Kamouflage: Loss-Resistant Password Management

Hristo Bojinov[1], Elie Bursztein[1], Xavier Boyen[2], and Dan Boneh[1]

[1] Stanford University
[2] Université de Liège, Belgium

Abstract. We introduce Kamouflage: a new architecture for building theft-resistant password managers. An attacker who steals a laptop or cell phone with a Kamouflage-based password manager is forced to carry out a considerable amount of online work before obtaining any user credentials. We implemented our proposal as a replacement for the built-in Firefox password manager, and provide performance measurements and the results from experiments with large real-world password sets to evaluate the feasibility and effectiveness of our approach. Kamouflage is well suited to become a standard architecture for password managers on mobile devices.

1 Introduction

All modern web browsers ship with a built-in password manager to help users manage the multitude of passwords needed for logging into online accounts. Most existing password managers store passwords encrypted using a master password. Firefox users, for example, can provide an optional master password to encrypt the password database. iPhone users can configure a PIN to unlock the iPhone before web passwords are available.

By stealing the user mobile device the attacker is able to obtain the password database encrypted under the master password. He or she can then run an offline dictionary attack using standard tools [17,14] to recover the master password and then decrypt the password database. We examined a long list of available password managers, both for laptops and smartphones, and found that all of them are vulnerable to offline attack. To address this threat, several potential defenses quickly come to mind:

The first one is to use salts and slow hash functions to slow down a dictionary attack on the master password. Unfortunately, these methods do not prevent dictionary attacks [5]; they merely slow them down. Moreover on mobile devices, we found that password managers tend to offer the use of a numerical PIN code to protect the database. While PIN codes are more easy to use on a smartphone, they also offer a smaller key space which makes the aforementioned attacks easier.

Another potential defense is to use a password generator [18] rather than a password manager. A password generator generates site-specific passwords from a master password. Users, however, want the ability to choose memorizable

passwords so that they can easily move from one machine to another; this can be quite difficult with the strings typically created by a password generator. As a result, if a password generator is not ubiquitous, and currently none are, then the majority of users will never use one.

Finally, another defense is to store passwords in the cloud [21,7] and use a master password to authenticate to the cloud. This solution only shifts the problem to the cloud; any employee at the cloud service can run an offline dictionary attack to expose account passwords for multiple users. Other potential defenses and their drawbacks are discussed in Section 6.

Our contribution. We propose a new architecture for building theft-resistant password managers called *Kamouflage*. Our goal is to force the attacker to mount an *online* attack before he can learn any user passwords. Since online attacks are easier to block (e.g. by detecting multiple failed login attempts on the user's account) we make it harder for an attacker to exploit a stolen password manager. Major websites already implement internal security mechanisms that throttle after several failures and therefore forcing an attacker to perform online work is an effective defense.

Kamouflage works as follows. While standard password managers store a single set S of user passwords, Kamouflage stores the set $S_0 = S$ along with $N-1$ decoy sets $S_1, \ldots, S_{N-1}$. A typical value for N is $N = 10,000$, but larger or smaller values are acceptable. Based on our personal experience $M = |S|$, the size of the real stored password set, is on the order of 100 (a pessimistic estimate: in reality users on average have fewer than ten passwords [6]). The key challenge for Kamouflage is to generate decoy sets that are statistically indistinguishable from the real set. This is difficult because as shown by our user survey, users tend to pick memorizable passwords that are closely related. If Kamouflage simply picked random decoy sets, an attacker would be able to easily distinguish the real password set from the decoys.

With Kamouflage, an attacker who steals the device will be forced to perform, on average, $N/2$ online login attempts (or $N/2M$ at *each* of the M websites) before recovering the user's credential. Web sites can detect these failed login attempts and react accordingly. With web site participation, Kamouflage can be further strengthened by having the user's device registering a few decoy passwords (say 10) at web sites where the user has an account. If a web site ever sees a login attempt with a decoy password, the site can immediately block the user's account and notify the user. This is similar in spirit to the warning displayed by GMail when an account is accessed from two different countries in a brief period of time.

Using decoy password sets is a practical approach because if we assume that each user has about 100 passwords and each password takes about 10 bytes, then the decoy sets will take about 10MB of storage, which is roughly the size of three MP3 files, which a negligible storage requirement for modern laptops and smartphones.

