CSCE315 Final Project Presentation

Sam Gwydir Chris Findeisen Martin Fracker Rafael Moreno Kyle Wilson

<2015-05-10 Sun>

Outline

In the final project students were tasked with creating 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.

DickGrayson

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.

Tools

`munchkincrypt` (aka `rsa-crypt`) RSA Encryption

`dorothy` (aka `rsa-attack`) RSA Attacks

`munchkinsteg` (aka `stego-crypt`) Steganography

`toto` (aka `stego-attack`) Steganography Attacks

Design Decisions

DickGrayson

Target OS	Linux `x86_64` (`build.tamu.edu`)
Compiler	GCC 4.9.2
Language	C++14
Build System	CMake
Numerics	GNU Multiple Precision Library
Unit Testing	Google Test
Continuous Integration	travis-ci
Code Coverage	coveralls

Implementation Decisions

RSA Encryption

- Especially, explain what attacks you chose to make for both RSA and stego, and how your two stego schemes work
- Show proofs of Test-Driven Development (screenshots of failing / passing tests, GitHub revision history, running test code in the demo, etc)

RSA Attacks

- Attacked using one general purpose attack, which would work on any rsa key, then created two more specific attacks that are efficient in some cases.

Implementation Decisions (cont.)

Steganography

- Use null byte to signify end of message
- Use a unified tool for both wav and bmp steganography

Steganography Attacks

- Attack using md5 and extract from Munchkinsteg BMP and WAV
- Attack using LSB scramble for BMP and WAV

Build System & Tools

- Travis
- Coveralls

rsa-crypt

- generate primes
- generate public and private keys
- encrypt and decrypt example messages

rsa-attack

factorization

Trial division method brute forces factorization only trying primes. Will break any RSA key no matter how big!* *given enough time

common modulus

Common modulus takes advantage of a shared n between two separate key/crypttext. Through some clever exponenetiation, we can obtain the original message.

low exponent

Low exponent attack works wherever you have e identical messages with different parties. Clearly, this is most effective and practical when $e = 3$.

stego-crypt

How it works

Our tool munchkinsteg supports image and audio least significant bit (LSB) steganography. The formats supported are Windows 8-bit BMP and PCM 16-bit WAV.

Embedding message

The data is broken up into bits and each bit is stored consecutively in the LSB of the individual pixels and samples of the bmp and wav file respectively. A null byte is stored using 8 additional pixels/samples to be used during extraction as a termination symbol.

`stego-crypt` (cont.)

Extracting message

The LSB of the individual pixels and samples of the bmp and wav file respectively are extracted and concatenated into a string. This process terminates once a null byte is reached.

Conclusion

We've shown RSA and Stegonography, and we've demonstrated attacks on both. We learned the difficulties and fun of encryption, and how exploits often rely on corner cases because general purpose exploits are typically slow.

Discussion

Problems

We had some difficulties using GMP and other helper libraries. As always, research was crucial in our success.

Future work

In the future, we'd like to make our solutions more robust–full base 64 encoding/decoding, writing to .pem/.priv formats, etc.

Links

GCC 4.9.2 https://gcc.gnu.org

GNU Multiple Precision Library https://gmplib.org

Google Test https://code.google.com/p/googletest/

travis-ci https://travis-ci.org

coveralls https://coveralls.io