Password Generators: Old Ideas and New

Fatma Al Maqbali, Chris J. Mitchell

Password Generators: Old Ideas and New

Fatma AL Maqbali and Chris J Mitchell

Information Security Group, Royal Holloway, Univ. of London
fatmaa.soh@cas.edu.om; me@chrismitchell.net

Abstract. *Password generators* that generate site-specific passwords on demand are an alternative to password managers. Over the last 15 years a range of such systems have been described. We propose the first general model for such systems, and critically examine options for instantiating it. The model enables an objective assessment of the design of such systems; it has also been used to sketch a possible new scheme, AutoPass, intended to incorporate the best features of the prior art while addressing many of the shortcomings of existing systems.

1 Introduction

Passwords remain a very widely used method for user authentication, despite widely shared concerns about the level of security they provide. There are many potential replacement technologies, including combinations of biometrics and trusted personal devices (e.g. as supported by protocols such as FIDO UAF [3]), but it seems likely that it will be some time before passwords are relegated to history. Given their current and likely future wide use, finding ways of improving the use and management of passwords remains a vitally important issue. We focus here on an important practical matter, namely how to make password-based user authentication to a website both more secure and more convenient.

An important class of schemes designed to ease password use are *password managers* (what McCarney [12] calls *retrieval password managers*). A password manager stores user passwords and produces them when required (e.g. by auto-filling-in login pages). Passwords can be stored either locally or on a trusted server; most browsers provide a local-storage password manager. However, the shortcomings of password managers have also been widely documented (see, e.g., McCarney [12]). Passwords stored on a user platform restrict user mobility, since they are not available when a user switches platform, e.g. from a laptop to a tablet or phone. However, if passwords are stored 'in the cloud', then there is a danger of compromise through poorly configured and managed servers, [2,8,13].

An alternative approach, which we consider here, involves generating site-specific passwords on demand from a combination of inputs, including those supplied by the user and those based on the site itself. A number of schemes have been proposed but, apart from a brief summary by McCarney [12], they have not been studied in a more general setting. The main purposes of this paper are to (a) provide a general model for password generation schemes, and (b) use the model to propose a new system combining the best features of existing schemes. This is the first time these schemes have been considered in a unified way.

2 Password Generators — A General Model

Password generators simplify user password management by generating site-specific passwords on demand from a small set of inputs. The term has also been used to describe the generation random or pseudorandom passwords which a user must remember; however, we use the term for schemes that generate a password in a repeatable way. We focus on the general properties of such schemes and options for operation. The schemes have been briefly considered previously by McCarney [12] under the name *generative password managers*.

2.1 A Model

A password generator is functionality on an end-user platform to support password-based user authentication to a remote server (assumed to be a web site). It generates, on demand, a site-unique password for use in authentication. Clearly this password also needs to be available to the web site authenticating the user; the *registration* step, in which the password is set up, is discussed further in section 2.3 below. A password generator has the following components.

- A set of *input values* is used to determine the password for a site; some must be site-specific so the generated password is site-specific. The values could be: stored (locally or online), based on characteristics of the authenticating site, user-entered, or some combination of types.
- A *password generation function* combines the input values to generate an appropriate password. This function must meet the requirements of the authenticating web site; e.g. one web site might forbid non-alphanumeric characters in a password, whereas another might insist that a password contains at least one such character. A password generation function must therefore be customisable.
- A *password output method* transfers the password to the authenticating site, e.g. by displaying the generated password to the user.

All this functionality needs to be implemented on the user platform. There are various possibilities for implementation, e.g. as a stand-alone application or a browser plug-in; these issues are discussed further in section 3 below.

2.2 Examples

We next briefly outline some existing proposals for password generation schemes, presented in chronological order of publication. The functional components of the various examples are considered in greater detail in section 3 below.

The *Site-Specific Passwords (SSP)* scheme proposed by Karp [7] in 2002/03 is one of the earliest proposed examples. It generates a site-specific password by combining a long-term (global) user master password and an easy-to-remember name for the web site, as chosen by the user. *PwdHash*, due to Ross et al. [14], generates a site-specific password by combining a long-term master password,

data associated with the web site, and (optionally) a second global password stored on the platform. The 2005 *Password Multiplier* scheme of Halderman, Waters and Felten, [4], computes a site-specific password as a function of a long-term master password, the web site name, and the user name for the user at the web site concerned. Wolf and Schneider's 2006 *PasswordSitter* [15] scheme generates a site-specific password as a function of a long-term master password, the user identity, the application/service name, and some configurable parameters. *Passpet*, due to Yee and Sitaker [16] and also published in 2006, takes a very similar approach to SSP, i.e. the site-specific password is a function of a long-term master password and a user-chosen name for the web site known as a *petname*. Each petname has an associated icon, which is automatically displayed to the user and is intended to reduce the risk of phishing attacks. *ObPwd*, due to Mannan et al. [1,9,10,11], first surfaced in 2008. It takes a somewhat different approach by generating a site-specific password as a function of a user-selected (site-specific) object (e.g. a file), together with a number of optional parameters, including a long-term master password (referred to as a *salt*), and the web site URL. Finally, *PALPAS* was published in 2015 [5]. PALPAS generates passwords complying with site-specific requirements using server-provided password policy data, a stored secret value (the *seed*), and a site- and user-specific secret value (the *salt*) that is synchronised across all the user devices using the server.