Here are the main difficulties we had to overcome to make Kamouflage work:

– **Human-memorable passwords**: Since decoys must look like human memorizable passwords, we need a model for generating such passwords. To build such a model we performed a study of human passwords, as discussed in Section 2. We also took advantage of previous work on this topic [24,13,23,22]. While most previous work studied this problem for the purpose of speeding up dictionary attacks, here we give the first "positive" application for these password models, namely hiding a real password in a set of decoys.
– **Related passwords**: Since humans tend to pick related passwords, Kamouflage must pick decoy sets that are both chosen according to a password model and related to each other as real users tend to do. We develop a model for *password sets* that mimics human behavior.
– **Relation to master password**: In some cases it makes sense to encrypt the password database using a master password. Unfortunately, our experiments found that users tend to pick master passwords that are themselves related to the passwords being protected. Hence, we had to develop an encryption scheme that cannot be used to rule out decoy sets. We present our approach in Section 5.1.
– **Site restrictions**: Finally, different sites have different password requirements. When generating decoy sets we have to make sure that all passwords in the decoy set are consistent with the corresponding site policy.

We built a Kamouflage prototype as a drop-in replacement for the Firefox password manager. We give performance numbers in Section 4.2 where we show that Kamouflage has little impact on user experience. Our design is complementary to mechanisms that aim at preventing password theft in-transit, such as key-logging malware. Ideally both off-line and on-line protections should be deployed in order to ensure password safety.

2 How users choose passwords

In this section we present the results of our experiments on user password behavior that we use to support our assumptions that users are uncomfortable with random independent passwords, and as a result tend to select predictable and related passwords across multiple sites. In order to test this and related hypotheses, we conducted two experiments: qualitative user interviews and quantitative analysis of large, real-world password databases (one of them containing over 30 million entries). The results complement existing work in the area, such as [6].

Empirical motivation from user interviews. Our first experiment was a survey conducted with undergraduate and post-doctoral students on our campus. The goal of the survey was not to learn people's passwords, but to elicit their approach to dealing with passwords. The interviews were performed via an in-person questionnaire, and completed by 87 individuals: 30 CS undergraduate students, and 57 post-doctoral researchers from various fields including biology, chemistry and psychology. 33% of the respondents were female, and over 61% were between 26

Table 1. General properties of password databases. phpbb data is skewed towards shorter lengths (selection bias) because we had to crack it before analyzing it. We confirm the observations about password shape made in [24].

Name	Entries	Size: 1-4	Size: 5-8	Size: 9-12	Size: 12+
RockYou	32.6M	0.2%	69.3%	27.0%	3.5%
phpbb	343K	4.2%	82.3%	13.4%	0.1%

and 35 years old. It can be said that our subjects represented the worst case scenario for an attacker in the sense that they were highly educated (more than 60% having a PhD) and sophisticated in their use of the Internet.

In the survey we asked direct questions about users' password habits. We briefly summarize the results due to space constraints: 81% of our subjects admitted to reusing the same password on many websites, thereby supporting our hypothesis on password reuse. This hypothesis is also supported by the number of passwords used by our subjects: 65% of the sample reported using at most 5 passwords, and 31% reported using between 6 and 10 passwords. In a separate question, 68% of the sample admitted selecting related but not necessarily identical passwords across sites. We also found that 83% of our subjects reported that they did not password-protect their smartphone, despite the presence of private data on the phone.

Password database analysis. In our second experiment we analyzed two real-world password databases that were recently leaked to the public. One was from the RockYou service: it contained 32 *million* entries [20], and we had access to all the entries in plaintext. As a consequence, the results from its analysis can be considered a highly reliable predictor of user behavior. The other password database was from the developer web site `phpbb.com`. We include it here for comparison purposes, because `phpbb.com` does not enforce any password requirements, so users were free to use whatever they want. The `phpbb.com` database contained 343 000 passwords. In this database, passwords were not listed in the clear, but as hashes created by one of two schemes: a simple MD5 and 2048-fold MD5 with salt. Using a cluster of computers over several months, we where able to recover 241 584 passwords, or 71% of the full database. Since we did not have direct access to the passwords as plaintext, our recovery process induces a *selection bias* towards easier passwords, hence the `phpbb.com` numbers should be used only as a secondary reference point.

Table 1 shows some basic properties of the analyzed databases. In our work, the primary concern was the structure of passwords: understanding this structure is the key to being able to generate high-quality plausible decoys. Accordingly we tested the hypothesis that people use known words in their passwords by comparing the databases to the dictionaries created by *openwall* to work in conjunction with the famous cracker "John the ripper" [17]. When combined these dictionaries contain around 4 millions words (note that not all of these

Table 2. The effectiveness of simple word-based rules in parsing passwords. Percent matched increases with each rule. The high coverage makes it possible to use this approach for password set analysis (Section 4).

Rule Name	Format	RockYou	RockYou %	phpbb.com	phpbb.com %
Strict	W	6.6M	20.2%	80.0K	33.2%
Post	Wd+	6.9M	41.4%	37.7K	48.8%
Concat	WW	6.1M	60.1%	50.0K	69.6%
Digit	d+	5.2M	76.1%	32.4K	83.0%
Concat-Post	WWd+	2.4M	83.4%	9.3K	86.9%

Table 3. Examples of passwords that did not match our rules. A large portion of the remaining passwords can be classified based on additional rules like "hax0r" letter-digit substitution and "iluv***" three word and letter concatenation.

