МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ С.В. Казмірчук
«_____»_____ 20__ р.

На правах рукопису

КВАЛІФІКАЦІЙНА РОБОТА
ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Система моніторингу мережі на наявність вразливого програмного забезпечення

Виконавець: М. С. Пазюк
Науковий керівник: к.т.н., доц. С.В. Єгоров
Нормоконтролер: к.т.н., доц. С.В. Єгоров

Київ 2020

ВСТУП

Актуальність. Пандемія COVID-19 кардинально змінила нашу реальність та вплинула на те як влаштований сучасний бізнес. Віддалена робота пропонується компаніями, коли це можливо - повністю або частково, передаючи домашній офіс підрядникам, які можуть собі це дозволити.

По мірі зростання кількості компаній що пропонують віддалену роботу, збільшуються і ризики пов’язані з підтримкою мережевої інфраструктури необхідної для ефективної віддаленої роботи[1]. Перед системними адміністраторами постає задача підтримки актуальності програмного та апаратного забезпечення, та швидкого реагування на появу вразливостей в ньому.

Кожного дня експерти знаходять нові вразливості у програмному забезпеченні. За 2019 рік Common Vulnerabilities and Exposures — база даних загальновідомих вразливостей інформаційної безпеки, поповнилась на 12174[2] нових вразливостей. Інциденти в галузі інформаційної безпеки здатні завдати серйозної шкоди компанії, в тому числі репутаційної. Від так, 2 вересня 2020 року стало відомо, що один з найбільших українських IT-аутсорсерів - компанія SoftServe - зазнала хакерської атаки, в результаті якої кілька сервісів компанії перестали працювати. Вже на наступний день в мережі з'явилися репозиторії проектів, які SoftServe розробляла для своїх клієнтів та персональні данні робітників компанії[3].

Для найбільш ефективного вирішення задач захисту мережі необхідний постійний моніторинг програмного забезпечення та швидке реагування на знаходження нових вразливостей. Часто перед фахівцями компаній для підвищення ефективності вирішення завдань захисту інформації виникає питання про автоматизацію сканування мережі. Проте зараз не існує відповідних

засобів, які можна було би швидко розгорнути та відразу ж задіяти без потреби у глибокому аналізі налаштувань та конфігурування.

Відомі підходи до вирішення поставленої задачі. На сьогоднішній день існує декілька систем для сканування та аналізу на наявність вразливого ПЗ у мережі. Нажаль більшість із них є закритими, а аналоги з відкритим вихідним кодом потребують ретельного вивчення документації для того щоб почати роботу з ними. Наявне ПЗ для проведення сканування та аналізу окремо один від одного має несумісні формати даних, а отже потребує мануальної роботи спеціаліста, або розробки власного програмного забезпечення для адаптації вихідних даних.

Метою роботи є розробка системи моніторингу мережі на наявність вразливого програмного забезпечення.

Для досягнення поставленої мети вирішуються такі **задачі:**

−дослідження відомих систем сканування та ідентифікації ПЗ встановленого на віддалених ПК та мережевому обладнанні;

−розробка системи аналізу та оцінки вразливостей на основі результатів сканування;

−розробка алгоритмів і програмного забезпечення та пакування їх в Docker контейнер для швидкої розгортки.

Галузь застосування. Розроблена система відносяться до галузі інформаційної безпеки і можуть бути використані для підвищення рівня захищеності мережі за рахунок автоматизації сканування та швидкого сповіщення про знайдені вразливості.

Об’єктом дослідження є процес захисту мережі від вторгнення через вразливості програмного забезпечення.

Предметом дослідження є методи, моделі та системи сканування та аналізу ПЗ у мережі.

Методи дослідження базуються на основі аналізу і порівняння принципів роботи ПЗ для сканування мережі та віддаленої ідентифікації встановленого ПЗ, та об'єктно орієнтованого програмування для програмної реалізації розробленої системи.

Новизна одержаних результатів полягає в наступному:

−Удосконалено процес сканування та аналізу вразливостей програмного забезпечення на віддалених хостах, за рахунок використання технології контейнеризації та методів аналізу та оцінки вразливостей ПЗ на основі результатів сканування, що дозволяє швидко автоматизувати процеси сканування, аналізу та оцінки ризиків.

Практичне значення отриманих результатів:

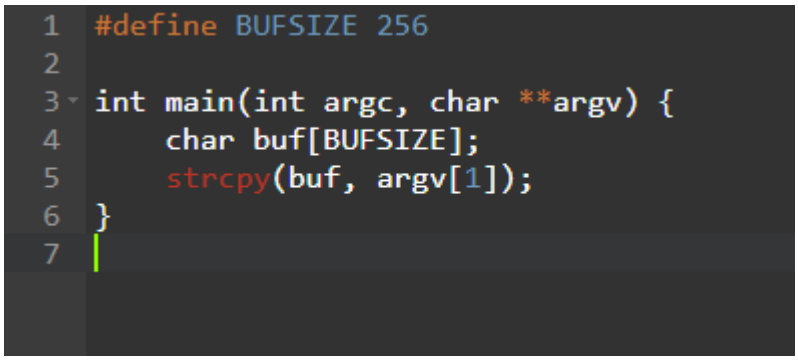
−роботи полягає у створенні удосконаленої системи для сканування і аналізу вразливостей ПЗ на основі рейтингу CVSSv3 мовою програмування python.

Розділ 1. ВРАЗЛИВОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. ІНСТРУМЕНТИ ВІДДАЛЕНОГО ВИЯВЛЕНЯ ВРАЗЛИВОГО ПЗ.

1.1. Помилки у коді що призводять до вразливостей у програмному забезпеченні

Переповнення буфера. Спираючись на данні з рейтингу CWE Top 25 2019, ми можемо зробити спостереження, що більшість помилок кодування (37,9%) трапляються в аспекті обробки даних. Вразливість програмного забезпечення, широко відома як "переповнення буфера" є найбільш поширеною в мові програмування C та C++[4]. Те саме стосується і читання та запису позва межами буферу. Ці помилки трапляються, коли буфер пам'яті налаштований отримувати більше даних, ніж дозволяє його ємність. Потім, під час виконання програми, дані записуються за межі блоку пам'яті. Як результат, операції зчитування або запису можуть проводитися з місця, що виходить за межі буфера. Через переповнення буфера неавторизований користувач може досить легко виконувати шкідливий код, читати конфіденційні дані або змінювати потік управління в програмі.

