Project Report: DickGrayson

Sam Gwydir & Martin Fracker & Christopher Findeisen
Rafael Moreno & Kyle Wilson

<2015-05-03 Sun>

Contents

1 Introduction

DickGrayson is a collection of tools that allow a user to encrypt and decrypt messages using the RSA encryption algorithm and embed and extract messages (either plaintext or ciphertext) from BMP images and WAV audio files.

2 Tools

`munchkincrypt` RSA Encryption

`dorothy` RSA Attacks

`munchkinsteg` Steganography

`toto` Steganography Attacks

3 Agile Programming

For this project we were to make use of the "Agile Software Development" ideology and in particular the "Scrum" method. In short this meant instead of using an ad-hoc organization method (read: none at all), we divided the four-week development time allocated into four "sprints". In addition we had five meetings, one every weekday at 15:00. On Mondays a planning meeting was held in which we decided on what tasks would define this weeks' sprint. In addition we estimated the difficulty of each card to avoid under or over estimating the amount of work one could accomplish in a week.

The canonical way to organize and store organization information for a project is a white board but because we did not have access to a physical white-board we opted to use trello.com instead.

3.1 Problems Encountered

4 Test-Driven Development

Test-Driven Development is development strategy where one writes out the tests for a segment of code, usually a single class or header, *before* one writes the code. The tests should initially all fail and then functionality should be implemented such that the tests pass. This ensures that one does not unconsciously write their tests to pass even if their code is broken.

Our team also made use of travis-io.com for continuous integration. Travis builds each push to the git repository for all branches and pull requests. Travis then signals whether or not the code built and passed the tests. After it has built and run all the tests, Travis can tell us which tests passed or failed.

In addition, we configured travis to build the code with code-coverage analytics which were then analyzed by coveralls.io. This allows us to analyze what parts of the codebase are being executed and how many times. This is particularly useful when checking what percentage of the codebase is actually being tested in our tests. Coverage also gives us some insight into what parts of the code are being "hit" hardest, e.g. a loop is being run 1,000,000 times. This sometimes allowed us to optimize the order of conditions in a loop such that the loop is run fewer times or breaks earlier.

4.1 Problems Encountered

5 RSA Encryption

5.1 Problems Encountered

6 Steganography

Our steganography tool, Munchkinsteg, supports two different embedding media: 8-bit Windows BMP (image), and PCM 16-bit WAV (sound). Both types of steganography are based on last significant bit (LSB). Before embedding the message, we append the null byte to it. When extracting, we stop at the null byte and return everything extracted minus the null byte.

6.1 BMP Image Files

Our tool supports only 8-bit Windows BMP which means only one subpixel per pixel. We used the EasyBMP library for interfacing with the BMP format. The LSB of each subpixel (and thus each pixel) contains one sequential bit of the embedded message.

6.2 WAV Audio Files

Our tool supports only PCM 16-bit WAV images. There are other types of WAV images; compression and rounding issues prevent LSB steganography from working correctly for other types of the WAV format. We used libsndfile for interfacing with the WAV format. The PCM 16-bit WAV format consists of an array of 16-bit sound samples. The LSB of each sound sample contains one sequential bit of the embedded message.

6.3 Problems Encountered

Originally we tried writing our own BMP library. We found that it was much easier to write an interface to EasyBMP. When we decided to implement the WAV format, we moved straight to the idea of writing an interface to libsndfile. At first, we didn't realize that only PCM 16-bit WAV would work. Once we did, we had no problem getting support for WAV steganography to work.

7 Division of Labor

7.1 Sam Gwydir

Responsibilities Build Tools, Travis-CI, Coveralls, Report, Presentation, Designated Pair-Programmer

Contribution 20%

7.2 Martin Fracker

Responsibilities Steganography Embedding/Extraction

Contribution 20%

7.3 Christopher Findeisen

Responsibilities RSA Attacks

Contribution 20%

7.4 Rafael Moreno

Responsibilities RSA Encryption/Decryption

Contribution 20%

7.5 Kyle Wilson

Responsibilities Steganography Attacks

Contribution 20%

8 Conclusion

9 Sprint Reports