Mnemonic Password Algorithms

Remembering Secure Passwords

I)ruid <druid@caughq.org>

What is a Mnemonic Password Algorithm?

- An MPA, or Mnemonic Password Algorithm, is a mental mapping of known elements that allows a user to quickly generate an adequately sized, adequately complex, and unique password for any given authenticating system.

Why are MPAs Needed?

- The security landscape today is cluttered with authentication systems
- Even with single-sign-on and multi-system authentication methods, there are still a number of disconnected systems a user must interface with (business, personal, websites, etc.)
- Remembering or managing storage of complex passwords for all these systems can be a nightmare
- The focus of attacks on authentication systems seems to be shifting toward intelligent guessing of passwords

Shifting The Focus

- As modern encryption, authentication, and message digest algorithms get stronger, the attack vector for authentication is shifting[3]:
  - Away from computationally breaking the password storage method (MD5, SHA-1, etc.) or brute-forcing the password
  - Toward intelligently guessing a user's password (dictionary attacks and context word targets)
- The user's choice in passwords is today's weakest link in authentication methods

The Numbers Game

- Most users, if left to their own devices, generally will not choose complex passwords themselves, instead opting for single dictionary words not using capitalization or special characters.
- These types of passwords require much less effort (and time) to crack than brute-forcing the key space if using an optimized dictionary attack method.
- For these reasons, most modern attacks on authentication systems target guessing the user's password first before attempting to brute-force the password.

Problems With Passwords Today

- If not allowed to write them down, users choose easy to remember passwords and tend to reuse passwords across multiple systems
- If allowed to write them down, users may choose harder to remember passwords, however:
  - They may store them in multiple insecure locations (home, work, etc.)
  - They may store them where they could become inaccessible and will require the password to be administratively reset

Failing Stupid

- Users today are tasked with remembering so many unique passwords, it's almost standard now for authentication systems (especially on the web) to provide a 'fail stupid' method of recovering a password.
- Bypassing the (hopefully) adequate authentication system, a user can generally reset a password or have it sent to them by answering a pre-chosen, easy question, such as:
  - What is your mother's maiden name?
  - What is your favorite color?
  - What is/was your high-school mascot?
- These types of questions are ripe targets for a user-context attack, the answers to which may be easily researchable via public information.

One Solution

- Recently, at a conference hosted by AusCERT, Microsoft's Jesper Johansson suggested[1] reversing decades of Information Security best practice of not writing down passwords
- He claims that the method of password security wherein users are prohibited from writing passwords down is absolutely wrong
- Instead, he advocates allowing users to write down their passwords

One Solution (cont.)

- While Mr. Johansson correctly identifies some of the problems of password security, his approach to solving the problems is not only short-sighted, but incomprehensive
- His solution solves the problem of users having to remember multiple complex passwords
- However, his solution creates the problems of the written passwords being physically less secure, or prone to require administrative reset due to loss

What Could Be Better?

A better solution to the password memorization problem might be to use one of a number of mnemonic techniques, such as:

- A password constructed from the first letters of an easy to remember phrase or sentence
- A modified version of a regular password
- Pass phrases

Mnemonic Passwords

- Mnemonic passwords have been used for quite some time and are nothing new.
- An example of a mnemonic password would be to use the first letters of an easy to remember phrase:

"Jack and Jill went up the hill"

which would become:

"JaJwuth"

The Problem With Mnemonic Passwords

- In order for mnemonic passwords to be effective, the original phrase must be easy to remember
- Easy to remember phrases generally don't contain non-alphanumeric or special characters and may yield passwords that are not very complex
- Mnemonic passwords might be reused across multiple systems; a user would still have to remember a unique phrase per system

More Secure Mnemonic Passwords

- MSMPs^2 are passwords that are derived from simple passwords that the user will easily remember but that use mnemonic substitutions to give the password a more random quality.
- “l33t-sp34k”ing your password is a simple example of this:

beerbash > b33rb4sh

catwoman > c@w0m4n

The Problem with MSMPs

- Not all passwords can be easily transformed, limiting either choice of available passwords or the password's pseudo-randomness
- Passwords might be reused across multiple systems; a user would still have to remember a unique MSMP per system

