitates applications to (1) generate passwords in accordance with the password requirements of a service, (2) perform logins, (3) change passwords, and (4) reset passwords on behalf of users. Second, in case that an automatic password management is not possible for a service (e.g. in case that users need to enter a CAPTCHA to change a password), a PPD provides all information to assist users in accessing and performing the password interfaces and procedures manually. In detail, a PPD provides the URLs to the password interfaces so that users can directly access them out of an application instead of finding them for each service by hand.

We conducted an analysis of password requirements as well as password procedures and interfaces of 200 representative services[1] in order to develop a comprehensive and representative specification for PPDs. Based on the results, we identified common patterns and created a universal description scheme for password requirements, password management procedures, and technologies of password interfaces. Finally, we transformed this common description into a XML Schema. XML is well-specified and supported by many programming languages, which enables an easy integration of PPDs in password generators, password managers, and other applications.

A PPD is presented by a XML element `<ppd>`, which has two attributes to identify and manage PPDs. The attribute `url` specifies for which URL (i.e. service) the PPD is valid. The attribute `version` defines a version number of a PPD. It allows applications to differentiate between different versions and to update a PPD in a reliable manner.

We describe in Section 4.1 how to express the password requirements and in Section 4.2 the password management of a service. An exemplary PPD can be found in Listing 1.

[1] The Alexa Top 500 US list [Al15] reduced by websites with pornographic and illegal content, non-english websites, and websites that do not have or allow the creation of user accounts (e.g. banking websites).

### 4.1 Password Requirements

A PPD allows specifying the following password requirements (see also [Ho16]).

- *Character Sets*. The `<characterSets>` element defines a list of allowed character sets. Each set is described by a `<characterSet>` element and defines a name (e.g. numbers) and the list of characters (e.g. 0..9).
- *Properties*. The `<properties>` element defines further restrictions for the password as a whole and the character sets. It contains information about the password length and its expiration. Character set restrictions are defined by a `<characterSettings>` element. A PPD allows the definition of minimum and maximum occurrences of characters, position restrictions for characters, and choices of character sets.

### 4.2 Password Management

In addition to the password requirements, a `<ppd>` element contains a `<service>` element that describes the password management of a service. It provides information about the registration, login, password change, and password reset which are represented by a `<register>`, `<login>`, `<passwordChange>`, and `<passwordReset>` element, respectively. All elements contain a `<url>` element which is described in the following:

- *Location of Password Interfaces.* The `<url>` contains the location of the HTML form for the registration, login, password change, and password reset at the service's website. This information is intended to directly guide users to these password interfaces to access them manually. The information for accessing these interfaces automatically are part of the `<routines>` element (see below).

In addition, the `<login>`, `<passwordChange>`, and `<passwordReset>` elements contain a `<maxTries>` and `<routines>` element which are described in the following:

- *Retry Counter for Passwords.* The `<maxTries>` element specifies the number of attempts to enter a password. In case of the login it defines the number of possible login attempts. In case of a password change it defines how often users can enter an incorrect old password. And in case of a password reset it defines how often users can answer security questions (or enter recovery information) before an account gets completely disabled. Similar to the `<url>` element, the `<maxTries>` element is intended for an manual interaction with the service's password interfaces.
- *Password Procedures.* The `<routines>` element describes the procedure for the login, password change, and password reset at a service. A routine is a set of instructions telling applications what actions need to be performed and how the execution of these actions can be verified. For example, a login routine describes how to log in to a user account and how to verify that the login was successful. We provide more technical information about routines in the following.

