

Jakub Kałuski,

Politechnika Warszawska, Wydział Informatyki i Technik Informatycznych

Sprawozdanie z projektu BSO - Antywirus dla systemu Linux

30 kwietnia 2021

Spis treści

1. Wstęp	1
1.1. Instrukcja dla użytkownika	2
1.2. Aplikacja natywna	2
2. Dynamiczne linkowanie	2
3. Napotkane problemy	2
3.1. Freeze przy skanowaniu niektórych plików	2
3.2. Skanowanie wszystkich plików w folderze	2
3.3. Dodawanie do kwarantanny plików o tej samej nazwie	2
4. Użyte technologie	3
5. Struktura projektu	4
5.1. Hashes	5
5.2. include	5
5.3. Kwarantanna	5
5.4. src	5
5.4.1. HashCalculator.cpp	5
5.4.2. FileHandler.cpp	5
5.4.3. VirusScanner.cpp	5
5.5. main	5
6. Wybór algorytmu kryptograficznego do porównywania skrótów plików	6
6.1. SHA-512	6
7. Instalacja aplikacji	6
8. Potrzebne uprawnienia aplikacji oraz uruchomienie	6
9. Działanie aplikacji	7
10. Sposób hashowania plików	7
11. Testy	8
11.1. Test1	8
11.2. Skanowanie całego systemu	10
12. Podsumowanie	10

1. Wstęp

Celem projektu było stworzenie prostego antywirusa dla systemu plików zapewniającego różne funkcje podane przez prowadzącego.

Aplikacja zapewnia:

- działanie na systemach Debian 10.7.0-amd64 oraz Ubuntu-18.04.5-desktop-amd64
- Prosty interfejs konsolowy

- Działanie na systemach `x86_64`
- Dokumentacje z omówieniem struktury projektu, konkretnych klas i funkcji
- Skanowanie pojedynczych plików oraz całych katalogów
- Mechanizm kwarantanny

Aplikacja działa w przestrzeni użytkownika. Należy uruchamiać ją z uprawnieniami roota.

1.1. Instrukcja dla użytkownika

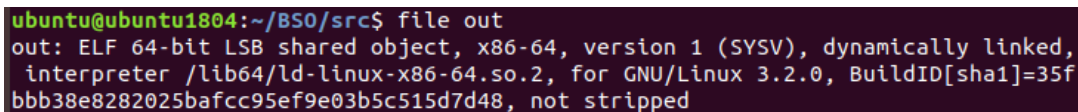
1.2. Aplikacja natywna

Omawiana aplikacja jest aplikacją natywną. Tzn. jest przeznaczona pod konkretną platformę, w naszym przypadku są to wyżej wymienione systemy Debian 10.7.0-amd64 oraz Ubuntu-18.04.5-desktop-amd64. Plusy takiego rozwiązania:

- szybkie działanie
- łatwy dostęp do specyficznego dla danej platformy hardware'u i software'u
- efektywna komunikacja z innymi aplikacjami natywnymi
- duża spójność

2. Dynamiczne linkowanie

W swoim projekcie stosuje dynamiczne linkowanie.



```
ubuntu@ubuntu1804:~/BSO/src$ file out
out: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=35f
bbb38e8282025bafcc95ef9e03b5c515d7d48, not stripped
```

Rys. 1.

Wybrałem je ze względu:

- mniejsze zużycie zasobów
- mniejsze koszty utrzymania aplikacji
- mniejsze koszty wsparcia aplikacji

3. Napotkane problemy

3.1. Freeze przy skanowaniu niektórych plików

Skanowanie niektórych plików np. `//proc/kmsg` freezuje program. Po przeszukaniu rozwiązań znalazłem funkcję zwracającą wartość typu `int`, w zależności od podanego pliku. Normalne pliki (możliwe do przeskanowania) po sprawdzeniu statfs zwracają wartość 61267. Jeśli skanuje się tym te pliki przy których antywirus się crashuje, funkcja ta nie zwraca wartości 61267. Wykorzystałem to porównanie, by sprawdzić, które pliki mogą być skanowane.

3.2. Skanowanie wszystkich plików w folderze

Tutaj z pomocą przyszedł stackoverflow. Znalazłem tam gotową funkcję, która rekurencyjnie przechodzi po plikach w podanym folderze i jego podfolderach.