Pass Phrases

- Pass phrases are essentially the root of mnemonic passwords
- They are easy to remember
- Pass phrases are much longer and therefore take more time to brute-force than mnemonic passwords
- Pass phrases are generally more complex than mnemonic passwords because they usually contain spaces, capitol and lowercase letters, and occasionally special characters, punctuation, or numbers

The Problem With Pass Phrases

- Many systems don’t support lengthy authentication tokens, thus pass phrases are not usable consistently
- Pass phrases might be reused across multiple systems; a user would still have to remember a unique pass phrase per system

Mnemonic Password Algorithms

(Why you're attending this presentation)

Mnemonic Password Algorithms

- Given a well designed MPA, the resultant password can be:
  - A seemingly random string of characters
  - Very complex and hard to crack
  - Easy to construct via memory of just the algorithm and knowledge of the target authenticating system
  - Unique for each user, class of access, and system the user is authenticating to

Algorithm Syntax

- For the purposes of this presentation, the following algorithm syntax will be used:
  - <X> : An element, where <X> is meant to be replaced by something known
  - | : (pipe) When used within angle brackets (< and >), represents an OR value choice
  - All other characters are literal

A Simple Algorithm

- First we'll look at a very simple MPA to demonstrate the concept
- Given the user authenticating and the system being authenticated to, we could construct an algorithm like this:

<user>!<hostname|lastoctet>

<user>!<hostname|lastoctet>

- This MPA would yield passwords such as:

druid!neo (druid@neo.jpl.nasa.gov)

intropy!intropy (intropy@intropy.net)

thegnome!nmrc (thegnome@nmrc.org)

druid!33 (druid@10.0.0.33)

etc…

<user>!<hostname|firstoctet>

- This MPA generally creates adequately long passwords and contains a special character, however:
  - The passwords are not very complex
  - Containing the full username and hostname may make the passwords easy to crack
  - The passwords may not be unique per system
  - The passwords are variable length and not guaranteed to meet password length requirements

A More Complex MPA

- Next we'll look at a slightly more complex MPA. Given the user authenticating and the system being authenticated to, we could construct an algorithm like this:

<u>!<h>.<d>

- In this algorithm:
  - <u> represents the first letter of the username
  - <h> represents the first letter of the hostname or first number of the first address octet
  - <d> represents the first letters of the remaining domain name parts or first numbers of the remaining address octets concatenated together.
- We also added another special character, the '.' between <h> and <d>

<u>!<h>.<d>

- This MPA would yield passwords such as:

d!n.jng (druid@neo.jpl.nasa.gov)

i!i.n (intropy@intropy.net)

t!n.o (thegnome@nmrc.org)

d!1.003 (druid@10.0.0.33)

etc…

<u>!<h>.<d>

- This MPA creates fairly complex passwords and contains two special characters, however:
  - The passwords are variable length and not guaranteed to meet minimum password length requirements
  - The MPA is beginning to become complex and may not be easily remembered

MPA Design Goals

- Contain enough elements to always yield a minimum password length
- Contain enough complex elements such as capitol letters and special characters to yield a complex password
- Elements must be unique enough to yield a unique password per system
- Elements must be easy to remember for the user

Meeting the Design Goals

- An adequately complex and easy to remember MPA may be something like this:

  <u>@<h>.<d>;

- In this algorithm:
  - <u> represents the first letter of the username
  - <h> represents the first letter of the hostname or first number of the first address octet
  - <d> represents the last letter of the domain name or last number of the last address octet

<u>@<h>.<d>;

- This MPA would yield passwords such as:

d@n.v; (druid@neo.jpl.nasa.gov)

i@i.t; (intropy@intropy.net)

t@n.g; (thegnome@nmrc.org)

d@1.3; (druid@10.0.0.33)

etc…

<u>@<h>.<d>;

- This MPA creates adequately complex passwords and contains multiple special characters
- This MPA always creates passwords 6 characters in length
- The MPA is fairly easy to remember:
  - The elements read in a natural way: “user at host dot domain”
  - It was designed by a C programmer, who will be able to remember to put a ‘;’ at the end.