Listing 1: Exemplary PPD. It provides information about the password requirements as well as routines to perform an automatic login and password change at the service.

```
<ppd url="https://www.example.com" version="1.0">
  <characterSets>
    <characterSet name="Letters">
      <characters>abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ</characters>
    </characterSet>
    <characterSet name="Numbers">
      <characters>0123456789</characters>
    </characterSet>
  </characterSets>
  <properties>
    <characterSettings>
      <characterSet name="Numbers">
        <minOccurs>1</minOccurs>
      </characterSet>
      <positionRestriction characterSet="Letters">
        <positions>1</positions>
      </positionRestriction>
    </characterSettings>
    <minLength>10</minLength>
    <maxLength>20</maxLength>
  </properties>
  <service>
    <register>
      <url>https://www.example.com/register</url>
    </register>
    <login>
      <url>http://www.example.com/login</url>
      <maxTries>3</maxTries>
      <routines>
        <htmlLoginRoutine>
          <get>
            <url>https://www.example.com/login/</url>
            <assert>
              <url><prefix>https://www.example.com/login/</prefix></url>
            </assert>
          </get>
          <form>
            <selector>#main form[action^="https://www.example.com/login/"]</selector>
            <element>
              <selector>#username</selector><value>{{username}}</value>
            </element>
            <element>
              <selector>#password</selector><value>{{password}}</value>
            </element>
            <assert>
              <select><selector>#userMenu</selector></select>
            </assert>
          </form>
        </htmlLoginRoutine>
      </routines>
    </login>
    <passwordChange>
      <routines>
        <htmlPasswordChangeRoutine login="htmlLoginRoutine">
          <get>
            <url>https://www.example.com/account/</url>
            <assert>
              <url><prefix>https://www.example.com/account/</prefix></url>
            </assert>
          </get>
          <form>
            <selector>#main form[action^="https://www.example.com/account/"]</selector>
            <element>
              <selector>#password</selector><value>{{password}}</value>
            </element>
            <element>
              <selector>#newPassword</selector><value>{{newPassword}}</value>
            </element>
            <element>
              <selector>#confirmNewPassword</selector><value>{{newPassword}}</value>
            </element>
            <assert>
              <select><selector>div.success</selector></select>
            </assert>
          </form>
        </htmlPasswordChangeRoutine>
      </routines>
    </passwordChange>
    <passwordReset>
      <url>https://www.example.com/account/recovery</url>
    </passwordReset>
  </service>
</ppd>
```

Due to the different technologies that services use for their password interfaces, we defined four different types of routines: HTTP, HTML, JavaScript, and Extended JavaScript. They are represented by the elements <*LoginRoutine>, <*PasswordChangeRoutine>, and <*PasswordResetRoutine> element where '*' indicates the technology (http, html, js, or extentedJS. For instance, <httpLoginRoutine>).

The basis for all these routines are HTTP POST and GET commands which are used to interact with the service's password interfaces and to perform a login, password change, or password reset. A POST or GET command is defined by a <post> or <get> element, respectively. Both contain an element <url> which defines the target URL of the command. Furthermore, both include an <assert> element which is used to define a list of assertions that need to be verified in order to check whether the POST or GET command was performed successfully or not. We provide more information about the assertions later. The <post> element additionally contains a list of <data> elements representing the data which is sent to the service within the POST command (e.g. the username and password within a login routine). In the following, we describe the routines in detail:

- *HTTP*. A HTTP-based routine contains a list of <post> and <get> elements. Such a routine is used in case that the password interfaces of a service can be accessed by using plain HTTP GET or POST commands.
- *HTML*. A HTML-based routine extends the HTTP routine by a <form> element. The element consists of a list of <selector> elements which define identifiers to select HTML input fields and enter values in a HTML form (cf. Listing 1). HTML-based routine must be in case that HTML forms contains hidden input field with random values such as a session identifier.
- *JavaScript*. A JavaScript-based routine provides the same functionality like HTML routines. The elements of the type js only indicate that applications need to support JavaScript in order to access the password interfaces of the service. This is necessary when services use JavaScript to manipulate the input of a HTML form before submitting it. For instance, services hash the password by JavaScript on the client-side before sending it to their servers.
- *Extended JavaScript*. This type of routine extends the HTTP routine and additionally defines a <javascript> element which contains plain JavaScript. In case that the aforementioned technologies are insufficient, developers can use this type to implement the procedure using the complete functionality of JavaScript. Applications need to browse the password interface and execute the JavaScript in order to perform the routine.