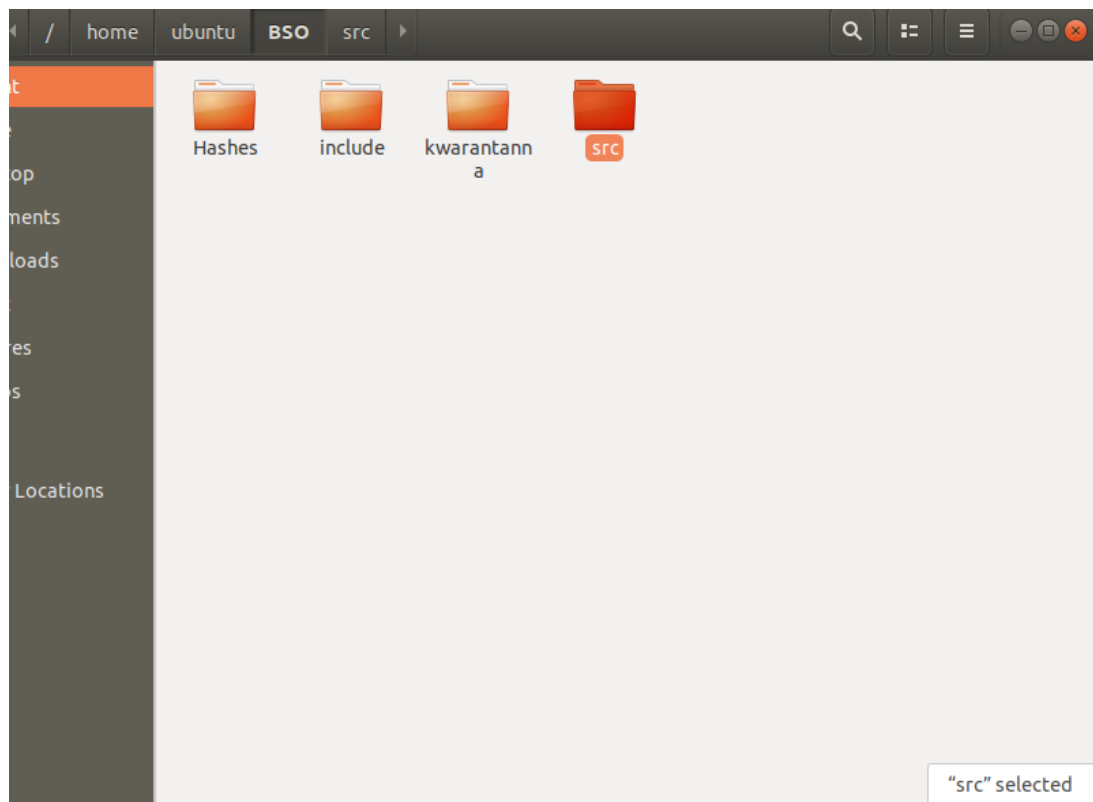
3.3. Dodawanie do kwarantanny plików o tej samej nazwie

Problem został rozwiązany poprzez podawanie liczby na początku nazwy pliku.

4. Użyte technologie

- C++
- G++
- nano
- Clang-Tidy

5. Struktura projektu



Rys. 2.

5.1. Hashes

W tym folderze znajduje się baza danych ze skrótami złośliwych aplikacji. Baza danych jest w postaci pliku .txt.

5.2. include

Tutaj znajdują się pliki nagłówkowe dla kompilowanych klas z katalogu src

- FileHandler.h
- HashCalculator.h
- VirusScanner.h

5.3. Kwarantanna

Folder z uprawnieniami 000, stworzony by przechowywać niebezpieczne pliki.

5.4. src

Folder z plikami .cpp.

5.4.1. HashCalculator.cpp

Metody:

- bool exists(const string &name) - sprawdza, czy dany plik istnieje
- string get_hash(string path) - funkcja obliczająca hash pliku, zwraca w postaci string

5.4.2. FileHandler.cpp

Metody:

- string exec(string command) - funkcja zwracająca rezultat wywołania systemowego w postaci stringa
- string get_filename(string path) - zwraca nazwę pliku w postaci string z pełnej ścieżki
- void create_file() - tworzy plik z hashami (pomocnicza metoda dla twórcy)
- int do_mkdir(const char *path, mode_t mode) - funkcja tworząca katalog o podanej ścieżce i podanych uprawnieniach
- void save_to_db(string hash) - funkcja zapisująca hash do bazy danych
- bool is_file(string path_string) - funkcja sprawdzająca, czy dany obiekt jest plikiem
- int move_to_quarantine2(string path) - funkcja przenosząca plik o danej ścieżce do kwarantanny
- string SplitFilename(string str) - funkcja zwracająca ostatni katalog w którym znajduje się plik

5.4.3. VirusScanner.cpp

Metody:

