COMP.SEC.300 Secure Programming Project Work: Simple Password Manager

Mikael Penttinen, H275377, mikael.penttinen@tuni.fi

Contents

Program Structure

General Functional Structure

The program consists of an internal manager instance that is manipulated via a set of functions called "services". User can use the services via the command line interface that runs continuously until exit has been requested. Cryptographic and file manipulation operations are separated from the services in the source code, as they use external libraries for certain functionalities. The general structure of the program is depicted in Figure 1: Internal program structure.

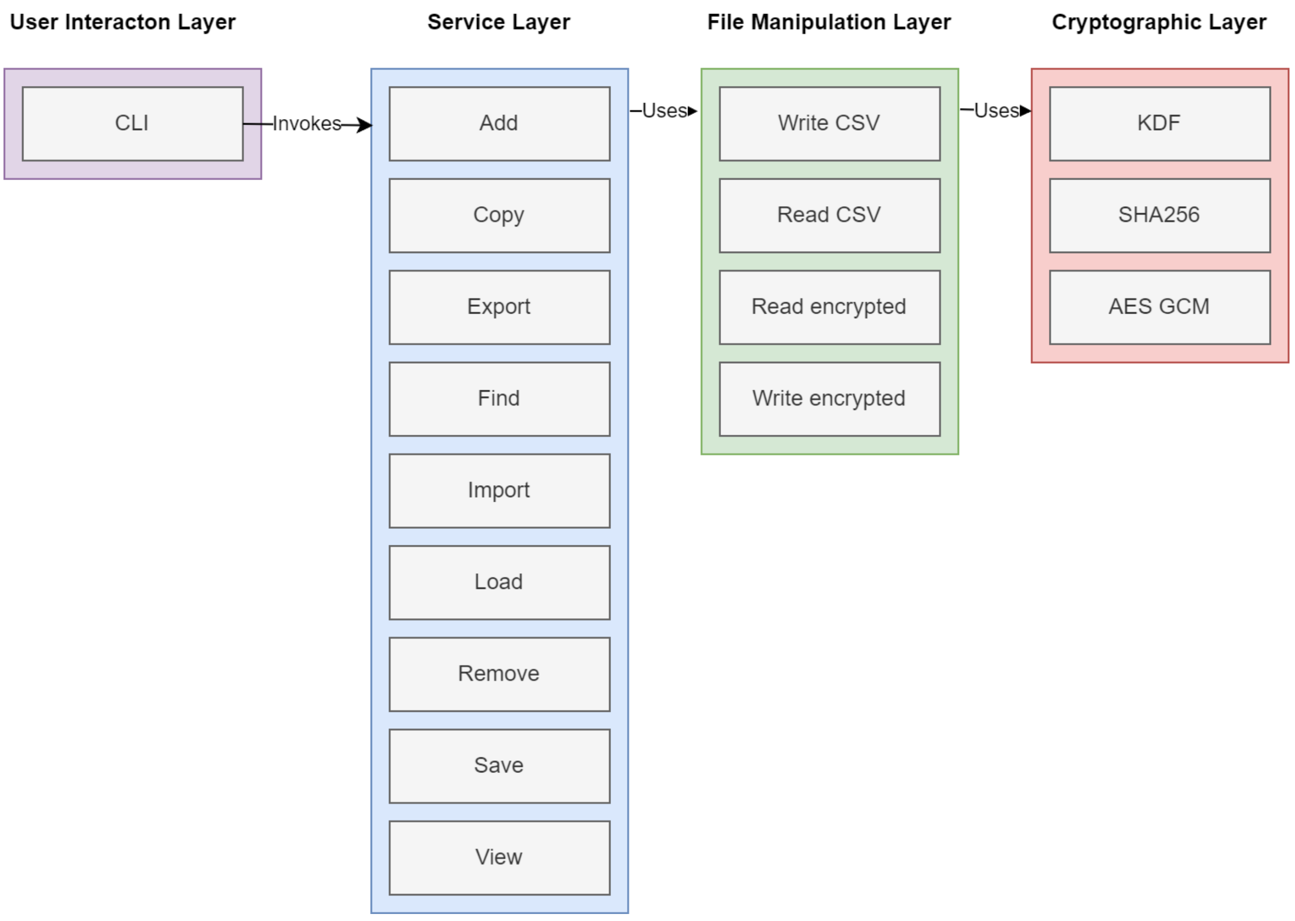

Figure 1: Internal program structure

Third-party Libraries

The manager program uses third-party libraries for certain functionalities.

BOOST Tokenizer

Boost Tokenizer is used for parsing CSV tokens. The library is available from: https://github.com/boostorg/tokenizer

Clip

Clip library by dacap is used for adding content to host clipboard. Clip library is available from: https://github.com/dacap/clip

Google Test

Google Test framework is used for unit testing the project components. Google Test is available from: https://github.com/google/googletest

OpenSSL

OpenSSL is used for implementing cryptographic functions of the project. OpenSSL is available from: https://www.openssl.org/

Implemented Services

Add

Add service allows user to add a login entry into the manager. Add uses a guided input sequence for easy user experience. For the new login entry be accepted, it needs to fulfill the following criteria:

1. URL and username fields cannot both be empty.
2. Password cannot be empty.
3. None of the fields can contain characters outside the supported character list. See Figure 4: Table of allowed characters in login entry data.
4. Matching entry cannot already exist in the manager.

Copy

Copy service allows copying of the password of a login entry to the system clipboard. The copy functionality is implemented using a third-party library and has some known limitations. The requested index must be valid for the copy operation to succeed.

Export

Export service allows the user to export the current manager login content to a new CSV file specified with the export arguments. For the export function to work, it needs to fulfill the following criteria:

1. The program must have permissions to read and write files in the requested directory.
2. The filename must not already exist. Overwriting of existing files is not allowed.

The CSV-format used in the export is the standard escaped CSV described in the chapter CSV.

Find