Password	Reason for not matching
`lordburnz`	Letter substituted ('s' → 'z')
`php4u`	Word-digit, word-letter, three tokens
`ilove$$$`	Non-alphanumeric, three tokens

are words from natural languages, but can rather be viewed broadly as "known password tokens").

Our analysis tool tried to match each password in the databases to one or two dictionary words according to several rules: direct match; direct match with a numerical suffix; match two words concatenated; etc. Just by using five rules we were able to match more than 80% of the passwords in both databases (Table 2). This result implies that for most users we can automatically produce the rules that were used in coming up with the passwords. Our password manager can then use the derived rules to generate new, plausible password sets that meet the same constraints. The remaining users appear to use more advanced password generation rules (Table 3), which can be emulated by building a simple N-gram model based on the extensive available data.

3 Threat model

The basic threat model. We consider an attacker who obtains a device, such as a laptop or smartphone, that contains user data stored in a password manager. The password manager stores user passwords for online sites (banking, shopping, corporate VPN) and possibly personal data such as social security and credit card numbers. The attacker's goal is to extract the user's data from the password manager.

We assume that the password manager encrypts the data using a master password. On Windows, for example, password managers often call the Windows DPAPI function `CryptProtectData` to encrypt data using a key derived from the user's login credentials. In this case we treat the user's login password as the

master password. Other passwords managers, such as the one in Firefox, let the user specify a master password separate from the login password.

An attacker can defeat existing password managers by an offline dictionary attack on the master password. Hence, simply encrypting with a master password cannot result in a secure password manager (unless the master password is quite strong). We capture this intuition in our threat model by saying that the attacker has "infinite" computing power and can therefore break the encryption used by the password manager. This is just a convenience to capture the fact that encryption based on a human password is weak.

In our basic threat model we assume that the attacker has no side information about the user being attacked such as the user's hobbies, age, etc. The only information known to the attacker are the bits that the password manager stores. We relax this assumption in the extended threat model discussed below. We measure security of a password manager by the expected number of online login attempts the attacker must try before he obtains some or all of the data stored in the password manager. The attacker is allowed to attempts to log on different and unrelated sites. We count the expected total number attempts across all sites before some sensitive data is exposed. To deal with web sites that have no online attack protections, Kamouflage will compartmentalize passwords into groups to ensure that the most sensitive passwords are protected in all cases, even if the less important ones are cracked successfully.

Extended model: taking computing time into account. In the basic model we ignored the attacker's computing time needed to break the encryption of the password manager. In the extended model we measure security more accurately. That is, security is represented as a pair of numbers: (1) expected offline computing time; and (2) expected number of online login attempts until information stored in the password manager is exposed. This allow us to more accurately compare schemes. For this extended model we are using encryption to slow down the attacker. The key challenge here is to design an encryption scheme that does provide information to the attacker that he can use to reduce the amount of his online work. Using encryption allows us to relax our basic-model assumptions. We permit the attacker to have side information beyond the device data, such as the user's age, gender, hobbies, etc. As we will see, we can offer some security even if the attacker has intimate knowledge of the victim.

Non-threats. In this paper we primarily focus on extracting data from long-term storage. We do not discuss attacks against the device or its operator while in active use (such as shoulder snooping and key loggers), and similarly omit hardware-based side-channel attacks, which can come in many guises, based, e.g., on electromagnetic emanations ("van Eck phreaking") and capacitive data retention ("cold-boot attacks"), to name but a few possibilities. Similarly, we do not address social engineering attacks such as phishing. While phishing is an effective way to steal user passwords, addressing it is orthogonal to our goal of providing security in case of device theft or loss.

4 Architecture

At any point in time, the password management software maintains a large collection of plausible password sets; exactly one of these sets is the real one, and the rest are *decoys*. Figure 1 illustrates the storage format. Apart from passwords, all other site information is kept in the clear. That is, an attacker looking at the database knows which sites the user has passwords for: she just has no idea what the passwords are. When the real user launches his web browser, he is prompted for the master password (MP) which is then cryptographically transformed to obtain the index of the real password set in the collection. During a dictionary attack, any attempt to guess the master password results in a different (plausible and valid, but incorrect) set of passwords.

Database operations. The following are the core operations that a password database needs to support, along with a description of our implementation for them. Note that in Kamouflage these operations are well-defined for any choice of master password: that is, by guessing a master password and attempting database operations the attacker does not obtain any new information about the validity of his guess.