2.3 Registration and Configuration

We only consider schemes whose operation is completely transparent to the authenticating website. As a result, the 'normal' website registration procedure, where the user selects a password and sends it to the site, is assumed to be used. This means that the password generation process needs to be in place *before* the registration procedure, or the introduction of a password generator will require the user to modify their password. A possible way of avoiding the need to change passwords is examined in section 4 below.

A password generator may need to store configuration data. Such data can be divided into: *global configuration data*, i.e. values unique to the user, used to help generate all passwords for that user, and *site-specific configuration data*, i.e. values used to help generate a password for a specific site. Not all schemes use configuration data, although building a workable system without at least some global configuration data seems challenging. However, the use of configuration data is clearly a major barrier to portability. That is, for a user with multiple platforms, the configuration data must be kept synchronised across all these platforms, a non-trivial task — exactly the issue addressed in a recent paper by Horsch, Hülsing and Buchmann [5].

3 Components of the Model

Inputs to Password Generation: The following data input types have been employed in existing schemes. A **master password** is a user-specific long-term

secret value; this could either be a **user-entered password**, i.e. entered by the user whenever a password is to be generated, or a **stored password**, i.e. a user-specific secret value stored as global configuration data. A **site name** is a name for the authenticating site; this could take a variety of forms, including a **user site name**, i.e., a name for a site chosen by a user, all or part of the site's **URL**, or a **site-specific secret**, e.g. a random value associated with the site URL. A **digital object** is anything available on the user platform which could be used as input to the password generation process, e.g. a file or a selected block of text on the target web site. A **password policy** is information governing the nature of the password generated, e.g. the set of acceptable symbols.

Generating the Password: Combining inputs to generate a password can be done variously. All approaches involve a 2-stage process, i.e. first combining inputs to generate a bit-string, then formatting the bit-string to obtain a password in the desired format. Horsch et al. [6] propose an XML syntax, the *Password Requirements Markup Language (PRML)*, designed specifically to enable requirements on passwords, as needed in the second stage, to be specified.

Password Output and Use: There are many ways in which a generated password could be transferred to the password field of a login page. Simplest is **manual copy and paste**, as used by SSP [7], where the password generator displays the password, and the user copies it to the login page. A slightly more automated approach is **copy to clipboard** in which the generated password is copied to the clipboard; for security reasons the password can be made to only reside in the clipboard for a limited period, e.g. in PasswordSitter the generated password is saved to the clipboard for 60 seconds before being deleted [15]. The simplest approach for the user is probably **automatic copying to the target password field**; this can either be done automatically, as is the case for PwdHash in the web page implementation [14] and the ObPwd Firefox browser extension [9]. Alternatively it can require the user to perform an action, e.g. clicking a specific key combination, before copying; PassPet requires the user to click on a screen button, [16], and Password Multiplier, [4], requires the user to double click the password field or press *ctrl+P* to trigger password copying.

Approaches to Implementation: A password generator can be implemented as a **browser add-on**, e.g. as a **browser plug-in**, **browser extension** or **signed browser applet**. Many existing password generator schemes adopt this approach, at least as one option, including [4,14,15,16]. An alternative is to implement the scheme as a **stand-alone application**, e.g. to run on a phone, tablet or desktop. Such an approach is adopted by SSP; ObPwd, [9], is also available as both a browser extension and a mobile app. A somewhat different approach is to use a **web-based application**, either running on a remote server or executing on the user platform as a dynamically downloaded JavaScript.

4 Improving System Operation

We next consider ways to improve password generators. In section 5 we consider how these might be integrated into a novel system.

We have mentioned certain types of configuration data, including global data, such as master secrets, and site-specific data, e.g. password policy values, possibly specified in PRML [6]. We now introduce two new configuration data types.