Find allows the user to search for login entries loaded into the manager. The search algorithm matches the keyword given as an argument to login entry username and URL content. Entries that contain the keyword are outputted by their index in the manager.

Import

Import service allows the user to import login entries from a CSV file. Multiple CSV formats are supported, and the service will automatically detect the order of the keys. For the import service to succeed, the following criteria must be met:

1. The program must have permissions to read files in the requested directory.
2. The file format must be valid CSV.
3. The CSV table must contain the following keys: “url”, “username” and “password”. The order of the keys does not matter.
4. The first line of the file must contain the keys.
5. At least one following line must contain valid CSV. Non valid lines are skipped.

Any login that already exists in the manager will be rejected when importing the login entries.

Load

Load service allows the user to load login entries from an encrypted container file created with the manager previously. Load takes as arguments a filename and a plaintext password. If no filename is specified, the service will attempt to load from a default file. For the load operation to succeed, the following criteria must be met:

1. The program must have permissions to read files in the requested directory.
2. The file must have the correct identification header.
3. The file must have the key salt and AES initialization vector.
4. The file must contain some encrypted data.
5. The password given to the load service must produce the key used to encrypt the container.
6. The SHA256 checksum of the decrypted data must match the checksum written to the end of the encrypted content.
7. The decrypted content must be valid CSV created by the manager.

Any login that already exists in the manager will be rejected when loading the login entries.

Remove

Remove service allows the user to remove a login entry from the manager. The index of the entry must exist in the manager.

Save

Save service allows the user to save the current login entries to an encrypted container file. Save takes as arguments a filename and a plaintext password. If no filename is specified, the service will attempt to save to a default file. For the save operation to succeed, the following criteria must be met:

1. The program must have permissions to read and write files in the requested directory.
2. If the file already exists, it must meet the following criteria:
    a. The file must have the correct identification header.
    b. The file must have been loaded into the manager previously. If container has not been loaded previously, the user will be prompted.