- **Add a new password to the database.** In our design, this amounts to adding a new password to each password set in the collection. The real password set gets the user-supplied value, while decoy sets get auto-generated entries influenced by the true one.
- **Remove.** Remove a password from the database. This is the reverse to adding a password. Only requires removing the web site's entry from each password set. No regeneration is needed.
- **Update.** Update a password entry, presumably when a password is changed. While the size of the password set collection will not change, the corresponding entry in each decoy password set must also be regenerated. On the one hand, this step ensures that the reuse patterns in the decoy sets continue to match those in the real password set (subject to random mutations, of course). On the other hand, we are also preventing an attacker from looking at the database image before and after an update, and finding the real password set based on the fact that it is the only one that has changed. Journaling file systems make this second attack vector very plausible.
- **Find.** Find the right password given some web page characteristics like URL and form field names. Given the master password, this is a straightforward database access in the real password set, ignoring all decoys. (Of course, the MP is needed to locate the correct set.)

4.1 Password set generation

Decoy password sets must be indistinguishable from the real password set. We extend the context-free grammar approach from [22] to work for the case where

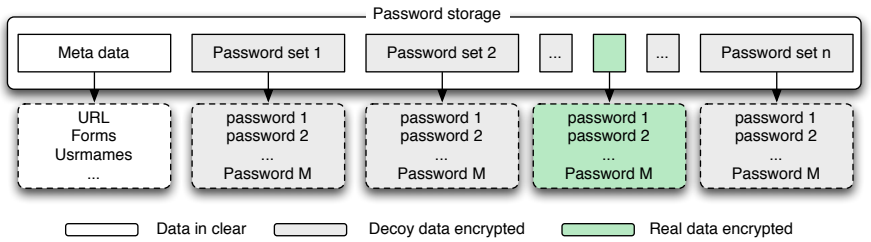

Fig. 1. Kamouflage database: cleartext metadata and encrypted real and decoy password sets. Encryption is discussed in Section 5.1.

multiple passwords are being generated for the same user. In [22], passwords candidates are generated by assigning probabilities to password templates (the templates look like "$l_4 d_2 s_1$", meaning "a four-letter word, followed by two digits, followed by a special character). The important insight there is that the likelihood of a password being used is determined by the likelihood of its template, in addition to the likelihood of its components. We point out that for the purpose of decoy password set generation, varying the templates is unnecessary, and even dangerous if done incorrectly: by performing asymmetric transformations on the template, the password manager might leak information about the correct password set.

In Kamouflage, decoy set generation proceeds in three steps:

1. **Tokenization:** The password manager converts the real user passwords into rules of the form $P_i \to T_{i_1} ... T_{i_i}$. The tokens typically stand for words or numbers of a certain size from a fixed dictionary[3]. In addition, tokens are reused across passwords, ensuring that portions that are shared by several passwords are correctly represented as equal by the rules. Tokenization is performed by attempting to partition each password according to the different rules from Section 2.
2. **Validation:** The system confirms that the tokenization is good: any tokens which are not dictionary words are flagged and reviewed by the user. This is the user's opportunity to remove from passwords any words that are specific to him, such as a last name or a birth date—such words can almost certainly be used to identify the correct password set among all decoys. In a perfect scenario, all of the users passwords will be readily tokenizable using standard dictionary words, possibly combined with apparently random characters. (As a consequence, decoys will be generated by using words of similar probability, along with similarly distributed random characters.)
3. **Generation:** Decoy sets are generated using the derived rules. We note that if the validation step completed without any tokens being flagged, then

[3] The dictionary used can be customizable to be able to accommodate different languages, or combinations of languages. However allowing arbitrary user customizations may lead to compromising security.

Table 4. Example decoy sets generated by the three-step algorithm.

Set	Description	Site #1	Site #2
S_0	Real	jones34monkey	jones34chuck
S_1	Decoy #1	apple10laptop	apple11quest
S_2	Decoy #2	tired93braces	frame93braces
S_3	Decoy #3	hills28highly	hills48canny

the fixed dictionary is as likely to generate the real password set as any of the decoys. The ease with which we can argue the statistical properties of generated decoys is the main reason we converged on this model.

A short example will clarify the password generation mechanism outlined above: let's assume the real set is of size two ($M = 2$), and we need to generate three different decoy sets ($N = 4$). Suppose the real password set consists of the passwords "jones34monkey" and "jones34chuck". The rules that are output by the tokenization step are: $P_1 \rightarrow ABC, P_2 \rightarrow ABD, A \rightarrow W_5, B \rightarrow D_2, C \rightarrow W_6, D \rightarrow W_5$. Depending on the dictionary used, the validation step could complete successfully, or maybe some of the words could be flagged as not present in the dictionary. For example the system could alert the user that "jones" is not a word from the dictionary, and as such does not blend in with generated decoy sets. The validation could also scan the contents of the user's device (e-mails, contacts, etc.) and specifically find words that are specific to the user even though they can be found in the dictionary—"chuck" is not likely a random 5-letter word if the user has a close relative called "Charles". After all issues are resolved at validation, the system generates decoy sets that might look like to ones in Table 4.
Optionally, the process of generating decoys can be customized by the user to better mimic the real password distribution. The customization choices should not be stored on disk as they can help the attacker.

