Prosty antywirus dla systemu operacyjnego Linux

BSO

Anna Luranc

Politechnika Warszawska, Cyberbezpieczeństwo

30 kwietnia 2021

Spis treści

1. Cel projektu

Celem projektu semestralnego z przedmiotu BSO było stworzenie prostego oprogramowania antywirusowego przeznaczonego dla systemu Linux.

2. Wymagania techniczne

Antywirus ma działać na systemie o architekturze x86_64, na systemie operacyjnym Linux Ubuntu 64 bitowym (wersja systemu: ubuntu-18.04.5-desktop-amd64).

3. Wymagania ogólne

Antywirus ma działać w dwóch trybach uruchomienia na żądanie - dla skanowania pojedynczego pliku, bądź do skanowania wskazanego folderu wraz z jego zawartością (plikami, podkatalogami itp.). Skanowanie odbywa się za pomocą porównywania skrótów plików z wcześniej przygotowaną bazą skrótów plików, które określamy jako niebezpieczne. Plik, który zostanie określony przez skanowanie jako "niebezpieczny", musi zostać przeniesiony do folderu z kwarantanną. Plik po przeniesieniu ma zaszyfrowaną zawartość oraz zmienione uprawnienia dostępu. Po skanowaniu wyświetlane są statystyki skanowania (nazwa skanowanego obiektu, ile plików zostało przeskanowanych, ile wirusów zostało wykrytych, nazwy wykrytych plików). Antywirus umożliwia również zobaczenie wszystkich statystyk skanowań wykonanych od początku włączenia programu.

3.1. Wybór funkcji hashowania

Aplikacja operuje na hashach i bazie hashy stworzonych na podstawie algorytmu MD5. Algorytm generuje 128 bitowy skrót pliku, na podstawie jego wersji binarnej. Implementacja algorytmu została przeprowadzona za pomocą biblioteki OpenSSL, biblioteki kryptograficznej dla języka C++. Generowany przez algorytm MD5 hash jest relatywnie krótki (32 znaki w formacie heksadecymalnym), co sprawia, że algorytm działa bardzo szybko, jednakże przez to algorytm nie jest odporny na kolizje. Zdarza się, że dla dwóch różnych plików wygenerowany hash będzie taki sam, co może generować fałszywie dodatnie wyniki skanowania. Ponadto ten algorytm hashowania nie jest kryptograficznie bezpieczny - algorytm nie jest odporny na ataki typu brute-force. Jednakże skrót jest potrzebny antywirusowi tylko do porównywania sygnatur, dlatego kwestia bezpieczeństwa jest w tym przypadku sprawą drugorzędną, gdyż zawartość pliku jest dodatkowo szyfrowana algorytmem DES. Kwestią pierwszorzędną jest szybkość działania programu. Ma to szczególne znaczenie przy dużej ilości skanowania plików. Dlatego zdecydowałam się na tę funkcję hashującą.
Dużym, nierozwiązanym jeszcze przeze mnie problemem jest możliwość kolizji. Aby zabezpieczyć się przed przeniesieniem pliku fałszywie wykrytego do kwarantanny (i przez to w pewnym rodzaju jego utraceniem) antywirus podczas skanowania informuje o wykrytym pliku i pyta, czy użytkownik chce przenieść dany plik do kwarantanny, przez co użytkownik ma możliwość własnej weryfikacji wyniku. To rozwiązanie oczywiście nie zabezpiecza w pełni przed pomyłkami, dlatego w następnym etapie rozwoju projektu należałoby dodać sprytniejszy mechanizm zabezpieczenia przed kolizjami np. dodatkowy algorytm hashujący.

