SAuth: Protecting User Accounts from Password Database Leaks

Georgios Kontaxis
Columbia University
kontaxis@cs.columbia.edu

Elias Athanasopoulos
Columbia University
elathan@cs.columbia.edu

Georgios Portokalidis
Stevens Inst. of Technology
gportoka@stevens.edu

Angelos D. Keromytis
Columbia University
angelos@cs.columbia.edu

ABSTRACT

Password-based authentication is the dominant form of access control in web services. Unfortunately, it proves to be more and more inadequate every year. Even if users choose long and complex passwords, vulnerabilities in the way they are managed by a service may leak them to an attacker. Recent incidents in popular services such as LinkedIn and Twitter demonstrate the impact that such an event could have. The use of one-way hash functions to mitigate the problem is countered by the evolution of hardware which enables powerful password-cracking platforms.

In this paper we propose SAuth, a protocol which employs authentication synergy among different services. Users wishing to access their account on service S will also have to authenticate for their account on service V, which acts as a vouching party. Both services S and V are regular sites visited by the user everyday (e.g., Twitter, Facebook, Gmail). Should an attacker acquire the password for service S he will be unable to log in unless he also compromises the password for service V and possibly more vouching services. SAuth is an extension and not a replacement of existing authentication methods. It operates one layer above without ties to a specific method, thus enabling different services to employ heterogeneous systems. Finally we employ password decoys to protect users that share a password across services.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Authentication*

General Terms

Security

Keywords

Authentication; Synergy; Password Leak; Decoys

CCS'13, November 04–08, 2013, Berlin, Germany.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2477-9/13/11 ...$15.00.
http://dx.doi.org/10.1145/2508859.2516746.

1. INTRODUCTION

Password theft can cause annoyance, financial damages, data loss, and loss of privacy [5, 9]. Services employing passwords urge their users to choose complex combinations, never write them down, change them frequently, select a different password on each service they use and verify the authenticity of the site before logging in. However, even users that manage to follow all these rules risk having their passwords stolen. Security pitfalls in a series of popular services have resulted in frequent and massive password leaks [4, 8, 16, 20]. Even though services usually store a digest or hash of the password (excluding rare incidents [12]), the emergence of powerful password-cracking platforms [14, 44] has enabled attackers to recover the original passwords in an efficient manner [13]. What is more, password-reuse practices by the users have enabled domino-like attacks [17, 34].

We propose Synergetic Authentication (SAuth), an authentication mechanism based on the synergy between different services that complements their individual procedures for verifying the identity of a given user. To successfully log into service S one is required to successfully authenticate both with S and a cooperating vouching service V. For example, a user logging into his Gmail account, after successfully submitting his password to the Gmail server, he will be required to also submit his Facebook password to the Facebook server. Once Gmail receives notice from Facebook that the same user has managed to authenticate successfully it will have an additional assurance that the user is the actual owner of those accounts. Our approach is founded upon the way most users access the web. In particular, users remain concurrently and constantly authenticated with many services such as e-mail and social networks [24, 42] unless they explicitly log out. A service the user is already logged on can transparently vouch for him, e.g., through client-side cookies. In the above example, had the user been already logged in to Facebook, he would just be required to enter his Gmail password to access his e-mails. Facebook would use his user agent cookies to transparently authenticate him and subsequently vouch for him to Gmail. At the same time, an attacker that has compromised the user's password for S is unable to access that account as he is lacking the password for vouching service V and thus cannot complete the authentication process. In other words, for an attacker to compromise one account, he must acquire multiple account passwords for that user located in different databases of distinct services.

Password-based authentication has received a lot of criticism lately with many large services like Google and PayPal looking for alternative means to authenticate users [7,15,35]. Some alternatives that have been proposed in the past include public-key mechanisms, such as TLS client certificates [19], graphical passwords [25,38], and many more. Unfortunately, none of the proposed alternatives has proven sufficiently enticing [29]. Passwords have been the de facto method for authenticating users for many decades, and have proven to be resilient to change.

Two-factor authentication [22] has probably been the most successful proposal to complement password-based systems by requiring that an additional password is provided, acquired through a second independent channel. Unfortunately the overhead both in cost and effort to deploy and maintain such system has led to adoption only by high-value services such as banking sites and e-mail providers. Moreover, it scales poorly when users are required to manage multiple secondary factors for distinct services. Finally, a study has shown that it can push users to weaker passwords [57].

Single-sign-on services like OpenID, as well as the OAuth-based interfaces of social networking services [6,46], offer the alternative of maintaining a single online identity, protected by one, hopefully strong, password. Users of such services, instead of creating separate, new accounts and passwords with third-party services, authenticate with a trusted identity provider (e.g., Facebook), that vouches for them. However, these identity providers present a single point of failure, may carry privacy-related risks, and can also suffer vulnerabilities themselves [55]. A recent study [54] attributes the limited adoption of such services to concerns regarding their availability and relinquishing control of the user base as part of outsourcing authentication.

SAuth does not suffer from any of the above problems, as it complements, rather than substitutes, existing authentication procedures at each site. Therefore, it does not degrade the security of a service, but allows heterogeneous authentication systems to operate at each site, while preserving each party's user accounting system. Finally, it encourages symmetric relations between services as it enables all participating sites to operate both as relying and vouching parties if they wish to.

To address the issue of password reuse [41,53], which can render vouching ineffective if a user recycles the same password across all vouching services, we employ decoy passwords. Each service automatically generates multiple decoy passwords that are similar to the one chosen by the user. Furthermore, for the decoys to blend in, they receive no special treatment by the service and are thus considered as legitimate credentials for user authentication. Note however that the user is never aware of them. Enabling decoy passwords requires no changes in the database schema, can be implemented with wrappers to existing password management functions and introduces multiple password entries per user instead of a single one. An attacker cracking the passwords will be unable to identify the actual one and resort to online guessing against the vouching services [1].

Overall, this work makes the following contributions:

[1]In this paper we model an on-line guessing attack according to the NIST specification [32] (see Section 6.1).

- We design, implement, and evaluate SAuth, a framework employing synergy between sites for stronger authentication.

- SAuth builds on widely used technologies and is orthogonal to the underlying authentication framework. Beyond providing stronger authentication, SAuth can be used as an alert system for password breaches.

- We leverage decoy passwords to tackle password reuse practices among the cooperating sites participating in an SAuth session.

The rest of the paper is organized as follows. We provide background information in §2. We present the design of SAuth in §3 and its implementation in §4. We discuss decoy passwords in §5 and evaluate their use in §6. Related work is presented in §7 and conclusions are in §8.

2. BACKGROUND

Password-based authentication has changed little in the many decades it is in use and today it is more popular than ever, with countless web applications using passwords to authenticate their users. In brief, when a user first registers with a service, he selects a username and password with which he will authenticate. The application stores the username in plain-text form, attaches a random prefix called salt to the password, gets the digest of this prefixed password using a cryptographic hash function such as MD5 or SHA1, stores the hash output along with the salt in the database and discards the plain-text password. Note that recently these general-purpose hash functions have received criticism [10] and alternatives such as bcrypt [48] have been proposed. The salt prefix ensures that even if a password is shared by multiple users, a different hash will be generated and stored in the database, and identical passwords cannot be inferred by their hash product. What is more, it defeats rainbow tables where the hashes for a large set of dictionary words and password-space permutations are precomputed and then compared against a password hash in real time. Upon login, users transmit their username and password to the service in plain text, hopefully through a secure communication protocol such as TLS. Subsequently the web service uses the stored salt and the provided password to compute their digest and compare it to the stored digest in the database. Note that unfortunately there are cases where passwords are simply stored verbatim in the database [3].