Хоча приклади переповнення буфера можуть бути досить складними, можна привести дуже простий, але все ще експлуатований приклад, переповнення буфера на основі стеку (рис. 1.1). Розмір буфера фіксований, але немає гарантії, що рядок з argv [1] не перевищить цей розмір та не спричинить переповнення.



```
#define BUFSIZE 256

int main(int argc, char **argv) {
    char buf[BUFSIZE];
    strcpy(buf, argv[1]);
}
```

Рис. 1.1. Приклад програми, що використовує незахищену функцію, без перевірки границь масиву

Ін'єкція команд та ін'єкція коду. Ін'єкція команд - це атака, метою якої є виконання довільних команд на операційній системі через вразливий додаток. Атаки ін'єкції команд можливі, коли програма передає небезпечні дані, надані користувачем (форми, файли cookie, заголовки HTTP тощо) в системну оболонку. У цій атаці команди операційної системи, що надаються зловмисником, зазвичай виконуються з привілеями вразливої програми. Атаки ін'єкції команд можливі здебільшого через недостатню перевірку вводу.

Ця атака відрізняється від ін'єкції коду тим, що ін'єкція коду дозволяє зловмисникові додати власний код у тіло програми, який потім виконується додатком. У включенні команд зловмисник розширює функціональність програми, яка виконує системні команди, без необхідності безпосередньо вводити код у тіло програми.

Код на рис. 1.2 - це обгортка навколо команди UNIX cat, яка друкує вміст файлу на стандартний вивід. Він є вразливим до ін'єкції команд

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv) {
 char cat[] = "cat ";
 char *command;
 size_t commandLength;

 commandLength = strlen(cat) + strlen(argv[1]) + 1;
 command = (char *) malloc(commandLength);
 strncpy(command, cat, commandLength);
 strncat(command, argv[1], (commandLength - strlen(cat)) );

 system(command);
 return (0);
}
```

Рис. 1.2. Код программи catWrapper

При звичайному використанні виводом програми буде лише вміст файлу (рис. 1.3).

```
$ ./catWrapper Story.txt
When last we left our heroes...
```

Рис. 1.3. Звичайний вивід програми catWrapper

Однак якщо ми додамо крапку з комою та іншу команду в кінець цього рядка, така команда виконається catWrapper (рис. 1.4).

```
$ ./catWrapper "Story.txt; ls"
When last we left our heroes...
Story.txt               doubFree.c              nullpointer.c
unstosig.c              www*                    a.out*
format.c                strlen.c                useFree*
catWrapper*             misnull.c               strlength.c
useFree.c
commandinjection.c      nodefault.c             trunc.c
writeWhatWhere.c
```

Рис. 1.4. Вивід програми catWrapper з ін’єкцією коду

Якби для catWrapper було встановлено вищий рівень привілеїв, ніж у стандартного користувача, довільні команди могли б бути виконані з підвищеними привілеями.

Відсутність звільнення ресурсів після використання. Якщо ресурс не звільняються після використання, це може дозволити зловмиснику викликати відмову в обслуговуванні. Зловмисник, який може впливати на розподіл ресурсів, які не були належним чином звільнені, може вичерпати доступний пул ресурсів і перешкодити всім іншим процесам отримати доступ до того самого типу ресурсів. До ресурсів, які часто попадають під атаку, належать пам’ять, файлові дескриптори, дисковий простір, ресурси центрального процесору, тощо.

Функція C приведена на рис. 1.5 не закриває дескриптор файлу, який він відкриває у разі виникнення помилки. Якщо процес довго живучий, то в нього можуть закінчитися файлові дескриптори.

```
int decodeFile(char* fName) {
    char buf[BUF_SZ];
    FILE* f = fopen(fName, "r");
    if (!f) {
        printf("cannot open %s\n", fName);
        return DECODE_FAIL;
    }
    else {
        while (fgets(buf, BUF_SZ, f)) {
            if (!checkChecksum(buf)) {
                return DECODE_FAIL;
            }
            else {
                decodeBlock(buf);
            }
        }
    }
    fclose(f);
    return DECODE_SUCCESS;
}
```

Рис. 1.5. Приклад функції що неправильно звільнює ресурси

Використання звільненої пам'яті. Посилання на пам’ять після її звільнення може спричинити збій програми та може мати будь-яку кількість

несприятливих наслідків[5] - починаючи від пошкодження даних і закінчуючи виконанням довільного коду.

Найпростіший спосіб пошкодження даних передбачає повторне використання звільненої пам'яті системою. У цьому випадку відповідна пам'ять призначається іншому покажчику в якийсь момент після його звільнення. Оригінальний покажчик на звільнену пам’ять використовується знову і вказує кудись у межах нового розподілу. Коли дані змінюються, вони пошкоджують дійсну пам’ять; це викликає невизначену поведінку в процесі.

Якщо нещодавно виділені дані можуть утримувати клас, як наприклад у C++, різні вказівники на функції можуть знаходитись всередині даних. Якщо один із цих покажчиків функцій перезапишеться адресою на код оболонки операційної системи, можна буде досягнути виконання довільного коду.

Код на рис. 1.6 ілюструє використання після вільної помилки. Коли виникає помилка, покажчик негайно звільняється. Однак пізніше цей покажчик неправильно використовується у функції logError.

```
char* ptr = (char*)malloc (SIZE);
if (err) {
    abrt = 1;
    free(ptr);
}
...
if (abrt) {
    logError("operation aborted before commit", ptr);
}
```

Рис. 1.6. Демонстрація помилки використання звільненої пам’яті

Використання жорстко закодованих облікових даних. Зазвичай закодовані облікові дані створюють значну дірку, яка дозволяє зловмисникові обійти автентифікацію, налаштовану адміністратором програмного забезпечення. Цю вразливість складно виявити системному адміністратору. Навіть після виявлення, її може бути важко виправити, через що адміністратор може бути змушений повністю вимкнути продукт. Є дві основні варіації:

- Вхідна: програмне забезпечення містить механізм автентифікації, який перевіряє вхідні дані проти жорстко закодованого набору облікових даних.
- Вихідна: програмне забезпечення підключається до іншої системи або компонента і містить жорстко закодовані облікові дані для підключення до цього компонента.

Приклад коду на рис. 1.7 намагається перевірити пароль за допомогою кодованого криптографічного ключа. Криптографічний ключ знаходиться в межах жорстко закодованого значення рядка, що порівнюється з паролем. Ймовірно, що зловмисник зможе прочитати ключ[17] і скомпрометувати систему.

