System skanowania urządzeń sieciowych z wykrywaniem potencjalnych zagrożeń oraz raportowaniem e-mail

Maciej Lipski, Kacper Średnicki

4 kwietnia 2024

Spis treści

1 Założenia projektowe

Ścieżka nowoczesna projektu z przedmiotu Bezpieczeństwo systemów i oprogramowania, realizowanego w semestrze 2024L, dotyczyła przygotowania automatycznego mechanizmu skanowania urządzeń sieciowych. Przygotowane rozwiązanie miało być przeznaczone do użytku przez niezaawansowanego użytkownika. Jego dwa główne zadania to:

- automatyczne skanowanie urządzenia sieciowego w celu wykrycia potencjalnych luk bezpieczeństwa;
- okresowe dostarczanie użytkownikowi raportów obejmujących wyniki przeprowadzonego skanowania.

Na potrzeby nadzoru stanu urządzeń sieciowych zdecydowano się skorzystać ze sprawdzonego narzędzia Greenbone OpenVAS. Raportowanie użytkownika o wynikach skanowania urządzeń odbywać się będzie poprzez wiadomości e-mail, z załączonymi raportami w formacie PDF. Wymienione wyżej komponenty będą natomiast konfigurowane w sposób automatyczny, przez skrypt przygotowany w języku Python. Planowane jest zamknięcie rozwiązania do kontenera wirtualizacyjnego Docker, a następnie opublikowane na koncie DockerHub.

Użytkownik, otrzymawszy kontener Docker/maszynę wirtualną, będzie musiał uruchomić dedykowany skrypt w języku Python, w celu konfiguracji i uruchomienia skanowania. Inną, choć nieco bardziej skomplikowaną dla niezaawansowanego użytkownika opcją uruchomienia mechanizmu, będzie zbudowanie niezbędnych narzędzi za pomocą pliku docker-compose (opisane w sekcji 3.2) zgodnie z dostarczoną instrukcją oraz uruchomienie dedykowanego skryptu w języku Python (opisany w sekcji 3.3). Ostatnim wymaganiem w stosunku do użytkownika będzie umiejętność sprawdzenia adresu IP urządzenia sieciowego, które ma zostać poddane skanowaniu. Adres ten, wraz z adresem e-mail użytkownika oraz wytycznymi dot. skanowania, będzie musiał zostać podany podczas wstępnej konfiguracji procesu.

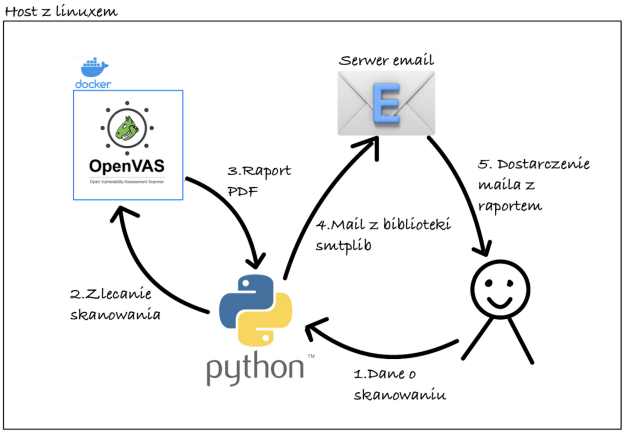
Przygotowane rozwiązanie jest kompatybilne z systemem Linux i może zostać uruchomione wyłącznie na urządzeniach na nim opartych. Wynika to ze sposobu wewnętrznej konfiguracji unix-socket'ów, opisanej w sekcji 3.1.1.

2 Schemat rozwiązania

Na rysunku 1 został przedstawiony schemat przygotowywanego rozwiązania. Widoczne są na nim ikony komponentów tworzących system oraz strzałki symbolizujące sposób komunikacji między nimi.

Użytkownik systemu skanowania przekazuje do skryptu w języku Python dane o skanowaniu, takie jak adres IP skanowanego urządzenia czy częstotliwość ska-

nowania. Przekazuje także swój adres e-mail, na który mają być wysyłane raporty w formacie PDF. Następnie, skrypt dokonuje odpowiednich konfiguracji i zleca menedżerowi GVMD (wyjaśnienie w sekcji 3.1) uruchomienie skanowania skanerem OpenVAS. Potrzebne do skanowania narzędzia są zbudowane jako zestaw kontenerów Docker. Po zakończeniu skanowania, automatycznie stworzony przez skaner raport jest obsługiwany przez wspomniany skrypt w języku Python. Skrypt tworzy wiadomość e-mail z załączonym raportem, która za pośrednictwem serwera e-mail jest dostarczana do użytkownika.



