Mnemonic Password Formulas

Remembering Secure Passwords

Nov, 2006

I)ruid, C^{2}ISSP
<druid@caughq.org>
http://druid.caughq.org

Abstract

The information technology landscape is cluttered with large numbers of information systems, many of which have their own individual authentication systems. Even with single-sign-on and multi-system authentication mechanisms, systems within disparate authentication domains are likely to be accessed by users of various levels of involvement with the landscape as a whole. Due to this inherent complexity and abundance of varying authentication requirements, users must manage volumes of password credentials for all of the systems that they interface with regularly. This has given rise to many different insecurities resulting from poor methods of password selection and management. This paper describes some security issues facing users and management of authentication systems that involve passwords, further discusses current approaches to mitigating those issues, and then finally introduces a new method for password recall and management termed Mnemonic Password Formulas.

Contents

Chapter 1

The Problem

1.1 Authentication Systems Abound

The current information systems landscape is cluttered with individual authentication systems. Even when considering that many systems existing under a single management domain utilize single-sign-on and multi-system authentication mechanisms, multiple systems within disparate management domains are likely to be accessed regularly by users. Even users of the most casual level of involvement with information systems can easily be expected to interface with a half-dozen or more authentication systems in a given day. On-line banking systems, corporate intranet web and database systems, e-mail systems, and social networking web sites are just a few of the various types of systems that generally require user authentication.

Due to this abundance of authentication systems, many users are required to manage the large numbers of passwords needed to authenticate to these various systems. This has given rise to many common insecurities resulting from poor selection and management of passwords.

In addition to the prevalence of insecurities in password selection and management, advances in authentication and cryptography systems have instigated a shift in attack methodologies against these systems. While recent advances in computing power have made shorter passwords such as six characters or less, regardless of the complexity of their content, vulnerable to cracking by brute force[1], many common attack methodologies are moving away from cryptanalytic and brute force methods against the password storage or authentication system itself in favor of the intelligent guessing of passwords such as optimized dictionary attacks and user context guesses. Also, attacks against other credentials required by the authentication system such as key-cards and password token devices, as well as attacks against the interaction between the human and the systems themselves have risen in popularity.

Due to all of the aforementioned factors, the user's password itself is commonly the weakest link in any given authentication system.

1.2 Managing Multiple Passwords

Two of the largest problems with password authentication relate directly to the user and how the user manages passwords. When users *are not* allowed to write down their passwords, they generally will choose easy to remember passwords which are usually much easier to crack than complex passwords. Users will also trend toward re-use of passwords across multiple authentication systems.

When users inevitably have difficulty memorizing assigned random passwords[1], or passwords of a mandated higher level of complexity chosen themselves, and thus *are* allowed to write down their passwords, they may do so in an insecure location such as a post-it note stuck to their computer monitor or on a notepad in their desk. Alternatively, they may store them securely at the risk of loosing access to their password store, such as storing a password in an encrypted file within a PDA; The user may forget the password to the encrypted file, or the PDA could be lost or stolen, resulting in such users who cannot recall their passwords from memory requiring an administrative reset of those passwords.

1.3 Poor Password Selection

When left to their own devices, users generally don't choose complex passwords themselves[1]. Rather, they tend to choose easy to crack dictionary words because they are easy to remember. Occasionally an attempt will be made at complexity by concatenating two words together or adding a number. In many cases, the word or words chosen will also be related to, or within the context of, the user themselves like a pet's name, phone number, or a birth date.

These types of passwords require much less effort to crack than with a brute-force trial of the entire range of potential passwords. By using an optimized dictionary attack method, common words and phrases are tried first, and usually succeed. Due to the high success rate of this method, most modern attacks on authentication systems target guessing the password first before attempting to brute-force the password or launch an in-depth attack on the authentication system itself.

1.4 Failing Stupid

When a user can't remember their password, likely because they have too many passwords to remember or the password was forced to be too complex for them to remember, many authentication systems provide a mechanism that I have termed *"failing stupid."*

When the user "fails stupid," they are asked a reminder question which is usually extremely easy for them to answer. If they answer correctly, they are presented with an option to either reset their password, have it e-mailed to them, or some other password recovery method depending on the implementation. When this recovery method is available, it effectively reduces the security of the authentication system from the strength of the password to the strength of this simple recovery question, the answer to which is likely found via publicly available information.

1.4.1 Case Study: Paris Hilton Screwed by Dog

A well publicized user context attack[2] was recently executed against the Hollywood celebrity Paris Hilton in which her cellular phone was compromised. The account password recovery question that she selected for use with her cellular provider's website was "What is your favorite pet's name?", the answer to which many fans can probably recollect from memory, not to mention all the fan websites, message boards, and tabloids that likely have that information available to anyone that cares to search out and read it. The attacker simply "failed stupid" and reset her website account password which then allowed access to her cellular device and it's data.