View

View allows the user to view the login entries loaded into the manager. The view supports three modes:

1. With no arguments the view service lists all login existing login entries.
2. With one argument the view service lists one page of login entries. Each page contains 10 entries, and the argument indicates the page number.
3. With two arguments the view service lists login entries between two indexes. The arguments indicate the start and end index.

When using view service with arguments, all arguments must be numeric and indicate a valid index.

Supported File Structures

CSV

The program uses a third-party library to decode CSV content. Multiple CSV formats are supported, and the order of the columns does not matter when parsing CSV documents. The CSV values may or may not be escaped. When parsing CSV input the following limitations are in place:

1. All characters in the value entries must match the allowed characters list.
2. The first line of the CSV file must contain the keys to identify the data in the columns.
3. The following keys must exist: “url”, “username” and “password”. The capitalization of the key strings does not matter. All keys are transformed to lowercase internally.
4. CSV line must contain the columns specified by the keys. The columns may be empty.

The following CSV formats have been tested to work with the program, additional keys are ignored:

Google Chrome output:
name,url,username,password,note

Mozilla Firefox output:
"url","username","password","httpRealm","formActionOrigin","guid","timeCreated","timeLastUsed","timePasswordChanged"

Manager output:
"url","username","password"

The CSV files created by the programs export function are in escaped format with using standard comma as column delimiter. The keys in the exported files are “url”, “username” and “password”.

Example lines from an exported file:

```
"url" "username" "password"
"https://example.com" "johndoe123" "Pa$$w0rd!"
```

Encrypted Files

The encrypted files created by the program have multiple fields to identify and initialize the content. The content of an encrypted file is the following:

1. 32 bytes (256 bits) constant value header used for identifying purposes.
2. 32 bytes (256 bits) salt used to derive a decryption key from a plaintext password.
3. 32 bytes (256 bits) initialization vector for AES GCM encryption.
4. N bytes of encrypted data that contains:
    a. The user specified data. The program uses CSV.
    b. SHA256 checksum of the user data for validating integrity.

The constant identification header is used to indemnify containers created by the program to ensure that non container files are not overwritten. The program allows overwriting containers that are created by the manager. If users attempt to save to existing container, the program will prompt the user, if the container has not been loaded previously. The salt field is used to identify loaded files, the program stores all salts loaded and created during runtime.

Program Usage

At Startup

The program supports entering command line arguments on the program startup. The supported arguments are:

- “--version”: prints out the program version and exits. All other arguments are ignored. Example:
```
PS C:\COMP.SEC.300-SecProg_project\build> .\manager.exe --version
Password manager: development build: Apr 28 2023 18:52:37
Version: 0.1 main-366054a
```

- All other arguments are fed into the programs command line interface as commands. The program starts normally but attempts to execute the command given at startup. Example of using command line arguments to launch command at startup:

```
PS C:\ COMP.SEC.300-SecProg_project\build> ./manager.exe import
examples/basic.csv
Parsed 5 logins, 5 added, 0 rejected.
manager:
```

During Runtime

The program is used by interacting a simple command line interface into which user inputs the commands. All commands can be written as whole words or by a single character that corresponds to the command. Some commands require arguments that are written after the command word separated by a whitespace characters. The amount whitespace between command tokens does not matter as excess whitespace is ignored. The command will fail if incorrect or incorrect count of arguments is given. The expected format of the input:

```
manager: command arg1 arg2 arg3 …
```

Example command on command line interface:

```
manager: import examples/basic.csv
Parsed 5 logins, 5 added, 0 rejected.
```

The command results and errors are printed into the console output after input is entered. All characters given as inputs to the CLI must be valid or the input will be rejected. See Figure 3: Table of allowed characters in user input.

List of Supported Commands

The supported commands of the program are listed in Figure 2: Table of supported commands. The command letter can be used in place of the command word for faster interaction.