Rysunek 1: Komponenty i sposób ich komunikacji

W dalszej części raportu, w sposób szczegółowy opisane zostały sposoby działania poszczególnych komponentów, konkretne techniki pozwalające na ich wzajemną komunikację oraz szczegóły implementacyjne.

3 Opis modułów

3.1 Greenbone Community Edition

Greenbone Community Edition to zestaw narzędzi (ang. framework) stanowiący podstawę tego projektu. Jego najważniejsze elementy to skaner podatności, menedżer GVM oraz narzędzia pozwalające na interakcję z GVM. Framework udostępniony jest do manualnej instalacji jako zestaw narzędzi w systemie Linux lub zestaw kontenerów Docker, co zastosowano w projekcie (szczegółowy opis w sekcji 3.2).

Najważniejszym skanerem w rozwiązaniu Greenbone Community Edition jest skaner OpenVAS. To on stanowi podstawę framework'u i był jego pierwotną częścią (Greenbone, „Background", marzec 2024, https://greenbone.github.io/docs/latest/background.html). Dodatkowo narzędzie zawiera skaner Notus, używany automatycznie równolegle ze skanerem OpenVAS.

„Sercem" framework'u Greenbone Community Edition jest menedżer Greenbone Vulnerability Management Daemon (GVMD). Odpowiada on za zarządzanie skanerami podatności oraz samymi skanowaniami. Komunikacja ze skanerami następuje przez specjalnie zaprojektowany przez Greenbone protokół OSP (Greenbone). Menedżer GVMD obsługuje także bazę danych przechowującą dane dotyczące skanowań. Ważną częścią Greenbone Community Edition jest tzw. feed, czyli zestaw plików zawierający informację o podatnościach, skanowaniu podatności oraz ustawienia skanowania. Część plików z feed'u dostarczana jest bezpośrednio do skanera, pozostała do GVMD.

Komunikacja użytkownika z menedżerem GVMD następuje poprzez protokół Greenbone Management Protocol (GMP), oparty na języku znaczników XML. Z wykorzystaniem tego protokołu komunikują się z GVMD wszystkie narzędzia dla użytkownika, udostępnione przez Greebone. Jedną z metod komunikacji z GVMD jest narzędzie Greenbone Security Assistant (GSA). GSA to aplikacja przeglądarkowa, napisana z wykorzystaniem framework'u języka JavaScript React. Z jej poziomu użytkownik może m.in. planować skanowania, zapoznawać się z ich wynikami oraz zarządzać wieloma elementami framework'u Greenbone Community Edition.

3.1.1 Biblioteka *python-gvm*

Inną, wykorzystywaną w tym projekcie, metodą komunikacji z menedżerem GVMD jest biblioteka *python-gvm*. Umożliwia ona komunikację z GVMD za pomocą języka programowania Python. Zawarte są w niej metody m.in. do planowania skanowań, ustalania ich harmonogramu oraz pobierania raportów (wykorzystanie ich w projekcie opisano w sekcji 3.3). Do kontaktu między GVMD a skryptem wykorzystywany jest ten sam protokół co do kontaktu między GVMD i GSA (protokół GMP). Do transportu danych, według dokumentacji (Greenbone, „Python-gvm documentation", marzec 2024, https://greenbone.github.io/python-gvm/usage.html), możliwe jest wykorzystywanie wielu protokołów - TLS, SSH i Unix domain socket. Choć za najciekawsze i najprzydatniejsze uznano zastosowanie SSH, jego konfiguracja okazała się być problematyczna. Brakuje również dokumentacji dotyczącej użycia zarówno protokołu SSH jak i TLS. Zdecydowano się wykorzystać najlepiej udokumentowane rozwiązanie - Unix domain socket. Ogranicza to niestety obsługiwane przez rozwiązanie projektowe systemy operacyjne do systemów UNIX.