- A *password offset* (site-specific configuration data) allows users to continue using existing passwords after introducing use of a password generator. It also help when specific password values are imposed on users, or when users need to change their passwords. Addressing such requirements previously has been problematic.
  A password offset is as an input to the second stage of password generation. A password offset induces this stage to generate a specific password value. E.g., suppose a password policy dictates that a password must be a string of letters and numerals, where each such character is internally represented as a numeric value in the range 0–61. After converting the bit-string to a string of alphanumeric characters of the desired length, and given a 'desired password' consisting of an alphanumeric string of the same length, the password offset could simply be the character-wise modulo 62 difference between the two strings[1]. Changing a password can be easily implemented by changing the offset, either to a random value (randomising the password choice), or to a chosen value if the new password value is chosen by the user.
  If implemented appropriately, this offset is not hugely confidential, since it need not reveal anything about the actual password value. Of course, if an 'old' password is compromised, and the old and new offsets are also revealed, then this could compromise the new password value.
- A password generator might generate a password for one site using a different set of input types to those used to generate a password for another site. E.g., a password for a mission-critical site (e.g. a bank account) might be generated using a large set of input values, e.g. including a digital object, whereas a password for a less sensitive site could be generated using a master secret and site name only. Such a possibility could be captured using site-specific configuration data, referred to here as *password input parameters*.
- A system might also store *password reminders* as site-specific configuration data. E.g., when using a digital object as input, a user could specify a phrase as a reminder of the chosen value (without specifying it precisely). This could be revealed on demand via the password generator user interface.

Storing configuration data on a user platform creates a major barrier to portability; it also poses a security risk through possible platform compromise, although much of the configuration data we have discussed is not necessarily confidential. The 'obvious' solution is to use a server to store configuration data, or at least the less sensitive such data, much as many password managers keep passwords in the cloud. While it would seem prudent to keep a master secret on the user platform, all site-specific configuration data could be held in the cloud. This type of solution is advocated by Horsch et al. [5,6].

[1] Such an idea is widely implemented to enable credit/debit card holders to select their own PIN value.

If the site-specific configuration data is all non-confidential, then there is no need for a highly trusted server, a great advantage by comparison with some server-based password managers. Server use need not impact password availability, since a password generator could cache a copy of the configuration data downloaded from the server, addressing short-term loss of server availability.

5 AutoPass: A New Proposal

We now outline AutoPass (from 'automatic password generator'), a novel password generation scheme combining the best features of the prior art together with the novel ideas introduced in this paper, particularly those devised to address some of the shortcomings of previously proposed schemes. AutoPass uses all the types of input given in section 3 to generate a password, since they all contribute to security in different ways. Following the approach of PALPAS, [5], we also make use of a server to store non-sensitive configuration data, such as website password policies.

5.1 Operation

Following section 2, to describe AutoPass we must define: (a) the input types, (b) how the password is generated, and (c) how the password is output, together with the implementation strategy. We cover these points in turn. Since we also propose the use of a cloud service to support AutoPass operation, we also briefly sketch its operation.

- For **inputs**, we propose the use of a **master password**, stored by the system (as global configuration data), and a password (or PIN) to be entered by the user. We also propose use of the first part of the **URL** of the site, where, depending on the implementation, this should also be stored as part of the site-specific configuration and used to retrieve the other site-specific data. The master password can be held encrypted by a key derived from the user password. We also propose the optional use of a **digital object**, where use of this option is indicated in the site-specific configuration data.
- The first stage of **password generation** adopts a two-level hash approach, giving some protection against brute force attacks. The second stage, i.e. **encoding**, uses the AutoPass cloud service to retrieve the password policy for the web site being visited (cf. PALPAS [5]); this policy could be encoded using PRML, [6]. It also uses other cloud-stored configuration data, notably the password offset, password input parameters, and password reminders introduced in section 2.3.
- The precise option for **password output and use** depends on the implementation. Where possible, auto-filling the password is desirable; where this is impossible, the copy to clipboard/paste buffer approach is advocated.
- **Implementation** as a browser add-on is probably the best option, not least in giving simple access to the web page of the target site, although a range of options may need to be pursued depending on the platform type.

We next consider the AutoPass Cloud Service, whch will be required to store two main types of data. *User-independent data* will be accessed by AutoPass users, and will include non-sensitive site-specific data, e.g. password policies. Even if corrupted by a malicious party, it would at worst cause a denial of service. *User-specific data* will only be accessed by a single user, and includes a range of password configuration data. Although this data is not highly confidential, access to it will need to be restricted to the user to whom it belongs, e.g. via a one-off login process in the local AutoPass application (with access permissions encoded in a cookie stored in the user platform).

Any cloud service has associated risks arising from non-availability; however, this can be addressed through caching. The local AutoPass app should maintain a copy of the data downloaded from the cloud service; since this data is not likely to change very quickly, the cached data should normally be sufficient. To avoid risks arising from fake AutoPass services, e.g. using DNS spoofing, the cloud service could sign all provided data, and the AutoPass app could verify signatures using a built-in copy of the cloud public key.