```
int VerifyAdmin(char *password) {
    if (strcmp(password,"68af404b513073584c4b6f22b6c63e6b")) {
        printf("Incorrect Password!\n");
        return(0);
    }
    printf("Entering Diagnostic Mode...\n");
    return(1);
}
```

Рис. 1.7. Приклад використання жорстко закодованих облікових даних

Обхід шляху до файлу. Зловмисник може використати спеціальні елементи для того щоб сконструювати шлях до файлів або папок за межами обмеженого каталогу[18]. Зловмисники використовують послідовність символів ../ для виходу з кореневого каталогу. Через цю вразливість програмного забезпечення можна читати файли з конфіденційними даними. Вони також можуть створювати, перезаписувати або видаляти важливі файли безпеки (/etc/passwd, тощо).

Розіменування нульового покажчика. Нульовий покажчик вказує на те, що покажчик не посилається на дійсний об'єкт. Коли він розіменовується, це зазвичай спричиняє негайне завершення програми. Винахідник нульового вказівника Тоні Хоаре назвав його своєю "помилкою на мільярд доларів". Як він прокоментував, нульовий вказівник призвів до численних вразливостей

програмного забезпечення та збоїв у роботі системи, що завдало величезної шкоди користувачам та бізнесу.

Необмежене завантаження файлу небезпечного типу. Ця помилка програмування переважає у технології веб-серверів як проблема споживання ресурсів. Розмір або кількість завантажених файлів не обмежується, тому зловмисники можуть просто завантажувати шкідливе програмне забезпечення у вигляді файлів, які згодом автоматично обробляються. Як результат, завантажений файл інтерпретується і виконується системою. Особливо це стосується мови програмування PHP[23]. Така вразливість існує, оскільки досить часто типи файлів .php розглядаються як виконувані автоматично.

Ненадійний шлях пошуку. Зловмисники надають зовнішній шлях пошуку, який вказує на місце або базу даних, що не підлягає безпосередньому контролю додатка. Уразливість дозволяє імплантувати шкідливе програмне забезпечення, отримувати несанкціонований доступ до даних та змінювати конфігурації. У цьому випадку будь-який тип критичного джерела інформації, яке система вважає довіреним, може опинитися під загрозою.

Використання компонентів із відомими вразливостями. Ця практика приховує хитру загрозу безпеці. Розробники часто використовують готові компоненти додатків для побудови складних систем і не перевіряють наявність помилок чи програмних вразливостей у бібліотечних залежностях цих компонентів. Відомі помилки кодування досить легко виявити, та використати. Ефекти можуть бути будь-якими з наведених вище.

1.2. Бази даних вразливостей

База даних вразливостей (VDB) - це платформа, спрямована на збір, підтримку та розповсюдження інформації про виявлені вразливості комп'ютерної безпеки. База даних зазвичай описує виявлену вразливість, оцінює потенційний вплив на уражені системи та способи обходу або усунення

вразливості. VDB присвоює унікальний ідентифікатор кожній каталогізованій вразливості (наприклад, VDB-2020-12345). Інформація в базі даних може бути доступною через веб-сторінки, експорт або API. VDB може надавати інформацію безкоштовно або платно.

Основні бази даних вразливостей, такі як база даних ISS X-Force, база даних Symantec / SecurityFocus BID та база даних вразливостей з відкритим кодом (OSVDB), об'єднують широкий спектр загальнодоступних вразливостей, включаючи CVE. Основною метою CVE, якою керує MITER, є спроба агрегувати загальнодоступні вразливості та надати їм унікальний ідентифікатор стандартизованого формату[19]. Багато баз даних вразливостей досліджують та обробляють отримані від CVE дані та проводять подальші дослідження, надаючи оцінки ризику вразливості, рейтинги впливу та шляхи усунення вразливості. В минулому індекс CVE використовувався для зв’язування баз даних вразливостей з критичними виправленнями ПЗ. Національна база даних про вразливості (NVD), якою керує Національний інститут стандартів і технологій (NIST), працює окремо від бази даних CVE, що працює під управлінням MITER, але включає лише інформацію про вразливість від CVE. NVD опрацьовує ці дані, надаючи оцінку ризиків за системою Common Vulnerability Scoring System (CVSS).