Word	Letter	Arguments	Description
add	a	none	Run add service
copy	c	[idx]	Attempt to copy password from login at [idx] to the host system clipboard
exit	x	none	Exit the program
export	e	[filepath]	Attempt to export login data to [filepath]
find	f	[keyword]	Run find service with keyword
help	h	none	Display all supported commands and their arguments
help	h	[cmd]	Display command arguments and description for [cmd]
import	i	[filepath]	Attempt to import login data from CSV file at [filepath]
load	l	[password]	Attempt to load encrypted login data from default file with [password]
load	l	[filepath] [password]	Attempt to load encrypted login data from [filepath] with [password]
remove	r	[idx]	Attempt to remove login entry with [idx]
save	s	[password]	Attempt to save encrypted login data to default file with [password]
save	s	[filepath] [password]	Attempt to save encrypted login data to [filepath] with [password]
view	v	none	Display all login entries
view	v	[page]	Attempt displaying login entry page [page]. 10 login entries per page
view	v	[from] [to]	Attempt displaying login entries beginning with index [from] until index [to]

Figure 2: Table of supported commands

Secure Programming Solutions

Cryptography

To achieve access control and to keep sensitive data secret, encryption algorithm is used for storing passwords. The program uses OpenSSL library to implement an AES GCM cypher for encrypting the container files. The program does not feature a master password or any other type of access control. Rather access to login information is entirely reliant on the user's ability to provide the correct plaintext password from which the decryption key can be derived. While it is most common for a user to have all their login information in one container, it is also possible to store different sets of passwords in different containers and to share only one container and the key. This enables to sharing of some encrypted information without exposing all of user's secret data.

The 256-bit encryption key is derived from a plaintext password using a key derivation function provided by OpenSSL. Every time a container is written a random key salt and AES initialization vector are created. Therefore, even using the same password to overwrite an existing container will result in different encrypted data.

Using a user provided plaintext password to create an encrypted container file consists of the following steps:

1. User plaintext password is read.
2. The login information is encoded into a CSV table and stored in a buffer.
3. A SHA256 checksum is calculated from the encoded CSV data and stored in the end of the buffer.
4. A random 256-bit salt is generated.
5. A 256-bit encryption key is derived from the plaintext password and the generated salt using OpenSSL HKDF algorithm.
6. A random 128-bit AES initialization vector (IV) is generated.
7. Using the 256-bit key and the 128-bit IV for the AES GCM cypher, the login CSV and the SHA256 checksum are encrypted into a buffer.
8. Constant ID header, the generated salt, the generated IV and the AES GCM tag are written into the beginning of the output file.
9. The encrypted data is written to the file.

Using a user provided plaintext password to read an encrypted container consists of the following steps:

1. The constant ID header, the salt and the IV is read from the beginning of the file.
2. A 256-bit decryption key is derived from the plaintext password and the read salt using OpenSSL HKDF algorithm.
3. Using the decryption key, the IV and the read tag, the container data is decrypted into a buffer with AES GCM cypher.
4. The SHA256 checksum is verified from the decrypted CSV data to ensure data integrity.
5. Login information is decoded from the CSV data and loaded into the manager.

Random Data Generation

Cryptographically safe random data needs to be generated for the encryption algorithms to work securely. In this project random data is generated using the OpenSSL Rand library. For the Rand to work correctly, a high entropy seed must be given before random data can be generated. In the event of failing to generate the seed (failure to initialize OpenSSL), the program will generate a runtime error and terminate the service execution. The user will be notified, and they may try again later.

Seed Generation on Linux

When running the Linux build of the program the seed is generated from `/dev/urandom`. This ensures cryptographically secure random data creation.

Seed Generation on Windows

When running the Windows build the seed is generated using `CryptGenRandom` function provided by the Windows platform. For The `CryptGenRandom` function to work correctly a context must be acquired first with CryptAcquireContext.

User Input Validation

User input into the program is validated to contain only allowed characters. If any command or argument contains illegal characters, the whole user input is rejected, and the user is notified. The user may then attempt to enter a new input.