3.2 Docker

W projekcie jest wykorzystywane narzędzie do konteneryzacji aplikacji - Docker. Pozwala ono na tworzenie, wdrażanie i uruchamianie aplikacji w kontenerach. Kontenery są izolowanymi, przenośnymi środowi-

skami, które zawierają wszystko, co potrzebne do uruchomienia aplikacji, w tym kod, zależności i konfigurację. Umożliwia to uruchamianie aplikacji na różnych środowiskach, niezależnie od infrastruktury i zainstalowanego na niej oprogramowania. W projekcie wykorzystywane jest także narzędzie Docker Compose, za pomocą którego można budować wielokontenerowe aplikacje, takie jak framework Greenbone Community Edition.

Firma Greenbone zapewnia swój zestaw narzędzi, opisany w sekcji 3.1, jako zestaw kontenerów Docker. Każde narzędzie jest udostępniane jako osobny kontener. Cały zestaw narzędzi można zbudować za pomocą zapewnianego przez Greenbone pliku docker-compose.

Aby osiągnąć założenia projektowe dokonano kilku zmian w pliku docker-compose zapewnianym przez Greenbone. Konieczna była konfiguracja unix-socket'ów, tak aby możliwa była komunikacja z wykorzystaniem biblioteki *python-gvm*. Komunikacja z kontenerem zawierającym narzędzie GVMD ma miejsce za pomocą unix-socket'ów. W związku z tym, z wykorzystaniem pliku docker-compose, konieczne jest wykonanie „bind mount" (Docker, „Setting up a mail transport agent inside Docker container", marzec 2024, https://docs.docker.com/storage/bind-mounts/), czyli umożliwienia kontenerom dostępu do folderu na dysku hosta, zgodnie z instrukcją producenta (Greenbone, „Exposing gvmd Unix socket for GMP Access", marzec 2024, https://greenbone.github.io/docs/latest/22.4/container/workflows.html#exposing-gvmd-unix-socket-for-gmp-access).

Próbowano również dokonać zmian w pliku docker-compose związanych z wysyłaniem raportów e-mailem, z wykorzystaniem alertów wbudowanych w narzędzie Greenbone, zgodnie z instrukcją (Greenbone, „Setting up a mail transport agent inside Docker container", marzec 2024, https://greenbone.github.io/docs/latest/22.4/container/workflows.html#setting-up-a-mail-transport-agent-inside-docker-container). Niestety, występowały przy tym problemy, dla których znaleziono niewielką ilość informacji dotyczących potencjalnej ich mitygacji, które jednak nie przynosiły efektu. W związku z tym zdecydowano się na rozwiązanie opisane w sekcji 3.3.2.

3.3 Skrypt w języku Python

Dedykowany dla projektu skrypt w języku Python odpowiedzialny jest za obsługę skanowań urządzeń sieciowych zgodnie z wytycznymi użytkownika, a także za automatyczne dostarczanie mu raportów bezpieczeństwa drogą mailową. Na potrzeby realizacji pierwszego z zadań wykorzystano opisaną w 3.1.1 bibliotekę *python-gvm*, odpowiedzialną za komunikację z menedżerem GVMD. W przypadku drugiego zadania skorzystano z biblioteki *smtplib*, umożliwiającej wysyłanie e-maili za pomocą protokołu SMTP. Poniżej przedstawiono ogólny schemat przygotowanego skryptu w pseudokodzie.

```
Main(nazwa_celu, adresy_hostów,
↪  częstotliwość_skanowań, email_adreata):
ustal_połączenie_z_serwerem_GVMD(ścieżka_do_unix_socket)
while True:
    utwórz_cel_skanowania(nazwa_celu,
    ↪  adresy_hostów, id_listy_portów)
    konfiguruj_skaner(id_konfiguracji_skanera)

    ↪  utwórz_harmonogram_skanowania(częstotliwość_skanowań,
    ↪  strefa_czasowa)
    utwórz_zadanie_skanowania(nazwa_zadania,
    ↪  id_skanera)
while True:
    czekaj_na_gotowość_zadania()
    pobierz_raport()
    zapisz_raport_do_pliku(ścieżka_do_zapisu)
    wyślij_raport_email(adres_email, hasło_email,
    ↪  email_adresat)
    while True:
        czekaj()
        if zmiana_statusu_skanowania():
            break
```

3.3.1 Obsługa skanowania

Na potrzeby obsługi skanowania urządzeń sieciowych w skrypcie ustanawiane jest połączenie z GVMD poprzez unix-socket wraz z uwierzytelnieniem do roli administratora, a także realizowane przygotowanie transformacji danych XML (ten język znaczników jest wykorzystywany przez protokół GMP) na obiekty języka Python.

