Bezpieczeństwo Systemów i Oprogramowania - Zajęcia Projektowe Temat 1

Prowadzący: Mateusz Nosek

Kontakt: mateusz.nosek.stud@pw.edu.pl / MS Teams

Semestr: 2022L

Temat projektu: **Prosty Antywirus dla systemu z jądrem Linux**

Proszę w zapoznanie się z dokumentem **"BSO_Projekt_Wprowadzenie.pdf"**.

Opis projektu

W ramach projektu należy stworzyć proste oprogramowanie antywirusowe przeznaczone dla systemu Linux. Oprogramowanie to może działać wyłącznie w trybie użytkownika, wyłącznie w trybie jądra lub składać się z komponentów działających w obu trybach.

Funkcjonalności oprogramowania w wersji podstawowej są jawnie wypisane poniżej. Dla wersji zaawansowanej należy uzgodnić z prowadzącym dodatkowe funkcjonalności.

Projekt należy realizować samodzielnie.

Zakres realizacji projektu

Część podstawowa

> Projekt należy pisać w języku C/C++ z użyciem systemu budowania make/cmake (w przypadku aplikacji działającej w przestrzeni użytkownika) lub języku C i systemu budowania make (dla trybu jądra).

1. Aplikacja natywna dla systemu operacyjnego Linux (statyczna vs dynamiczna linkowalność pozostają w gestii Twórcy). Aplikacja powinna działać na systemie o architekturze x86_64 (zarówno wersja 32 jak i 64 bitowa), nie wymaga się natomiast jej działania na innych architekturach.
2. Mechanizm wykrywania niebezpiecznych plików powinien opierać się na porównaniu ich skrótów z bazą skrótów. Dobór algorytmów dla tego mechanizmu pozostają w gestii Twórcy.
    - Dokumentacja aplikacji musi zawierać uzasadnienie wyboru oraz omówienie zalet i ograniczeń przyjętego podejścia.

> Proszę zwrócić uwagę na wybraną funkcję skrótu pod kątem szybkości reakcji programu. (UWAGA. Proszę uważać przy testowaniu by nie zniszczyć kluczowych aspektów systemu. Nie zaleca się testowania na maszynie fizycznej lecz z użyciem maszyn wirtualnych.)

3. Sposób uruchamiania i wyłączania uwzględniający wygodę użytkownika oraz konieczność *niesiłowego* zwolnienia zasobów.

> Siłowe zabicie procesu przez dostarczenie sygnału nie jest właściwym podejściem.

4. Jeżeli aplikacja jest realizowana w przestrzeni użytkownika (ang. *User Space Mode*) - należy zaplanować system uprawnień wymaganych dla uruchomienia aplikacji.

> Najważniejszą kwestią do rozwiązania jest to czy aplikacja musi być uruchamiana z uprawnieniami roota. (W tym kontekście można póki co pominąć kwestie potencjalnych LSM)

5. Aplikacja powinna wyświetlać statystyki podczas wykonywanych akcji.
6. Aplikacja powinna dostarczać możliwość uruchomienia jej na żądanie w dwóch trybach:
    - na potrzeby skanowania wskazanego pliku (ścieżka lub nazwa w folderze, w którym uruchamiana jest aplikacja),
    - na potrzeby skanowania wszystkich plików we wskazanym katalogu i rekursywnie we wszystkich podkatalogach i na wszystkich poziomach (wskazanie ścieżki do folderu głównego).
7. Aplikacja powinna dostarczyć mechanizm kwarantanny.
    - Kwarantanna powinna być aplikowana na pliku uznanym za niebezpieczny na podstawie sygnatury według mechanizmu zaimplementowanego na podstawie Wymagania 3.
    - Kwarantanna sama w sobie powinna być *bezpieczna* (nic poza uprawnionymi bytami nie powinno móc uzyskać dostępu do plików w kwarantannie).
    - Dokumentacja aplikacji musi zawierać uzasadnienie wyboru oraz omówienie zalet i ograniczeń przyjętego podejścia.
8. Źródła aplikacji powinny zachowywać jakość kodu uwzględniając: styl, nazewnictwo zmiennych i funkcji oraz logiczny podział na pliki i katalogi projektu

> Sugerowane narzędzie: clang-format do automatycznego dbania o styl). Przykład stylu kodu: https://google.github.io/styleguide/cppguide.html

9. W procesie wytwarzania aplikacji należy uwzględnić stosowanie statycznej analizy kodu przez kompilator oraz jedno wybrane narzędzie zewnętrzne (np. clang-tidy dla C/C++).
    - Dokumentacja aplikacji musi zawierać informację o tym procesie.
10. Aplikacja powinna działać w systemach:
    - ubuntu-20.04.3-desktop-amd64 lub debian-11.0.0-amd64
11. Pisząc aplikację należy starać się unikać błędów bezpieczeństwa.

Część zaawansowana

12. Aplikacja powinna działać w tle.
13. Aplikacja powinna wspierać wielowątkowość.
14. Aplikacja powinna dostarczyć jedną inną i dodatkową funkcję bezpieczeństwa - według wyboru Twórcy.
    - Dokumentacja aplikacji musi zawierać uzasadnienie wyboru oraz omówienie zalet i ograniczeń przyjętego podejścia.

> Przykłady: śledzenie połączeń sieciowych lub innych istotnych zdarzeń systemowych, monitorowanie wskazanego procesu pod kątem nietypowych zachowań, ochrona dla wskazanych plików (np. przed zaszyfrowaniem), sprytniejszy mechanizm hashowania (np. pojedynczych sekcji aplikacji).

15. Aplikacja nie powinna obciążać systemu niewspółmiernie do wykonywanej pracy.

16. Aplikacja powinna dostarczyć podstawowy mechanizm aktualizacji bazy sygnatur w trakcie działania.

> Restart aplikacji i ładowanie zaktualizowanej bazy sygnatur nie jest dostatecznym rozwiązaniem.