Lightweight Antivirus Application (LAVA)

Architecture Document

Table of Contents

8. Naming and Coding Standards

1. Introduction

1.1 Purpose

The purpose of this document is to provide insight as to how LAVA will be implemented from an architectural standpoint. This serves to ensure components within LAVA behave as expected and feature improvements as well as bug fixes are facilitated through the use of components that can easily be replaced and maintained as needed.

1.2 Document Overview

This document provides an in-depth description of the architecture of the LAVA application.

1.3 Definitions, acronyms, and abbreviations

Word, Acronym, or Abbreviation	Definition
API	Application Programming Interface
IDE	Integrated Development Environment

1.4 References

- Google's C++ Coding format and standards for development: https://google.github.io/styleguide/cppguide.html

- https://htcrecap.atlassian.net/wiki/spaces/SAD/pages/7176223/Software+Architecture+Document

2. Project Goals

2.1 Scope and Objectives

The intention of this document is to help the development team to determine how the system will be structured at the highest level. It is also made for any interested external viewers to see how the system is meant to be structured and behave.

2.2 Architecture Constraints

LAVA's scanning capability is made with the open-sourced library ClamAV. ClamAV requires a certain database setup (cvd) for the virus signatures. LAVA cannot change this without altering the source code of ClamAV. LAVA has to build its system around this and with this in mind.

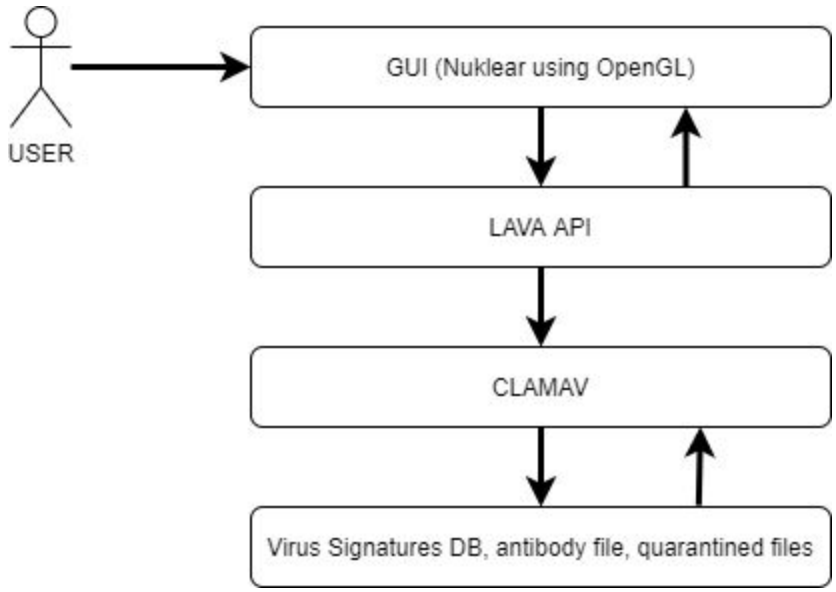
3. Architecture Overview

3.1 High Level Architecture

The ultimate goal of LAVA is to provide a secondary option virus scanner for users. LAVA is a simple C++ program for Windows 10. It will provide a simple GUI for its users. The frontend, nuklear, will be connected to LAVA wrapper for ClamAV. The GUI will utilize OpenGL to draw images on the screen. The GUI will interact with an internal API for LAVA that wraps ClamAV's functionality. This API utilizes ClamAV's

development header file and libraries (clamav.h and libclamav.lib, respectively). ClamAV interacts with its own virus database via a cvd-type file. This file is updated with ClamAV's freshclam.

3.2 Layered Architecture



3.2.1 Presentation Layer

The presentation layer is a GUI built using the nuklear library that will allow the user to interact with LAVA. It is a single-file header library that can easily interact with OpenGL (3 or 2) and multimedia libraries such as SFML (Simple and Fast Multimedia Library). The presentation layer will directly interact with LAVA's business layer (through buttons and interactions in the GUI).