Бази даних вразливостей містять широкий спектр виявлених вразливостей. Однак небагато організацій мають досвід, персонал та час для перегляду, оцінки та виправлення всіх потенційних вразливостей системи, отже, оцінка вразливості є методом кількісного визначення тяжкості порушення системи. У базах даних вразливостей існує безліч методів оцінювання, але загальна система оцінки вразливості (CVSS) є переважаючою технікою для більшості баз даних вразливості, включаючи OSVDB, vFeed та NVD[20]. CVSS базується на трьох основних показниках: базовому, часовому та контекстному, кожен з яких впливає на кінцевий рейтинг вразливості[6].

Базові метрики описують характеристики уразливості, не змінюються з плином часу і не залежать від середовища виконання. Цими метриками описується складність експлуатації уразливості і потенційний збиток для конфіденційності, цілісності та доступності інформації.

Часові метрики, як видно з назви, вносять в загальну оцінку поправку на повноту наявної інформації про вразливість, зрілість коду експлойтів (при їх наявності) і доступність виправлень.

За допомогою контекстних метрик експерти з безпеки можуть внести в результуючу оцінку поправки з урахуванням характеристик інформаційного середовища.

Часові і контекстні метрики є опційними[21] і застосовуються для більш точної оцінки небезпеки, яку представляє дана уразливість для більш-менш конкретної інфраструктури. Значення метрики прийнято публікувати у вигляді пари з вектора (конкретні значення окремих показників) і числового значення, розрахованого на основі всіх показників за допомогою формули, визначеної в стандарті.

1.3. Сканування портів

Сканер портів — це програма, призначена для перевірки сервера або хоста на наявність відкритих сокет-портів. Адміністратори часто використовують для перевірки політики безпеки своїх мереж. Хакери теж використовують для ідентифікації мережевих сервісів, що працюють на хості, та пошуку вразливостей.

Сканування портів — це процес, який надсилає клієнтські запити до діапазону адрес портів сервера на хості з метою пошуку активного порту, він не є зловмисним за задумом[7, с.128]. Більшість застосувань сканування портів є не атаками, а простими перевірками для визначення послуг, доступних на віддаленому хості.

Архітектура та принципи інтернету основані на IP-протоколі. Відповідно до цієї системи, мережеві послуги надаються за допомогою пари компонент: адреси хоста та номера порту. Всього існує 65536 різних доступних для використання портів[25]. Більшість мережевих сервісів використовують тільки декілька портів для виконання поставлених задач. Деякі сканери портів сканують тільки найпоширеніші номери портів або порти, що найчастіше пов'язані з уразливими службами, на певному хості. Усі форми сканування портів спираються на припущення, що кінцевий пристрій сумісний із стандартом RFC 793[22]. Результат сканування на порту зазвичай зводиться до одного із трьох варіантів:

- Порт відкритий: хост відправив відповідь, яка вказує на те, що дані, відправлені на цей порт прослуховуються і обробляються хостом
- Порт закритий: хост відправив відповідь, яка вказує на те, що дані, відправлені на цей порт не оброблятимуться і будуть відкинуті
- Відповідь не отримана — хост не відправив відповідь на запит, або вона не дійшла.

Типи сканувань:

1.3.1. SYN-сканування.

Даний тип сканування є найбільш популярним. Замість використання мережевих функцій операційної системи, сканер портів сам генерує IP пакети, і відстежує відповіді на них. Цю техніку часто називають скануванням з використанням напіввідкритих з'єднань, оскільки повне з'єднання TCP / IP ніколи не відкривається. Сканер портів генерує пакет SYN. Якщо порт на цільовому хості відкритий, з нього прийде пакет SYN-ACK. Хост сканера відповідає пакетом RST, закриваючи тим самим з'єднання до того, як процес встановлення з'єднання завершився. Використання самостійно сформованих мережевих пакетів має ряд переваг, даючи скануючій програмі повний контроль

над пакетами що відсилаються і відповідями на них, затримками відповідей, і дозволяючи отримати детальні результати сканування.