2.1 Password Leaks

In recent years we have witnessed an increasing amount of password leaks [8, 12, 16, 20] from major Internet sites. Between early 2012 and now these incidents have been occurring at an alarming rate of roughly one every couple of months. We use the term password leak to describe the exfiltration of user passwords stored by some service's accounting system. This can be due to malicious insiders or front-end bug exploitation, e.g., through SQL injection.

It is important to stress that password leaks are different in nature from phishing or social engineering attacks which may also be used to compromise user passwords. Leaks happen on the server side and usually lead to large-scale password exfiltration. As they don't involve the client side, in contrast to phishing attacks, even diligent users may have their password compromised.

2.2 Password Cracking

Following a successful database leak the attacker has in his possession all the elements necessary to attempt to crack the stolen passwords; he holds their digests, the corresponding salt values and has knowledge of the hashing function used.

Subsequently he can employ various cracking methodologies [44] ranging from brute-force attacks to dictionary attacks and a combination of the two. In the first case he can simply compute the salted hashes for all possible permutations in the given password space, e.g., passwords 1 to 8 characters in length involving upper and lower case English letters, numbers and symbols. Whenever a computed hash, for which the attacker knows the input, matches one of the hashes in the stolen set, that user's password is revealed. Clearly this approach requires abundant processing power and a lot of time. Alternatively, the attacker can compute the salted hashes for a pre-constructed dictionary of potential passwords. As users tend to choose passwords that are easy to remember [28], more often than not they select words such as "password" or phrases such as "letmein" or people's names. Therefore a dictionary-based attack can be very fast and efficient. Finally, a hybrid approach involving permutations of dictionary words can crack passwords that meet the bare minimum of otherwise complex password policies. "password1" or "letmein!" and "s3cur3" are such examples.

Even if an attacker designs his password cracking process in such an elaborate manner he still requires ample processing power to achieve high success rates in a reasonably short amount of time. The parallel architectures of modern GPUs can greatly shorten the password cracking process. Recently, GPU-based architectures were able to crack eight-character-long NTLM passwords for Microsoft Windows in just about five hours [14]. At the same time, cracking platforms utilizing the resources and scalability of the cloud have emerged [1]. Overall the sophistication of password cracking methodologies, along with the availability of increasingly more powerful hardware, significantly extends an attacker's capability to reveal complex passwords.

To mitigate the increase in the processing power of cracking platforms, alternatives to the standard cryptographic hash functions have been proposed. As general purpose functions like MD5 or SHA1 were designed to be fast and efficient, they serve in favor of the attacker who wants to compute a large number of digests in a short amount of time. On the other hand, adaptive hash functions, such as bcrypt [48] and scrypt [47], can be configured in terms of the work factor they introduce. In other words, as processing power increases over time these functions can be tuned to take constant amount of time. Therefore it is possible to slow down the cracking process significantly. Note however that these functions have a much lesser impact on dictionary attacks compared to brute-force attacks.

Our proposal does not make any assumptions about the way the password is stored in the database or the particular hash function employed. Nevertheless we assume that, as the way users select passwords favors password-cracking, leaked password hashes, or at least a significant portion of them, will eventually be cracked.

2.3 Out of Scope Threats

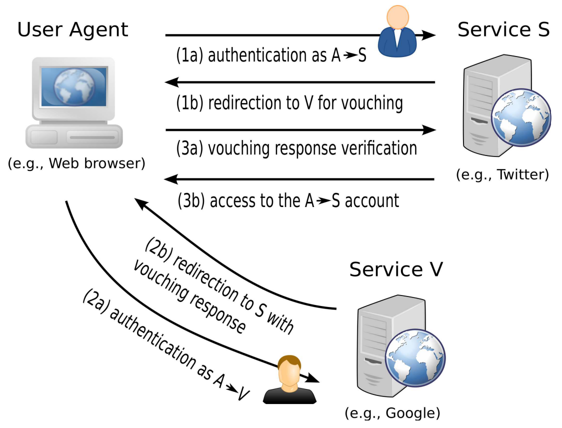
User Agent
Service S
(1a) authentication as A→S
(1b) redirection to V for vouching
(3a) vouching response verification
(3b) access to the A→S account
(e.g., Web browser)
(e.g., Twitter)
Service V
(2b) redirection to S with vouching response
(2a) authentication as A→V
(e.g., Google)

Figure 1: Overview of synergy-enhanced authentication. A user, with accounts in services S and V (e.g., Twitter and Google) tries to log in S. A standard authentication process takes place which involves the user's password for S (step 1a). Subsequently service S initiates our proposed protocol which involves service V vouching for the user (step 1b). The user is redirected to V with which he also engages in a standard authentication process (step 2a) at the end of which service V generates a vouching token for the user to return to S (step 2b). Finally, service S allows the user to access his account if and only if the vouching token from V is verified (step 3a) and optionally returns a persistent authentication token (e.g., cookie) that will bypass this authentication process for subsequent interactions (e.g., HTTP requests) with S.

We do not attempt to address password compromise due to social engineering, phishing, or man-in-the-middle attacks. Social engineering exploits the human psychology and we argue that technical means may not be able to offer adequate protection on their own. As far as phishing, malware, and man-in-the-middle attacks are concerned we stress that there are already sufficient protection mechanisms that can be employed independently to our proposal. For example, researchers have proposed PwdHash [50], BeamAuth [22] and Visual Skins [39] against phishing. They employ something unique to the original site, such as a URL or its appearance, to alert the user or safeguard against transmitting his password to the wrong site. Nevertheless, they do not have any effect on the server side nor do they present any benefit once the user's password has been leaked.

3. SAuth ARCHITECTURE

Alice has registered and maintains accounts with web sites S and V like she does today. We consider the case where Alice tries to authenticate to S using SAuth, while V is acting as a vouching party for Alice. We use the terms $A_{\rightarrow S}$ and $A_{\rightarrow V}$ for indicating Alice as announced to S and V, respectively. We use this notation for stressing that Alice has a different identity with each party. Below, we provide the steps executed for Alice to successfully authenticate with service S in the presence of service V, which is trusted by S and has agreed to vouch for its own users to S. The process is also depicted in Figure 1.

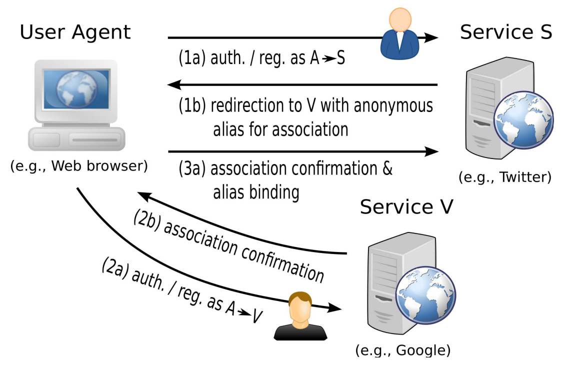
User Agent
(e.g., Web browser)
Service S
(e.g., Twitter)
Service V
(e.g., Google)
(1a) auth. / reg. as A→S
(1b) redirection to V with anonymous alias for association
(3a) association confirmation & alias binding
(2b) association confirmation
(2a) auth. / reg. as A→V

Figure 2: Overview of the association process to enable synergy-enhanced authentication. A user tries to register for a new account in service S (e.g., Twitter). Alternatively he logs in to an existing account with S for which he wants to enable SAuth (step 1a). S generates a unique anonymous alias for the user and redirects him to vouching service V (e.g., Google) (step 1b). The user is expected to authenticate to V or create a new account (step 2a). Note that the user may have already enabled SAuth in his V account. Upon authentication or registration, the anonymous alias from S is associated with the current account on V and user is redirected back to S with a confirmation message (step 2b). Finally, S binds that alias to the user's account and enables SAuth (step 3) so that subsequent authentication attempts can involve V as a vouching service and, if so, will require the receipt of that alias from V.