3.2. Wybór systemu uprawnień wymaganych do włączenia aplikacji

Aplikacja może zostać uruchomiona przez każdego użytkownika poleceniem **./antivirus**. Decyzja ta wynika z tego, iż każdy użytkownik powinien móc zadbać o bezpieczeństwo systemu na którym pracuje. Aplikacja została zaprojektowana tak, by nie stanowiła zagrożenia dla integralności systemu operacyjnego - program wykrywa, które pliki nie są "typowymi" plikami tylko np. systemowymi i tych nie skanuje. Jedyne zagrożenie wynika z tego, że wybrana funkcja hashująca nie jest idealna i zdarza się, że generuje taki sam hash dla dwóch różnych plików, co oznacza, że niedoświadczona osoba może umieścić w kwarantannie plik, dla którego skanowanie dało wynik fałszywie dodatni. Na ten moment aplikacja nie ma zaimplementowanego mechanizmu przywracania plików z kwarantanny. Jest to funkcjonalność, która zostałaby dodana przy następnym etapie rozwoju projektu.

3.3. Mechanizm kwarantanny

Antywirus po wykryciu niebezpiecznego pliku daje użytkownikowi możliwość przeniesienia pliku do kwarantanny. Kwarantanna jest folderem znajdującym się w folderze projektowym o nazwie **quarantine**. Folder sam w sobie nie ma żadnych ograniczeń dostępu. Program, przenosząc pliki do kwarantanny, nakłada na nie następujące ograniczenia dostępu za pomocą funkcji **chmod**:
— S_IRUSR - uprawnienie "tylko do odczytu" (read only) dla właściciela pliku
— S_IROTH - uprawnienie "tylko do odczytu" (read only) dla użytkowników innych niż właściciel pliku
— S_IRGRP - uprawnienie "tylko do odczytu" (read only) dla grupy pliku (file's group)
Te ograniczenia zostały wprowadzone po to, by kwarantanna była bezpieczna - nikt nie ma dostępu do jakiejkolwiek akcji poza zobaczeniem zawartości pliku, która i tak jest już zaszyfrowana.

4. Struktura programu

Program składa się z trzech plików header (**antivirus.h**, **controller.h** i **dataStructure.h**, dwóch plików source (**antivirus.cpp** i **controller.cpp** i main.cpp. Pliki typu header znajdują się w folderze o nazwie **Headers** a pliki źródłowe w folderze **Source**.

5. dataStructure.h

Jest to plik zawierający strukturę danych, tworzącą statystyki skanowania. Są w niej takie pola jak czy plik został przeskanowany, ilość przeskanowanych plików, nazwy wykrytych plików itp.

6. main.cpp

Jest to plik, w którym zawiera się tylko jedna funkcja - funkcja **main**. Jest to pierwsza funkcja wywoływana przez program, która tworzy obiekty ze statystykami i przekazuje je do funkcji **performMainLoop** - funkcji, która implementuje interaktywny interfejs programu.

7. controller.cpp

Jest to plik, który zawiera funkcje sterujące programem.

7.1. void performMainLoop

Funkcja, która zawiera pętle for, ze switchem, dzięki któremu użytkownik może wybrać interesującą go funkcjonalność programu (skanowanie pliku/folderu/pokazanie statystyk/wyjście z programu).

7.2. void eraseStatistics

To funkcja służąca do wyzerowania statystyk skanowania. Jest wywoływana przed każdym skanowaniem.

7.3. void performShowAllStatistics

Jest to funkcja służąca do pokazania wszystkich statystyk skanowania od momentu włączenia programu.

7.4. void performShowStatistics

Jest to funkcja, która pokazuje wyniki pojedynczego skanowania, w zależności od tego, czy skanowanie było pomyślne, czy nie. Jeśli było, funkcja wypisze ile plików zostało przeskanowanych, ile z nich zostało zakwalifikowane jako wirusy i ile z nich zostało przeniesionych do kwarantanny.

7.5. void performScanning

Funkcja odpowiedzialna za wywoływanie odpowiednich funkcji dotyczących skanowania, w zależności od tego czy plik udało się otworzyć czy nie, oraz od tego, czy skanowany obiekt jest plikiem czy folderem.

