Architecture and Design Overview

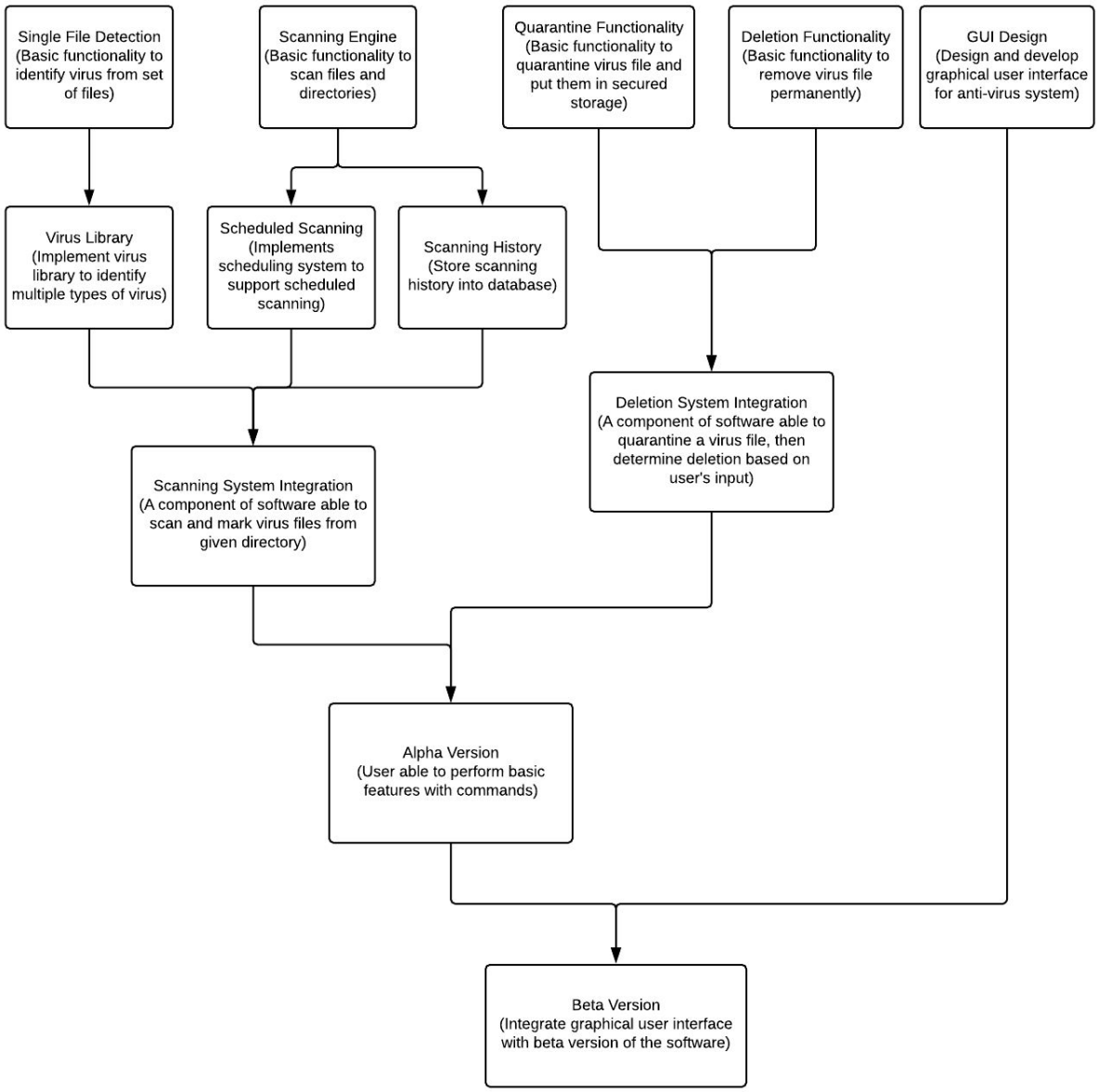
By: Kelly Rose, Kestrel Bridges, Travis Shen,

Tyrone Harmon, Yuliang Xue and Kunal Patel

Project:

Malware scanning application, written in Python and equipped with scanning, detection, quarantine, and deletion features.

Artitecture Diagram:



Detection Functionality:

For detection functionality, we implemented Pyclamd to connect with ClamAV through clamd (ClamAV multi-threaded daemon), and modified the configuration file of ClamAV daemon to fit our requirements. We are able to adapt the detection solution from ClamAV to detect if a file contains malicious code. The hard part for this implementation is to get the dependencies, and separate loading process of clamd so user only have to load the library once when they start the program. When user input a filename as a argument to detect function, the program first check for suspicious string (etc. SQLi), then scan the file with ClamAV detection system to detect if the file matches any signature from the virus library. If a virus file is detected, then the quarantine system is activated to move the file to a closed data storage.

