CSE 543 - Fall 2017 - Project 1: Password Management

1 Dates

- **Out:** *August 26, 2017*
- **Due:** *September 20, 2017*

2 Introduction

In this project, you will complete the implementation of a password management system. Password management systems store domain-password mappings for users to enable users to "remember" complex, per-domain passwords for logging in to those domains. The idea would be that the password management system would store domain-password pairs for domains to which users have created passwords, and then the password management system would replay the password when the domain requests the password for that user.

There are several security concerns that apply to the construction of a password management system. First, the password management system should store strong passwords for each domain. If the user creates a weak password, then an adversary may compromise that user's access to the domain and the password management system would be no help. We will apply password guess estimation techniques to test and strengthen suggested passwords.

Second, the password management system will need to store passwords on the host, e.g., when not running or to back them up, and we must ensure that adversaries do not learn about the passwords or other aspects of the passwords, such as their length. Also, domains for which passwords have been produced should not be revealed to adversaries. We will use the OpenSSL library's cryptographic functions to protect domains and passwords. I have gather example OpenSSL library usage from the Internet (cited in cse543-ssl.c) for performing major cryptographic operations, and you will have to choose when to use the particular functionality. There will be questions about how this example code works to test your understanding.

Third, the password management system must have a method for generating keys for protecting domains and passwords that does not involve hardcoding a key into the program, which can easily be discovered by adversaries (even in binary versions of the program). We will have each user supply a password on the command line when launching the password management system, which will initiate all the key generation activities for protecting domains and passwords.

3 Overview

The password management system you will build will work as follows.

Users can enter the following command line: `cse543-p1 <kvs-file> <master-passwd> <crack-file>` where: (1) the `<kvs-file>` is the name of the file that stores the domain-password key-value store data (may be empty at start); (2) `<master-passwd>` is the value of the master password

for the password management system that will be used on each invocation using the same <kvs-file>; and (3) <crack-file> contains the data from the password cracking analysis used to compute the guess number for each new password and any passwords strengthened. In this configuration, the user (you) will enter domain and password pairs via the command. When the user has entered all the pairs, they enter a EOF (Ctrl-D) to end input and switch to lookup mode. The user then enters domains to retrieve passwords from those entered in the input phase or in previous runs.

There are two additional optional arguments that may be specified: (4) <input-file>, which contains domain and password values (one per line, alternating) for populating the key-value store without manual input and (5) <lookup-file>, which contains the domain names to use for password retrieval (one per line). In this configuration, the key-value store and lookups are performed using the two sets of file inputs, so no manual input is required.

Given this input the program `cse543-p1` will run the following sequence of steps.

Populate the key-value store: The password management system reads the <kvs-file> to load the current key-value pairs for domains and passwords into memory (if any). This operation is provided for you.

Create the cryptographical keys from the master password: The <master-password> is used to create key values for encrypting passwords and producing message authentication codes (using HMAC) for the domains. The key values must be produced from digests of the <master-password>.

Collect new domain-password pairs: Next, the password management system will obtain the domain-password pairs. There are two ways of entering this information, determined by how the program is invoked. If the command line is `cse543-p1 <kvs-file> <master-passwd> <crack-file>`, then the user enters domain-password pairs prompted by the program. Data entry must be terminated by the EOF character (Ctrl-D). If the command line is `cse543-p1 <kvs-file> <master-passwd> <crack-file> <input-file> <lookup-file>` then domain-password pairs are supplied in the input-file.

Strengthen passwords where necessary: Each password will be checked by computing its "guess number" (see below) using the <crack-file> supplied from the command line, and strengthened if the guess number is below a pre-specified threshold. Strengthening will be repeated until the password passes the threshold. **Note that only character may be changed per pass and no characters may be added.**

Create protected key-value pairs for the domain-password pairs: The domain and its strong-enough password will be entered into a key-value store. Your implementation must protect the secrecy and integrity of both the domain and the password using cryptographic operations implemented by using the OpenSSL API. An HMAC of the domain will serve as the key and the password must be encrypted in a manner that protects its secrecy and integrity and length.