Listing 1 shows an example for a HTML-based login and password change routine. Please note that the <htmlPasswordChangeRoutine> has an attribute login which refers to the login routine. This allows to reuse existing routines and makes the creation of PPDs easier and more efficient.

It is essential that applications can verify the correct execution of a routine, e.g., verify that a login was successful. To this end, a routine contains one or more assertions that applications need to verify. PPML defines the following types of assertions:

- *Existence of a Cookie*. A certain cookie is set after performing a routine.
- *Non-existence of a Cookie*. A certain cookie is not set after performing a routine.
- *Content*. The content of the response from the service contains a particular content. For instance, the text ”the password is wrong”.
- *Location*. The response redirects to a certain URL.

The routines of PPML are a powerful tool to describe password interfaces and procedures. They are highly flexible and support the wide range of different technologies and methods of password implementations used by services. In the next Section we provide more information of putting PPDs into practice.

## 5 Prototype

In the previous section we explained the syntax of PPDs in detail. We now show how to employ PPDs in practice and how to enable their broad application. We developed the PPD Distribution Service (PPDDS) which makes PPDs available to applications. Applications can query the PPDDS using the service's URL to search for a corresponding PPD. Furthermore, we developed a password manager and created PPDs for 5 services to demonstrate the feasibility of our solution. The key component of our Java-based password manager is the *Routine Engine*. It is capable of executing routines defined by a PPD and performs logins, password changes, and password resets. We use the Web Engine of JavaFX to interact with the services' password interface, which allows us to support all types of technologies of routines.

## 6 Conclusion and Future Work

In this paper, we presented the Password Policy Markup Language which closes the gap between the different implementations of password-based authentication on the service-side and applications managing passwords on the user-side. PPML facilitates a fully-fledged assistance along the complete password life cycle. Furthermore, it eliminates the need for error-prone heuristics to find password forms. Our password manager shows the practicality of our solution and provides an open and easily extendable basis for further research and development. To support further services users just need to create the corresponding PPDs for the services. The development of tools that assist in a fast and convenient creation of PPDs is part of future work.

## References


[Al12] AlFayyadh, Bander; Thorsheim, Per; Jsang, Audun; Klevjer, Henning: Improving Usability of Password Management with Standardized Password Policies. In: The Seventh Conference on Network and Information Systems Security. pp. 38–45, May 2012.

[Al15] Alexa Internet: , The top 500 sites on the web, 2015. `http://www.alexa.com/topsites`.

[ASL97] Adams, Anne; Sasse, Martina Angela; Lunt, Peter: Making Passwords Secure and Usable. In (Thimbleby, Harold W.; O'Conaill, Brid; Thomas, Peter, eds): People and Computers XII, Proceedings of HCI '97. Springer, pp. 1–19, 1997.

[BK95] Bishop, Matt; Klein, Daniel V.: Improving system security via proactive password checking. Computers & Security, 14(3):233–249, 1995.

[Bo12a] Bonneau, Joseph: The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords. In: IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA. IEEE Computer Society, pp. 538–552, 2012.

[Bo12b] Bonneau, Joseph; Herley, Cormac; van Oorschot, Paul C.; Stajano, Frank: The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In: IEEE Symposium on Security and Privacy, SP 2012, San Francisco, California, USA. IEEE Computer Society, pp. 553–567, 2012.

[Ca13] Castelluccia, Claude; Abdelberi, Chaabane; Dürmuth, Markus; Perito, Daniele: When Privacy meets Security: Leveraging personal information for password cracking. CoRR, abs/1304.6584, 2013.

[Ch14] Choong, Yee-Yin: A Cognitive-Behavioral Framework of User Password Management Lifecycle. In (Tryfonas, Theo; Askoxylakis, Ioannis G., eds): Human Aspects of Information Security, Privacy, and Trust - Second International Conference, HAS 2014, Held as Part of HCI International 2014, Heraklion, Crete, Greece, June 22-27, 2014. Proceedings. volume 8533 of Lecture Notes in Computer Science. Springer, pp. 127–137, 2014.