W skrypcie szczegółowo definiowany jest cel skanowania. Do metody `create_target()` (z biblioteki *python-gvm*) podawane są takie argumenty jak adres IP hosta przeznaczonego do skanowania, czy identyfikator listy portów (w formacie UUID), które mają zostać uwzględnione podczas skanowania. Konfigurowane są również: typ skanowania (domyślnie „full and fast") oraz rodzaj skanera (używany domyślny skaner OpenVAS).

W kodzie, z wykorzystaniem biblioteki *icalendar*, szczegółowo konfigurowany jest kalendarz skanowań. Między innymi za pomocą `vRecur` tworzona jest reguła powtarzania skanowania, wraz z określoną jego częstotliwością. Korzystając ze skonfigurowanego obiektu kalendarza tworzony jest harmonogram skanowania z wykorzystaniem GVMD, za pomocą metody `create_schedule()` z *python-gvm*.

Wreszcie tworzone jest zadanie skanowania za pomocą metody `create_task()` z *python-gvm*. Do metody podawane są identyfikatory opisanych wyżej: typu skanowania, hosta będącego celem skanowania, rodzaju skanera, harmonogramu skanowania. Skanowanie uru-

chamiane jest automatycznie przez GVMD zgodnie z ustalonym harmonogramem.

Następnie skrypt w pętli monitoruje jego status. Sprawdzenie statusu odbywa się co określony w funkcji `sleep()` czas (wstępnie ustawione $30s$). Za pomocą metody `get_task()` pobierany jest obiekt reprezentujący dane zadania skanowania w formacie XML. Następnie na obiekcie wywoływana jest funkcja `find('.//status')` z argumentem będącym ścieżką do elementu XML, który zawiera informację o aktualnym statusie wykonania zadania skanowania. Jeśli tekstowa postać tego elementu to `'Done'`, ustawiana jest flaga określająca zakończenie skanowania, co kończy wykonywanie pętli monitorującej status.

Mechanizm sprawdzania statusu skanowania wykorzystywany jest do pobierania raportu. Po zakończeniu skanowania, w podobny sposób z wykorzystaniem funkcji `find()` (tym razem z atrybutami `'.//task'` oraz `'.//report'`) uzyskiwany jest identyfikator powstałego raportu. Metoda `get_report()` pozwala pobrać obiekt raportu o tym identyfikatorze. Następnie wykonywana jest sekwencja operacji prowadzących do zapisania raportu PDF w postaci binarnej pod odpowiednią ścieżką.

3.3.2 Wysyłanie raportów

Do realizacji funkcjonalności wysyłania raportów PDF do użytkownika drogą mailową skorzystano z konta pocztowego Onet[1]. W ustawieniach konta, w zakładce bezpieczeństwo, wygenerowano losowe hasło przeznaczone do użytku w usłudze zewnętrznej. Dzięki temu, w kodzie nie musi zostać podawane główne hasło, umożliwiające klasyczne zalogowanie do konta.

W skrypcie nawiązywane jest połączenie z serwerem SMTP Onet. Z wykorzystaniem konstruktora `SMTP()` z biblioteki *smtplib* tworzona jest instancja serwera SMTP ('smtp.poczta.onet.pl' na porcie 587). Następnie metoda `starttls()` rozpoczyna szyfrowane połączenie TLS z serwerem, a metoda `login()` dokonuje uwierzytelnienia za pomocą adresu e-mail wspomnianego konta pocztowego Onet oraz wygenerowanego na tę potrzebę hasła.

Do budowy wiadomości mailowych zdecydowano się na użycie biblioteki *e-mail.mime*. Każdorazowo tworzony jest obiekt klasy *MIMEMultipart*, reprezentujący wiadomość e-mail. Ustawiane są pola nagłówka wiadomości: `'From'`, `'To'` i `'Subject'`. Do pola `'To'` wiadomości przypisywany jest adres mailowy użytkownika systemu skanowania. Z wykorzystaniem konstruktora klasy *MIMEApplication* oraz metody `add_header`, plik PDF przygotowywany jest do załączenia. Na wcześniej utworzonym obiekcie klasy *MIMEMultipart* wykonywane są operacje `attach()`, załączające do obiektu wiadomości: treść (jako string) oraz plik PDF. Na koniec, metoda `as_string()` wywołana na obiekcie wiadomości, zwraca reprezentację całej sformatowanej wiadomości w postaci stringa.