Scanning Engine:

For scanning engine, we are using the scanning methods from Pyclamd library. User can either specify a file location or a directory for scanning. When user select a directory for scanning, we use the constantly scanning method so it will keep scanning recursively until every file under that directory is scanned. The program then output a scanning report for user to review, and store the result with our Scanning History feature.

Scheduled Scanning:

Scheduled Scanning feature is key component from the user interface, where it asks the user what is the frequency that user wants the program to scan the system. The drop down menu contains the following options, every day (24 hours), every other day (48 hours), every week (168 hours), every month (720 hours). Once user selects a frequency of scanning, it will trigger a program clock that reads the system time. It will then start the countdown from the current time based on user selection. For example, if the user selects the daily option option at 9 am on December 1st. The moment the setting is submitted into the program, it will trigger the record of the current operating system time, and it will start the countdown of 24 hours from that time. If the user decides to cancel the scheduled scanning feature while countdown is on going (interrupt), the clock will then stop.

There will two scenarios when the countdown reaches the end. If the machine is turned on, it will then call the scanning function to perform a system scanning. If the machine is off and the countdown is interrupted because of that, the program will record the time when machine powers off, and record the time again when machine turns back on. A calculation of time passed will be performed. If the time has surpassed the countdown overall time, it will pop up a message from the program to notify the user that a scheduled scan is overdue and it will immediately perform a scanning.

If the machine is off for 12 hours more than the scheduled time, it will ask the user if the user wants to keep the scheduled scanning in place and it will pause the scheduled scanning feature until user responds.

Scanning History Process:

After termination of each scanning, the application is required to mark the scan in a scan history record, for the user's convenience. In order to accomplish, we initially started by simply saving the current time and date, and appending that information to a text file within the user's root folder. However, we have improved and optimized this process, by adapting a simple database to store this information. This decision was made because we decided it would make full application integration much easier. Pushing a text file from the users root folder also has its risk. For example, if the user deletes this file, or if this file is corrupted somehow on the user's computer, than all of the user's scan history, findings, and data would be lost.

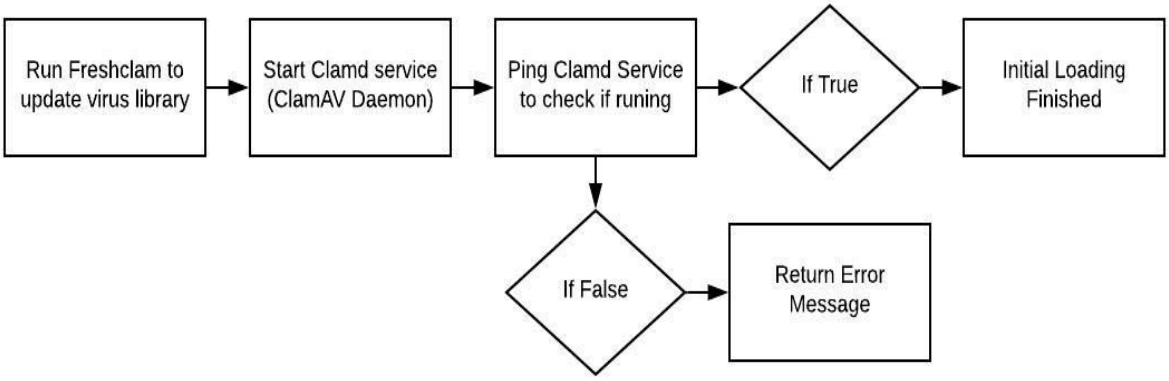
By implementing and adopting the simple database approach, we can assure much strong data persistence, and ascertain the user's information swiftly and more accurately. Additionally, it is a lot easier to store and retrieve different elements of data with a database. With our database, not only can we store the user's scan time and date, but we may as well store how many items where found, which items where of high risk, etc. and this level of additional abstractions makes our application much more scalable in the event that future requirements dictate for additional program features.

More structurally speaking, the database will be implemented utilizing SQL. The intentions are to have user table than contains elements for the time, date, malicious file count, and one for the list of malicious file names for that scanned instance. With these elements in place, our front end user interface can make an easy call to the database to receive the complete scan history, and for further detail about the results of the scans, upon user request.

Virus Library:

For the virus library, we adapted libclamav (ClamAV signature database) with clamd (ClamAV multi-threaded daemon). We first set up configuration for Freshclam (Virus database update tool), and run Freshclam to update ClamAV virus database every time when the program starts. The process of updating virus library is before the loading of ClamAV daemon, so the daemon can be based on the newer version of virus library.