Memory Management

C++ allows manual memory management by the programmer. However manually allocating memory must be freed when no longer used. Manual memory management introduces unnecessary risk and therefore no manual memory allocation is done within this project. STL containers are widely used for storing data within the application.

Runtime Exception Handling

The program is written with multiple levels of exception handling to ensure that the program does not exit unexpectedly. Whenever some resource is allocated, and it must be freed later the exception handling is designed in such a way where resource freeing can always occur. For example, during file operations all possible exceptions after file opening, are caught such that file will always be closed. Similarly, when starting a service function all exceptions are caught and the user is notified in case a runtime error occurs.

Runtime errors are thrown whenever some function fails, or some operation is not possible. The user is notified whenever the program was not able to complete a requested operation.

File Management

The status of file objects is checked before starting any file manipulation operations. If opening a file result in an error, the user is notified, and the service is terminated. The exception handling ensures that the opened file objects are always closed even if a runtime exception occurs.

Additionally, before writing to a file, it is opened in read only mode first to ensure that the program cannot overwrite any data that does not belong to it. If a file operation is not possible, the user is notified, and the service terminated.

Known Issues

Clipboard Functionality

The clipboard functionality of the program has been implemented with a third-party library (See Clip). It is currently known that at least copying to X11 clipboard while running the application in Pop OS does not work. Windows clipboard works correctly. These issues could be solved in the future by writing the clipboard support from scratch or solving the issues with the library.

Partial Export

Partial export (exporting only a part of the loaded login entries) to CSV or encrypted container is not supported. Therefore, currently if user wants to export only a small part of entries, they need to be first loaded separately in a single program runtime. Alternatively, user needs to remove entries that should not be exported.

In the future it is intended that the user could export only a part of login entries. For this to work some sort of selection method needs to be implemented.

Tables

Table of Allowed Characters in User Input

Character	ASCII code	Character	ASCII code
Space	32		
a	97	A	65
b	98	B	66
c	99	C	67
d	100	D	68
e	101	E	69
f	102	F	70
g	103	G	71
h	104	H	72
i	105	I	73
j	106	J	74
k	107	K	75
l	108	L	76
m	109	M	77
n	110	N	78
o	111	O	79
p	112	P	80
q	113	Q	81
r	114	R	82
s	115	S	83
t	116	T	84
u	117	U	85
v	118	V	86
w	119	W	87
x	120	X	88
y	121	Y	89
z	122	Z	90
ä	132	Ä	142
ö	148	Ö	753
0	48	(	40
1	49	)	41
2	50	_	95
3	51	-	45
4	52	,	44
5	53	.	46
6	54	/	47
7	55	\	92
8	56	:	58
9	57	;	59

Figure 3: Table of allowed characters in user input

Table of Allowed Characters in Login Enty

Character	ASCII code	Character	ASCII code
Space	32		
a	97	A	65
b	98	B	66
c	99	C	67
d	100	D	68
e	101	E	69
f	102	F	70
g	103	G	71
h	104	H	72
i	105	I	73
j	106	J	74
k	107	K	75
l	108	L	76
m	109	M	77
n	110	N	78
o	111	O	79
p	112	P	80
q	113	Q	81
r	114	R	82
s	115	S	83
t	116	T	84
u	117	U	85
v	118	V	86
w	119	W	87
x	120	X	88
y	121	Y	89
z	122	Z	90
ä	132	Ä	142
ö	148	Ö	753
0	48	(	40
1	49	)	41
2	50	_	95
3	51	-	45
4	52	,	44
5	53	.	46
6	54	/	47
7	55	\	92
8	56	:	58
9	57	;	59
!	33	^	94
@	64	&	38
#	35	*	42
$	36	+	43
%	37	=	61
[	91	{	123
]	93	}	125
<	60	?	63
>	62	~	126

Figure 4: Table of allowed characters in login entry data