1. Alice visits service S and is challenged for her authentication credentials (e.g., an HTML form asking for her name and password or a cookie storing an authentication token). Note that these authentication credentials were established when Alice created her account, $A_{\rightarrow S}$, on service S. This is currently a standard method for authentication employed by web services. This step is depicted as (1a) in Figure 1. After providing the correct credentials for $A_{\rightarrow S}$, the synergy-based authentication protocol is initiated (step (1b) in Figure 1) and Alice is prompted to choose from a list of trusted vouching services, including service V, to complete the enhanced authentication process. Notice, that Alice has specified her vouching services in S when she registered or enabled SAuth. Alice chooses service V and a vouching request is issued towards V by S, while the user is being redirected to service V.

2. Service V receives the vouching request from S and challenges the current user for her authentication credentials (step (2a) in Figure 1). Note that these authentication credentials were established when Alice created her account, $A_{\rightarrow V}$, on service V. After successful authentication of user $A_{\rightarrow V}$, a verification response is returned and the user is redirected back to service S (step (2b) in Figure 1).

3. Service S receives a verification response from V signaling that current user, $A_{\rightarrow S}$, has managed to prove ownership of an account, $A_{\rightarrow V}$, on service V (step (3a) in Figure 1). As any user can prove ownership of two accounts on two distinct services, we require that accounts $A_{\rightarrow S}$ and $A_{\rightarrow V}$ have been associated in the past when user Alice opted in this enhanced authentication process. This association means that if some user manages to authenticate as $A_{\rightarrow S}$ but then authenticates as $B_{\rightarrow V}$, the enhanced authentication process will be terminated by service S as unsuccessful. The only way for the process to succeed is for the current user to authenticate as both $A_{\rightarrow S}$, on service S, and $A_{\rightarrow V}$, on service V. Once the enhanced authentication process concludes with a positive result on service S the current user is given an authentication token (e.g., HTTP cookie), which replaces the need for challenging her authentication credentials for every subsequent request, currently a common practice in the web and elsewhere. Note that at this point the current user has authenticated fully on S and V in case both services are vouching for each other. If a different service is selected to vouch for V then, as expected, the user has to follow a similar process for authenticating with V.

SAuth is proposed as an opt-in feature, designed to complement existing authentication methods. Sites providing SAuth can let users enable the mechanism at registration or at any later time. We now discuss the protocol details. We assume that SAuth operates above SSL.

3.1 Protocol Details

Security and Trust. When service S receives a vouching token from service V about some user, S trusts that V has indeed authenticated the user and its security practices do not allow some other user to generate the same vouching token while interacting with V. If V fails to meet those expectations, the security of the process followed by S is as if SAuth was not in place at all and S was operating alone. In other words, as S does not nullify its own authentication system, even if V implicitly or explicitly misbehaves, S will not be negatively impacted. As the first step in the SAuth protocol involves correctly authenticating with the target service, the security procedures of S take precedence over the ones by vouching parties. Note that including more than one vouching service reduces the dependence on a single external party and increases the redundancy in the face of vouching unavailability.

Activation. Registering for a new account in a synergy-supporting service or enabling synergy-enabled authentication for an existing account entails an association step between the target service S and potential vouching services V. When registering for a new account, the traditional registration process is carried out and then followed by the association step. Note that activation of SAuth can be deferred for a later time to limit the workload for new users as it is not strongly tied to the registration process. For existing accounts, only the association step needs to take place. The association step is necessary so that a link is established between the current account in the target service S and an account the user controls in a vouching service V. As mentioned earlier, a vouching party produces proof that the current user has managed to successfully authenticate. This alone is not useful to the target service as it is not necessary for the owner of the account in S to be the owner of the account in V. In other words, an attacker who manages to steal the password for an account in S could authenticate with his own account in V and have service V return that as proof of authentication. For that matter, upon registration of a new account or enabling of SAuth, service S generates an anonymous alias for the user. The user is then expected to provide that alias to a vouching service and associate it with an account there. Once this association is made, the vouching service will return this alias as part of the authentication proof to service S. Service S will check the returned alias against the one bound to the account of the current user and enable him to access the account if and only if the two are a match (Fig. 2).

Authenticity. As the user agent, i.e., web browser, is tasked to facilitate the communication between the target site and vouching parties, special measures are necessary to ensure the authenticity of protocol messages exchanged. The reason is that the user agent is both untrusted and motivated to misbehave when challenged to prove his identity. Secrecy is also important but, since the user is privy to all exchanged information, it can be achieved through secure layer, e.g., SSL, connections between the user and the sites. To safeguard authenticity, each protocol message is required to carry the `service`, `signature` and `signed_fields` parameters. The first one is the identifier of the sender service and could be the service's domain, a URI under that domain or an alias. The second one is a cryptographic signature, such as RSA-SHA1, computed over the rest of the parameters using the private key of the service sending the message. The final parameter is a list with the names of parameters and the order in which they have been signed. As soon as the user-agent relays a protocol message to the receiving service, it will parse the `signed_fields` parameter, verify the signature based on those fields and accept the values of those fields while discarding all other fields. Verifying the signature entails looking up the public key for the service, specified by the `service` parameter. To do so efficiently we employ the X.509 certificate it has for HTTP over SSL.

Password reset. In the general case, without SAuth present, users have the ability to be reminded of their password or better yet reset it and choose a new one. For this to happen, users may be asked some security questions, such as the name of their pet, that they chose during account registration and subsequently receive an e-mail, to an address they also chose during registration, with either their current password or a secure URL to select a new password. Note that this current practice is already one form of synergy-based authentication, since the service places trust in the e-mail provider and expects only the owner of that address to be able to read the password-reset message. Unfortunately, this model fails when an attacker manages to get access to the e-mail address and then proceeds to initiate password-reset procedures for the victim's online accounts. Enhancing the current model with SAuth means that the service will proceed as detailed above only after the current user proves ownership of an account in a vouching service. Assuming the user hasn't forgotten all of his passwords in all the vouching services, he is first prompted by target service S to authenticate with vouching service V before S proceeds with the steps to reset the user's password for S. Note that this process is not meant to replace the current security-question model, whose security has been questioned by researchers [51], but to precede it and thereby couple it.

3.2 Usability

SAuth is built on standard technologies and is founded upon the existing browsing habits of users. Thereby it does not affect their perceived web experience. We are taking advantage of the fact that users prefer to maintain concurrently many browser tabs open [58] and spend at least 57.4% of their time in switching tabs [42]. This suggests the use of multiple web applications in one session. Furthermore, researchers have observed long-lived sessions in online social networks [24], which is another indicator suggesting that users are concurrently active in multiple sites.