5.2 Assessment

AutoPass incorporates both the best features of the existing password generation schemes and certain novel features, notably the use of password configuration data (see section 4). A full assessment of AutoPass will require prototyping and user testing. Nonetheless, we can at least consider the known issues in existing systems and see whether AutoPass addresses these concerns.

By using a combination of stored secret and memorised password/PIN as inputs to the generation process, we enable strong passwords to be generated while protecting against compromise through platform loss. Use of the URL enables automatic generation of site-specific passwords, and optional use of digital objects enables passwords of varying strength to be generated without imposing an unnecessarily onerous load on the user for 'everyday' use. URL use has residual problems, notably if a site URL changes, but user site names also have problems. In this connection, an advantage of AutoPass is that password offsets enable a generated password to remain constant even if a URL changes.

Use of cloud-served password policies solves problems relating to site-specific password requirements. Continued use of existing passwords and the need to change passwords are addressed by use of cloud-stored password configuration data. Password generation/synchronisation issues arising from use of multiple platforms can also be addressed through use of a cloud service. Of course, use of a cloud service brings with it certain availability and security risks; however, by only storing non-sensitive data in the cloud and using caching, these problems are largely addressed.

6 Concluding Remarks

We introduced a general model for password generation, and considered existing proposals in the context of this model. The model enables us to propose

certain new options to enhance such schemes. The operation of a novel scheme, AutoPass, has been sketched, but has not yet been tested in practice. The next step is to prototype AutoPass, both to verify that the underlying idea works and also as a basis for practical user trials.

References

1. Biddle, R., Mannan, M., van Oorschot, P.C., Whalen, T.: User study, analysis, and usable security of passwords based on digital objects. IEEE Transactions on Information Forensics and Security 6(3-2), 970–979 (2011)
2. Cluley, G.: Lastpass vulnerability potentially exposed passwords for internet explorer users (August 2013), https://www.grahamcluley.com/2013/08/lastpass-vulnerability/
3. FIDO Alliance: FIDO UAF Protocol Specification v1.0: FIDO Alliance Proposed Standard 08 (December 2014)
4. Halderman, J.A., Waters, B., Felten, E.W.: A convenient method for securely managing passwords. In: Ellis, A., Hagino, T. (eds.) Proc. WWW 2005, May 2005. pp. 471–479. ACM (2005)
5. Horsch, M., Hülsing, A., Buchmann, J.A.: PALPAS - passwordless password synchronization (2015), arXiv:1506.04549v1 [cs.CR], http://arxiv.org/abs/1506.04549
6. Horsch, M., Schlipf, M., Braun, J., Buchmann, J.A.: Password requirements markup language. In: Proc. ACISP 2016, July 2016. pp. 426–439. Lecture Notes in Computer Science, to appear, Springer-Verlag (2016)
7. Karp, A.H.: Site-specific passwords. Tech. Rep. HPL-2002-39 (R.1), HP Laboratories, Palo Alto (May 2003)
8. Kelly, S.M.: Lastpass passwords exposed for some internet explorer users (August 2013), mashableUK, http://mashable.com/2013/08/19/lastpass-password-bug/
9. Mannan, M., Van Oorschot, P.C.: Passwords for both mobile and desktop computers: ObPwd for Firefox and Android. USENIX ;login 37(4), 28–37 (August 2012)
10. Mannan, M., van Oorschot, P.C.: Digital objects as passwords. In: Provos, N. (ed.) Proc. HotSec'08, July 2008. USENIX Association (2008)
11. Mannan, M., Whalen, T., Biddle, R., van Oorschot, P.C.: The usable security of passwords based on digital objects: From design and analysis to user study. Tech. Rep. TR-10-02, School of Computer Sciemce, Carleton University (February 2010), https://www.scs.carleton.ca/sites/default/files/tr/TR-10-02.pdf
12. McCarney, D.: Password managers: Comparative evaluation, design, implementation and empirical analysis. Master's thesis, Carleton University (August 2013)
13. Pauli, D.: KeePass looter: Password plunderer rinses pwned sysadmins. The Register (November 2015), http://www.theregister.co.uk/2015/11/03/keepass_looter_the_password_plunderer_to_hose_pwned_sys
14. Ross, B., Jackson, C., Miyake, N., Boneh, D., Mitchell, J.C.: Stronger password authentication using browser extensions. In: McDaniel, P. (ed.) Proc. 14th USENIX Security Symposium, July/August 2005. USENIX Association (2005)
15. Wolf, R., Schneider, M.: The passwordsitter. Tech. rep., Fraunhofer Institute for Secure Information Technology (SIT) (May 2006)
16. Yee, K.P., Sitaker, K.: Passpet: Convenient password management and phishing protection. In: Cranor, L.F. (ed.) Proc. SOUPS 2006, July 2006. ACM International Conference Proceeding Series, vol. 149, pp. 32–43. ACM (2006)