3.2.2 Business Layer

LAVA's business layer wraps around ClamAV to provide the functional requirements to the users. This layer will contain the functions used for the underlying system such as the scans. All of these actions will be initiated by the user's interactions with the presentation layer or through external events that cause the program to deem a scan necessary. Functions like Quick Scans, Complete Scans, etc. are created in this layer.

3.2.3 Data Access

LAVA does not use a traditional database such as SQL or any non-relational one. LAVA's database is a cvd file used for virus signatures, thus LAVA does not have an ORM of any sorts. LAVA uses ClamAV (and more specifically the submodule freshclam) to abstract the process of updating this database. Besides this cvd file, LAVA also manages an antibody file, and this is seperate from ClamAV.

3.2.4 Data Layer

As mentioned before, LAVA's main "database" is the virus signature database that is managed by freshclam. The database is upheld by ClamAV's community. LAVA's job is to ensure the database stays updated.

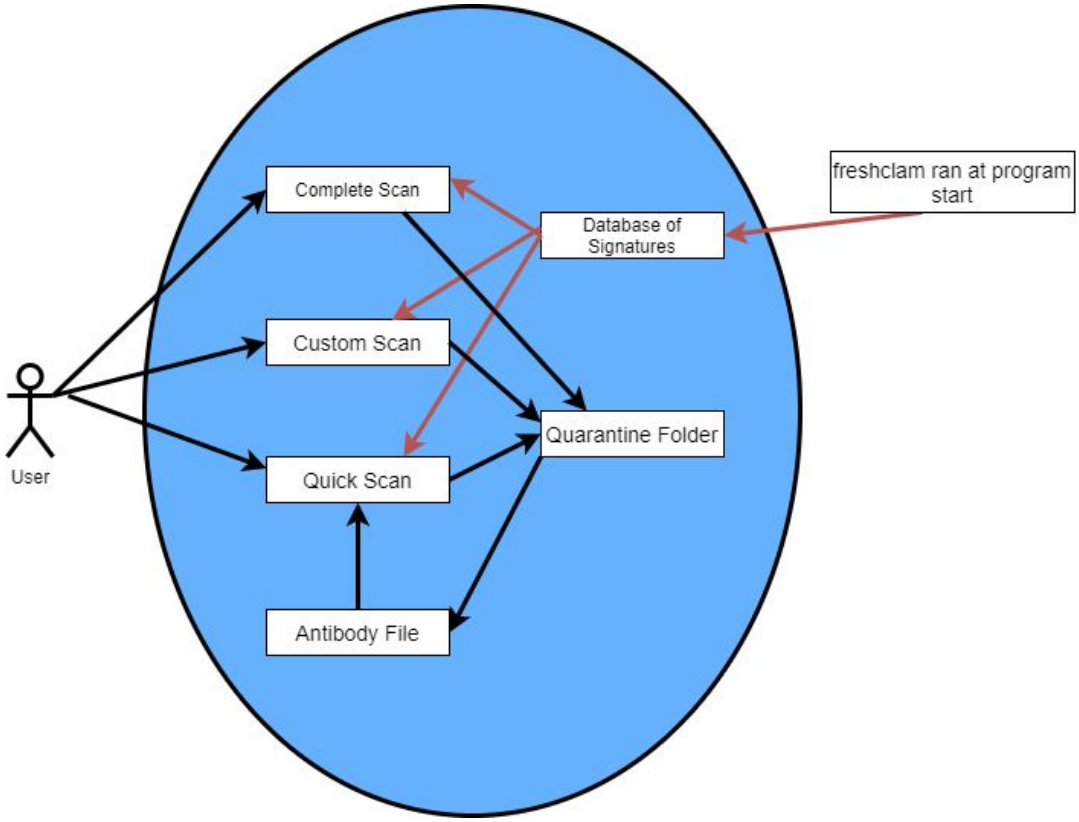
4. Architectural Views

4.1 Business Process Overview

Currently, the users will be able to send files and directories to the scanning engine to check for viruses or other malware. If there are viruses, LAVA will move the folder to a

quaratine folder that the user will then delete the viruses from. In the future users will be able to restore infected files from the quaratine folder.

4.2 Functional Requirements View



Functional Requirements

4.2.1 Quick, Complete, and Custom Scanning

The user will have these three options when it comes to scanning. Complete scans the whole filesystem. Quick scans target commonly infected directories and files while also learning from previous infections by keeping track of directories in which infections have

occured. Custom scans allow the user to indicate which directory and files they want to scan.