Chapter 2

Existing Approaches

2.1 Write Down Passwords

During the AusCERT 2005 information security conference, Jesper Johansson, Senior Program Manager for Security Policy at Microsoft, suggested[3] reversing decades of information security best practice of not writing down passwords. He claimed that the method of password security wherein users are prohibited from writing down passwords is absolutely wrong. Instead, he advocated allowing users to write down their passwords. The reasoning behind his claim is an attempt at solving one of the problems mentioned previously; when users are not allowed to write down their passwords they tend to choose easy to remember, and therefore easy to crack passwords. He believes that allowing users to write down their passwords will result in more complex passwords actually being used.

While Mr. Johansson correctly identifies some of the problems of password security, his approach to solving those problems is not only short-sighted, but also not comprehensive. His solution solves the problem of users having to remember multiple complex passwords, however his solution also creates the other problems previously mentioned regarding the written passwords being physically less secure and prone to require administrative reset due to loss.

2.2 Mnemonic Passwords

A mnemonic password is a password that is easily recalled by utilizing a memory trick such as constructing the password from the first letters of an easily remembered phrase, poem, or song lyric. As an example, using the first letters of each word in the phrase "Jack and Jill went up the hill" would result in the password "JaJwuth".

For mnemonic passwords to be useful the phrase must be easy for the user to remember.

While previous research has shown[1] that passwords built from phrase recollection like the example above yield passwords with complexity akin to true random character distribution, mnemonic passwords share a weakness with regular passwords in that users may reuse them across multiple authentication systems. Also, such passwords are commonly created using well known selections of text from famous literature or music lyrics. Password cracking dictionaries have been developed that contain many of these common mnemonics.

2.3 More Secure Mnemonic Passwords

More Secure Mnemonic Passwords[4], or "MSMPs", are passwords that are derived from simple passwords that the user will more easily remember but that use mnemonic substitutions to give the password a more complex quality. "Leet-speaking" a password is a simple example of this technique. For example, converting the passwords "beerbash" and "catwoman" into leet-speak would result in the passwords "b33rb4sh" and "c@tw0m4n", respectively.

A problem unique to MSMPs is that not all passwords can be easily transformed. This limits either the choice of available passwords or alternatively the password's seemingly complex quality. MSMPs also rely on permutations of an underlying dictionary word or set of words which *are* easy to remember. Various cracking dictionaries have been developed to attack specific methods of permutation such as the "leet-speak" method mentioned above. Also, as with mnemonic passwords, these passwords might be reused across multiple authentication systems.

2.4 Pass Phrases

Pass phrases[5] are essentially what is used as the root of a mnemonic password. They are easier to remember and much longer which results in a password much more resilient to attack by brute force. Also, pass phrases tend to be much more complex due to containing upper and lowercase characters, white-space characters, and special characters like punctuation and numbers.

Pass phrases however have their own set of problems. Many authentication systems don't support lengthy authentication tokens, thus pass phrases are not usable consistently. Also, like the previous other methods, the same pass phrase may also be reused across multiple authentication systems.

Chapter 3

Mnemonic Password Formulas

3.1 Definition

A Mnemonic Password Formula, or MPF, is a memory technique utilizing a predefined, memorized formula to construct a password on the fly from various information that the user has available to them.

3.2 Properties

Given a well designed MPF, the resultant password should have the following properties:

1. It is a seemingly random string of characters.
2. It is long and very complex, therefore difficult to crack via brute force.
3. It is easy to reconstruct by a user via knowledge of only the formula, themselves, and the target authentication system.
4. It is unique for each user, class of access, and authenticating system.

3.3 Formula Design

3.3.1 Syntax

For the purposes of this paper, the following formula syntax will be used:

- $< X >$: An element, where $< X >$ is meant to be entirely replaced by something known as described by X.
- $|$: When used within an element's angle brackets ($<$ and $>$), represents an OR value choice.
- All other characters are literal.

3.3.2 A Simple MPF

The following simple formula should be sufficient to demonstrate the concept. Given the user authenticating and the system being authenticated to, a formula like the following could be constructed. It contains two elements, the user and the target system identified either by hostname *or* the most significant octet of the IP address.

$$< user >! < hostname|lastoctet >$$

The above MPF would yield such passwords as:

- "druid!neo" for user druid at system neo.jpl.nasa.gov
- "intropy!intropy" for user intropy at system intropy.net
- "thegnome!nmrc" for user thegnome at system nmrc.org
- "druid!33" for user druid at system 10.0.0.33

This simple MPF creates fairly long passwords containing a special character, and is very easy to remember, however the passwords yielded are not very complex. A diligent attacker may include the target user and hostname as some of the first combinations of dictionary words used in a brute force attack against the password. Also, these passwords may not be unique per system due to the fact that only the hostname or last octet of the IP address is used as a component of the password. If the same user has an account on two different web servers, both with hostname "www", or two different servers with the same last address octet value within two different sub-nets, the resultant passwords will be identical. Finally, the passwords yielded are variable in length and may not comply with a given system's password length policies.