[Co14] Codenomicon: , The Heartbleed Bug, 2014. `http://heartbleed.com`.

[Da14] Daniel Humphries: , 67 Percent of Internet Users Haven't Changed Passwords After Heartbleed, 2014. `http://intelligent-defense.softwareadvice.com/67-percent-havent-changed-passwords-after-heartbleed-0414/`.

[Da16] Dashlane, Inc: Best Password Manager, Free Form Filler, Secure Digital Wallet. 2016. `https://www.dashlane.com`.

[dCdCM14] de Carné de Carnavalet, Xavier; Mannan, Mohammad: From Very Weak to Very Strong: Analyzing Password-Strength Meters. In: 21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014. The Internet Society, 2014.

[DMR10] Dell'Amico, Matteo; Michiardi, Pietro; Roudier, Yves: Password Strength: An Empirical Analysis. In: INFOCOM 2010. 29th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 15-19 March 2010, San Diego, CA, USA. IEEE, pp. 983–991, 2010.

[Eg13] Egelman, Serge; Sotirakopoulos, Andreas; Muslukhov, Ildar; Beznosov, Konstantin; Herley, Cormac: Does my password go up to eleven?: the impact of password meters on password selection. In (Mackay, Wendy E.; Brewster, Stephen A.; Bødker, Susanne, eds): 2013 ACM SIGCHI Conference on Human Factors in Computing Systems, CHI '13, Paris, France, April 27 - May 2, 2013. ACM, pp. 2379–2388, 2013.

[FH07] Florêncio, Dinei A. F.; Herley, Cormac: A large-scale study of web password habits. In (Williamson, Carey L.; Zurko, Mary Ellen; Patel-Schneider, Peter F.; Shenoy, Prashant J., eds): Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007. ACM, pp. 657–666, 2007.

[FH10] Florêncio, Dinei A. F.; Herley, Cormac: Where do security policies come from? In (Cranor, Lorrie Faith, ed.): Proceedings of the Sixth Symposium on Usable Privacy and Security, SOUPS 2010, Redmond, Washington, USA, July 14-16, 2010. volume 485 of ACM International Conference Proceeding Series. ACM, 2010.

[Fu07] Furnell, Steven: An assessment of website password practices. Computers and Security, 26(7-8):445–451, 2007.

[Fu11] Furnell, Steven: Assessing password guidance and enforcement on leading websites. Computer Fraud & Security, 2011(12):10–18, 2011.

[Ha15] Harris Interactive; Various sources (Dashlane): , Which of the following online security precautions did you take within the past 30 days?, March 2015. `http://www.statista.com/statistics/418676/us-online-security-precuations/`.

[Ho16] Horsch, Moritz; Schlipf, Mario; Braun, Johannes; Buchmann, Johannes A.: Password Requirements Markup Language. In (Liu, Joseph K.; Steinfeld, Ron, eds): Information Security and Privacy - 21st Australasian Conference, ACISP 2016, Melbourne, VIC, Australia, July 4-6, 2016, Proceedings, Part I. volume 9722 of Lecture Notes in Computer Science. Springer, pp. 426–439, 2016.

[Ke12] Kelley, Patrick Gage; Komanduri, Saranga; Mazurek, Michelle L.; Shay, Richard; Vidas, Timothy; Bauer, Lujo; Christin, Nicolas; Cranor, Lorrie Faith; Lopez, Julio: Guess Again (and Again and Again): Measuring Password Strength by Simulating Password-Cracking Algorithms. In: IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA. IEEE Computer Society, pp. 523–537, 2012.

[La16] LastPass Corporate: LastPass - The Last Password You Have to Remember. 2016. `https://lastpass.com`.

[Pe14] Pew Research Center: , Heartbleed's Impact, 2014. `http://www.pewinternet.org/2014/04/30/heartbleeds-impact/`.