4.2 Implementation

In order to prove the feasibility of our architecture, we built a proof-of-concept extension for the Firefox web browser, called Kamouflage. The extension implements the `nsILoginManagerStorage` interface and acts as an alternative storage for login credentials.

The main goal in developing the extension was to show that the overhead of maintaining decoy password sets is acceptable, particularly from a user's point of view. We intentionally used no optimizations in the handling of the password database, because we wanted to get a sense of the worst-case performance implied by our approach. Completing the extension to a point where it can be deployed for real-world use is straight forward.

When it is loaded, Kamouflage registers with Firefox as a login manager storage provider (`nsILoginManagerStorage`). Each of the implemented API methods (`addLogin`, `findLogin`, etc.) calls some internal methods that deal with

Table 5. User-visible performance of the Kamouflage Firefox extension for three typical use cases. The estimate of 20 passwords per user is realistic, while 100 passwords are a worst-case scenario unlikely to occur in practice [6].

Collection size (number of decoy sets)	10^3	10^4	10^4
Password set size (number of user passwords)	100	100	20
Database size on disk	2MB	20MB	4MB
Measured performance (access and update time)	< 1 sec	5 sec	< 1 sec

reading and writing the password database file from and to persistent storage, as outlined earlier (see Figure 1). If the password storage file does not exist, it is assumed that the user's password set is the empty set.

Performance. We measured how individual API calls are impacted by various password set collection sizes. We show that performance in Table 5. In our implementation the password file is read in its entirety every time a password is accessed, and written out completely for every update. From a user's point of view, there is no impact when maintaining approximately 10^3 decoy password sets; at 10^4 decoy sets the performance drop becomes clearly noticeable.

In practice, the performance of our prototype could be further improved in a number of ways:

- **Caching.** Our measurements of user-visible latency often include several invocations of the `nsILoginManagerStorage` interface, each of which reads the whole file from scratch. The login manager could cache the database contents, reading the file only once, at launch time.
- **Read size.** Password storage does not need to be read in its entirety. Given a master password (input by the user), only one password set needs to be read from disk. In the context of a Firefox extension this would require writing a native implementation for the read function: the JavaScript file I/O API available does not allow random access inside a file.
- **Write size.** Password sets do not all have to be rewritten on every `addLogin` or `updateLogin` operation, if we can guarantee that older versions are overwritten and unavailable to an attacker.

5 Extensions

The system described in Section 4 does not encrypt the password database with a master password. The master password is only used to identify the location of the real password set. As we will see, encrypting the password database without exposing the system to an offline dictionary attack is not trivial. In this section we extend Kamouflage to address this issue and others.

5.1 Why and How to Encrypt

Side information is dangerous. An attacker armed with side information about the victim can be successful, being able to guess the correct password set in the collection by searching for victim-related keywords in the password storage, hoping that those keywords appear as part of a password. Alternatively, if the user elects to use a very weak password at a specific, unimportant web site, the attacker may be able to recover it in an online attack, and use that information later on to crack the master password offline. Both of these attacks are reason enough to consider using encryption techniques similar to those used by current systems.

Password managers often encrypt the password database with a master password, denoted MP. In our settings this is non-trivial, and if done incorrectly, can cause more harm than good. To see why, suppose the password database is encrypted using an MP. Our user study from Section 2 shows that people tend to choose master passwords that are related to the passwords being protected. An attacker who obtains the encrypted password database can find the MP with an offline dictionary attack and then quickly identify the real password set by looking for a set containing passwords related to the MP.

Our approach to encryption. We use the following technique to avoid the preceding problem. Recall that each password set S_i contains a set of related decoy passwords generated as discussed in Section 4.1. We use the same approach to generate a master password MP_i for the set S_i, so that MP_i will likewise be related to the passwords in S_i. The master password for the real set is the user-selected MP. Now, for each set S_i do:

- generate a fresh random value IV_i to be stored in the clear with the set S_i, and
- use two key derivation functions (KDF) to generate two values K_i and L_i from MP_i as follows: $K_i \leftarrow \mathrm{KDF}_1(\mathrm{MP}_i,\ \mathrm{IV}_i)$; $L_i \leftarrow \mathrm{KDF}_2(\mathrm{MP}_i)$

The key K_i is used to encrypt the set S_i. The index L_i determines the position of the set S_i in the password database. In other words, the index of the set S_i in the database is determined by the master password for the set S_i. Collisions (i.e., two sets that have the same index L) are handled by simply discarding any new set that attempts to claim a busy slot, optionally regenerating it, and allowing some small fraction of decoy slots to remain unused in order to ensure short completion time. Once the whole database is generated, all master passwords MP_i are deleted. When the user enters the real master password MP the system can recompute the index L to locate the encrypted set and its IV and then recompute K to decrypt the set.