- int check_file() - metoda sprawdzająca czy dany plik może być skanowany
- void setup_directory(string folder) - funkcja zerująca statystyki przed skanowaniem i wywołująca funkcje directory
- void directory(string folder) - funkcja, która przechodzi rekurencyjnie po wszystkich katalogach
- bool scan_hashes(string path) - funkcja sprawdzająca czy dany plik jest złośliwy

5.5. main

Metody:

- void interface_scan() - interfejs dla skanowania pojedynczego pliku
- void interface_directory() - interfejs dla skanowania folderu
- void interface() - podstawowy interfejs
- int main()

6. Wybór algorytmu kryptograficznego do porównywania skrótów plików

6.1. SHA-512

Jako algorytm do hashowania plików wybrałem SHA-512, ponieważ jest on wystarczająco bezpieczny, odporny na kolizje oraz szybki.

Brałem pod uwagę również SHA-256, ale z moich obserwacji wynika, że jest on wolniejszy od wybranego przeze mnie SHA-512. Z informacji, które zebrałem wynika, że SHA-512 działa prosto szybciej od SHA-256 na systemach 64-bitowych. SHA-256 może być wyjątkowo szybszy, ale dla mniejszych plików Hash MD5 nie jest natomiast odporny na kolizje oraz jest wolniejszy od obu wspomnianych wcześniej hashy. Rozpatrywałem również SHA-1 ze względu na jego szybkość, ale doszedłem do wniosku, że nie jest on bezpieczny. Fragment z wikipedii:

W 2004 zgłoszono udane ataki na funkcje skrótu mające strukturę podobną do SHA-1, co podniosło kwestię długotrwałego bezpieczeństwa SHA-1.

Pomiędzy rokiem 2005 a 2008 opublikowano szereg ataków zarówno na uproszczoną wersję SHA-1, jak i na pełną. Najlepszy z tych ataków wymaga jedynie około 263 operacji funkcji kompresującej (w porównaniu do 280 metodą brute-force).

NIST ogłosił, że do 2010 zaprzestanie stosowania SHA-1 na rzecz różnych wariantów SHA-2

Zobacz wiadomość w serwisie Wikinews pt. Pierwsza kolizja funkcji kryptograficznej SHA-1 dokonana przez Google W 2017 Google wraz z Centrum Wiskunde & Informatica(ang.) ogłosiło, że przeprowadziło praktyczny atak, generując dwa różne pliki PDF o tym samym skrócie SHA-1

7. Instalacja aplikacji

Należy wypakować BSO_Jakub_Kałuski.zip z uprawnieniami roota by wypakowały się również pliki znajdujące się w folderze kwarantanna co sprawi, że w katalogu będziemy mieć cały projekt. Aplikacje uruchamiamy poleceniem `sudo ./(nazwa pliku wykonywalnego)` z katalogu `src`.

8. Potrzebne uprawnienia aplikacji oraz uruchomienie

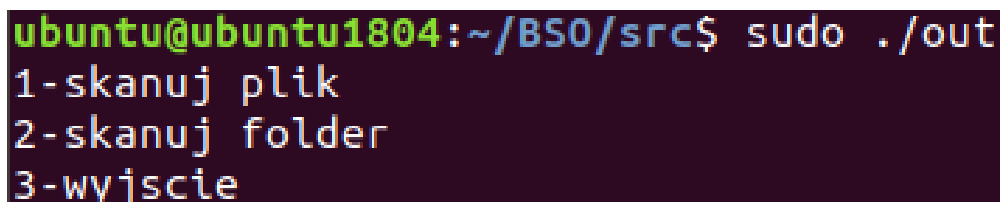
Folder kwarantanna jest tworzony z uprawnieniami 000. Żeby podjąć jakieś akcji z nim związane trzeba mieć więc uprawnienia roota. Konkretnie chodzi tu o przenoszenie plików do kwarantanny. Naszego antywirusa uruchamiamy więc z poleceniem `sudo ./(nazwa pliku wykonywalnego)`. Z analizy źródeł można wywnioskować, że różne popularne antywirusy działają właśnie z uprawnieniami roota.

W folderze BSO znajdują się 2 wirusy. Ich sygnatury są dodane do bazy danych. Po uruchomieniu skanowania folderu BSO program powinien znaleźć 2 wirusy `wirus1.cpp` oraz `wirus2.cpp`.

