pman

Architecture and design description

09.04.2021

Contents

1 Introduction

This documents serves as an introduction to the architecture and design of pman, which is a command-line based password manager for Linux platforms. pman will be written in C and it revolves around managing a file-based password database that the user can manage with different commands.

Before the introduction of the architecture, the build system of pman is shortly described. This will include the chosen build system, build tools, some relevant compiler options, and the different linters and static analyzers used in the project.

Given the relatively simplicity pman as a program, the architecture is succinct and contains only a few critical architectural views. Despite this, making an architecture design is critical, as then secure design and, most importantly, secure handling of passwords can be emphasized in the construction of pman as early as possible.

The utilized architecture views consist of activity, logical and deployment views. The activity view was decided instead of the usual use-case view due to only real user being the user of the command-line. Logical view was an obvious decision given the security requirements described in the previous paragraph. Finally, the deployment view provides a look in to the different libraries that will be built as a part of the pman project.

At the very end, this document will also go in to the detailed design of pman, which will include detailed interfaces of the different components described in the architecture description including the component's respective interface documentations. Even though C does not support classes, to which UML is quite heavily biased to, class diagrams will be utilized in the interface descriptions.

2 Project structure and build tools

2.1 CMake build system

CMake was an rather obvious choice for a cross-platform build system, as the only real other candidate would have been Make. Modern CMake is well integrated with different C compiler options and tools.

2.1.1 Folder structure

The pman project is split to its architectural components using concise folder structure. Each subfolder/library will contain their own *CMakeLists.txt* that specifies how the possible source files in the folder are built. The folders of the project with their purposes are listed in table 1.

Folder	Purpose
cmake/	CMake scripts and modules
include/	Header files (*.h*)
src/	Source files (*.c*)
tests/	Unit test files

Table 1: Folder structure of pman

2.1.2 Clang and C17

The C compiler for the project was chosen to be Clang due to its superior tool capabilities and integration that is heavily utilized. Clang provides many highly capable static analysis and linting tools, which for a large C project are more than welcome.

The newest complete C standard, C17, was chosen due to all of the useful features provided in C11. pman does not have portability constraints, so choosing the newest C standard poses no clear downsides.

2.1.3 CTest and unit testing

CTest is a unit test driver integrated with CMake, so it was a self-explanatory choice for the project. CTest implementation for the project can be seen in *tests/* subfolder of the project.

The pman project utilizes unit testing heavily, which is why a Unity library is used to extend the capabilities of the unit tests. Unity provides easy-to-use unit testing interface similar to Google Test etc. while also fully supporting C.

2.2 Static analysis and linting

As security is one of the key design principles of pman, the build system integrates several tools to facilitate this principle. Static analysis and linting are used during the building process to eliminate as many bugs and potential security risks as possible, as early as possible.

2.2.1 Compiler warnings

All C compilers have a way of generating warnings from a possibly buggy code, and Clang is no exception. To utilize warnings to their full effect, the `pman` build process enables every single feasible warning provided by the Clang compiler. Additionally, every warning produced by the compiler is transformed in to an error, which forces fixing each and every one of them.

2.2.2 Clang-Tidy linter

`clang-tidy` is a linting tool for C/C++, which provides an extensible framework for diagnosing typical programming errors. These include style violations, interface misuse, and bugs.

`clang-tidy` is integrated in to the build process of `pman`, and all of its warnings are interpreted as errors. Additionally, the project also supports using CLion IDE, which further integrates with `clang-tidy`.

2.2.3 Clang Static Analyzer

Along with the `clang-tidy` mentioned in the previous section, Clang also provides a powerful static analyzer purpose-built for C/C++code. For such a small project as `pman`, the only minor drawback is increase in compilation-time, which is well worth the benefits.

The Clang Static Analyzer is integrates with `clang-tidy`, and therefore, is also ran during the normal build process. The project also implements a script for running the static analyzer separately in *scripts/* folder.

3 Architecture

3.1 Activity view

3.1.1 Common tasks

3.1.2 Main commands

3.2 Logical view

3.3 Deployment view

4 Detailed design

4.1 Component 1

4.2 Component 2

4.3 Component 3

4.4 Encryption library