Antywirus

Dokumentacja projektu BSO - część podstawowa

Jan Stempkowski

29 kwietnia 2021

Spis treści

1 Cel projektu

Celem projektu jest utworzenie prostego oprogramowania antywirusowego dla systemu Linux. Program powinien umożliwiać skanowanie plików w poszukiwaniu malware'u. Skanowanie polegać ma na generowaniu skrótów (hashy) plików i porównywaniu ich z bazą skrótów. Program spełniać ma także inne przyjęte wymagania, takie jak wyświetlanie statystyk lub zastosowanie mechanizmu kwarantanny.

2 Środowisko pracy

Projekt zdecydowałem się stworzyć w języku *C++* na maszynie wirtualnej z systemem operacyjnym *Ubuntu 20.04*.

W celu usprawnienia pracy postanowiłem wykorzystać IDE *CLion* od *JetBrains*, którego licencję posiadamy w ramach studiów. *CLion* ułatwia między innymi zarządzanie strukturą projektu, jego budowaniem oraz formatowaniem kodu.

CLion posiada wbudowane narzędzie *clang-tidy*, które często pomagało mi w utrzymaniu poprawnego stylu kodu.

Budowanie (*build*) projektu odbywa się za pomocą systemu *CMake*, także wygodnie wbudowanego w IDE.

3 Specyfikacja

Rozpoczynając projekt nie zaplanowałem dokładnie każdego jego elementu. Stosowałem raczej proces iteracyjny. Starałem się budować solidną podstawę dla różnego rodzaju funkcji które będę mógł chcieć dodać później. Ostatecznie lista funkcjonalności aplikacji przedstawia się następująco:

- skanowanie pojedynczego pliku
- skanowanie folderu liniowo (bez rekursji)
- skanowanie folderu rekursywnie
- wczytywanie hashy malware'u z wielu plików tekstowych
- zapisywanie raportu ze skanu do wielu plików tekstowych
- odczytywanie najświeższego raportu z poziomu konsoli
- zatrzymywanie wszystkich aktywnych skanów
- umieszczanie niebezpiecznych plików w kwarantannie
- skanowanie online w postaci zapytań do rejestru malware'u

4 Strona techniczna

4.1 Zależności

Kod programu korzysta z bibliotek:

- `OpenSSL` - hashowanie plików (MD5)
- `Boost/filesystem` - iteracja plików w katalogach, ścieżki
- `Boost/iostreams` - odczytywanie plików do hashowania

Dodatkowo, skanowanie online wymaga zainstalowania *whois*.

4.2 Struktura

Struktura kodu wygląda następująco:

- `src/main.cpp` - *entry point*, obsługuje komendy i wykonuje akcje niezwiązane ze skanowaniem
- `src/scan.cpp` - obsługuje cały zakres funkcji związanych ze skanowaniem
- `src/scan.h` - header dla `scan.cpp`
- `CMakeLists.txt` - ustawienia budowania projektu dla systemu *CMake*

4.3 Pliki

Program przechowuje ważne pliki w katalogu `/avir`. Znajdują się tam:

- `/avir/reports` - katalog ze wszystkimi raportami
- `/avir/quarantine` - katalog stanowiący kwarantannę dla plików
- `/avir/hashlist.txt` - domyślny plik z hashami

W celach testowania do projektu dołączony jest także katalog `scanme`, zawierający kilka plików udających malware, oraz plik `malwarelist.txt` będący listą ich hashy.

4.4 Instalacja

Aby ułatwić testowanie i instalację programu stworzyłem skrypt `installer.sh`, który buduje projekt i instaluje plik wykonywalny w katalogu `usr/local/bin`. Dzięki temu program dostępny jest w konsoli z dowolnego miejsca. Instalacja programu wymaga uprawnień roota.

4.5 Uprawnienia

Początkowo aplikacja miała nie wymagać uprawnień *roota* do funkcjonowania oraz zapisywać swoje pliki w katalogu *home* użytkownika.

Wprowadzenie mechanizmu kwarantanny pokazało jednak, że znacznie czystszym podejściem jest wymaganie uprawnień *roota* do każdego wywołania programu oraz umieszczanie plików, wraz z katalogiem kwarantanny, w przestrzeni na dysku, która jest współdzielona dla wszystkich użytkowników i wymaga uprawnień.

Każde wywołanie programu (także wyświetlanie raportu) wymaga zatem wpisania prefiksu `sudo`.