What is more, in favor of users' convenience, current web applications maintain authentication state, usually in the form of HTTP cookies, for each established session, which can be destroyed only if the user is explicitly logged off or decides to erase the browser cache. In other words, like the user is not prompted for his password for every HTTP request the browser makes when rendering a site, he won't be interrupted when a service he is already logged in is involved as a vouching party in SAuth. In addition, browsers implement auto-completion features for filling out password forms. Therefore, even if the user is not currently authenticated with one or more of the services involved in a SAuth session, this feature will alleviate his inconvenience.

Finally, one may be concerned with the involvement of multiple accounts in different services if a user needs to register for all of them at once to activate SAuth. We expect that such cases will be rare in practice as vouching services will be selected due to their popularity and dependability. As such we expect that users will almost always have an account with one of the supported vouching services so, when registering with an SAuth-enabled service, they will not need to additionally register with one of the vouching services. Our use case is aligned with the existing "sign in with Google" and "sign in with Facebook" single-sign-on mechanisms; users most probably will hold an account with one of those services. Moreover, SAuth's design enables users to include a vouching service of their choosing by supplying its domain name to the target service (Sec. 4). This clearly increases the flexibility of SAuth. Nevertheless, in the unlikely case that the user is unable to provide an account with an existing, or introduce a new, vouching party to the target service, activating SAuth could also be deferred to the future.

3.3 Availability

The synergistic nature of SAuth creates dependencies between sites during their user authentication process. The unavailability of a vouching site V means the target service S will have to rely solely on its own authentication procedure which, as mentioned earlier, is the first step in an SAuth session and takes precedence over any vouching responses. In other words, should a vouching site be unavailable, the target service will still be able to function and operate, as it would do without the use of SAuth. Administrators can make a policy decision on whether they want to operate without enhanced authentication, admit the user into his account on a provisional basis with limited functionality, or engage additional checks such as security questions. Note that once a user is authenticated and receives back a token, for instance an HTTP cookie, he is not affected by any changes in the availability of the vouching sites. He will not have to initiate SAuth unless he relinquishes that token.

3.4 Password Compromise Alerts

The multi-party authentication process of SAuth enables a warning system for when passwords from one or more parties are compromised. In a password leakage an attacker would gain access to all password hashes stored by a target site. We assume that he will eventually recover the plain-text passwords using any of the techniques described in Sec. 2.2. As he will still be lacking the corresponding passwords for the vouching sites he will resort to online guessing attempts which will result in failed SAuth sessions. Even if the user has chosen the same password for both the target and vouching site, as described in Section 5, the attacker will still end up guessing online. Note that for a vouching site to receive a request from a target site, the user must have already successfully authenticated with the latter. Therefore, a vouching site dealing with a user repeatedly failing to authenticate with it can suspect that the user is trying to log in to an account he doesn't own. Considering some leniency for typing errors, this creates a reliable system for triggering alerts. The vouching site is in a position to notify the target site that a specific user is failing to authenticate even through he has managed to do so at the target site. At the same time, the target site will notice repeated SAuth sessions failing at the vouching parties even though local authentication goes through. Overall, the nature of the SAuth design offers indications of anomalous activity that would otherwise go unnoticed.

4. IMPLEMENTATION

We define the SAuth protocol messages as a set of URIs [21] which makes it easier to project them on a URI-oriented application-level protocol such as HTTP [11]. We assume that the user is represented by an agent program, e.g., a web browser. Although we focus on HTTP, as it is employed by web services, our design can be applied to any other application-level protocol provided it supports the concept of end-point redirection. We group our messages into two categories; registration and authentication messages defined as registration and authentication URIs respectively.

For a target service and a vouching service to engage in an SAuth session, they need to be aware of each other's end-points. Registration and authentication end-points may be explicitly exchanged offline upon prior agreement, or they may be retrieved automatically from an XML file, named `sauth.xml` by convention, hosted under the domain of each service and served over a secure network layer such as SSL, e.g., `https://www.example.org/sauth.xml`. This enables users of a target service to include a vouching service of their choosing by supplying the voucher's domain name in the target service's SAuth activation page. Alternatively, users can select one of the pre-configured vouching services.

To realize the technical issues from the adoption of our protocol by a web application we have implemented it in its entirety in PHP and subsequently developed adopting sample applications including both a front and back end. We argue that once SAuth is offered as a module or library it requires effort comparable to the use of OAuth and similar authorization protocols, something which a plethora of sites today uses. Our implementation is less than 1000 LoC.

Registration. A user who registers with S selects a vouching party from a list of cooperating services or may specify one of his choosing by specifying its domain name as described earlier (Listing 1, line 1). Note that selection of a vouching service or introduction of a new one is only

```
1 schema://TARGET_SERVICE_S/registration
2         [username, password, vouching_service]
3
4 schema://TARGET_SERVICE_S/registration
5         [action="commit",service=
              VOUCHING_SERVICE_V,alias=
              FOREIGN_ALIAS,nonce=NONCE,signature,
              signed_fields="action,service,alias,
              nonce"]
```

Listing 1: SAuth registration messages as URIs.

```
1  schema://VOUCHING_SERVICE_V/authentication
2          [action="register_alias", alias=
               FOREIGN_ALIAS, service=
               TARGET_SERVICE_S, nonce=NONCE,
               signature, signed_fields="action,
               alias,service,nonce"]
3
4  schema://TARGET_SERVICE_S/authentication
5          [username, password, vouching_service]
6
7  schema://VOUCHING_SERVICE_V/authentication
8          [action="vouch",service=TARGET_SERVICE_S,
               nonce=NONCE,signature,signed_fields="
               action,service,nonce"]
9
10 schema://TARGET_SERVICE_S/authentication
11         [action="verify", alias=FOREIGN_ALIAS,
               service=VOUCHING_SERVICE_S, nonce=
               NONCE, signature, signed_fields="
               action,alias,service,nonce"]
```

Listing 2: SAuth authentication messages as URIs.

possible upon registering a new account with the target service or after successfully authenticating to an existing one, through SAuth if enabled. The response of service S to the user-agent's registration request is a redirection towards the authentication end-point of the selected vouching service V (Listing 2, line 1) with the parameter `action` set to instruct the vouching service to first authenticate the user and then associate the resulting account with an anonymous alias that has been just generated. Service S also binds that alias with the newly registered account once it receives confirmation from V. Assuming the current user has an account with service V, he provides his credentials to authenticate. If the current user has enabled SAuth on service V, a synergy-enhanced authentication process will follow. Note that if the user does not have an account with V, he can optionally create one at this point, but even if he does not, such cases can be handled in the manner discussed in Sec. 3.2. Eventually the user successfully authenticates or creates a new account with V. Service V then redirects the user's agent back to service S while setting parameter `action` to signal service S that it should bind the generated alias to the current user's account. This conveys to S that the alias has been associated with the user's account on V and it will be part of a future vouching authentication process (Listing 1, line 4). This completes the registration process under SAuth. Note that activating SAuth is not strongly tied to creating a user account on S and can take place independently.

