Hashtabula 1.0

Hashtable-based password cracking system

Hashtabula is a password-cracking system designed to help penetration testers crack passwords in milliseconds rather than days or weeks. It works by taking advantage of the time-space trade-off made possible by modern storage technology.

Hashtabula supports any hashing algorithm supported by the Java platform on which it runs. By default, Java 6 supports SHA-256, SHA-512, SHA, SHA-384, MD5, and MD2 hash algorithms, while other algorithms can be added and will work seamlessly with Hashtabula.

Download

This software can be download from http://www.nick-brown.com/hashtabula

Usage

Hashtables can be very large, and they grow in size exponentially on both the password length and the number of characters used to create them. To use Hashtabula, you will need a fast computer with a large disk and a large amount of memory. On such a system:

1. Install Java if it is not already installed, and make sure the directory containing Java is in the system's PATH. On a Windows system, java.exe is typically found in "C:\Program Files\Java\jre6\bin". Java is automatically added to the PATH on Unix systems.
2. Download Hashtabula and extract it to a large disk partition. On a Unix system, you may need to make the extracted files "generate" and "search" executable.
3. Open a command window in the location to which you extracted Hashtabula.
4. View the help message with the command "generate", which explains which hash algorithms are supported and which character sets are available. More character sets may be created by editing "character_sets.utf8.conf".

   ```
   generate
   ```

5. To create a hashtable, you must specify a table name, an algorithm, and a character set. Below we will use the name "snowcrash", the algorithm SHA-256, and both lowercase letters and numbers for password selection.

   ```
   generate -name=snowcrash -algorithm=SHA-256 -character-set=lowercase+numbers
   ```

6. The search command may now be used to instantly look-up any password in the hashtable based on a known hash value.

   ```
   search -name=snowcrash
   -hash=68d6afc9d50755eb78ca3fe35eff54a6389fbe07423c7f9bd3067f34e71e10e0
   ```

You will see the password which corresponds to that hash.

To create hashtables for more passwords, extend the lengths of the passwords generated with the -maximum-length argument (example: -maximum-length=5).

To generate passwords composed of different sets of characters, use different options to -character-set. For example, to generate passwords which contain every key on a standard US keyboard, the argument would be “-character-set=uppercase+lowercase+numbers+symbols”.

Note that table size grows very quickly. When using a the full US keyboard character set above, each additional character in the password length increases the hashtable size by a factor of 100. If disk space is limited, it may be necessary to use more limited character sets for longer passwords. It is recommended that you generate tables for short passwords first, so that you can extrapolate the time and disk requirements needed to generate your intended hashtable on your system.

To generate a hashtable which is effective against a uniformly-salted system, use the -prefix-salt or -postfix-salt options, depending on how the salt is concatenated with the passwords.

Password-cracking theory

Cryptography is a complex and nuanced subject. Consequently, many software developers and companies make mistakes when implementing cryptographic systems, because they fail to consult with a security expert during the design phase, and they fail to have their software checked for security weaknesses before shipping it.

Many of the most common mistakes are related to password storage. Because databases can be compromised in many ways, most developers realize that it is unwise to save users' passwords directly in the database. To address this concern, password hashing is often employed. Proper password hashing requires the use of a hash algorithm that is expensive to compute, and the use of a random, unique salt for every password. Bcrypt is a popular example of such a system.

Unfortunately, few software development platforms ship with systems similar Bcrypt by default, and few developers comprehend the importance of password algorithms and salts. As such, a great deal of software in use today stores password hashes in either unsalted or uniformly-salted formats generated by cheap-to-execute hashing algorithms, such as SHA.

Hashtabula defeats both unsalted and uniformly-salted password systems based on such hashing algorithms.

Versions and changelog

- 1.0 released 2010-04-24

Features planned for future versions

- Better Unicode support
- Hashtable compression
- Automatically-adjusting disk-cache sorting
- Better error-handling
- Resume support
- Heuristic password generation (passwords most likely to be selected by humans)
- Dictionary-based password generation

License