Tak przygotowana wiadomość jest wysyłana za pomocą metody `sendmail()`, wywołanej na instancji serwera SMTP. Do metody podawane są: adres e-mail nadawcy, adres e-mail odbiorcy (użytkownika systemu skanowania) oraz wiadomość w postaci stringa. Po wysłaniu wiadomości mailowej metoda `quit()` zamyka połączenie z serwerem SMTP.

Po wykonaniu zadania, skrypt okresowo sprawdza, czy nastąpiła zmiana statusu skanowania, oznaczająca rozpoczęcie jego nowej instancji. Gdy to nastąpi, powtarza opisane wyżej kroki od momentu oczekiwania na zakończenie skanowania.

4 Podsumowanie i perspektywy rozwoju aplikacji

W ramach etapu I zebrano właściwą dokumentację oraz udowodniono możliwość realizacji założeń projektowych. Opracowano roboczą wersję aplikacji (opisaną w sekcji 3.3), która umożliwia zlecanie okresowych skanowań skanerowi OpenVas, pobieranie raportów oraz wysyłanie ich mailem. Skonfigurowano także maszynę wirtualną, opartą na dystrybucji systemu Linux Ubuntu, zawierającą uruchomiony zestaw kontenerów firmy Greenbone, opisany w sekcjach 3.1, 3.2. Pozwoliło to na udowodnienie możliwości pełnego zrealizowania założeń w II etapie projektu. Udowodniona została możliwość dostarczenia rozwiązania w postaci maszyny wirtualnej. Użytkownik będzie uruchamiał skrypt w języku Python (sekcja 3.3) wraz z odpowiednimi parametrami - adresem IP skanowanego urządzenia sieciowego i częstotliwością skanowań. W ten sposób możliwe będzie zrealizowanie założeń projektowych.

Planowaną opcją jest zamknięcie aplikacji do kontenera Docker. Pewne wątpliwości budzi jednak możliwość komunikacji z wykorzystaniem unix-socket'ów między aplikacją zamkniętą w kontenerze, a menedżerem GVMD, ze względu na brak udokumentowanych przypadków takiej komunikacji. Zebrano jednak stosowną dokumentację, dotyczącą bind mounts (Docker, „Setting up a mail transport agent inside Docker container"), rozwiązania którego zastosowanie potencjalnie może zapewnić komunikację między kontenerami. Rozwiązania tego jednak na obecnym etapie nie testowano. Planowane jest jego przetestowanie na etapie II i w zależności od wyników testu implementacja w docelowej aplikacji.

1. Próbowano także serwery innych dostawców np. Gmail, jednak zabezpieczenia uniemożliwiały połączenie się z serwerem z SMTP poziomu aplikacji.

Bibliografia

Docker. „Docker Compose overview”, marzec 2024. https://docs.docker.com/compose/.

———. „Setting up a mail transport agent inside Docker container”, marzec 2024. https://docs.docker.com/storage/bind-mounts/.

Greenbone. „Background”, marzec 2024. https://greenbone.github.io/docs/latest/background.html.

———. „Exposing gvmd Unix socket for GMP Access”, marzec 2024. https://greenbone.github.io/docs/latest/22.4/container/workflows.html#exposing-gvmd-unix-socket-for-gmp-access.

———. „Greenbone Community Containers”, marzec 2024. https://greenbone.github.io/docs/latest/22.4/container/index.html.

———. „Python-gvm documentation”, marzec 2024. https://greenbone.github.io/python-gvm/usage.html.

———. „Python-gvm documentation - create_schedule”, marzec 2024. https://greenbone.github.io/python-gvm/api/gmpv208.html#gvm.protocols.gmpv208.Gmp.create_schedule.

———. „Setting up a mail transport agent inside Docker container”, marzec 2024. https://greenbone.github.io/docs/latest/22.4/container/workflows.html#setting-up-a-mail-transport-agent-inside-docker-container.

PythonDocs. „smtplib — SMTP protocol client”, marzec 2024. https://docs.python.org/3/library/smtplib.html.