7.6. void getFileOrDirectory

Funkcja pomocnicza, która w zależności czy skanownay jest plik czy folder, wypisze odpowiednią prośbę o ścieżkę. Ta funkcja jest wykorzystywana w **performMainLoop** podczas proszenia użytkownika o ścieżkę do obiektu, który chce przeskanować.

7.7. bool checkIfDirectory

Funkcja, która sprawdza, czy podany plik jest folderem. Jeśli tak, zwraca TRUE, jeśli nie zwraca FALSE.

7.8. bool checkIfRegularFile

Funkcja, która sprawdza, czy skanowany plik jest normalnym plikiem a nie np. plikiem systemowym. Zwraca TRUE, jeśli jest normalnym plikiem, zaś FALSE kiedy nie jest.

8. antivirus.cpp

Jest to plik, który zawiera implementacje funkcji używanych podczas skanowania.

8.1. bool compareFileWithDatabase

Jest to funkcja, która przyjmuje hash sprawdzanego pliku, otwiera plik z bazą danych i sprawdza, czy dany hash się w tej bazie znajduje. Jeśli tak, zwraca TRUE, jeśli nie - FALSE. Jeśli baza danych nie zostanie znaleziona, funkcja wypisze komunikat "Unable to open database."

8.2. void fileEncryption

Jest to funkcja, która otwiera plik i linijka po linijce będzie go szyfrowała, za pomocą wywołania funkcji **des_encrypt**, która została opisana w punkcie poniższym (8.3). Od razu jest podawany klucz szyfrowania "key12".

8.3. string des_encrypt

Jest to funkcja szyfrująca za pomocą szyfru DES. Implementacja funkcji została zaczerpnięta ze strony [1]

8.4. string get$_md5hash$

. Jest to funkcja hashująca algorytmem MD5. Implementacja funkcji została zaczerpnięta ze strony [2]

8.5. string getFileNameFromPath

Jest to funkcja pomocnicza, dzięki której można dostać samą nazwę pliku wraz z rozszerzeniem, bez ściezki. Funkcja jest używana np. podczas zmiany nazewnictwa plików w kwarantannie w przypadku gdyby już taki plik w niej istniał.

8.6. string checkNamingInQuarantine

Jest to funkcja, która sprawdza czy nazwa nowododanego pliku do kwarantanny nie pokrywa się z plikami, które już w nim istnieją. Jeśli tak, to nazwa pliku jest modyfikowana, poprzez dodanie "_" i liczby powtórzenia. Dzięki temu każdy plik będzie miał unikalną nazwę. Funkcja zwraca nową ścieżkę pliku (ścieżka do kwarantanny + nowa nazwa pliku).

8.7. void moveFileToQuarantine

Jest to funkcja, która jest odpowiedzialna za przeniesienie pliku do kwaranatanny. Najpierw nazwenictwo pliku jest sprawdzane za pomocą funkcji **checkNamingInQuarantine**, po czym jest szyfrowany, a następnie kopiowany do nowej lokalizacji. Plik z pierwotnej lokalizacji jest następnie usuwany.

8.8. vector¡string¿ getAllFilesRecursive

Jest to funkcja, która dodaje do wektora ścieżki wszystkich plików zawartych we wskazanym katalogu i jego podkatalogach.

8.9. vector¡string¿ getAllFilesFromQuarantine

Jest to funkcja, która zbiera wszystkie pliki, które są już zawarte w folderze z kwarantanną. Jest to potrzebne, w celu sprawdzenia nazewnictwa funkcją **checkNamingInQuarantine**.