Authentication. To authenticate under the enhanced process of SAuth a user initially visits the service he wants to access, labeled as target service S. He is prompted for his name and password. He is also asked to select a vouching service V (Listing 2, line 4). This selection may either

```
[Authentication Request to example.com (S)]
POST /login HTTP/1.1
Host: www.example.com

username=bob&password=password&vouching_service=
     example.org

[Authentication Response from example.com]
HTTP/1.1 303 See other
Location: https://www.example.org/login?action=
     vouch&service=example.com&nonce=...&signature
     =...&signed_fields=action%2Cservice%2Cnonce

[Authentication Request to example.org (V)]
GET /login HTTP/1.1
Host: www.example.org?action=vouch&service=example
     .com&nonce=...&signature=...&signed_fields=
     action%2Cservice%2Cnonce

...
```

Listing 3: Example use of HTTP redirection to relay the first message of the SAuth protocol between the target and vouching service through the user agent.

take place in the same screen as the log-in form or after his credentials are authenticated by S. In the first case he is given the option of selecting any of the vouching services S supports. This is to avoid revealing to attackers the vouching service(s) a given user employs. In the second case, he is given the option of selecting only from the vouching services that have already been associated with his account through a foreign alias. Note that if the user specified a vouching party of his choosing through its domain name that party will be available as an option at this point. Target service S then redirects the user's agent to V while setting the parameter `action` to signal that a vouching for current user is expected from the remote service (Listing 2, line 7).

The user then presents his credentials in an authentication request towards service V for his respective account. On successful local authentication with V, the service's response to the user agent redirects it to the target service S while setting parameter `action` to signal that current user has successfully authenticated with some account, that the associated foreign alias is included in the response, and that service S should verify this vouching response and decide whether the returned foreign alias matches the alias bound to the user's account on S. On match, service S has successfully authenticated the user using SAuth and can optionally return a persistent authentication token, such as an HTTP cookie, to the user's agent so that future interactions with service S can skip the enhanced authentication in a manner similar to the way users don't have to type their password for each HTTP request their web browser makes.

HTTP User-agent redirection. To facilitate message relaying through the user agent in HTTP we employ 3xx redirection messages [11]. For service S to redirect the user agent to service V, it responds to the user agent's request with the 303 "See other" status code. It also includes the `Location` header with its value being that of a URI under the service V domain that targets the desired end-point and carries the information it wants to communicate to V in the form of parameters. Listing 3 presents an example where the user agent is redirected to service V with a vouching request after successfully authenticating to service S.

Cryptographic Signatures. As the user agent is responsible for relaying messages between the target service and vouching services, it is necessary to ensure the integrity of those messages. We assume that the secrecy of the messages is preserved as long as the user agent maintains SSL connections with the two services and that there is no need for the messages to be hidden from the user agent. We implement cryptographic signatures using 1024-bit RSA key pairs and the SHA-1 digest algorithm. Each protocol message must contain the parameters `service`, `signature` and `signed_fields`. The first parameter identifies the sender and is used to retrieve the necessary information for verifying the signature. The last parameter specifies which parameters are contained in the signature.

Web services use the same private key that supports their HTTPS connections with clients and thus the corresponding public key is protected under an X.509 public key certificate. For a service to verify a signature in SAuth, it uses the `service` identifier to locate the corresponding URL for the sender. If the hostname of the sender service is used as the identifier, the service connects to the sender and downloads its public-key certificate. There are other ways to resolve an identifier to a hostname or URL but they are outside the scope of this paper. After retrieving the certificate, the service tries to verify the signature over the parameters specified by the `signed_fields` parameter. On success, the parameters are committed to the current user session. HTTP parameters not covered by the signature are discarded. If a parameter is specified twice, only the instance carrying a value which causes the signature verification to succeed is kept. Finally, if a signature fails the request is terminated immediately and no processing takes place.

Apart from the integrity of protocol messages, it is very important to ensure their freshness and avoid replay attacks. For that matter, a nonce is generated per message and per user and is bound to the current user's session state that a service maintains. That nonce is included as a parameter to the protocol message sent to a remote service and the respective response is expected to carry the same nonce. Nonces do not survive the termination of a user session.

5. PASSWORD REUSE

SAuth is rendered moot when a user is sharing the same password across web sites acting as vouching services. Unfortunately this is quite common in practice where a password is reused across six different web sites [40] on average. In essence reuse intensifies the problem of password leakage [17, 34]. Password managing software, nowadays offered natively by web browsers, could remedy the situation but usability issues are presently obstructing wide adoption.

5.1 Decoys

In the spirit of recent research [27, 43], we propose placing decoy passwords in databases to introduce uncertainty about the actual passwords chosen by users. In other words, anyone examining the password database, including an attacker who has compromised it, will discover that every user account has N passwords instead of one. Any of those passwords can successfully authenticate the user to the service and that is a key difference between our proposal and Kamouflage [27] as well as Honeywords [43]. That is because decoy passwords carry no marks and receive no special treat-

	(a) normal login	(b) normal login (pwd reuse)	(c) leaked	(d) leaked (pwd reuse)
(1) Single-site authentication	PS^{-1}	PS^{-1}	1 ✗	1 ✗
(2) Two-site authentication without decoys	$(PS^2)^{-1}$	PS^{-1}	PS^{-1} ✓	1 ✗
(3) Two-site authentication with decoys	$K1 \cdot K2 \cdot (PS^2)^{-1}$	PS^{-1}	$K2 \cdot PS^{-1}$ ✓	$K1^{-1}$ ✓

Table 1: Probabilities of different log-in events under current and our proposed authentication systems. Case (a) refers to the normal scenario where someone without any prior knowledge of the password attempts to log in to a service while case (b) is a special sub-case where the user of the target account is reusing his password in more than one services. Cases (c) and (d) refer to the scenario where a service has been compromised and the user's password for that service leaked. Our approach (2) offers a significant increase in the security of cases (a), (b) and (c) but not (d). For that matter we couple our proposed system with decoys which overall decreases the probability of an attacker logging in all four cases.

ment from the service to eliminate heuristics that could distinguish them from the actual user-set passwords.

Decoy Generation. Generating decoys indistinguishable from actual data is an interesting research area. Bowen et al. have worked on providing believable decoys at the network and host level [30, 31]. Researchers have already proposed techniques for generating sets of similar tokens in an automated manner and applied them towards decoy passwords. In Honeywords [43] a tweaking algorithm with variations is used. The authors propose various transformations, such as chaffing-by-tail-tweaking, which basically produce new words similar to a core word, which is the original user's password. In Kamouflage [27] they extend the context free grammar rules proposed by Weir et al. [56] for generating similar passwords.

We consider the above works to be towards the right direction and extend them in our approach. The key idea is to identify the context of the password and randomly produce tokens that match it. However, it is important to keep two requirements in mind. First, there should be no single mask describing the set of decoy passwords. In other words, it should not be possible to determine the structure or the properties of all decoy passwords, including the actual password, by looking at any small subset of them. Therefore, we introduce variations in the structure and properties of the actual password to achieve diversity. Second, random generation might produce passwords that are unlikely to be chosen by actual humans. For instance, analysis of the leaked passwords from the RockYou service revealed that just 4% of passwords begin with a digit as opposed to a character. Randomly producing decoys with mixed characters and digits would produce a roughly equal amount of tokens starting with a character and a digit. The attacker could subsequently ignore any password starting with a digit without risking discarding the actual password. Therefore, we place weights on the generation process to favor human norms.

The process of generating decoys begins by analyzing the user-selected password. It is grammatically decomposed [56] and new passwords are generated by applying transformations to the core parts of the initial password. Transformations can be applied in the same fashion that users change their own password [59], like for example certain substitutions such as leetspeak [26]. We can also generate the corresponding parts of speech tags [45], and use them to produce random permutations that are still grammatically similar and valid. For example, "she runs fast" is written as "pronoun verb adverb" and could generate "she plays carefully" or "he speaks loudly". In the case where natural language processing fails to identify the presence of human language, we treat the password as a random string with certain statistical properties. For example, "6rZ" can be written as "digit lowercase uppercase" and generate "0xB" or "5cM" but not "5cm".