9. Działanie aplikacji

Aplikacja udostępnia użytkownikowi prosty interfejs konsolowy. Są trzy opcje do wyboru:

- 1-Skanowanie pliku. Użytkownik podaje ścieżkę do danego pliku, który chce przeskanować. W przypadku złej ścieżki wypisywany jest komunikat informacyjny
- 2-Skanowanie folderu. Użytkownik podaje ścieżkę do folderu, który chce przeskanować. Wszystkie pliki w tym folderze oraz w podfolderach skanowane są rekurencyjnie. W przypadku podania złej ścieżki wypisywany jest komunikat informacyjny.
- 3-Wyjście. Wyjście z aplikacji wraz z zwolnieniem zasobów wywołaniem funkcji `exit(3)`.



```
ubuntu@ubuntu1804:~/BS0/src$ sudo ./out
1-skanuj plik
2-skanuj folder
3-wyjście
█
```

Rys. 3.

Do aplikacji dołączona jest baza danych z sygnaturami złośliwych plików w postaci pliku z rozszerzeniem `.txt`. Aplikacja skanuje plik w następujący sposób: Plik, który ma być skanowany jest otwierany, liczony jest z niego hash algorytmem SHA-512, a następnie hash ten porównywany jest z hashami zapisanym w pliku `txt`. Jeśli wykryty plik jest uznany za złośliwy(jego wartość funkcji skrótu zgadza się z jednym z hashów zapisanych w bazie danych), jest on przenoszony do folderu kwarantanna z uprawnieniami `0000`.

10. Sposób hashowania plików

Linux ma wbudowane różne hashe. Są one dostępne pod komendą np. `sha512sum [nazwa pliku]`. W moim programie korzystam właśnie z takiego wywołania systemowego zwracającego hash. Hash zwraca się w postaci stringu i jest przekazywany do dalszej części programu.

11. Testy

11.1. Test1

```
ubuntu@ubuntu1804:~/BSO/src$ sudo ./out
1-skanuj plik
2-skanuj folder
3-wyjscie
2
Podaj sciezke do folderu
../
```

Rys. 4.

Rozmieściłem na w katalogu BSO 4 pliki test.cpp, fibonacci.cpp, skrypt.cpp oraz pascal.cpp, a następnie obliczyłem ich hashe i dodałem do bazy danych. Potem uruchomiłem skanowanie katalogu BSO. Oto rezultat:

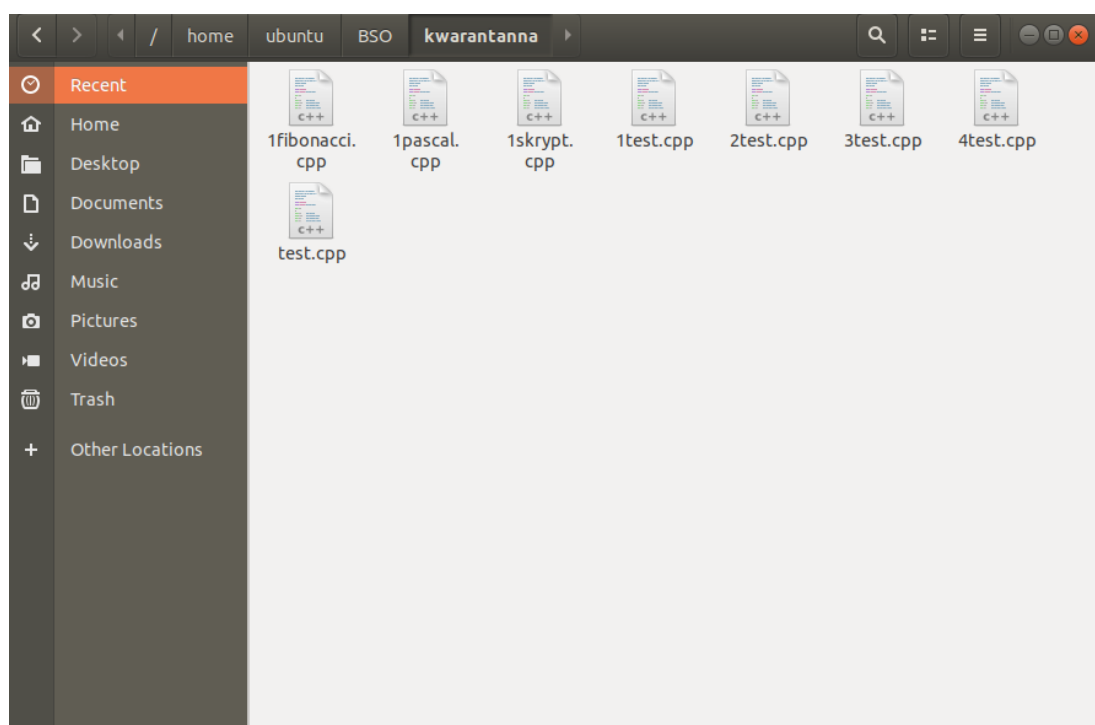
```
wirus
  ../a.out
  ../kwarantanna/test.cpp
  ../kwarantanna/1test.cpp
  ../Hashes/test.cpp
wirus
  ../Hashes/ashes.txt
  ../pascal.cpp
wirus
  ../out
  ../include/test.cpp
wirus
  ../include/FileHandler.h
  ../include/HashCalculator.h
  ../include/VirusScanner.h
  ../.swp
  ../fibonacci.cpp
wirus
  ../skrypt.cpp
wirus
  ../src/test.cpp
wirus
  ../src/FileHandler.cpp
```