8.10. Biblioteki

Do wykonania projektu zostały podlinkowane biblioteki:
— OpenSSL - biblioteka kryptograficzna, która umożliwiła działanie funkcji hashującej MD5 i algorytmowi DES
— experimental/filesystem - biblioteka umożliwiająca operacje na plikach (np. sprawdzenie czy obiekt jest folderem, czy plikiem (**is_directory**), sprawdzenie czy plik nie jest plikiem systemowym (**is_regular_file**) oraz rekursywne iterowanie po folderze i jego zawartości (**recursive_directory_iterator**)

9. Testy

9.1. Poprawne wywołanie skanowania

9.1.1. Pliku

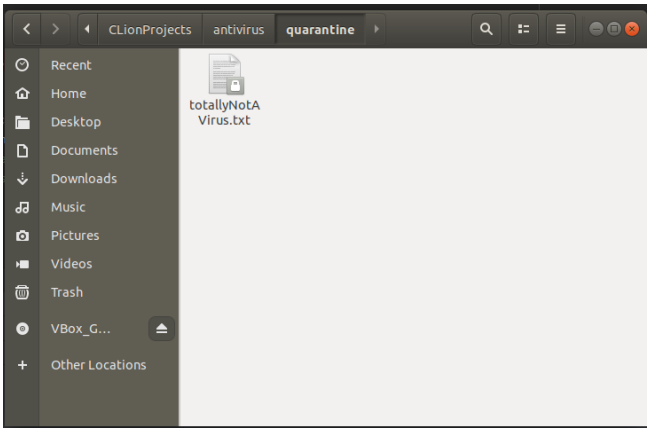
Test polega na wywołaniu pierwszej funkcjonalności antywirusa - skanowania pojedynczego pliku, poprzez podanie ścieżki absolutnej do pliku. Sygnatura skanowanego pliku o nazwie **totallyNotAVirus.txt** znajduje się w bazie sygnatur wirusów.

```
***********************************************
*                   ANTIVIRUS                 *
* Choose operation:                           *
*1. Single file scan                          *
*2. Scan directory                            *
*3. Show all today's scanning statistics      *
*4. Exit                                      *
***********************************************
1
Provide the path of the file:
/home/a/Desktop/projects/folder/totallyNotAVirus.txt
Virus detected
Do you want to move the file to quarantine? Yes/No
No

SCANNING RESULTS
Scanned item: /home/a/Desktop/projects/folder/totallyNotAVirus.txt
Number of scanned files: 1
Number of detected viruses: 1 in the following files:
/home/a/Desktop/projects/folder/totallyNotAVirus.txt

Quarantine applied on 0 out of 1 detected files in total.
Press ENTER To continue
```

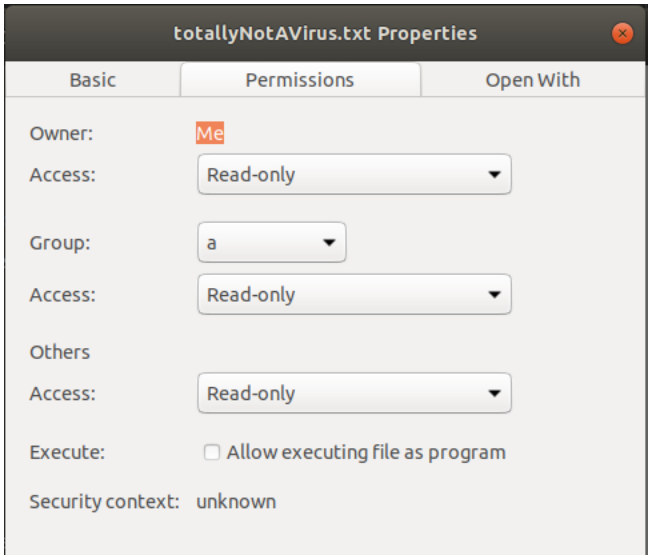
Rys. 1. Poprawne wywołanie skanowania pojedynczego pliku bez przeniesienia do kwarantanny

```
**********************************************
*                 ANTIVIRUS                  *
* Choose operation:                          *
*1. Single file scan                         *
*2. Scan directory                           *
*3. Show all today's scanning statistics     *
*4. Exit                                     *
**********************************************
1
Provide the path of the file:
/home/a/Desktop/projects/folder/totallyNotAVirus.txt
Virus detected
Do you want to move the file to quarantine? Yes/No
Yes
Successfully moved

SCANNING RESULTS
Scanned item: /home/a/Desktop/projects/folder/totallyNotAVirus.txt
Number of scanned files: 1
Number of detected viruses: 1 in the following files:
/home/a/Desktop/projects/folder/totallyNotAVirus.txt

Quarantine applied on 1 out of 1 detected files in total.
Following files has been moved to quarantine:
../quarantine/totallyNotAVirus.txt
Press ENTER To continue
```