5.2 Incorporating Decoys

A site can transition gracefully into the use of decoy passwords. We have implemented generic wrappers of existing hash functions. Currently sites need to have a function that creates a password hash and one that validates it. Password decoys do not interfere with existing hashing schemes nor do they dictate any changes in the database schema. Instead, a site needs to enable `create(password, N)` and `validate(hash)` wrappers. The former takes as input a plaintext password and a positive integer `N`, produces `N` similar passwords and hashes each of them using the existing hashing functionality of the site. The function returns a password vector encapsulating `N` hashes per user and which is subsequently inserted into the database. The latter function looks up a given hash key in the user's password vector.

6. SECURITY EVALUATION

Here we evaluate the security of SAuth and compare it with the current practice of single-site authentication. We account for SAuth coupled with and without decoys and discuss the trade-offs. As already mentioned, there is no special marking for the service to distinguish the actual password from the decoys; any decoy password will successfully authenticate the user although the user is never made aware of them. Therefore if the number of generated decoys is too large in relation to the password space, typing a random password will result with good probability in authentication.

We consider the probability of two events, G_b and G_s, for someone guessing both passwords in a two-entity enhanced authentication process and someone guessing just the second password provided the first one is stolen. Considering the above, we wish the size of the decoy set to be as small as possible for event G_b and as large as possible for the second event, G_s. For a given password space, PS, and assuming $D(s)$ is the distribution of users that do share the same password with both services:

$$P(G_b) = \begin{cases} \frac{1}{PS} \cdot \frac{1}{PS-1} \approx \frac{1}{PS} \cdot \frac{1}{PS}, & \text{for all users,} \\ \frac{1}{PS} \cdot 1 & \text{for } D(s) \text{ of the users.} \end{cases}$$

In other words, for all users the probability of successful authentication is the probability of authenticating to the first service and the probability of authenticating to the second. However, for $D(s)$ of the users, after successfully authenticating to the first service, just trying the same password in the second will grant access to that user's account (we assume $D(s) = 70\%$ [40]). Similarly, the probability when the first password is known becomes:

$$P(G_s) = \begin{cases} 1 \cdot \frac{1}{PS-1} \approx 1 \cdot \frac{1}{PS} & \text{for all users,} \\ 1 \cdot 1 & \text{for } D(s) \text{ of the users.} \end{cases}$$

This means that the original design of synergy-based authentication will not be able to improve the security of accounts that reuse the password in the vouching service against a password-leak incident. It will however be able to significantly improve the security in the case where different passwords are employed and also when no site has been compromised.[2] By introducing $K1$ and $K2$ decoys in each service, respectively, the event probabilities become:

$$P(G_b) = \begin{cases} \frac{K1}{PS} \cdot \frac{K2}{PS-1} \approx \frac{K1}{PS} \cdot \frac{K2}{PS} & \text{for all users,} \\ \frac{K1}{PS} \cdot \frac{1}{K1} = \frac{1}{PS} & \text{for } D(s) \text{ of the users.} \end{cases}$$

We assume no overlap between decoy sets $K1$ and $K2$. As decoy sets are chosen at random without cooperation between the different services there may be some overlap. However, this is small and there is no hint for the attacker to determine the existence of overlap or its size. Finally, the probability of a successful login when the first password has been leaked becomes:

$$P(G_b) = \begin{cases} 1 \cdot \frac{K2}{PS-1} \approx 1 \cdot \frac{K2}{PS} & \text{for all users,} \\ 1 \cdot \frac{1}{K1} & \text{for } D(s) \text{ of the users.} \end{cases}$$

Table 1 summarizes and compares the probabilities of different events, with SAuth in place or not. Rows marked (1)-(3) characterize three different approaches, each one containing four cases marked as columns (a)-(d). Our initial approach (2) significantly improves the security of the authentication system when someone is trying to guess both passwords (column (a) in Table 1) unless the password is being reused in which case our approach offers the same security guarantees as current systems (column (b)). Moreover, while the original system would allow anyone with probability 1 to access the accounts of a service using leaked credentials, SAuth significantly reduces the probability to $^1/_{PS}$ (column (c)). Unfortunately, if the user of the target account has chosen to reuse his password in the vouching service, our method is rendered ineffective. For that matter we propose the decoy-enhanced approach (3) in which we can see that we maintain the same order of security in cases (a) through (b) and reduce probability 1 in case (c) to $^1/_K$ where K is the size of the decoy password set.

6.1 Decoy Set Size

[2]Consider an attacker that performs non-targeted online dictionary attack, where a single password is tried against all users of a particular web site.

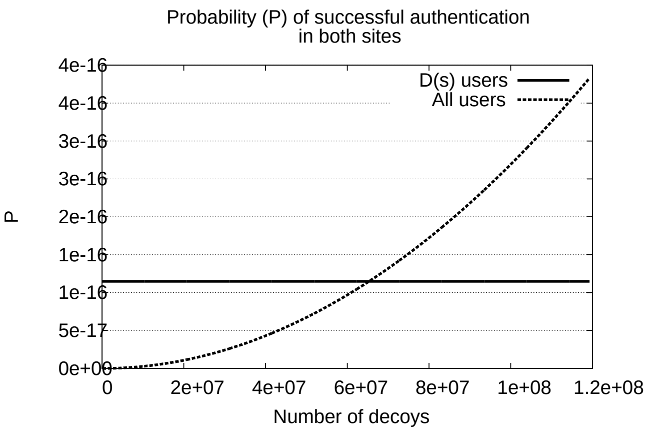

Figure 3: Probability of successful authentication in target site S and vouching site V when decoy passwords are used. As we do not distinguish between actual and decoy passwords, we require their number to be small over the password space.

In Honeywords [43] the authors suggest a typical set of 20 decoys and note that some sensitive accounts, such as the administrator's, may be associated with a larger set of 200 honey tokens. One could argue that adding more decoys increases the level of protection for the actual password as it creates more noise around it. In SAuth, where password decoys can actually be used for user authentication, reasoning about the size of the decoy set is not as straightforward. Here we theorize about the ideal size but then proceed to propose a more practical approach.

Based on Table 1 one observes in Figure 3 that in the absence of a security leak, and provided passwords are not recycled, expanding the decoy set actually makes guessing easier for an attacker (case (a)) as he can now figure out any one of the decoys as opposed to finding the actual user password. Case (b) is independent of the number of decoys and case (c), where the target site's passwords have been leaked, also requires a small number of decoys in relation to the password space. To justify the need for decoys we examine case (d) where users share the same password in both the target service and the vouching service. Since the attacker's probability of success is reduced to $1/K$ we want as many decoys as possible. Figure 4 presents this trade-off between the security of users with unique and shared passwords following a password database leak. Note that the password space drawn is $2 \cdot 10^{-6}\%$ of the entire space and has been selected as such to focus on the trade-off point between the two cases. Ideally, the size of the decoy set should result in an equal probability of success in both cases (c) and (d). In other words, there should be no incentive for the attacker to concentrate on either category of users. For example, given 70% probability that a user will reuse a certain password and a password space in the order of 94^8 ([a-zA-Z0-9] and symbols) we find that the decoy set should contain $O(94^4)$ decoys per user. For anything below that point, an attacker would prefer to guess among the already leaked decoy set, for a chance to attack 70% of the users of a service, rather than attempting a completely random password in hopes of falling within the decoy set of the second service. Note