1.3.2. TCP-сканування.

Даний метод є більш простим бо використовує мережеві функції операційної системи, і застосовується, коли SYN-сканування з тих чи інших причин здійснити неможливо. Операційна система, в разі, якщо порт відкритий, завершує трьохетапну процедуру встановлення з'єднання, і потім відразу закриває з'єднання. В іншому випадку, повертається код помилки. Перевагою даного методу є те, що він не вимагає від користувача спеціальних прав доступу. Проте, цей метод не дозволяє вести низькорівневий контроль відсилаємих пакетів, тому використовується не настільки широко. Головним недоліком даного методу є велика кількість відкритих і відразу перерваних з'єднань, що створює навантаження на скановану систему і дозволяє просто виявити активність сканера портів.

1.3.3. UDP-сканування.

Сканування за допомогою пакетів UDP також можливо, хоча має ряд особливостей. Для UDP відсутнє поняття з'єднання, і немає еквівалента TCP-пакету SYN. Проте, якщо послати UDP-пакет на закритий порт, система відповість повідомленням ICMP «порт недоступний». Відсутність такого повідомлення тлумачиться як сигнал того, що порт відкритий. Однак, якщо порт блокується брандмауером, метод невірно покаже, що порт відкритий. Якщо заблоковані ICMP-повідомлення про недоступність порту, всі порти будуть здаватися відкритими. Також, може бути встановлено обмеження на частоту використання ICMP-пакетів, що також впливає на результати, що даються методом.

Альтернативним підходом є відправка UDP-пакетів, специфічних для певного додатка, в розрахунку на отримання відповіді з рівня додатка. Наприклад, відправка запиту DNS на порт 53 приведе до відповіді, якщо по адресі запиту є DNS-сервер. Проблема в даному випадку полягає в наявності відповідного «пробного» пакета для кожного з портів. У деяких випадках, сервіс може бути присутнім, але бути налаштований таким чином, щоб не відповідати на відомі «пробні» пакети.

Також можливий комбінований підхід, що поєднує в собі обидва вищевказаних методи. Сканування може починатися відправкою UDP-пакета для перевірки на ICMP-відповідь «порт недоступний», а потім порти з невизначеним результатом «відкритий або заблокований» можуть повторно перевірятись на специфічні для програм відповіді.

1.3.4. ACK-сканування.

Дане сканування використовується для визначення, чи фільтрується даний порт, і особливо ефективний для визначення наявності брандмауерів і з'ясування їх правил. Проста фільтрація пакетів дозволить проходження пакетів з встановленим бітом ACK (використовується для вже встановлених з'єднань), тоді як більш складні брандмауери цього не дозволять[8].

1.3.5. FIN-сканування.

Деякі сервери здатні виявити спробу SYN-сканування портів. Наприклад, спробу SYN-сканування можна розпізнати по надходженню «підроблених» SYN-пакетів на закриті порти сервера, і в разі опитування декількох портів сервер розриває з'єднання для захисту від сканування.

Сканування з використанням FIN-пакетів дозволяє обійти подібні засоби захисту. Згідно RFC 793[24], якщо FIN-пакет прибуває на закритий порт сервер

повинен відповісти пакетом RST. FIN-пакети на відкриті порти повинні ігноруватися сервером. Завдяки цій відмінності стає можливим відрізнити закритий порт від відкритого. Проте варто зазначити, що рекомендації RFC 793 дотримуються не всі операційні системи.

1.3.6. Сканування в режимі очікування.

Перевірка в режимі очікування - це метод сканування портів TCP, який полягає у надсиланні підроблених пакетів на хост для з’ясування наявних послуг. Це досягається видаванням себе за інший комп’ютер, мережевий трафік якого є дуже повільним або взагалі не існує (тобто не передає та не отримує інформацію). Це може бути простоюючий комп’ютер, який називається «зомбі»[9].

Сканування в режимі очікування використовує властивості передбачуваного значення поля ідентифікації із заголовка IP: кожен пакет IP із даного джерела має ідентифікатор, який однозначно ідентифікує фрагменти оригінальної дейтаграми IP; реалізація протоколу призначає значення цьому обов’язковому полю, як правило, із збільшенням фіксованого значення (1). Оскільки передані пакети нумеруються послідовно, можна сказати, скільки пакетів було передано між двома отриманими пакетами.