Rys. 2. Poprawne wywołanie skanowania pojedynczego pliku z przeniesieniem do kwarantanny



Rys. 3. Zawartość kwarantanny po przeniesieniu wykrytego pliku

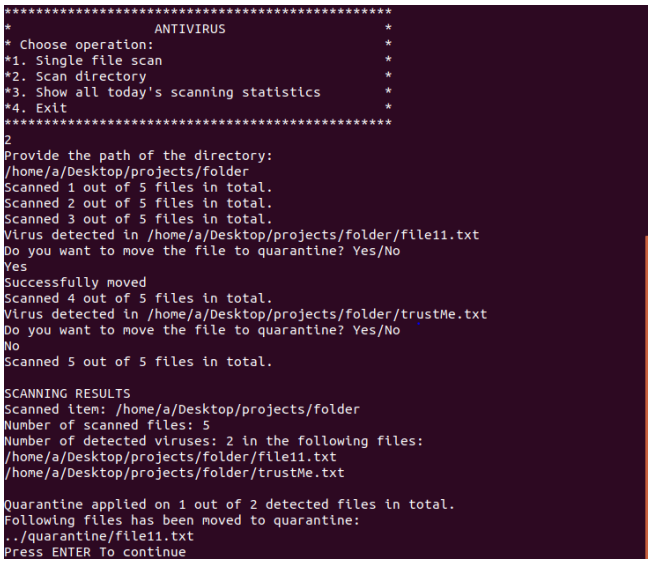


Rys. 4. Uprawnienia przeniesionego pliku

Wyniki testu są zgodne z oczekiwanymi. Plik uznany za niebezpieczny odpowiednio przenosi się do kwarantanny i zmieniane są do niego uprawnienia.

9.2. Folderu

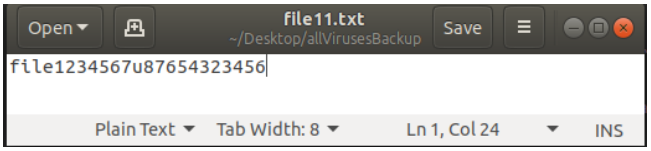
Test polega na wywołaniu drugiej funkcjonalności antywirusa - skanowania folderu, poprzez podanie jego ścieżki. Podczas skanowania pierwszy wykryty plik zostanie przeniesiony do kwarantanny, drugi już nie.



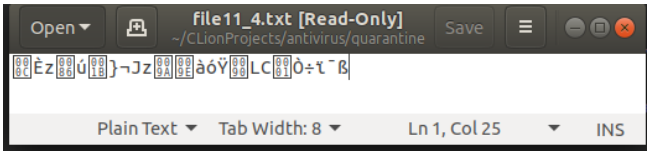
Rys. 5. Poprawne wywołanie skanowania folderu z przeniesieniem pliku **file11.txt** do kwarantanny, a pliku **trustMe.txt** nie.

9.3. Sprawdzenie, czy plik się szyfruje

Test polega na przeniesieniu wykrytego pliku do folderu z kwarantanną i sprawdzeniu czy plik się szyfruje.