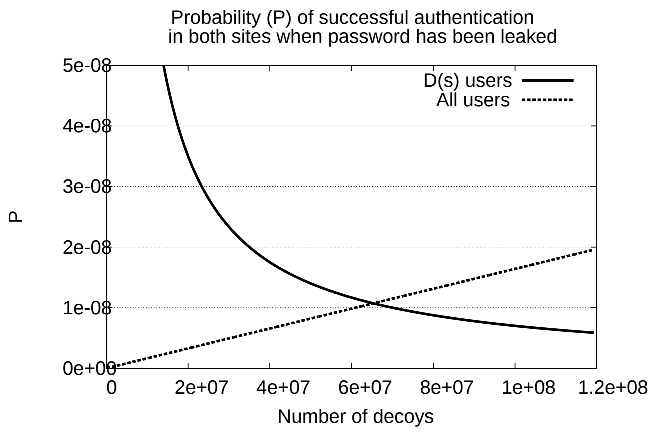

Figure 4: Probability of successful authentication in target service S following a password leak. Vouching site V and decoy passwords are employed. As the size of the decoy set for each user increases, the probability of attacking users who reuse their password across services is reduced. However, the probability of guessing a random password which falls within the decoy set increases in relation to the size of the password space. These conflicting requirements define the ideal size for the decoy set.

that this is a theoretical approximation which could however become practical as storage technology advances.

In practice, according to NIST [32], an online guessing attack for Level 1 should succeed with a probability of 1 in 1,024 and for Level 2 with a probability of 1 in 16,384. Thus, a site that meets the specification for Level 1 under case (d) should maintain at least 1,024 decoy passwords per user, while a site that meets the specification for Level 2 should maintain at least 16,384 decoy passwords per user. At the same time, the probabilities for case (c) are $\approx 10^{-12}$ and $\approx 10^{-11}$ respectively. Notice that even in the case of Level 2, the site needs to store an additional megabyte of information per user which we consider reasonable given that it is less than the size of some photos that users might upload online. [3]

6.2 Network Overhead

SAuth introduces additional network messages during the authentication process (see Fig. 1). During a traditional authentication the network time required is $t_{auth} = RTT_S$ where RTT_S is the round-trip time from browser to service S. With SAuth in place this time is $t_{sauth} = 2RTT_S + RTT_V$, where RTT_V is the round-trip time from browser to the vouching service V. Note that a single SAuth session can result in the user logging in to both the target site and the vouching site, something which in the traditional case requires $t = RTT_S + RTT_V$ if done in succession. In other words, SAuth introduces $RTT_S + RTT_V$ which can be just RTT if logging in to both services. Moreover, this overhead is incurred only in the absence of a persistent authentication token, e.g., a cookie, something which we expect to happen rarely.

[3] We assume a hash key of 512 bits.

7. RELATED WORK

Decoy Passwords. In Kamouflage [27] the authors propose injecting dummy passwords in password managers to protect the actual passwords if the master secret that unlocks the database is compromised. Similarly, honeywords [43] are fake passwords that are stored on the server together with the authentic password. Honeywords require a trusted third party for making sure, each time the user logs in, that the password entered is the authentic one and not a decoy.

k-Secret Sharing. Shamir et al. [52] proposed splitting up a secret into k pieces and distributing them to distinct parties in such a way at any given moment if $m < k$ pieces are present the original secret can be reconstructed. The same primitive has been applied to distributed user authentication by splitting up the stored password into multiple tokens [23,33]. Our proposal, SAuth, can be seen as a manifestation of this technique. In our terminology, the process of verifying each piece of the original password is called vouching. Instead of generating a password and splitting it up, we combine passwords the user already has to benefit from the security of a long and distributed secret.

Bounded Retrieval. Crescenzo et al. [36] propose making the authentication database too big for the attacker to retrieve. The server maintains a set of very large random files and each password is mapped to locations within them. The sum of those locations represents the password's digest requiring the full database to brute-force leaked hashes.

Authentication/Authorization. OAuth 2.0 [18] enables a third party to request access to a credential-restricted resource from its owner and receive that access without knowledge of the owner's credentials. OpenID 2.0 [49] provides a way for an end user to prove ownership of a claimed identity to a third party without a separate account. Facebook Connect [46] builds on top of both OAuth and OpenID to produce an authentication and authorization framework combined with the social information and graph its users form. BrowserID [2] or Mozilla Persona is a single-sign-on mechanism which uses e-mail addresses to represent user identities. PseudoID [37] employs blind cryptographic signatures to eliminate this privacy concern.

In this paper we propose a federated login system which may additionally facilitate the following use case; passwordless logins for a plethora of web services, much like Facebook Connect works today, but instead of using a single sign-on service, like Facebook, our system enables the use of a federated sign-on service which could include Facebook, Google and Twitter all vouching during log-in and thus making it harder for an attacker to exploit the single sign-on system.

8. CONCLUSION

We have presented SAuth, a novel protocol for synergy-based enhanced authentication in the face a password database leak. Users wishing to access their account on service S also have to authenticate for their account on service V, which acts as a vouching party. Both services can be regular sites already visited by the user every day, such as e-mail providers or social networks. The vouching process implicitly takes place through the user's browser and remains transparent to a large degree. With SAuth, should an attacker acquire the password for service S he will be unable to log in unless he also compromises the password for V. To mitigate password reuse habits we employ decoy passwords to introduce uncertainty regarding the actual one. SAuth is an extension and not a replacement of existing authentication methods, operates one layer above them and thus enables services to employ heterogeneous systems.

We make SAuth publicly available at `http://www.cs.columbia.edu/~kontaxis/sauth/`.

9. ACKNOWLEDGMENTS

We thank the anonymous reviewers and our shepherd, Anupam Datta, for the preparation of the final paper. We also thank Mariana Raykova and Michalis Polychronakis for early discussions. This work was supported by DARPA through contract FA8650-11-C-7190. Any opinions, findings, conclusions or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the US Government or DARPA. It was also supported in part by the FP7-PEOPLE-2010-IOF project XHUNTER, No. 273765, and SysSec, funded by the European Commission under Grant Agreement No. 257007.

10. REFERENCES

[1] CloudCracker :: Online Hash Cracker. `https://www.cloudcracker.com`.

[2] BrowserID. `https://github.com/mozilla/id-specs/blob/prod/browserid/index.md`.

[3] Directory of web sites storing passwords in plain text. `http://plaintextoffenders.com`.

[4] E-mail discussion at Debian about the wiki.debian.org security breach. `https://lwn.net/Articles/531727/`.

[5] Gmail account security in Iran. `http://googleonlinesecurity.blogspot.com/2011/09/gmail-account-security-in-iran.html`.

[6] Google Accounts Authentication and Authorization. `https://developers.google.com/accounts/docs/GettingStarted`.

[7] Google Declares War on the Password. `http://www.wired.com/wiredenterprise/2013/01/google-password/all/`.

[8] Hacker Posts 6.4 Million LinkedIn Passwords. `http://www.technewsdaily.com/7839-linked-passwords-hack.html`.

[9] How apple and amazon security flaws led to my epic hacking. `http://www.wired.com/gadgetlab/2012/08/apple-amazon-mat-honan-hacking/`.

[10] How to Safely Store a Password. `http://codahale.com/how-to-safely-store-a-password/`.

[11] HTTP 1.1. `http://tools.ietf.org/html/rfc2616`.

[12] IEEE data breach: 100K passwords leak in plain text. `http://www.neowin.net/news/ieee-data-breach-100k-passwords-leak-in-plain-text`.