Спочатку зловмисник сканує хост із послідовним та передбачуваним порядковим номером (IPID). Останні версії Linux, Solaris, OpenBSD та Windows починаючи з Vista не підходять на роль зомбі, оскільки IPID реалізовано з виправленнями, які рандомізували IPID. Комп’ютери, обрані для використання на цьому етапі, відомі як «зомбі».

Як тільки підходящий зомбі буде знайдений, наступним кроком буде спроба встановити TCP-з'єднання із заданою службою (портом) цільової системи, видаючи себе за зомбі. Це робиться шляхом відправки пакету SYN на

цільовий комп'ютер, підробляючи IP-адресу від зомбі, тобто з вихідною адресою, рівною IP-адресі зомбі.

Якщо порт цільового комп'ютера відкритий, він прийме підключення до послуги, відповідаючи пакетом SYN / ACK назад зомбі.

Потім зомбі-комп’ютер надішле RST-пакет цільовому комп’ютеру (для скидання з’єднання), оскільки фактично спочатку не надсилав пакет SYN.

Оскільки зомбі повинен був відправити пакет RST, він збільшить свій IPID. Ось як зловмисник дізнається, чи відкрито порт цілі. Зловмисник надішле зомбі ще один пакет. Якщо IPID збільшується лише на крок, то зловмисник знає, що конкретний порт закритий.

Метод передбачає, що зомбі не має інших взаємодій: якщо між «зомбі» та іншим хостом відбудеться обмін пакетами у проміжок часу між першою взаємодією зловмисника та зомбі та їх другою взаємодією, результатом стане помилково позитивний результат.

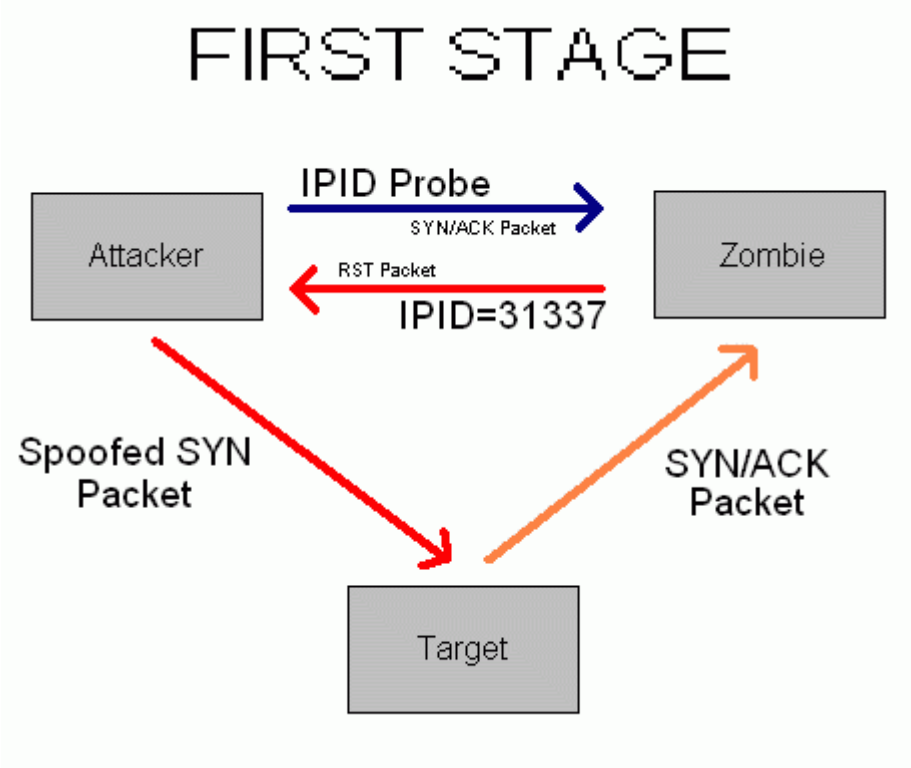


Рис. 1.8. Схема сканування в режимі очікування, перша фаза

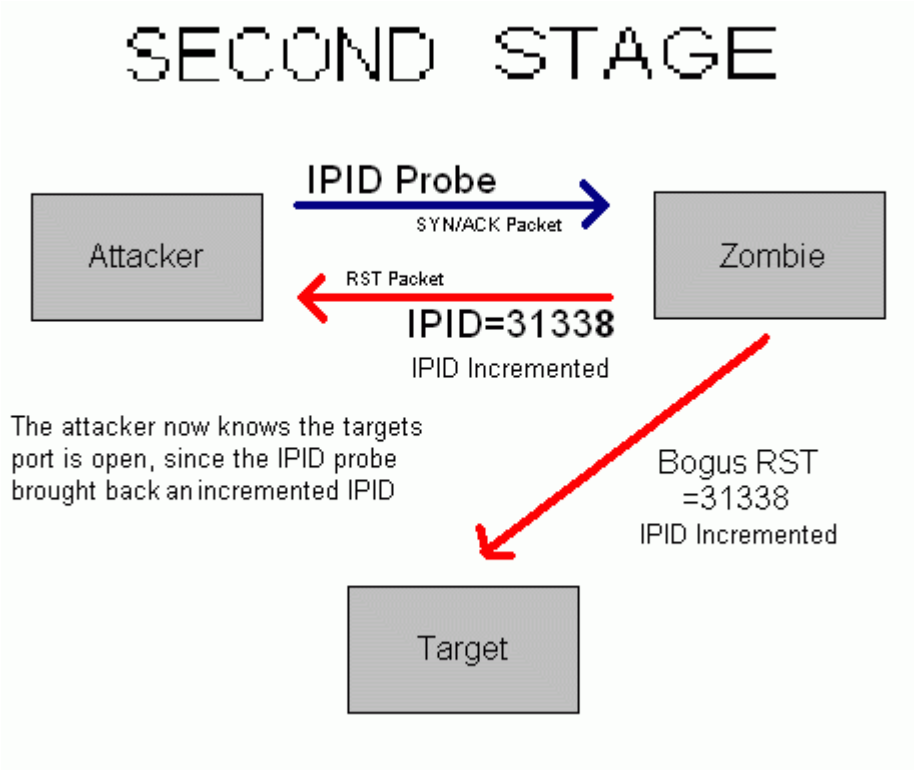


Рис. 1.9. Схема сканування в режимі очікування, друга фаза

1.4. Сканери портів з відкритим вихідним кодом. Їх переваги та недоліки.

1.4.1 Nmap

Nmap – opensource[16] утиліта для сканування мереж, є одним з найпопулярніших інструментів у фахівців з кібербезпеки та системних адміністраторів. В першу чергу використовується для сканування портів, крім цього, має величезну масу корисних функцій, що, по суті, робить Nmap фреймворком для дослідження мереж.

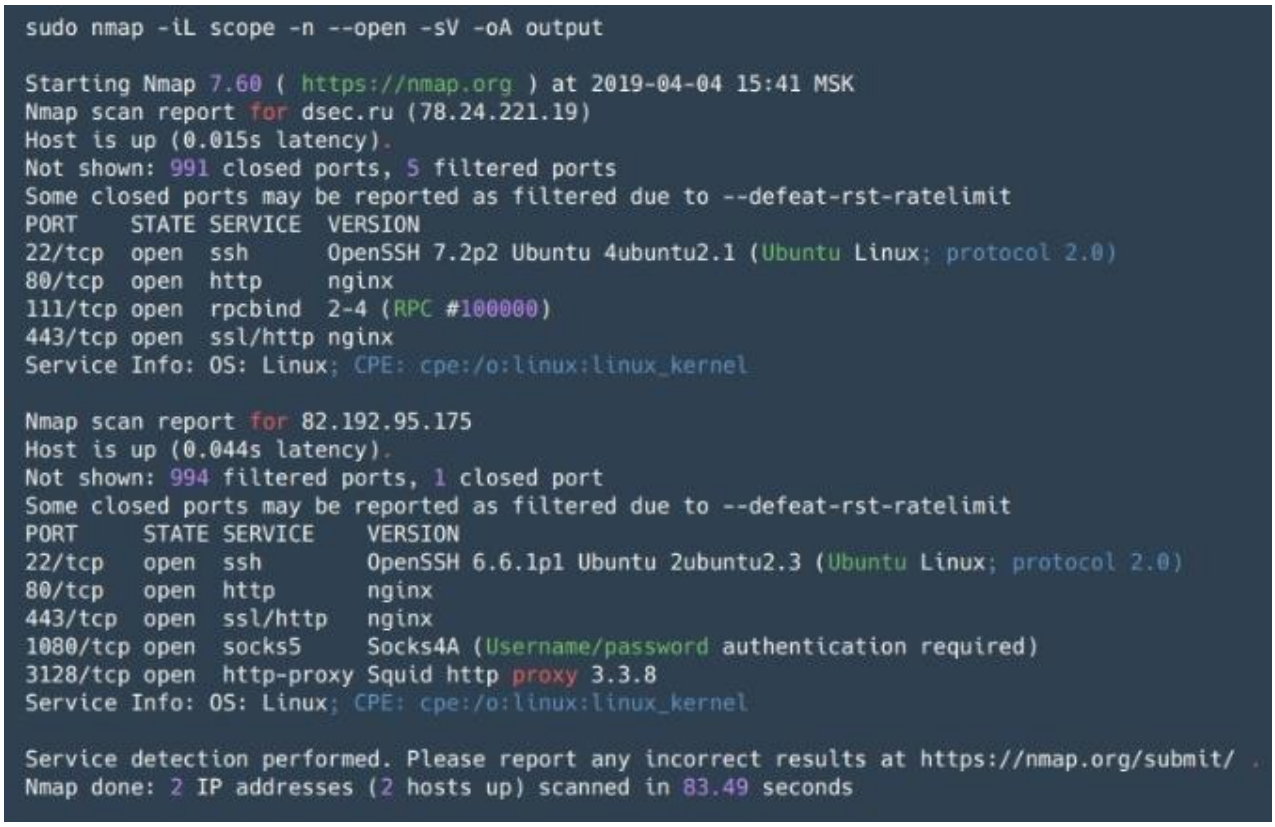


Рис. 1.10. Типовий вивід Nmap

Крім перевірки відкритих / закритих портів Nmap може ідентифікувати сервіси[10], що слухають відкритий порт та їх версії, а іноді допомагає визначити ОС[11]. У Nmap є можливості запуску програм для сканування (NSE - Nmap Scripting Engine)[12]. З використанням скриптів можливо перевірити уразливості для різних сервісів (якщо, для них є скрипт, або можна написати свій).

Таким чином, Nmap дозволяє скласти детальну карту мережі, отримати максимум інформації про запущені сервіси на хостах в мережі, а також превентивно перевірити деякі уразливості. Nmap також має гнучкі налаштування сканування, можлива настройка швидкості сканування, кількості потоків, кількості груп для сканування і т.д.

Переваги:

- Швидка робота з невеликим діапазоном хостів.
- Гнучкість налаштувань - можна комбінувати опції таким чином, щоб отримати максимально інформативні дані за прийнятний час.

- Паралельне сканування - список цільових хостів розділяється на групи, кожна група сканується по черзі, всередині групи використовується паралельне сканування.
- Готові набори скриптів для різних задач, можливість створювати свої.
- Виведення результатів у 5 різних форматів, включаючи XML, який може бути імпортований в інші інструменти.

Недоліки:

- Під час сканування групи хостів інформація про конкретний хост недоступна, поки не закінчиться сканування всієї групи.
- При скануванні Nmap відправляє SYN-пакети на цільової порт і чекає на відповідь або настання таймаута, в разі коли відповіді немає. Це негативно позначається на продуктивності сканера в цілому, в порівнянні з асинхронними сканерами.
- При скануванні великих мереж з використанням флагів для прискорення сканування (--min-rate, --min-parallelism) може давати false-negative результати, пропускаючи відкриті порти на хості.

1.4.2. Zmap

Zmap - сканер з відкритим вихідним кодом, створювався як більш швидка альтернатива Nmap.