3.3.3 A More Complex MPF

By modifying the simple MPF above, complexity can be improved. Given the user authenticating and the system being authenticated to, an MPF like the following could be constructed:

$$< u >! < h|n > . < d, d, ...|n, n, ... >$$

This MPF contains three elements; $< u >$ represents the first letter of the username, $< h|n >$ represents the first letter of the hostname *or* first number of the first address octet, and $< d, d, ...|n, n, ... >$ represents the first letters of the remaining domain name parts *or* first numbers of the remaining address octets, concatenated together. This MPF also contains another special character in addition to the exclamation mark, the period between the second and third element.

The above MPF would yield such passwords as:

- "d!n.jng" for user druid at system neo.jpl.nasa.gov
- "i!i.n" for user intropy at system intropy.net
- "t!n.o" for user thegnome at system nmrc.org
- "d!1.003" for user druid at system 10.0.0.33

This more complex MPF contains two special characters and yields more complex passwords, however the passwords are still variable length and may not comply with the authenticating system's password length policies. Our example MPF is also increasing in complexity and may not be easily remembered.

3.3.4 Design Goals

The ideal MPF should meet as many of the following design goals as possible:

1. Contain enough elements and literals to always yield a minimum password length.
2. Contain enough complex elements and literals such as capitol letters and special characters to yield a complex password.
3. Elements must be unique enough to yield a unique password per authenticating system.
4. Must be easily remembered by the user.

3.3.5 Layered Mnemonics

Due to the fact that MPFs can become fairly complex while attempting to meet the first three design goals listed above, a second layer of mnemonic properties applied to the MPF may be useful when attempting to meet the final design goal. The MPF, by definition, is a mnemonic technique due to it's property of allowing the user to reconstruct the password for any given system by remembering only the MPF and having knowledge of themselves and the system. Other mnemonic techniques can also be applied to help remember the MPF itself. This second layer of mnemonics may also be tailored to the user of the MPF.

Given the user authenticating and the system being authenticated to, an adequately complex, adequately long, and easy to remember MPF like the following could be constructed:

$$< u > @ < h|n > . < d|n >;$$

This MPF contains three elements; $< u >$ represents the first letter of the username, $< h|n >$ represents the first letter of the hostname *or* first number of the first address octet, and $< d|n >$ represents the last letter of the domain name suffix *or* last number of the last address octet. This MPF also contains another special character in addition to the exclamation mark and period; the semicolon after the final element.

The above MPF would yield such passwords as:

- "d@n.v;" for user druid at system neo.jpl.nasa.gov
- "i@i.t;" for user intropy at system intropy.net
- "t@n.g;" for user thegnome at system nmrc.org
- "d@1.3;" for user druid at 10.0.0.33

Unlike the other MPFs previously discussed, the one mentioned above employs a secondary mnemonic technique by having the MPF read in a natural way and is thus easier for a user to remember. The MPF can be read and remembered as "user at host dot domain." Also, a secondary mnemonic technique specific to the user of this MPF was used by appending the literal semicolon character. This MPF was designed by a C programmer who would naturally remember to terminate her passwords with semicolons.

3.3.6 Advanced Elements

MPFs can be made even more complex through use of various advanced elements. Unlike simple elements which are meant to be replaced entirely by some static

value like a username, first letter of a username, or some part of the hostname, advanced elements such as repeating elements, variable elements, and rotating or incrementing elements can be used to vastly improve the MPF's output complexity. Note however that overuse of these types of elements may cause the MPF to not meet design goal number four by making the MPF too difficult for the user to remember.

Repeating Elements

MPFs can yield longer passwords by repeating simple elements. For example, an element such as the first letter of the hostname may be used twice:

$$< u > @ < h|n >< h|n > . < d >;$$

Note, such repeating elements are not required to be sequential, and may be inserted at any point within the MPF.

Variable Elements

MPFs can yield more complex passwords by including variable elements. For example, the MPF designer could include an element indicating whether the target system is a personal or business system by prepending the characters "p:" *or* "b:" to the beginning of the MPF:

$$< p|b >:< u > @ < h|n > . < d|n >;$$

To further expand this example, consider a user who performs system administration work for multiple entities. In this case the variable element being prepended could be the first letter of the system's managing entity:

$$< x >:< u > @ < hi|n > . < d|n >;$$

$< x >$ could be replaced by "p" for a personal system, "E" for a system within Exxon-Mobil's management domain, or "A" for a system managed by the Austin Hackers Association. As you may have realized, most of the elements used thus far are in a sense actually simple variable elements that derive their value from other known information like username or system name. The difference here is that those elements are variable only in how their value changes when the MPF is applied to different systems, whereas these variable elements change values in relation to the context of the user, system, class of access, and any number of other factors you may care to take into consideration.