Retrieve passwords for domains: Users can then retrieve passwords from the key-value store by specifying "lookup" domains. As for inputs, the same two methods may be used to supply domains, but domains are obtained from the <lookup-file>. Given a domain name, the password management system will retrieve the corresponding password, decrypting the password to present to the user (in lieu of sending to the domain). Please terminate data entry by entering the EOF symbol (Ctrl-D, manually).

Store the key-value pairs in a file: Once the lookups are complete, the password management system writes the current key-value store to the <kvs-file> specified in the command line. This operation is provided for you.

4 Project Tasks

This project has three types of tasks: (1) data entry; (2) password checking and strengthening; and (3) cryptographic protection of domains and passwords. The specific project tasks are listed below and detailed in the rest of this section.

1. **Download project tarball**:

   From `http://www.cse.psu.edu/~trj1/cse543-f17/p1.tgz`. The project includes source code and a file with information for cracking passwords in `rockyou.txt.6.4.a.mc1`.

2. **Make keys from master password**: Given the master password as input, produce separate encryption and HMAC keys for use to protect domains and passwords.

3. **Obtain input domain and password pairs for key-value store**: Obtain the domain-password pairs either from command line or input file, if provided.

4. **Check and strengthen passwords**: Using the provided `<crack-file>`, check the strength of each entered password using the supplied function (`get_markov_guess_number`) and develop a method to strengthen the password to pass the minimum guess number (MIN_GUESS_NUMBER).

5. **Upload password into key-value store**: Use the HMAC of the domain as the key to retrieve an encrypted and integrity protected (authenticated encryption) value of the password and add the key and value to the key value store using `kvs_auth_set`. Ensure protection from leaking both the length and value of the password.

6. **Obtain domain names for retrieving passwords:** Obtain the domain-password pairs either from command line or lookup file, if provided.

7. **Retrieve password using domain name:** Compute the HMAC of the domain name to retrieve the corresponding password from the key-value store. Decrypt and print the password retrieved.

4.1 Download Tarball

The tarball consists of 5 C files and associated header files (for all but `cse543-pwdmgr.c`, a Makefile for compiling the program code and producing tarballs, and a file with information for cracking passwords `rockyou.txt.6.4.a.mc1` which you will use as the `<crack-file>`.

The C file `cse543-pwdmgr.c` is the main file in the project and the file in which all your programming tasks must be completed.

The C file `cse543-ssl.c` contains the code for leveraging the OpenSSL API. I have downloaded most of this code from the internet, and included the URLs of the sites, so you can read the associated text for these operations. There are some questions on using the OpenSSL API below.

The C file `cse543-kvs.c` is a simple key-value store (KVS) for the project. The KVS has a simple API specified in `cse543-kvs.h`.

The C file `cse543-cracker.c` computes the guess number for each password supplied using the `<crack-file>`, `rockyou.txt.6.4.a.mc1` for processing. There is code to call the key cracking function already in the supplied code.

The C file `cse543-util.c` provides a file processing utility function used by the key value store.

4.2 Make Keys from Master Password

In Task #1, given the master password as input from the command line, produce separate encryption and HMAC keys for use to protect domains and passwords. This operation is performed in the function `make_key_from_password` (NOTE: make both keys in this function). Both keys are 256 bits long (ENC_KEY_LEN). Note that the size of the master password is limited to 16 characters (bytes) by MASTER_PASSWD_LEN. The encryption and HMAC keys must be different, and both must be dependent on the master password as the only secret information for constructing key values. Note that the keys should

be produced by generating digests (cryptographic hashes) utilizing the master password as input. The cryptographic protocol used to generate both keys must be specified for Question #1.

4.3 Obtain Input Domain and Password Pairs

In Task #2, obtain the input values for the domain and password pairs to be loaded into the key-value store. Use the supplied file pointer `fp` for the source of this data. If <input-file> is supplied, then the file pointer will refer to this file. If not, the file pointer will refer to `stdin`.

The implementation for obtaining domain-password pairs should be the same either way (i.e., read input from the file pointer for domain and password). Note that each domain must be of the form www.<name>.com, whereas input passwords should be 8 characters or more long.