```
sudo zmap -r 1000 -w input.ranges -p 443 -o output.csv
Apr 16 18:32:44.307 [INFO] zmap: output module: csv
 0:00 0%; send: 0 0 p/s (0 p/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate:
0.00%
 0:00 0%; send: 0 0 p/s (0 p/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate:
0.00%
 0:01 6%; send: 1008 1.01 Kp/s (997 p/s avg); recv: 392 391 p/s (387 p/s avg); drops: 0 p/s (0 p/s
avg); hitrate: 38.89%
 0:02 13%; send: 2008 999 p/s (998 p/s avg); recv: 1119 726 p/s (556 p/s avg); drops: 0 p/s (0 p/s
avg); hitrate: 55.73%
 0:03 19%; send: 3007 998 p/s (998 p/s avg); recv: 2039 919 p/s (677 p/s avg); drops: 0 p/s (0 p/s
avg); hitrate: 67.81%
 0:04 26%; send: 4006 998 p/s (998 p/s avg); recv: 2911 871 p/s (725 p/s avg); drops: 0 p/s (0 p/s
avg); hitrate: 72.67%
 0:05 32% (11s left); send: 5007 1.00 Kp/s (999 p/s avg); recv: 3567 655 p/s (711 p/s avg); drops: 0
p/s (0 p/s avg); hitrate: 71.24%
 0:06 39% (10s left); send: 6006 998 p/s (998 p/s avg); recv: 4422 854 p/s (735 p/s avg); drops: 0 p/s
(0 p/s avg); hitrate: 73.63%
 0:07 45% (9s left); send: 7006 999 p/s (999 p/s avg); recv: 5294 871 p/s (754 p/s avg); drops: 0 p/s
(0 p/s avg); hitrate: 75.56%
 0:08 52% (8s left); send: 7543 done (999 p/s avg); recv: 5816 521 p/s (725 p/s avg); drops: 0 p/s (0
p/s avg); hitrate: 77.10%
 0:09 58% (7s left); send: 7543 done (999 p/s avg); recv: 6131 314 p/s (680 p/s avg); drops: 0 p/s (0
p/s avg); hitrate: 81.28%
```

Рис. 1.11. Типовий вивід zmap

На відміну від Nmap - Zmap при відправці SYN-пакетів не чекає поки повернеться відповідь, а продовжує сканування, паралельно чекаючи відповіді від усіх хостів, таким чином фактично він не підтримує стан з'єднання. Коли відповідь на SYN-пакет прийде Zmap за змістом пакету зрозуміє який порт і на якому хості був відкритий. Крім того, Zmap відправляє тільки один SYN-пакет на сканований порт. Також є можливість використання PF_RING[13] для швидкого сканування великих мереж, при наявності 10-гігабітного інтерфейсу і сумісної мережевої карти.

Переваги:

- Висока швидкість сканування.
- Zmap генерує Ethernet-фрейми минаючи системний стек TCP / IP.
- Можливість використання PF_RING.
- ZMap рандомізує цілі для рівномірного розподілу навантаження на скануємій стороні.

Недоліки:

- Може стати причиною відмови в обслуговуванні мережевого обладнання, наприклад, вивести з ладу проміжні маршрутизатори, незважаючи на розподілене навантаження, оскільки всі пакети будуть проходити через один маршрутизатор[13].
- Станом на 2020-й рік можливо задати лише один порт для одного проходу сканування[14]. Для сканування інших портів сканер потрібно запускати заново.

1.4.3. Masscan

Masscan - сканер з відкритим вихідним кодом, який створювався з однією метою - сканувати Інтернет ще швидше (менше, ніж за 6 хвилин зі швидкістю ~ 10 млн пакетів / с). Працює майже так само як і Zmap, тільки ще швидше.