For example, using the same MPF for a super-user and an unprivileged user account on the same system may result in passwords that differ only slightly. Including a

variable element can help to mitigate this similarity. Prepending the characters "0:" or "1:" to the resultant password to indicate super-user versus unprivileged user access, respectively, by inclusion of a variable element in the MPF can result in the resultant password's complexity being increased as well as indicating class of access:

$$< 0|1 >:< u > @ < h|n > . < d|n >;$$

Note that variable elements are not required to be prepended to the beginning of the formula like all of the examples above; they could just as easily be appended or inserted anywhere else within the MPF.

Rotating and Incrementing Elements

Rotating and incrementing elements can be included to assist in managing password changes required to conform to password rotation policies. A rotating element is an element who's value rotates through a predefined list of values such as "apple", "orange", "banana", etc. An incrementing element such as the one represented below by $< \# >$ is an element who's value is derived from an open-ended linear sequence of values to be incremented through such as "1", "2", "3" or "one", "two", "three". When a password rotation policy dictates that a password must be changed, rotate or increment the appropriate elements:

$$< u > @ < h|n > . < d|n >;< \# >$$

The above MPF results in passwords like "d@c.g:1", "d@c.g:2", "d@c.g:3", etc.

$$< u > @ < h|n > . < d|n >;< fruit >$$

The above MPF, when used with the predefined list of fruit values mentioned in the paragraph above, results in passwords like "d@c.g:apple", "d@c.g:orange", "d@c.g:banana", etc.

The only additional pieces of information that the user must remember other than the MPF itself is the predefined list of values in the case of a rotating element and the current value of the rotating or incrementing element.

In the case of rotating elements this list of values may potentially be written down for easy reference without compromising the security of the password itself. Lists may further be obscured by utilizing certain values, like a grocery list or a list of company employees and telephone extensions that may already be posted within the user's environment. In the case of incrementing elements, knowledge of the current value should be all that is required to determine the next value.

3.4 Enterprise Considerations

Large organizations could use MPFs assigned to specific users to facilitate dual-access to a user's accounts across the enterprise. If the enterprise's Security Operations group assigns unique MPFs to it's users, Security Officers would then be able to access the user's accounts without intrusively modifying the user's account. This type of management could be used for account access when user is absent or indisposed, shared account access among multiple staff members or within an operational group, or even surveillance of a suspected user by the Security Operations group.

3.5 Weaknesses

3.5.1 The "Skeleton Key"Effect

The most significant weakness of passwords generated by MPFs is that in the case that the formula becomes compromised, all passwords to systems for which the user is using that particular MPF are potentially compromised. This, however, is no worse than a user simply using the same password on all systems; in fact, it's significantly better. When using a password generated by an MPF, the password should be unique per system and ideally should appear to be a random string of characters. In order to compromise the formula, an attacker would likely have to crack a significant number of system's passwords which were generated by the formula before being able to identify the correlation between them.

3.5.2 Complexity Through Password Policy

A second weakness of MPF generated passwords is that without rotating or incrementing elements they are not very resilient to password expiration or rotation policies. There exists a trade-off between increased password security via expiring passwords and MPF complexity, however the trade-off is to either have both, or neither. The more secure choice is to use both, however that increases the complexity of the MPF, potentially causing the MPF to not meet design goal number four.

Chapter 4

Conclusion

MPFs can effectively mitigate many of the existing risks of complex password selection and management, however their complexity and mnemonic properties must be managed very carefully in order to achieve a comfortable level of password security while maintaining memorability. When MPFs become too complex for users to easily remember, they may then reintroduce many of the problems they intend to solve.

Bibliography

[1] Jeff Jianxin Yan, Alan F. Blackwell, Ross J. Anderson, and Alasdair Grant. Password memorability and security: Empirical results. *IEEE Security & Privacy*, 2(5):25–31, 2004.

[2] Brian McWilliams. How paris got hacked? http://www.macdevcenter.com/pub/a/mac/2005/01/01/paris.html, February 2005.

[3] Munir Kotadia. Microsoft security guru: Jot down your passwords. http://news.com.com/Microsoft+security+guru+Jot+down+your+passwords/2100-7355_3-5716590.html, May 2005.

[4] Stephan Vladimir Bugaj. More secure mnemonic-passwords: User-friendly passwords for real humans. http://www.cs.uno.edu/Resources/FAQ/faq4.html, August 2006.

[5] Randall T. Williams. The passphrase faq. http://www.iusmentis.com/security/passphrasefaq/, August 2006.