[SB09] Shay, Richard; Bertino, Elisa: A comprehensive simulation tool for the analysis of password policies. Int. J. Inf. Sec., 8(4):275–289, 2009.

[SB14] Stobert, Elizabeth; Biddle, Robert: The Password Life Cycle: User Behaviour in Managing Passwords. In (Cranor, Lorrie Faith; Bauer, Lujo; Biddle, Robert, eds): Tenth Symposium on Usable Privacy and Security, SOUPS 2014, Menlo Park, CA, USA, July 9-11, 2014. USENIX Association, pp. 243–255, 2014.

[SBB07] Shay, Richard; Bhargav-Spantzel, Abhilasha; Bertino, Elisa: Password policy simulation and analysis. In (Goto, Atsuhiro, ed.): Proceedings of the 2007 Workshop on Digital Identity Management, Fairfax, VA, USA, November 2, 2007. ACM, pp. 1–10, 2007.

[Si14] Silver, David; Jana, Suman; Boneh, Dan; Chen, Eric Yawei; Jackson, Collin: Password Managers: Attacks and Defenses. In (Fu, Kevin; Jung, Jaeyeon, eds): Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014. USENIX Association, pp. 449–464, 2014.

[Sq07] Squicciarini, Anna Cinzia; Bhargav-Spantzel, Abhilasha; Bertino, Elisa; Czeksis, Alexei B.: Auth-SL - A System for the Specification and Enforcement of Quality-Based Authentication Policies. In: ICICS. volume 4861 of Lecture Notes in Computer Science. Springer, pp. 386–397, 2007.

[SS16] Sparell, Peder; Simovits, Mikael: Linguistic Cracking of Passphrases Using Markov Chains. IACR Cryptology ePrint Archive, 2016:246, 2016.

[St14] Stajano, Frank; Spencer, Max; Jenkinson, Graeme; Stafford-Fraser, Quentin: Password-Manager Friendly (PMF): Semantic Annotations to Improve the Effectiveness of Password Managers. In: PASSWORDS. volume 9393 of Lecture Notes in Computer Science. Springer, pp. 61–73, 2014.

[THB15] Taneski, Viktor; Hericko, Marjan; Brumen, Bostjan: Impact of security education on password change. In: MIPRO. IEEE, pp. 1350–1355, 2015.

[Ur12] Ur, Blase; Kelley, Patrick Gage; Komanduri, Saranga; Lee, Joel; Maass, Michael; Mazurek, Michelle L.; Passaro, Timothy; Shay, Richard; Vidas, Timothy; Bauer, Lujo; Christin, Nicolas; Cranor, Lorrie Faith: How Does Your Password Measure Up? The Effect of Strength Meters on Password Creation. In: Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA. USENIX Association, pp. 65–80, 2012.

[We09] Weir, Matt; Aggarwal, Sudhir; de Medeiros, Breno; Glodek, Bill: Password Cracking Using Probabilistic Context-Free Grammars. In: IEEE Symposium on Security and Privacy. IEEE Computer Society, pp. 391–405, 2009.

[We10] Weir, Matt; Aggarwal, Sudhir; Collins, Michael P.; Stern, Henry: Testing metrics for password creation policies by attacking large sets of revealed passwords. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA. ACM, pp. 162–175, 2010.

[WW15] Wang, Ding; Wang, Ping: The Emperor's New Password Creation Policies: An Evaluation of Leading Web Services and the Effect of Role in Resisting Against Online Guessing. In: Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria. volume 9327 of Lecture Notes in Computer Science. Springer, pp. 456–477, 2015.

[Ya04] Yan, Jeff Jianxin; Blackwell, Alan F.; Anderson, Ross J.; Grant, Alasdair: Password Memorability and Security: Empirical Results. IEEE Security and Privacy, 2(5):25–31, 2004.

[ZH99] Zviran, Moshe; Haga, William J.: Password Security: An Empirical Study. J. of Management Information Systems, 15(4):161–186, 1999.