Rys. 6. Zawartość pliku przed skanowaniem.



Rys. 7. Zawartość pliku po przeniesieniu i zaszyfrowaniu. Zawartość jest zaszyfrowana.

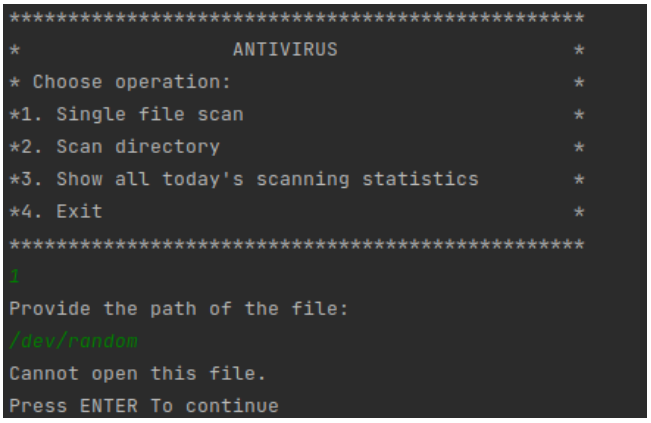
Wynik testu zgadza się z założeniami.

9.4. Wywołanie skanowania pliku/folderu systemowego

Test polegał na próbie skanowania folderu systemowego np. **/dev**.

```
Cannot open this file: /dev/random
Cannot open this file: /dev/full
Cannot open this file: /dev/zero
Cannot open this file: /dev/port
Cannot open this file: /dev/null
Cannot open this file: /dev/mem
Cannot open this file: /dev/rfkill
Cannot open this file: /dev/vga_arbiter

SCANNING RESULTS
Scanned item: /dev
Unable to open this file.
Press ENTER To continue
```

Rys. 8. Kawałek wyniku skanowania dla folderu systemowego. Program odmówi skanowania "Cannot open this file"

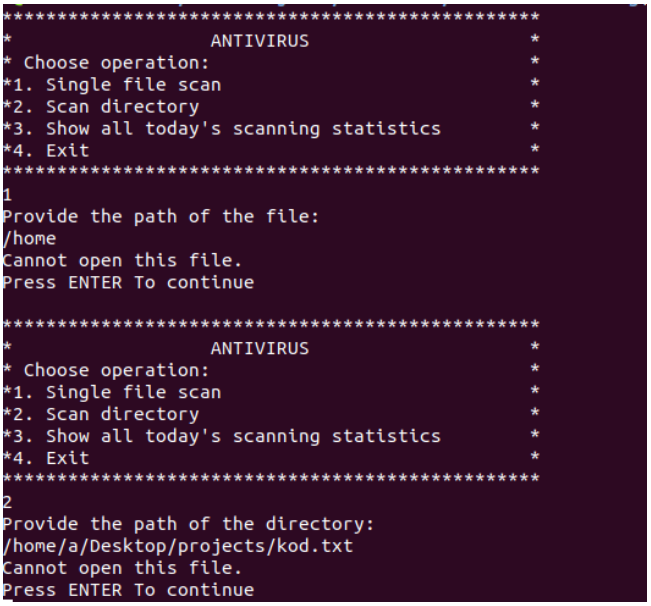


Rys. 9. Kawałek wyniku skanowania dla folderu systemowego. Program odmówi skanowania "Cannot open this file"

Wynik testu jest zgodny z oczekiwanym.

9.5. Niepoprawne wywołanie skanowania pliku/folderu

Test polega na podaniu ścieżki do folderu podczas wywołania skanowania pliku i odwrotnie.



Rys. 10. Wywołanie skanowania pliku i wywołanie folderu. W następnym kroku wywołanie skanowania folderu i wywołanie pliku. W obu przypadkach program zwraca, że nie może otworzyć tego pliku.

