Report: Keysmith

Overview

Keysmith is a cryptographically secure authentication token generator. It uses a true random number generator (provided by RANDOM.ORG) to produce a specified number (degree) of nonces that are used as indices to a word list. The entry at a given index is used as a concatenated section (tooth) of the key. This method is very similar to diceware except that it can use simpler, more diverse, more scalable word lists; however, unlike diceware implementations, it requires a network connection to collect random numbers.

Motivation

For awhile now, I have had an idea for an account manager that does more than simply remember the user's passwords. I have named the project Sentinel and one of its key features is a passkey generator. The design would place related accounts in groups which would then be assigned a tier based on the level of security demanded by the accounts within it. Each tier specifies how to generate an authentication key and how long that key is valid. Keysmith is the generator Sentinel will use, and for this project, I created it with support for a single tier and the ability to make strong, diceware-style keys as described above.

Architecture

The simple architecture of Keysmith consists of 4 parts: option handling (parsing), word list checking (I/O), random number retrieval (networking), and key assembly (string operations).

The first part involved implementing the 3 standard options: help (h), verbose (v), version (V), and, in the C implementation only, details (d) which prints additional information about the generated key (e.g. nonces used, key length, etc.). The C and Haskell implementations have native and very similar interfaces for accomplishing this; however, Java requires the third party library JCommander to parse command-line input. The Java version cannot handle combined options (-dv instead of -d -v) like C, and the C parser does not have an automated way to generate help menus. However, Haskell excels in both.

The next part consists of checking for the word list (etc/word.list by default)and counting the number of indexable lines (one word per line, presumably). This count is used to determine the range of random numbers to generate in the next part. The Haskell implementation's affinity for higher-order functions makes this task a breeze. C and Java require a bit more error-checking.

The third part controls the obtainment of random nonces. For this, the pseudo-random generator supplied by most systems is not adequate. RANDOM.ORG offers a true random number generator online for free. It supplies random numbers generated through the observation of atmospheric noise to any HTTP or HTTPS conforming client. To deploy Keysmith to any kind of real audience of users, HTTPS would be a necessity due to the secure objective of this application; however, for now, each implementation only supports HTTP. The C version utilizes the cURL library to initiate a GET request to

http://www.random.org/integers/ with parameters describing the desired result set (minimum value, maximum value, quantity, format, etc.). The Haskell implementation uses the Network.HTTP module available on Hackage to make a similar request. Java has a native library for achieving this task.

The final part is very string operation and I/O heavy. Each implementation looks up the word at the offset indicated by a nonce. that word is then read and appended to the initially empty key. When every random number has been used, the finished key is delivered to the user, and the program exits.

Operation

Below are some screenshots of different results the program produces.

```
Animus:keysmith david$ bin/keysmith-c
unabridgedspinnakermoustache
```

1. A basic run of the C implementation

```
http://www.random.org/integers/?num=3&min=0&max=41237&...
39293 => vainglory
1563 => apace
29681 => refuter
---------------------
vaingloryapacerefuter
---------------------
21 characters
```

2. The C implementation's unique -d (detailed) results

```
Animus:keysmith david$ bin/keysmith-h -h
keysmith-h
  -h  --help     Show help
  -v  --verbose  Enable verbose messages
  -V  --version  Print version
```

3. Haskell implementation showing help.

```
Animus:keysmith david$ java -jar bin/keysmith.jar -v
counting words in etc/word.list...41238
retrieving random numbers...done
generating a key...curranteightiethtroth
```

4. Java implementation with verbose messages enabled.

Implementation

My original proposal for this project was to do the other main body of code needed by Sentinel: AES encrypted key storage and retrieval. I ended up switching my focus to the generator for a couple reasons. One reason was that I recently implemented that portion of Sentinel for a different course last quarter (CMPS 183). That design was web-based, though, (web2py) and I want to pursue a more offline-friendly application. That aside, the main consideration that pushed me to do the generator instead is its potential for use outside the Sentinel application. I feel the key generator is a more useful tool that could appeal to more users. I think that it would be more beneficial if it could be made to run on a variety of platforms accordingly. This is exactly what the project called for.

As far as the implementation timeline I proposed, I was very backloaded this quarter due to other class projects, and I tended to focus on one project at a time. The decision to do the generator came late and resulted in constant development for about four days straight (about 30 hours).

category	C*	Haskell	Java
lines	188	90	148
libraries	7	7	11
hours	12	8	10

***C includes an additional feature (-d details).**

5. Implementation comparison

Conclusion

To be honest, this project taught me a lot about the communities that support the C, Haskell, and Java languages. I discovered how helpful using IRC for technical assistance and less commonly found explanations can be. Also, I realize the imperative nature of computers more now by seeing the necessity for every language/paradigm I've encountered incorporate some imperative facility.

If I could have done anything differently, I would have tried to spend more time getting to know each language before implementing the project in it.