Initial Loading Process:



File Quarantine Process:

To implement the quarantine function, we initially had to find a dynamic solution that could properly identify the host operating system, and discern how to proceed from said information. Due to the use of python as our foundational programming language, arriving to a solution for this problem was very easy. This only required the implementation of the os library generally comes packaged with all python versions. At the initial running instance of our program, the software initiates this check and assigns a variable to the result.

With this input, the program will understand how to proceed and which functions to implement, because functional requirements and implementation are slightly different depending on whether or not the program is running on windows or a unix-based system. This is due to the fact that each operating system aggregates and handles its directories and subdirectories slightly differently. After accomplishing this, we initiate an empty array in order to store the filenames of any corrupt files that may be discovered during the run time of our program.

From here we initiate an additional variable to keep the path of the root directory, and then we enter a loop that will walk all sub directories of this root directory. Within this loop process, the program checks if the file has any behavior similar to what you might find within our anti virus signature library, and if it does, it appends the full and absolute file path into the array that was instantiated at the beginning of the program. After the program has successfully compared all file against our anti virus signature library, the program creates a new folder to have all suspicious files moved to and quarantined. In order to accomplish this, the program checks to see if the quarantine folder currently exists on the system, if so it appends all files to said folder. To accomplish the implementation of this feature, the os.rename function was

utilized in order move the files into this quarantine folder, and then modify their privileges in order to insure that these files aren't being utilized in any malicious way.

Deletion:

For deletion, once we have the files quarantined, the user then is prompted with the option of removing the files from their device. If they decide to delete the files from their device, these files will then be placed into their trash bin, and then wiped completely from the recycling bin. In order to accomplish this program feature, we have decided to be very critical about the exact method for removing the file. Moreover, sometimes removing a file with python's built in os.remove method isn't sufficient enough to complete remove the effects of an infected file on a computer system.

We want to ensure that the file is completely removed from the system, because removing the file will still keep its instance within the user's recycling bin for a set period of time depending on the user's recycling bin settings. From here, there are certain operations that a malicious program, or file could utilize in order to reface the infected file or process onto the user's system. For example, if the malicious file is connected to another file or batch job that checks to see if the file has been moved to the recycling bin and then returns the file to a new subdirectory, this would cause an issue for our users. Thus, we have decided to, in a sense, encapsulate our own variation of the file remove operation within python, and utilize this method to ensure and persist security for our users.

The modified delete method will be actualized as follows: once an affected file has been flagged against our anti virus signature library, and additionally once the user has approve of the removal of this file. We shall remove all instances of this file from the system, and then completely wipe its existence from the user's recycling bin as well. This is more of an issue in windows operating systems, and is not as precedent in unix-like operating systems. This, is where the initial operating system variable comes into play. This design choice, is very reflexive throughout our program, and its utilization will continue within this process. By identifying the operating system, the software has a complete understanding of the necessary function calls.

For the deletion process, we only have two functions calls, one for each operating system. The unix-like environment crafted deletion process is far more simple, due to the design of unix-like systems, and the lack of use of a recycling bin. Thus, the python method for that implementation, will follow the simple, os.remove system call. However, for windows systems, the more complex version of the program described above will be utilized. Post-software completion will require the deleted filenames to be issued to the database, in order to keep a record for the users.

Front-end Development:

For the front-end production, we will be using an open source GUI software called Tkinter, a cross platform service written in Python and Cython. Licensed by MIT, Tkinter runs on Android, iOS, OS X, Linux, and Windows, but for our project we will be utilizing its Windows capabilities. As it says in our Requirements Specification document, we are devoting 4 weeks (28 days) to front end development. Tasks to be completed within this timeframe include

design development, quarantine interface, history interface, and scan interface. We chose Tkinter as our front-end framework because of its accessibility and wide range of widgets, as well as its active developer community, which make us easier to solve any technical problem met. Since Tkinter only support the basic looking of the widgets (ex. Buttons), we decided to adapt customized button through importing images to buttons. At the end of our development cycle, we will merge the front-end and back-end of our application together, and then proceed with more quality assurance and bug testing.

For the project, we will be creating a interface that upon opening prompts the user to peruse their computer and open the directory they would like to scan for viruses and malicious code. Once the scan is complete, system will take the result and store it under logs folder. If there's any virus found during the process, the quarantine system will be triggered to move the virus file into quarantine folder. User should be able to access and perform actions to both logs and quarantined files through our GUI interface.

We first created a controller to control the display of each frame we designed, when a button is pressed, the show_frame function will be triggered to display the new frame to user.

For the logger system to work in GUI, we created event handler to bind the update function of logger to automatically update when the Scanning History frame is being opened through controller. The event handler is also triggered in Quarantine frame so update the list of quarantined files.