Bezpieczeństwo Systemów i Oprogramowania

System skanowania urządzeń sieciowych z wykrywaniem potencjalnych zagrożeń oraz raportowaniem email

Rafał Dadura, Juliusz Kuzyka

2024L

Spis treści

1 Wstęp **2**

2 Narzędzia **2**

3 Kontener **2**

4 Koncepcja działania aplikacji **2**

5 Aplikacja **3**

6 Opis funkcjonalny klas **3**
6.1 Klasa Mail 3
6.2 Klasa Scan 3
6.3 Klasa Main 4

7 Schemat funkcjonalny **4**

8 Podsumowanie **4**

9 Bibliografia **4**

1 Wstęp

Naszym celem przy tworzeniu tego narzędzia było zapewnienie prostego w użyciu rozwiązania, które automatycznie generuje raporty bezpieczeństwa dla niezaawansowanych użytkowników. Zależało nam na zapewnieniu intuicyjnego interfejsu, który nie wymaga specjalistycznej wiedzy technicznej, dzięki czemu użytkownicy mogą łatwo korzystać z aplikacji bez konieczności dogłębnego zrozumienia technicznych aspektów bezpieczeństwa informatycznego, czyli wyłącznie otrzymywanie maili. Nasza główna uwaga skupiała się na zapewnieniu wydajności i prostoty działania narzędzia. Chcieliśmy, aby proces generowania raportów był jak najbardziej efektywny i nie obciążał użytkowników skomplikowanymi krokami czy niepotrzebnymi ustawieniami.

2 Narzędzia

Przy tworzeniu naszego projektu korzystaliśmy z narzędzi, dzięki którym udało nam się efektywnie zaimplementować funkcjonalności związane z zarządzaniem pocztą, skanowaniem maszyn, czy generowaniem raportów bezpieczeństwa.

- **smtplib**[1] - biblioteka, która została wykorzystana w naszej aplikacji do wysyłania raportów bezpieczeństwa użytkownikowi poprzez e-mail.
- **python-gvm**[2] - biblioteka Pythona służąca do integracji z OpenVAS (Open Vulnerability Assessment System), umożliwiająca wykonywanie skanów bezpieczeństwa i pobieranie wyników skanowania. Użyliśmy jej do implementacji funkcjonalności związanych ze skanowaniem maszyny wirtualnej i generowaniem raportów bezpieczeństwa na podstawie wyników skanowania.
- **lxml**[4] - użyliśmy w naszej aplikacji do analizy i przetwarzania danych uzyskanych z wyników skanowania maszyny wirtualnej, umożliwiając generowanie czytelnych raportów bezpieczeństwa dla użytkownika.

3 Kontener

W naszej aplikacji wykorzystaliśmy kontener Dockerowy do uruchamiania skanowania bezpieczeństwa przy użyciu obrazu **A Greenbone Vulnerability Management docker image**[3]. Wybraliśmy go ze względu na liczne korzyści:

- **Możliwość uruchomienia pełnego skanera w pojedynczym obrazie z lub bez woluminów** - dzięki obrazowi Immauss mamy możliwość uruchomienia pełnego skanera bezpieczeństwa w jednym kontenerze Dockerowym, bez konieczności zarządzania wieloma obrazami czy woluminami danych. To upraszcza proces konfiguracji i wdrażania aplikacji.
- **Obraz zawiera pełną bazę danych** - obraz zawiera pełną bazę danych, co eliminuje potrzebę osobnego zarządzania i aktualizacji bazy danych. To znacznie ułatwia utrzymanie i aktualizację aplikacji.
- **Szybkość skanowania** - obraz oferuje szybkość skanowania. Jest to kluczowe dla efektywnego przeprowadzania skanowań bezpieczeństwa i minimalizacji czasu przestoju.
- **Regularne aktualizacje** - obraz dostępny na Docker Hub jest regularnie aktualizowany co tydzień, aby zapewnić, że baza danych jest zawsze aktualna. Regularne aktualizacje są kluczowe dla zapewnienia skuteczności skanowania i wykrywania najnowszych zagrożeń.