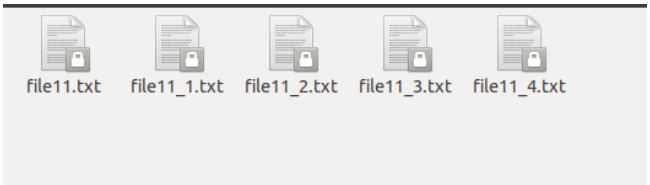
Wynik testu zgadza się z założeniami.

9.6. Umieszczenie w kwarantannie pliku, który już w niej się zawiera

Test polega na umieszczeniu w kwarantannie pliku, który już się w niej zawiera.

```
SCANNING RESULTS
Scanned item: /home/a/Desktop/projects/file11.txt
Number of scanned files: 1
Number of detected viruses: 1 in the following files:
/home/a/Desktop/projects/file11.txt

Quarantine applied on 1 out of 1 detected files in total.
Following files has been moved to quarantine:
/home/a/CLionProjects/antivirus/quarantine/file11_4.txt
Press ENTER To continue
```

Rys. 11. Wynik skanowania w programie. Nazwa pliku nadpisuje się poprzez dodanie _ i licznika.



Rys. 12. Zawartość plików w kwarantannie

Wyniki testów są zgodne z założeniami.

9.7. Skanowanie nieistniejącego pliku

```
**********************************************
*                  ANTIVIRUS                 *
* Choose operation:                          *
*1. Single file scan                         *
*2. Scan directory                           *
*3. Show all today's scanning statistics     *
*4. Exit                                     *
**********************************************
1
Provide the path of the file:
asdfgfds
Cannot open this file.
Press ENTER To continue

**********************************************
*                  ANTIVIRUS                 *
* Choose operation:                          *
*1. Single file scan                         *
*2. Scan directory                           *
*3. Show all today's scanning statistics     *
*4. Exit                                     *
**********************************************
2
Provide the path of the directory:
/adfgtrfs
Cannot open this file.
Press ENTER To continue
```

Rys. 13. Wywołanie skanowania dla nieistniejącego pliku, następnie dla nieistniejącego folderu. W obu przypadkach program zwraca "Cannot open this file"

Wynik testu zgadza się z założeniami.

10. Problemy

Głównymi problemami z jakimi się spotkałam podczas wykonywania projektu była implementacja kryptografii - funkcji hashującej MD5 i algorytmu szyfrującego DES. Mimo użycia biblioteki kryptograficznej OpenSSL, nie udało mi się samodzielnie zaimplementować algorytmów. Wykorzystałam gotowe funkcje z Internetu - algorytm DES [1] i algorytm MD5[2].

11. Rozwój projektu

Następujące ulepszenia mogłyby zostać wprowadzone w dalszych etapach rozwoju projektu:
— możliwość przywrócenia pliku z kwarantanny
— sprytniejszy algorytm hashowania
— możliwość podawania klucza do algorytmu DES na początku działania
— ładowanie bazy hashy do struktury danych (np. vector) na początku programu i operowaniu na niej, zamiast porównywania hashy na otwartym pliku **database.txt** - poprawiłoby to wydajność programu i kosztowałoby mniej pamięci

12. Wnioski

Realizacja projektu umożliwiła mi zapoznanie się z zasadami działania programów antywirusowych. Poznałam algorytm hashujący MD5 i algorytm szyfrujący DES. Ponadto mogłam zapoznać się z językiem C++ i zrozumieć jego podstawy.

Literatura

[1] C++ uses openssl to encrypt and decrypt data. `https://www.programmersought.com/article/33695211858/`.
[2] How can I get the MD5 hash?, 2015. `https://www.quora.com/How-can-I-get-the-MD5-or-SHA-hash-of-a-file-in-C?fbclid=IwAR3n7aMLhVJ8QJp52nmVXMO1-lypfTb7g67KD_0uVaeHKv9fOoDd74_eEOk`.