[13] LinkedIn cleartext passwords. `http://dazzlepod.com/linkedin/`.

[14] New 25 GPU Monster Devours Passwords In Seconds. `http://securityledger.com/new-25-gpu-monster-devours-passwords-in-seconds/`.

[15] PayPal Leads Industry Effort to Move Beyond Passwords. `https://www.thepaypalblog.com/2013/02/paypal-leads-industry-effort-to-move-beyond-passwords/`.

[16] Sony Hacked Again, 1 Million Passwords Exposed. `http://www.informationweek.com/security/attacks/sony-hacked-again-1-million-passwords-ex/229900111`.

[17] The Domino Effect of the Password Leak at Gawker. `http://voices.yahoo.com/the-domino-effect-password-leak-gawker-10566853.html`.

[18] The OAuth 2.0 Authorization Framework. `http://www.ietf.org/rfc/rfc6749.txt`.

[19] TLS 1.2. `https://tools.ietf.org/html/rfc5246`.

[20] Twitter detects and shuts down password data hack in progress. `http://arstechnica.com/security/2013/02/twitter-detects-and-shuts-down-password-data-hack-in-progress/`.

[21] URI. `http://www.ietf.org/rfc/rfc2396.txt`.

[22] B. Adida. Beamauth: Two-factor web authentication with a bookmark. In *Proceedings of the 14th ACM conference on Computer and Communications Security*, 2007.

[23] A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu. Password-protected secret sharing. In *Proceedings of the 18th ACM conference on Computer and Communications Security*, 2011.

[24] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida. Characterizing user behavior in online social networks. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, 2009.

[25] R. Biddle, S. Chiasson, and P. Van Oorschot. Graphical passwords: Learning from the first twelve years. *ACM Computing Surveys*, 44(4), Sept. 2012.

[26] K. Blashki and S. Nichol. Game geek's goss: Linguistic creativity in young males within an online university forum (94//3 933k'5 9055oneone). *Australian Journal of Emerging Technologies and Society*, 3(2), 2005.

[27] H. Bojinov, E. Bursztein, D. Boneh, and X. Boyen. Kamouflage: Loss-resistant password management. In *Proceedings of the 15th European Symposium On Research In Computer Security*, September 2010.

[28] J. Bonneau. Statistical metrics for individual password strength. In *Proceedings of the 20th international conference on Security Protocols*. Springer, 2012.

[29] J. Bonneau, C. Herley, P. C. v. Oorschot, and F. Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, 2012.

[30] B. M. Bowen, V. P. Kemerlis, P. V. Prabhu, A. D. Keromytis, and S. J. Stolfo. A system for generating and injecting indistinguishable network decoys. *Journal of Computer Security*, 20(2-3), 2012.

[31] B. M. Bowen, P. Prabhu, V. P. Kemerlis, S. Sidiroglou, A. D. Keromytis, and S. J. Stolfo. BotSwindler: tamper resistant injection of believable decoys in VM-based hosts for crimeware detection. In *Proceedings of the 13th international conference on Recent Advances in Intrusion Detection*, 2010.

[32] W. E. Burr, D. F. Dodson, and W. T. Polk. *Electronic authentication guideline*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 2004.

[33] J. Camenisch, A. Lysyanskaya, and G. Neven. Practical yet universally composable two-server

password-authenticated secret sharing. In *Proceedings of the 2012 ACM conference on Computer and Communications Security*. ACM, 2012.

[34] W. Cheswick. Rethinking passwords. *Communications of the ACM*, 56(2), 2013.

[35] L. S. Clair, L. Johansen, W. Enck, M. Pirretti, P. Traynor, P. McDaniel, and T. Jaeger. Password exhaustion: predicting the end of password usefulness. In *Proceedings of the 2nd international conference on Information Systems Security*. Springer-Verlag, 2006.

[36] G. D. Crescenzo, R. J. Lipton, and S. Walfish. Perfectly secure password protocols in the bounded retrieval model. In *Theory of Cryptography Conference*. Springer, 2006.

[37] A. Dey and S. Weis. Pseudoid: Enhancing privacy in federated login. In *Hot Topics in Privacy Enhancing Technologies*, 2010.

[38] R. Dhamija and A. Perrig. Deja vu: a user study using images for authentication. In *Proceedings of the 9th USENIX Security Symposium*, 2000.

[39] R. Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *Proceedings of the Symposium on Usable Privacy and Security*, 2005.

[40] D. Florencio and C. Herley. A large-scale study of web password habits. In *Proceedings of the international conference on World Wide Web*. ACM, 2007.

[41] S. Gaw and E. W. Felten. Password management strategies for online accounts. In *Proceedings of the Symposium on Usable Privacy and Security*, 2006.

[42] J. Huang and R. W. White. Parallel browsing behavior on the web. In *Proceedings of the 21st ACM conference on Hypertext and Hypermedia*, 2010.

[43] A. Juels and R. L. Rivest. Honeywords: Making password-cracking detectable, 2013.

[44] P. G. Kelley, S. Komanduri, M. L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, and J. Lopez. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, 2012.

[45] C. D. Manning and H. Schütze. *Foundations of statistical natural language processing*. MIT, 1999.

[46] M. Miculan and C. Urban. Formal analysis of facebook connect single sign-on authentication protocol. In *Proceedings of the 37th International Conference on Current Trends in Theory and Practice of Computer Science*. Springer, 2011.

[47] C. Percival. Stronger key derivation via sequential memory-hard functions. `http://tools.ietf.org/html/draft-josefsson-scrypt-kdf-00`.

[48] N. Provos and D. Mazières. A future-adaptive password scheme. In *Proceedings of the USENIX Annual Technical Conference*, 1999.

[49] D. Recordon and D. Reed. Openid 2.0: a platform for user-centric identity management. In *Proceedings of the ACM workshop on Digital Identity Management*, 2006.

[50] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger password authentication using browser extensions. In *Proceedings of the 14th USENIX Security Symposium*, 2005.

[51] S. Schechter, A. J. B. Brush, and S. Egelman. It's no secret. Measuring the security and reliability of authentication via secret questions. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, 2009.

[52] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11), 1979.

[53] R. Shay, S. Komanduri, P. G. Kelley, P. G. Leon, M. L. Mazurek, L. Bauer, N. Christin, and L. F. Cranor. Encountering stronger password requirements: user attitudes and behaviors. In *Proceedings of the Symposium on Usable Privacy and Security*, 2010.

[54] S.-T. Sun, Y. Boshmaf, K. Hawkey, and K. Beznosov. A billion keys, but few locks: the crisis of web single sign-on. In *Proceedings of the New Security Paradigms Workshop*. ACM, 2010.

[55] R. Wang, S. Chen, and X. Wang. Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, 2012.

[56] M. Weir, S. Aggarwal, B. d. Medeiros, and B. Glodek. Password cracking using probabilistic context-free grammars. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, 2009.

[57] H. Wimberly and L. M. Liebrock. Using fingerprint authentication to reduce system security: An empirical study. In *Proceedings of the 32nd IEEE Symposium on Security and Privacy*, 2011.

[58] M. Wu, R. C. Miller, and S. L. Garfinkel. Do security toolbars actually prevent phishing attacks? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2006.

[59] Y. Zhang, F. Monrose, and M. K. Reiter. The security of modern password expiration: an algorithmic framework and empirical analysis. In *Proceedings of the 17th ACM conference on Computer and Communications Security*.