The best strategy for an attack on this system is to run through all dictionary words (candidate MPs) and for each one to compute the corresponding index L and candidate key K. Whenever the attacker eventually tries to decrypt a password set using the actual MP that was used to encrypt it, he can generally

recognize this fact, causing that MP to become exposed along with the corresponding password set. However, in the end, even after decrypting all the sets in the password database with their respective correct master passwords, the attacker must still do substantial online work to determine the good set.

Table 6 shows how the master password strength affects the offline computation effort and the number of online login attempts that an attacker needs to perform, even if the attacker has perfect knowledge of the real master password distribution.

Note that when the user adds or updates a password, the system cannot add passwords to the decoy sets since it does not have the master passwords for the decoy sets. Instead, when the real set is updated, all the decoy sets and their master passwords are regenerated, and all sets are re-encrypted using new random values IV_i. The previous IVs must be securely purged from the database to thwart attacks that compare the current contents from past snapshots.

Our performance numbers from Table 5 show that the running time to perform this whole update operation remains acceptable.

By adding encryption to Kamouflage, we have neutralized the threat of attacks based on side information: depending on the MP strength, the attacker may need to spend considerable offline effort before he is in a position to mount the online attack.

Table 6. Comparison of attack difficulty in traditional and the new password management schemes, for different master password strengths (distribution known by the attacker).

	Master Password Strength		
Traditional	**Weak**	**Medium**	**Strong**
Offline (# of decryptions)	10^4	10^7	10^{10}
Online (# of login attempts)	1	1	1
Kamouflage	**Weak**	**Medium**	**Strong**
Offline (# of decryptions)	10^4	10^7	10^{10}
Online (# of login attempts)	10^4	10^4	10^4

5.2 Website Policy Compatibility

Restrictive password policies can be detrimental to the security of passwords stored using a mechanism like Kamouflage. Imagine that a web site requires that user passwords consist only of digits, while the decoy set generator randomly uses letters and digits when generating all passwords. In this scenario, an attacker looking at all the password sets in the collection can zero in on the real password set, because most likely it is the only one which contains a numeric password for that specific web site. We surveyed the top 10 web sites listed by Alexa, along with a small list of bank web sites, and tabulated their password requirements.

Table 7. Password strength requirements at top sites ranked by Alexa and a small group of finance-related web sites.

Web Site	**Password Requirement**
Google	at least 8 characters
Yahoo!	at least 6 characters
YouTube	at least 8 characters
Facebook	at least 6 characters
Windows Live	at least 6 characters
MSN	at least 6 characters
MySpace	6 to 10 characters, at least 1 digit or punctuation
Fidelity	6 to 12 characters, *digits only*
Bank of America	8 to 20 characters, $\geq$ 1 digit and $\geq$ 1 letter, no `$ < > & ^ ! [ ]`
Wells Fargo	8 to 10 characters, $\geq$ 3 of: uppercase, digit, or special characters

The results are shown in Table 7, and clearly demonstrate that this danger is real.

It is evident that the major Internet web sites already allow arbitrary passwords, subject only to a minimum length requirement. Security-savvy companies such as Google, Yahoo, and Facebook realize that forcing specific password patterns on users results in a system that is more difficult to use, prone to human error, and ultimately less secure.

Kamouflage effectively deals with this challenge by mimicking the composition of a user's passwords, and ensuring that passwords containing specific classes of characters (lowercase letters, uppercase letters, digits, special characters) continue to contain those classes of characters in the generated decoy sets.

It is conceivable that some web sites will implement weak security, or will not be significant for the user and as a result their passwords will be easy to guess. In order to prevent this weakness from helping the attacker find out the master password, and along with it the passwords for more secure and important to the user web sites, it is also preferable for the password manager to allow the grouping of web sites according to their importance. This grouping can be suggested by the user or inferred automatically, and will determine whether site passwords are kept in the protected database, or in a separate, unprotected area.

5.3 "Honeywords": Using Decoys as Attacker Traps

Some web sites are averse to blocking a user's account when they see a large number of failed login attempts. This is usually due to fear that a user's account will effectively suffer a denial-of-service attack. The reasoning behind this is sound: it is much more likely an attacker is trying to block a user's account, than trying to guess her password. It is attacks that are exceptions to this rule that have the greatest potential to cause damage however: an unauthorized user of an account could transfer money, attack other related accounts, or steal personal information to be used later for identity fraud.