4.2.2 Scheduling

The user will be able to schedule daily, weekly, or monthly scans.

4.2.3 Quaratine and Antibody File

When infected files are found, the virus gets moved to the Quaratine folder where it will not be able to do any damage to the operating system. When it is moved the path will be stored in an Antibody file so that future Quick scans can target that directory in the future.

4.2.4 Malware Database Updates

LAVA will update its virus signature database every time a scan is launched.

4.2.5 Working in the Background

Lava will run on boot up unless the user disables that function and will scan external storage devices as well as the C drive.

5. Development Environment

5.1 Implementation View

LAVA will be implemented using the Visual Studio 2019 Community IDE. The application will be written in C/C++ languages in order to keep it lightweight and compatible with most machines. The repository for the application will be hosted on GitHub.

5.2 Data View

The application data, including scan history, the antibody file and the quarantined files, will be stored locally on the users machine. The daily bytecode files will be downloaded from the internet at boot and will also be stored locally.

5.3 Deployment View

LAVA will be installed onto the users machine using an installer that is provided within the Visual Studio IDE. These releases will be available to download on GitHub.

6. Non-Functional Expectations

6.1 Scalability

In terms of scalability, LAVA is expected to perform linearly the same when more files are added to be scanned. To ensure the best results, libclamav.lib will need to be able to scan multi-threaded. Besides that, there are no scalability limitations with the database or backend, because the virus scanner will always need to check every file selected for scanning.

6.2 Usability

LAVA is expected to be a simple application. It should have a basic GUI and the scanning should be available always. The user expects there to be no performance hiccups when performing their scans.

6.3 Availability

LAVA is expected to always be available to scan. Besides the very first run where the initial database needs to be downloaded, LAVA will be able to use previous databases for subsequent scans. The only downtime that can accrue is when ClamAV's database host is not up. If a user has previously ran ClamAV, a previous database version will be used for scans.

6.4 Performance

LAVA will be a lightweight application and thus not consume much memory or CPU. Although LAVA may use a couple threads in the background, they will be doing simple tasks such as checking files for viruses or moving infected files.

6.5 Security

LAVA does not expose any user data nor does it store any. The only files saved are the virus signatures and the infected file list. Thus there are no architectural changes that need to be done to ensure security of user data. Each PC LAVA is installed on is an

isolated working environment. However, as with any program, there exists a risk of an external exploit being discovered. With this in mind, LAVA will be designed to minimize this risk and will be monitored throughout development with this in mind.

7. Design Constraints

7.1 All Impacted Applications/Components

The idea behind a well-made antivirus application is to offer protection with minimal system impact. As such, some applications may be slowed to a minor when LAVA is scanning or updating. Additionally, files detected to be malicious (provided they exist outside of a folder deemed “critical” such as C:/WINDOWS) will be quarantined. Applications that rely on files deemed to be malicious may therefore be affected as well.

7.2 Application Controls

The application will have administrative rights to be able to both schedule scans and quarantine files in the users file system.

7.3 Development Must Comply With Standards

LAVA will be designed in such a way as to ensure communication between group members as well as readability of the code by an external party is maximized. As such, different functions will be built into different files which allow them to be easily replaced as necessary and allow an external party to easily review a specific part of the code.

7.4 Use of Open Source Libraries

LAVA is designed to use the following open source libraries:

- ClamAV
- Dirent
- OpenSSL
- Nuklear

As additional libraries are added to the project, this document will be updated to reflect their inclusion.

7.5 Use of Framework Services

Because LAVA is built using Microsoft's Visual Studio C++, it will be built on the .NET framework and will require Microsoft C++ redistributable.

7.6 Use of Mainstream/Mid-tier Technologies

The Visual Studio IDE is the only mainstream technology that is used in LAVA. The ClamAV engine and OpenSSL are mid-tier technologies that are used within our application.

8. Naming and Coding Standards

LAVA will follow Google's C++ Coding format and standards for development. For more in-depth information on that, see https://google.github.io/styleguide/cppguide.html .