4 Koncepcja działania aplikacji

Koncepcja projektu opiera się na stworzeniu jak najprostszego w użyciu narzędzia umożliwiającego automatyczne generowanie raportów bezpieczeństwa dla niezaawansowanych użytkowników. Projekt składa się z trzech głównych elementów:

- **Użytkownik** - jest to osoba korzystająca z narzędzia, która otrzymuje raporty bezpieczeństwa. Użytkownik jedynie dostaje raporty skanowania w postaci e-maili.
- **Maszyna główna** - w roli maszyny głównej występuje Greenbone Vulnerability Management docker image z zainstalowanym oprogramowaniem OpenVAS[5]. Jest to centralny punkt, w którym znajduje się główny kod aplikacji odpowiedzialny za generowanie raportów bezpieczeństwa. Maszyna główna zarządza procesem skanowania oraz analizy bezpieczeństwa.
- **Urządzenie sieciowe** - jest to oddzielna maszyna, która jest poddawana testom i skanowaniom w poszukiwaniu potencjalnych zagrożeń bezpieczeństwa. Na tej maszynie wykonywane są skany, a uzyskane wyniki są przekazywane do maszyny głównej w celu wygenerowania raportów bezpieczeństwa.

Narzędzie będzie działać w sposób cykliczny, regularnie skanując urządzenia sieciowe i generując raporty bezpieczeństwa na podstawie wyników skanowania. Dzięki takiemu podejściu narzędzie umożliwi automatyczne monitorowanie bezpieczeństwa systemu i regularne informowanie użytkownika o ewentualnych zagrożeniach.

Rysunek 1: Schematyczna koncepcja projektu

5 Aplikacja

Projekt będzie realizowany przy użyciu języka Python, który jest wyborem optymalnym ze względu na jego wszechstronność, popularność i dostępność bibliotek (takich jak OpenVAS) wspomagających takie zadania jak zarządzanie pocztą, skanowanie maszyn czy tworzenie interfejsu użytkownika. W ramach tego podejścia, stworzone zostaną trzy klasy: **Main** jako klasa główna, **Scan** do skanowania maszyny wirtualnej i generowaniu raportów bezpieczeństwa oraz **Mail** do zarządzania pocztą. Każda z tych klas będzie integralną częścią aplikacji maszyny głównej:

- **Klasa Main** - jest to główna klasa uruchomieniowa aplikacji. Inicjuje ona klasy Mail i Scan, a następnie uruchamia interfejs użytkownika.

- **Klasa Mail (Zarządzanie Pocztą)** - klasa odpowiedzialna za zarządzanie funkcjonalnością związaną z wysyłaniem. Będzie obsługiwać operacje takie jak wysyłanie raportów bezpieczeństwa użytkownikowi oraz przyjmowanie informacji zwrotnych od użytkownika dotyczących preferencji wysyłania i odbierania raportów.

- **Klasa Scan (Skanowanie Maszyny Wirtualnej)** - ta klasa będzie zarządzać procesem skanowania urządzeń sieciowych w celu identyfikacji potencjalnych zagrożeń. Będzie odpowiedzialna za inicjowanie skanowania, analizowanie wyników skanowania oraz generowanie raportów bezpieczeństwa na podstawie tych wyników.

Te klasy będą współpracować ze sobą w ramach maszyny głównej, aby umożliwić użytkownikowi otrzymywanie raportów bezpieczeństwa oraz monitorowanie stanu bezpieczeństwa systemu.

6 Opis funkcjonalny klas

6.1 Klasa Mail

Klasa **Mail** umożliwia wysyłanie wiadomości e-mail z załącznikami za pomocą protokołu SMTP. Wykorzystuje do tego bibliotekę smtplib, która zapewnia interfejs do wysyłania e-maili w języku Python.

1. Ustanawia połączenie z serwerem SMTP, zwykle zabezpieczone protokołem SSL.
2. Autentykuje się na koncie mailowym nadawcy za pomocą podanego hasła.
3. Tworzy wiadomość e-mail z podanymi parametrami, takimi jak temat, treść i odbiorcy.
4. Dołącza ewentualne załączniki do wiadomości.
5. Wysyła wiadomość e-mail do określonych odbiorców.
6. Zamyka połączenie z serwerem SMTP.

6.2 Klasa Scan

Klasa **Scan** umożliwia interakcję z systemem OpenVAS/Greenbone w celu przeprowadzania skanowań bezpieczeństwa oraz zarządzania nimi.