We have seen that decoy password sets carry certain risks when deployed without care. At the same time, they provide an opportunity to cooperate with web sites in detecting and blocking targeted attacks on user accounts, alleviating concerns over potential DoS vulnerabilities of the lock-out logic [15].

Supplying web sites with some of their corresponding decoy passwords can provide them with an effective tool for identifying attacks that are based on compromised password files, and encourage them to take steps to block the user account in such scenarios. This presents little risk on the part of the web site, because the likelihood that a casual DoS attacker hits a decoy password, without having access to the user's device, should be very low. In other words, knowing that an attack is not a random DoS but a genuine impersonation attempt will make web sites more willing to take immediate and decisive actions to stop the attack. This idea has been previously explored in the context of network security for identifying and rapidly blocking intrusions via honeypots [19,16,3].

5.4 Master Password Fingerprinting

The flip side of using decoy traps as a defense mechanism, is that it becomes vital to provide the user with positive feedback on the correctness of the master password being entered. With the honeyword mechanism, a mistake on the master password is indeed much more likely to result into a locked-out account than a mistake on the account's login password itself.

A simple technique similar to Dynamic Security Skins [4] can solve this problem. When the user selects his master password, he can be presented with an icon selected pseudo-randomly from several thousand possible ones, based on the master password. The user remembers the icon and uses its presence as a cue that he typed the correct master password when he logs in again later on. It is very unlikely that the user will mistype his password and at the same time get the same icon, believing the password he typed was correct. In particular, error-correction codes can be employed to ensure that single-character errors always result in different validation icons.

5.5 Kamouflage Summarized

It is instructive to take a step back and compare our extended Kamouflage architecture to a "traditional" password manager design.

The encryption step we added ensures that our password database is at least as hard to crack as a traditional one: an attacker that guesses the master password can test her guess offline, however even upon successful decryption of a password set, there will be no guarantee that the decrypted set is the real one and not a decoy. Successfully uncovering all password sets takes time proportional to the size of the master password space, which is just the same as with a traditional design. In other words, using decoy sets we are requiring the attacker to perform a significant amount of on-line work even when the whole space of master passwords has been explored offline.

Kamouflage also provides opportunities for additional security mechanisms to be deployed by web sites. We mentioned the use of honeywords, whereby a subset of the decoy password sets could be provided to web sites by the password manager, enabling them to identify and quickly respond to a targeted attack, without providing new opportunities for DoS.

Finally, the visual fingerprinting technique ensures that users have feedback on whether they entered the correct master password, without leaking any information to an attacker, and thus without weakening the strength of the master password.

6 Additional Related work

RSA key camouflage. The idea of camouflaging a cryptographic key in a list of junk keys was previously used by Arcot systems [12] to protect RSA signing keys used for authentication. Arcot hid an RSA private key among ten thousand dummy private keys. The user's password was a 4 digit PIN identifying the correct private key. The public key was also kept secret. An attacker who obtained the list of ten thousand private keys could not determine which is the correct one. Camouflaging an RSA private key is much easier than camouflaging a password since the distribution of an RSA private key is uniform in the space of keys.

Remote password storage. Several password management systems store passwords on a remote third party server. As examples, we mention Verisign's PIP [21], the Ford-Kaliski system [7], and Boyen's Hidden Credential Retrieval [2], while noting that many other proposals exist. These systems, unlike ours, require additional network infrastructure for password management. Moreover, in some systems, like Verisign's, there is considerable trust in the third party since it holds all user passwords. An exception is Boyen's HCR scheme [2], which is designed to exploit the limited redundancy of stored passwords to prevent the third-party storage facility from validating the master password offline. Compared to [2], Kamouflage can also deal with (sets of) passwords with large amounts of redundancy.

Trusted Computing Chips. Another approach to protecting password storage is to rely on disk encryption, such as Windows BitLocker which uses special hardware (a TPM) to manage the disk encryption key. An attacker who steals the laptop will be unable to decrypt the disk, unless he or she can extract the key from the TPM. This solution, however, cannot be used on devices that have no TPM chip, such as smartphones and some laptops; it also suffers from portability problems. Clearly, we prefer a solution that does not rely on special hardware.

Intelligent dictionary attacks. Several recent papers propose models for how humans generate passwords [13,8,22]. These results apply their models to speeding dictionary attacks. Here we apply these models defensively for hiding passwords in a long list of dummy passwords.

Slow hash functions and halting functions. Many password management proposals discuss slow hash functions for slowing down dictionary attacks [1,5,10]. These methods are based on the assumption that the attacker has limited computing power. They can be used to protect the user's master password against dictionary attacks. Our approach, which is secure in the face of an attacker with significant computing power, is complementary and can be used in conjunction with slow hashing methods for additional security.