Rys. 5.


```
przeskanowane
26
wirusy
7
bezpieczne
19
pliki w kwarantannie
0
1-skanuj plik
2-skanuj folder
3-wyjscie

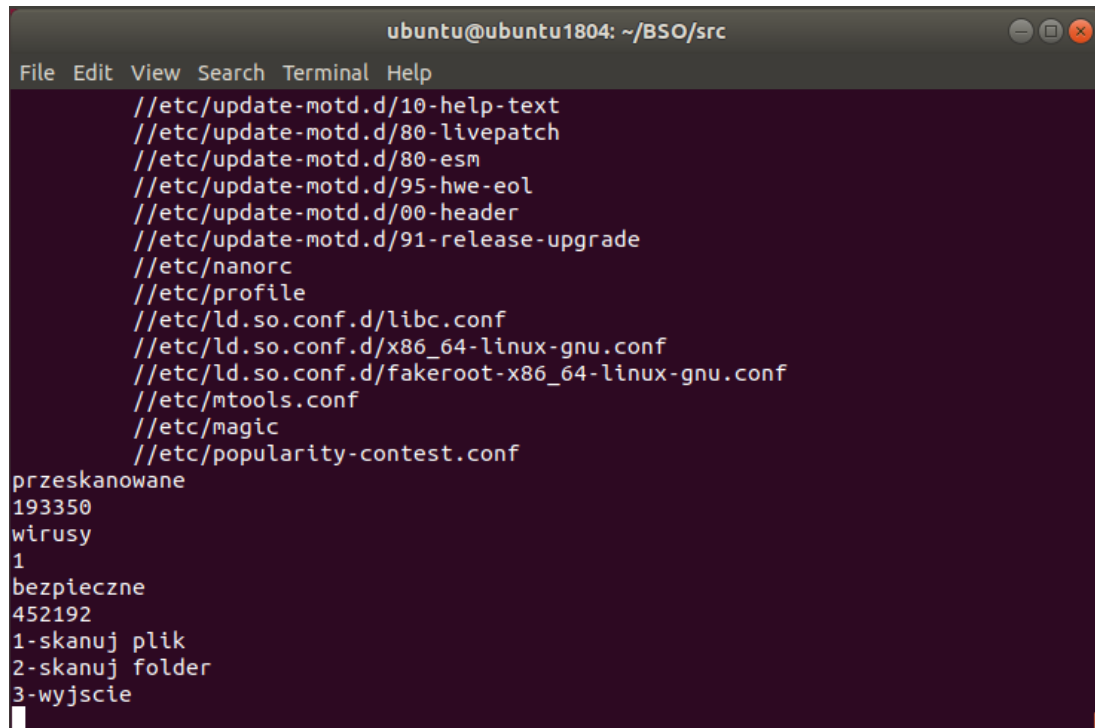
```

Rys. 6. Rezultat skanowania



Rys. 7. Widok w folderze kwarantanna

11.2. Skanowanie całego systemu



```
ubuntu@ubuntu1804: ~/BSO/src
File Edit View Search Terminal Help

//etc/update-motd.d/10-help-text
//etc/update-motd.d/80-livepatch
//etc/update-motd.d/80-esm
//etc/update-motd.d/95-hwe-eol
//etc/update-motd.d/00-header
//etc/update-motd.d/91-release-upgrade
//etc/nanorc
//etc/profile
//etc/ld.so.conf.d/libc.conf
//etc/ld.so.conf.d/x86_64-linux-gnu.conf
//etc/ld.so.conf.d/fakeroot-x86_64-linux-gnu.conf
//etc/mtools.conf
//etc/magic
//etc/popularity-contest.conf
przeskanowane
193350
wirusy
1
bezpieczne
452192
1-skanuj plik
2-skanuj folder
3-wyjście
```

Rys. 8.

12. Podsumowanie

Podsumowując, myślę, że w udało się wykonać założony cel projektu. Oczywiście nie zostało to wykonane w idealny sposób więc, znajdują się rzeczy, które będzie można udoskonalić, np system kwarantanny plików.