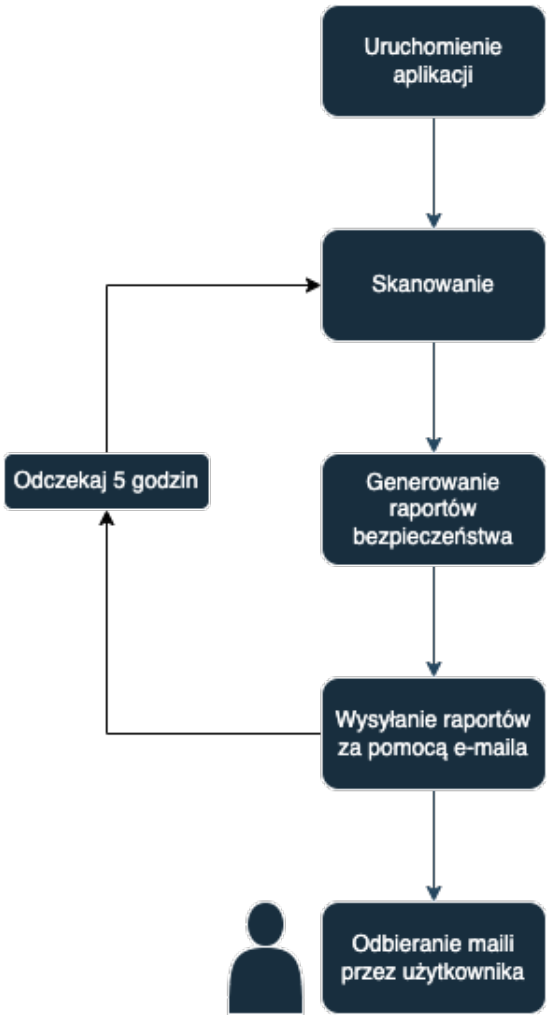
1. Ustanawia połączenie z serwerem OpenVAS/Greenbone - tworzenie połączenia z usługą OpenVAS/Greenbone poprzez protokół GMP.
2. Wykonuje skanowanie - uruchamianie skanowania bezpieczeństwa w systemie OpenVAS/Greenbone.
3. Monitoruje postęp skanowania - sprawdzanie stanu postępu wykonywania skanowania, umożliwiające śledzenie jego statusu.
4. Zapisywanie raportów skanowania - zapisywanie raportów skanowania w odpowiednim formacie, na przykład jako plik PDF, po zakończeniu skanowania.

6.3 Klasa Main

Klasa **Main** pełni rolę koordynatora, łącząc kluczowe elementy aplikacji i inicjując jej działanie.

7 Schemat funkcjonalny

Aby zaprezentować działanie naszej aplikacji w bardziej szczegółowy sposób, przedstawiamy poniżej schemat funkcjonalny, który ilustruje interakcje następujące po sobie w naszej działającej aplikacji:



Rysunek 2: Schemat funkcjonalny projektu

Operator uruchamia aplikację na kontenerze. Cały program aplikacji odbywa się wewnątrz kontenera, gdzie dochodzi do procesu skanowania urządzeń sieciowych. Następnie generowane są raporty bezpieczeństwa, które są wysyłane do użytkownika i odbierane przez niego. Po wysyłaniu wiadomości program odczekuje 5 godzin, aby ponownie rozpocząć skanowanie urządzeń sieciowych.

8 Podsumowanie

Podstawową zasadą projektu było zapewnienie łatwości obsługi, aby nawet użytkownicy niezaawansowani mogli skorzystać z narzędzia bez konieczności posiadania specjalistycznej wiedzy z zakresu bezpieczeństwa informatycznego. Dzięki automatyzacji procesu skanowania i wysyłania raportów, nasza aplikacja pozwala na systematyczne monitorowanie bezpieczeństwa systemu przy minimalnym nakładzie pracy ze strony użytkownika. Zakładamy, że nasze ostateczne rozwiązanie projektu nie będzie odbiegało od koncepcji zaproponowanej w tym artykule.

9 Bibliografia

[1] Python Software Foundation. *smtplib - SMTP protocol client*. URL: `https://docs.python.org/3/library/smtplib.html` (term. wiz. 31. 03. 2024).

[2] Greenbone Networks GmbH. *Greenbone Vulnerability Management Python Library*. URL: `https://greenbone.github.io/python-gvm/` (term. wiz. 31. 03. 2024).

[3] Immauss. *A Greenbone Vulnerability Management docker image*. URL: `https://immauss.github.io/openvas/` (term. wiz. 31. 03. 2024).

[4] lxml developers. *XML and HTML with Python*. URL: `https://lxml.de/` (term. wiz. 31. 03. 2024).

[5] OpenVAS Developers. *OpenVAS - Open Vulnerability Assessment System*. 2024. URL: `https://www.openvas.org/` (term. wiz. 31. 03. 2024).