Graphical passwords. Graphical passwords [11,9] are an alternative to text passwords. While they appear to have less entropy than textual passwords, our methods can, in principle, also be used to protect graphical passwords. One would need a model for generating dummy graphical passwords that look identical to human generated passwords. We did not explore this direction.

7 Conclusions

We presented a system to secure the password database on a mobile device from attacks that are often ignored by deployed password managers. The system leverages our knowledge of user password selection behavior to substantially increase the expected online work required to exploit a stolen password database. The mechanism can be further strengthened with the cooperation of web sites to detect decoy passwords. Using a prototype implementation we demonstrated that the system can be used as a drop-in replacement to existing systems with minimal impact on the user experience. Our experiments on real password databases suggest that the proposed decoy generation algorithm produces decoy sets that are indistinguishable from the real set.

References

1. X. Boyen. Halting password puzzles: hard-to-break encryption from human-memorable keys. In *16th USENIX Security Symposium—SECURITY 2007*, pages 119–134, 2007. 16
2. X. Boyen. Hidden credential retrieval from a reusable password. In *ACM Symp. on Information, Computer & Communication Security—ASIACCS 2009*, pages 228–238, 2009. 15
3. V. V. Das. Honeypot scheme for distributed denial-of-service. *Advanced Computer Control, International Conference on*, 0:497–501, 2009. 14
4. R. Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *SOUPS '05: Proceedings of the 2005 symposium on Usable privacy and security*, pages 77–88, 2005. 14
5. D. Feldmeier and P. Karn. UNIX password security – 10 years later. In *Proceedings of Crypto 1989*, pages 44–63, 1989. 1, 16
6. D. Florencio and C. Herley. A large-scale study of web password habits. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 657–666, New York, NY, USA, 2007. ACM. 2, 3, 10

7. W. Ford and B. Kaliski. Server-assisted generation of a strong secret from a password. In *Proc. 9th IEEE International Workshops on Enabling Technologies*, pages 176–180, 2000. 2, 15
8. W. Glodek. Using a specialized grammar to generate probable passwords. Master's thesis, Florida state university, 2008. 15
9. J. Goldberg, J. Hagman, and V. Sazawal. Doodling our way to better authentication. In *proceedings CHI 2002*, pages 868–869, 2002. 16
10. J. A. Halderman, B. Waters, and E. W. Felten. A convenient method for securely managing passwords. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 471–479. ACM, 2005. 16
11. I. Jermyn, A. Mayer, F. Monrose, M. Reiter, and A. Rubin. The design and analysis of graphical passwords. In *Proc. 8th USENIX Security Symposium*, pages 135–150, 1999. 16
12. B. Kausik. Method and apparatus for cryptographically camouflaged cryptographic key, 2001. US patent 6170058. 15
13. A. Narayanan and V. Shmatikov. Fast dictionary attacks on passwords using time-space trade-off. In *Proc. of ACM CCS 2005*, pages 364–372, 2005. 3, 15
14. P. Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Proceedings of CRYPTO 2003*, pages 617–630, 2003. 1
15. Account lockout attack, 2009. http://www.owasp.org/index.php/Account_lockout_attack. 14
16. G. Portokalidis and H. Bos. Sweetbait: Zero-hour worm detection and containment using honeypots. Technical report, Journal on Computer Networks, Special Issue on Security through Self-Protecting and Self-Healing Systems), TR IR-CS-015. Technical report, Vrije Universiteit, 2005. 14
17. O. Project. John the ripper password cracker, 2005. http://www.openwall.com/john. 1, 4
18. B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. Mitchell. Stronger password authentication using browser extensions. In *proceedings of Usenix security*, 2005. 1
19. A. Sardana and R. Joshi. An auto-responsive honeypot architecture for dynamic resource allocation and qos adaptation in ddos attacked networks. *Comput. Commun.*, 32(12):1384–1399, 2009. 14
20. TechCrunch. One of the 32 million with a rockyou account? you may want to change all your passwords. like now., 2009. http://techcrunch.com/2009/12/14/rockyou-hacked/. 4
21. Verisign. Personal identity portal, 2008. https://pip.verisignlabs.com/. 2, 15
22. M. Weir, S. Aggarwal, B. Glodek, and B. de Medeiros. Password cracking using probabilistic context-free grammars. In *proceedings of IEEE Security and Privacy*, 2009. 3, 7, 8, 15
23. R. Yampolskiy. Analyzing user passwords selection behavior for reduction of password space. In *Proc. IEEE Int. Carnahan Conference on Security Technology*, pages 109–115, 2006. 3
24. J. Yan, A. Blackwell, R. Anderson, and A. Grant. Password memorability and security: Empirical results. *IEEE Security and Privacy magazine*, 2(5):25–31, 2004. 3, 4