Even More Complexity

- You can make your MPA's output even more complex with a few simple techniques:
  - Repeating Elements
  - Variable Elements
  - Rotating and Incrementing Elements
- These techniques may make the MPA harder to remember, so use sparingly

Repeating Elements

- Your MPA can generate longer and even more complex passwords by repeating elements
- For example, you may want to repeat an element such as the first letter of the hostname:

<u>@<h><h>.<d>;

Variable Elements

- Your MPA can generate even more complex passwords by including variable elements
- For example, you may want to include an element indicating whether the target system is personal or business by adding the characters ‘p:’ or ‘b:’ to the beginning of the algorithm:

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

Variable Elements (cont.)

- To take the previous example even further, let's say you are perform contract administration work for multiple entities. The variable element could be the first letter of the system's managing entity:

  <x>:<u>@<h>.<d>;

- <x> could be:
  - personal: p
  - Exxon-Mobil: E
  - dc214: d

Variable Elements (cont.)

- Variable elements can be used to differentiate between classes of access
- For example, using the same algorithm for super-user and normal-user access on the same system may result in passwords that have only minor differences
- Adding a variable element helps to mitigate this similarity. By adding the characters ‘0:’ (super-user) or ‘1:’ (normal-user) to the beginning of the algorithm, we can increase complexity and identify class of access:

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

Rotating and Incrementing Elements

- You can use rotating or incrementing elements to help manage password changes required to conform to a password policy
- A rotating element is a variable element that rotates through a list of values such as:
  - apple, orange, banana, etc.
- An incrementing element is a linear sequence of values to be stepped through such as:
  - 1,2,3, etc.
  - one, two, three, etc.

Rotating and Incrementing Elements (cont.)

Each time a system requests that you change your password, rotate or increment the variable element:

<u>@<h>.<d>:<#> would result in passwords like d@c.g:1, d@c.g:2, d@c.g:3, etc.

<u>@<h>.<d>:<fruit> would result in passwords like d@c.g:apple, d@c.g:orange, d@c.g:banana, etc.

The only additional piece of information the user must remember in addition to the algorithm itself is the current value of the rotating or incrementing element.

Managing Enterprise MPAs

- Large organizations could use assigned MPAs for dual-access to a user's accounts across the enterprise.
- If the enterprise's Security group assigns unique MPAs to it's staff, Security Officers are then able to access the user's accounts without intrusively modifying the account.
- This type of management may be used for:
  - Account access when staff are absent
  - Shared accounts between multiple staff members
  - Staff surveillance by the Security group

MPA Weaknesses

- If the MPA is compromised, all passwords to systems that the user uses the MPA for are compromised
- Without rotating or incrementing elements, MPA generated passwords are not resilient to password expiration policies

Q&A

Any questions?

References and Further Reading

- [1] Microsoft Security Guru: Jot Down Your Passwords (Munir Kotadia, News.com.com, May 23, 2005)
  - http://news.com.com/Microsoft+security+guru+Jot+down+your+passwords
- [2] More Secure Mnemonic-Passwords (Stephan Vladimir Bugaj)
  - http://www.cs.uno.edu/Resources/FAQ/faq4.html
- [3] How to Choose a Passphrase FAQ
  - http://www.skuz.net/passfaq.html
- The Memorability and Security of Passwords – Some Empirical Results (Yan, Blackwell, Anderson, & Grant, Cambridge University Computer Laboratory)
  - http://www.ftp.cl.cam.ac.uk/ftp/users/rja14/tr500.pdf