4.4 Check and Strengthen Passwords

In Task #3, passwords are checked to determine whether they are strong enough based on their guess number estimate, and if they fail to meet a prescribed threshold, they must be strengthened to satisfy that threshold. **Note that each strengthening operation must be limited to only character, and that no new characters may be added to the password.**

The "Guess Again" paper [2] (from class) describes how Markov and PCFG methods may be used to estimate the "guess number" for a password, approximating the strength of the password. See the paper for details.

Researchers have suggested that modifying a few characters in a user-supplied password such that the guess number exceeds a threshold. For example, Houshmand and Aggarwal [1] suggest that changes to only one or two characters in a user-defined password (either by replacement or insertion) may be sufficient to strength the password such that its guess number exceeds a threshold.

The aim in this task is to devise a method using the password and data available in the supplied `<crack-file>` **to use the minimum number of iterations to produce a strengthened password whose guess number surpasses the MIN_GUESS_NUMBER requirement. A project bonus will be awarded to the students to produce minimal number of iterations (approximately 10%).**

4.5 Upload Domain-Password Pairs

In Task #4, you must upload the domain-password pairs into the key-value store in a manner that protects their secrecy and integrity.

The domain is the key in the key-value pair. To protect its secrecy and integrity, you must generate an HMAC value from the domain for the key using the HMAC key computed earlier. An HMAC function is a form of a keyed-hash function that takes the value (domain) and a secrect key (HMAC key) as input. See the course notes on HMAC.

The password is the value in the key-value pair. To protect its secrecy and integrity, you must perform an authenticated encryption on the password. An authentication encryption function takes a symmetric encryption key and a value (password) as input and produces an encrypted value of the password and a message authentication code (MAC), which is called a tag in OpenSSL. You must store the encrypted password and tag as the value. See cse543-kvs.c and cse543-kvs.h as to how this is done already for you when you call `kvs_auth_set`.

An important and non-trivial requirement of this project is to protect the secrecy of the length of the password. Use the entire value (128 bytes) to encrypt the password, crafting your method for encoding the password value for encryption to prevent an adversary from deducing the length of the password from the encrypted value.

4.6 Obtain Domain for Password Retrieval

In Task #5, you must obtain the domain input to retrieve (lookup) its corresponding password. Use the supplied file pointer `fp` for the source of this data. As for Task #2, the file pointer will either reference a specified file (<`lookup-file`>) or stdin. The code you write will be the same either way. Note that each domain must be of the form www.<name>.com.

4.7 Lookup Password for Domain

In Task #6, you must retrieve the password from the key-value store using the domain. To do this, you must once again compute the HMAC of the domain to produce the key of the key-value pair, and then use this key value to retrieve the encrypted password and MAC tag. You must decrypt the password value and extract the original value from whatever encoding you chose to protect its length, so the plaintext password (matching the input password generated for that domain) can be printed by the provided code.

5 Project Questions

1. Using cryptographic notation, specify the cryptographic protocol used to generate the encryption and HMAC keys from the master password.
2. Using cryptographic notation, specify the cryptographic operations used to produce an encrypted password that prevents leakage of both the length and value of the password when written to disk.
3. Specify your password strengthening approach.
4. Given the findings of attacks by Schmidt and Jaeger [3] against methods to strengthen passwords, suggest how you would improve your password strengthening approach to avoid such attacks?
5. Specify how the tag is computed during the authenticated encryption using the OpenSSL API.

6 Deliverables

Please submit the following:

1. A tarball of your password produced with the provided Makefile using `make tar`.
2. Answers to the questions in the Questions section above.

7 Grading

The assignment is worth 100 points total broken down as follows.

1. I can build and run what you have submitted without incident (10 points).
2. Program executes as expected on my test input and lookup files (70 points).
3. Answers to questions (20 points).

In addition, the password strengthening algorithms that take the fewest iterations per my testing will win a 10% bonus.

References

[1] S. Houshmand and S. Aggarwal. Building better passwords using probabilistic techniques. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 109–118. ACM, 2012.

[2] P. Kelley, S. Komanduri, M. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. Cranor, and J. Lopez. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, Oakland '12, pages 523–537. IEEE Computer Society, 2012.

[3] D. Schmidt and T. Jaeger. Pitfalls in the automated strengthening of passwords. In *Proceedings of the 29th Annual Computer Security Applications Conference*, ACSAC '13, pages 129–138. ACM, 2013.