5 Działanie

Pierwszym krokiem wykonania programu jest wczytanie i analiza podanych argumentów, następnie wykonywane są zależne od tych argumentów funkcje, między innymi skanowanie.

5.1 Akcje

Możliwe są do wykonania trzy podstawowe akcje:

- skanowanie
- wyświetlenie raportu
- zatrzymanie procesów

5.1.1 Skanowanie

Skanowanie może odbyć się na różne sposoby, zależnie od wybranego zasięgu (plik, katalog liniowo, katalog rekursywnie) oraz sposobu skanowania (lokalnie, online). Właściwy proces skanowania odbywa się w tle i pozwala na zamknięcie okna konsoli z której był wywołany. Generalnie skan przebiega w następujący sposób:

- wczytanie argumentów (ścieżka skanu, zasięg, sposób, ścieżki do hashy i raportów
- odczepienie procesu-dziecka - rozpoczęcie pracy w tle
- załadowanie hashów do pamięci (jeśli sposób lokalny)
- utworzenie pliku raportu (lub wielu plików)
- skanowanie plików (lokalnie - jeden po drugim, online - asynchronicznie):
  - kalkulacja skrótu MD5 pliku
  - porównanie skrótu z bazą skrótów lub wysłanie zapytania
  - przeniesienie pliku do kwarantanny jeśli jest niebezpieczny
  - periodyczne zapisywanie dotychczasowych wyników do raportu
- ostateczna modyfikacja raportu

5.1.2 Wyświetlanie raportu

Każdy raport zapisywany jest w katalogu `/avir/reports` (poza innymi lokalizacjami wyznaczonymi przez użytkownika). Każdy plik w tym katalogu posiada nazwę `report_{timestamp}.txt`, gdzie `timestamp` to aktualny *Unix epoch time* w momencie rozpoczęcia skanu. Oznacza to, że najnowszy raport posiada największy `timestamp`.

Wyświetlanie raportu polega na wyszukaniu pliku z największym `timestamp` i wyświetleniu jego zawartości w konsoli.

5.1.3 Zatrzymywanie procesów

Zatrzymywanie skanów polega na wyszukaniu wszystkich `PID` procesów o nazwie `avir`, a następnie wysłaniu do nich sygnałów `SIGTERM`, które są przez nie obsługiwane - powodują zapisanie raportu z dopiskiem `status: terminated` i następnie zakończenie procesu. Wyszukiwanie procesów odbywa się poprzez przeszukiwanie katalogu `/proc` oraz odczytywanie plików `/proc/{pid}/cmdline`.

5.2 Hashowanie

Program do generowania skrótów plików wykorzystuje algorytm MD5. Decyzja wyboru tego algorytmu wynika z następujących czynników:

- jest on popularny, istnieje wiele baz skrótów MD5 malware'u w porównaniu z innymi algorytmami
- jest on prosty do zaimplementowania i jest już zaimplementowany w wielu bibliotekach
- jest on wystarczająco szybki jak na potrzeby programu
- cechuje się wystarczająco małą szansą kolizji

Hashowanie plików w programie odbywa się za pomocą funkcji biblioteki *OpenSSL*.

5.3 Kwarantanna

Mechanizm kwarantanny polega na przenoszeniu plików do specjalnie utworzonego katalogu, którego właścicielem jest *root* i który nie posiada uprawnień do uruchomienia (`x`). Powoduje to, że użytkownik nie może otworzyć katalogu i nie ma dostępu do zawartych w nim plików. Jest to bardzo prosty mechanizm który może zostać rozszerzony na wiele sposobów (więcej w sekcji Wnioski).

Pliki przenoszone są do kwarantanny natychmiast po stwierdzeniu, że są niebezpieczne.

6 Efekt końcowy

Projekt dostępny jest publicznie na GitHub.

6.1 Składnia

Program wywoływany jest z dowolnego miejsca z konsoli, z poniższą składnią. Nie ma określonego limitu opcji.

```
Usage: avir [action] [options]
 Scan actions
   -sf <path>       scan a single file
   -sl <path>       scan a directory linearly
   -sr <path>       scan a directory recursively
 Other actions
   --show           show last scan report
   --stop           stop all ongoing scans
 Scan options
   -h <path>        specify an additional hash list file
   -r <path>        specify an additional output/report file
   -o, --online     check hashes online instead of locally
   -u, --unread     list unreadable files in report
```

6.2 Przykłady wywołania

Poniżej znajdują się przykładowe wywołania programu.

```
user@ubuntu:~$ sudo avir -sr ~/Avir
user@ubuntu:~/Avir$ sudo avir -l ./scanme -o ./report.txt -o ./report_copy.txt
user@ubuntu:~$ sudo avir -sf ~/Avir/scanme/malware1.txt -h ~/Avir/malwarelist.txt
user@ubuntu:~$ sudo avir -sr ~/Avir/scanme --online --unread
user@ubuntu:~$ sudo avir --stop
user@ubuntu:~$ sudo avir --show
```

6.3 Przykład raportu

Poniżej znajduje się przykładowa treść pliku raportu.

```
AVIR SCAN REPORT
 ---
Start time:     Wed Apr 21 13:49:51 2021
Elapsed:        5.72194 seconds
Status:         in progress (54%)
 ---
Scan path:      /home/user
Scan scope:     recursive directory scan
Scan method:    local
 ---
File count:     8948
  Unsafe:       2
  Unreadable:   38
  Safe:         8908
 ---
UNSAFE FILES:
  /home/user/Avir/scanme/malware2.txt
  /home/user/Avir/scanme/malware1.txt
```

7 Problemy

- iteracja po niedostępnych katalogach

  Mechanizm rekursywnej iteracji po plikach rzucał wyjątek gdy próbował zejść do katalogu `quarantine`, który to nie posiadał uprawnień do uruchomienia (`x`). Aby rozwiązać problem musiałem łapać te wyjątki i kontynuować działanie programu. W efekcie niedostępne pliki były omijane.

  Problem ten został rozwiązany wprowadzając wymaganie uprawnień roota, nadal musiałem natomiast wykluczyć katalog `quarantine`.

- pliki nie-hashowalne

  Niektóre pliki nie mogą zostać poddane algorytmowi MD5 i są wtedy uznawane za *"unreadable"*. Dokładne przyczyny takiego zachowania nie są mi znane.

- instalacja

  Początkowo budowa i instalacja programu odbywała się w sposób, który okazał się być dostępny jedynie w *CMake* wersji *13.5*, co powodowało że instalacja programu nie działała na systemie *Debian 10.9*. Wymagało to połączenia procesu budowy i instalacji w jedną komendę `cmake -build [dir] -target install`.

8 Testy

Program został stworzony i w pełni przetestowany na maszynie wirtualnej z systemem *Ubuntu 20.04*.

Przetestowałem także instalację i działanie programu na nowo utworzonej maszynie wirtualnej z systemem *Debian 10.9*. Aby zbudować i zainstalować program należy wykonać następujące kroki:

```
root@debian:~/# apt-get install cmake
root@debian:~/# apt-get install build-essential
root@debian:~/# apt-get install libssl-dev
root@debian:~/# apt-get install boost-filesystem-dev
root@debian:~/# apt-get install boost-iostreams-dev

root@debian:~/# apt-get install git
user@debian:~/# git clone https://github.com/TheJonu/Avir.git
user@debian:~/# ./Avir/installer.sh
```

9 Przemyślenia i wnioski

- kwarantanna

  Zaimplementowany przeze mnie mechanizm kwarantanny jest bardzo prosty i zdecydowanie mógłby zostać ulepszony, na przykład poprzez szyfrowanie plików. Należałoby też stworzyć system, który umożliwiałby odzyskiwanie plików z kwarantanny. Odbywałoby się to dzięki plikowi zawierającemu listę obiektów znajdujących się aktualnie w kwarantannie, wraz z ich oryginalnymi lokalizacjami i sygnaturami.

- asynchroniczność i wielowątkowość

  W aktualnym stanie programu asynchronicznie wykonywane są jedynie skany *online*. Możliwe byłoby wprowadzenie wielowątkowości także do skanowania lokalnego polegającego na odczytywaniu skrótów z pliku. Taki mechanizm wymagałby kontroli liczby współbieżnych wątków, a liczba ta mogłaby być zależna od zasobów systemu.

- statystyki i skanowanie w tle

  Początkowo aplikacja działała w oknie terminala i na bieżąco wypisywała efekty swoich działań na konsolę. Ten system jednak postanowiłem w całości zamienić działaniem aplikacji w tle, zapisywaniem plików raportu i podglądaniem ich. Możliwe byłoby dodanie opcji `-verbose`, która powodowałaby działanie aplikacji jawnie w oknie terminala. Zdałem sobie także sprawę, że działanie w tle i zapisywanie raportu jest całkowicie niepotrzebne w przypadku skanowania pojedynczego pliku.