```
sudo masscan -p 139,445,3389 --rate 1000 10.0.0.0/12

Starting masscan 1.0.4 (http://bit.ly/14GZzcT) at 2019-04-04 13:39:26 GMT
 -- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
Scanning 1048576 hosts [3 ports/host]
Discovered open port 445/tcp on 10.0.0.10
Discovered open port 139/tcp on 10.0.9.3
Discovered open port 445/tcp on 10.0.2.231
Discovered open port 445/tcp on 10.0.88.18
```

Рис. 1.12. Типовий вивід Masscan

Переваги:

- Синтаксис схожий з Nmap, програма підтримує деякі сумісні з Nmap опції[15].
- Швидкість роботи - один з найшвидших асинхронних сканерів.
- Гнучкий механізм сканування - відновлення перерваного сканування, розподіл навантаження на декілька пристроїв.
- Можливість одночасного сканування різних портів.

Недоліки:

- Навантаження на саму мережу вкрай високе, що може привести до DoS[15].
- Немає можливості сканувати на прикладному рівні L7.

1.5. Висновки до розділу.

У даному розділі дипломної роботи було розглянуто та проаналізовано типові помилки у коді програмних додатків що призводять до появи вразливостей у цих додатках. Було розглянуто як такі вразливості можуть бути використані зловмисниками та для яких цілей.

Ми розглянули існуючі бази даних вразливостей та принципи за якими вони класифікують такі вразливості та присвоюють їм рейтинг.

Крім цього було розібрано існуючі типи та техніки сканування мережі та збору інформації про неї. Розглянуто мережеві сканери з відкритим вихідним кодом, їх переваги та недоліки.

Підсумовуючи викладене у розділі, можна зробити висновок що жодне програмне забезпечення не застраховано від помилок що призводять до появи вразливостей, тому важливо знаходити їх, описувати, класифікувати та присвоювати їм рейтинг. Така інформація має зберігатися у централізованій відкритій базі даних. Спеціальні додатки сканери при сумісному використанні з базами даних вразливостей можуть бути застосовуватись для виявлення та усунення вразливостей у інфраструктурі мережі. Перевірити усі можливі вразливі місця комплексної мережі без використання подібних інструментів є практично нереальною задачею.

Розділ 2. СУЧАСНІ РІШЕННЯ У СФЕРІ МОНІТОРИНГУ ВРАЗЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ШВИДКОЇ РОЗГОРТКИ ПРОГРАМНИХ ДОДАТКІВ.

2.1 Контейнери як рішення проблеми швидкої розгортки та надійного запуску програмного забезпечення у різних обчислювальних середовищах.

Контейнери - це рішення проблеми забезпечення надійної роботи програмного забезпечення у різних обчислювальних середовищах. Від ноутбука розробника до тестового середовища, від проміжного середовища до production серверу і, можливо, від фізичної машини в центрі обробки даних до віртуальної машини в приватній або загальнодоступній хмарі. Проблеми виникають, коли середовище необхідне для роботи програмного забезпечення, не ідентичне тому що використовувалось для його розробки та тестування. Як заявляє творець Docker Соломон Хайкс: "Ви збираєтесь провести тестування за допомогою Python 2.7, а у production ваш додаток буде працювати на Python 3, і відбудеться щось дивне. Або ви покладаєтесь на поведінку певної версії бібліотеки SSL, і буде встановлена інша. Ви будете запускати свої тести на Debian, а production - на Red Hat, і трапляються всілякі дивні речі ". І не лише різне програмне забезпечення може спричинити проблеми, додав він. "Топологія мережі може бути іншою, або політики безпеки можуть бути різними, але програмне забезпечення має працювати."

Контейнер складається з повного середовища виконання: програми, а також усіх залежностей, бібліотек, а також конфігураційних файлів, необхідних для її запуску, об'єднаних в один пакет. Контейнеризуючи платформу додатків та їх залежності, усуваються відмінності в середовищах операційних систем та базовій інфраструктурі.

Постає питання у чому ж відмінність контейнерів від технологій віртуалізації? З технологією віртуалізації, пакет, з яким ми працюємо являє

собою віртуальну машину, і вона включає в себе всю операційну систему та її додатки. Фізичний сервер, на якому працюють три віртуальні машини, мав би гіпервізор і три окремі операційні системи, що працюють поверх нього.

На відміну від цього, сервер, що запускає три контейнерні програми, запускає одну операційну систему, і кожен контейнер ділить ядро операційної системи з іншими контейнерами. Спільні частини операційної системи доступні лише для читання, тоді як кожен контейнер має власну точку монування (тобто спосіб доступу до контейнера) для запису. На практиці це означає, що контейнери набагато легші і використовують набагато менше ресурсів, ніж віртуальні машини.

Розмір контейнера може бути лише десятки мегабайт, тоді як віртуальна машина з цілою власною операційною системою може мати розмір у кілька гігабайт. Через це на одному сервері може бути розміщено набагато більше контейнерів, ніж віртуальних машин.

Ще однією перевагою є те, що віртуальним машинам може знадобитися кілька хвилин для завантаження своїх операційних систем і запуску програм, які вони розміщують, тоді як контейнерні програми можна запускати майже миттєво. Це означає, що контейнери можуть бути створені "just in time", тобто коли вони потрібні, і можуть зникнути, коли вони більше не потрібні, звільняючи ресурси на своїх хостах.

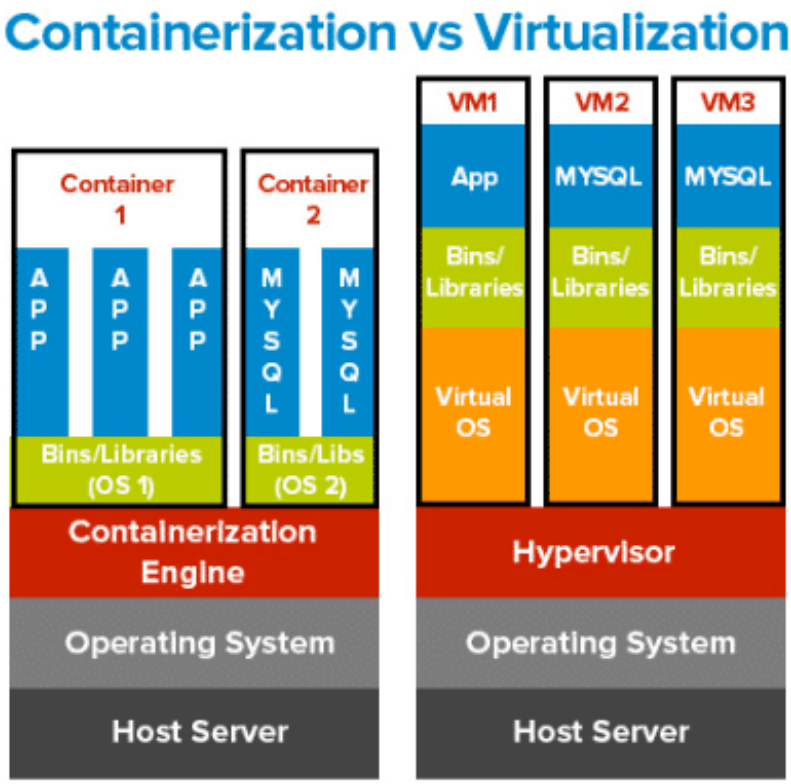


Рис. 2.1. Порівняння влаштування контейнеризації та віртуалізації

Третя перевага полягає в тому, що контейнеризація забезпечує більшу модульність. Замість того, щоб запускати цілу складну програму всередині одного контейнера, програму можна розділити на модулі (наприклад, базу даних, інтерфейс програми тощо). Це так звана мікросервісна архітектура. Управління програмами побудованими таким чином простіше, оскільки кожен модуль є відносно простим, і модулі можуть бути внесені без необхідності перебудовувати весь додаток. Оскільки контейнери настільки легкі, окремі модулі (або мікропослуги) можуть бути створені лише тоді, коли вони потрібні і вони будуть доступні майже відразу.

Ще одним важливим питанням в контексті контейнерів та управління ними є стандартизація. У 2015 році була оголошена ініціатива під назвою «Відкритий контейнерний проект», яка пізніше була перейменована в «Ініціатива відкритих контейнерів». Метою ініціативи, що працює під егідою Linux Foundation, є розробка галузевих стандартів для формату контейнера та програмного забезпечення для виконання контейнерів для всіх платформах. Відправною

точкою стандартів була технологія Docker, і Docker пожертвував на проект близько 5 відсотків своєї кодової бази, щоб зрушити ініціативу з місця. Спонсорами проекту є AWS, Google, IBM, HP, Microsoft, VMware, Red Hat, Oracle, Twitter і HP, а також Docker і CoreOS. Замість того, щоб витрачати ресурси на розробку конкуруючих контейнерних технологій, організації можуть зосередитись на розробці додаткового програмного забезпечення, необхідного для підтримки використання стандартизованих контейнерів в корпоративному або хмарному середовищі. Тип необхідного програмного забезпечення включає системи організації та управління контейнерами та системи безпеки контейнерів.

Рис. 2.2. Логотип Ініціативи відкритих контейнерів

Багато людей вважають, що контейнери менш безпечні, ніж віртуальні машини, тому що якщо в ядрі хоста контейнера є вразливість, це може забезпечити доступ до контейнерів, що його використовують. Це також вірно для гіпервізора, але оскільки гіпервізор надає набагато менше функціональних можливостей, ніж ядро ОС (яке зазвичай реалізує роботу з файловою системою, мережеві функції, засоби керування процесом програми тощо), він представляє набагато меншу цінність як вектор для атаки. Тому останні пару років багато зусиль було приділено розробці програмного забезпечення для підвищення захищенності контейнерів. Наприклад, Docker (та інші контейнерні системи) тепер включають інфраструктуру цифрового підпису, що дозволяє адміністраторам підписувати образи контейнерів, щоб запобігти розгортанню

ненадійних контейнерів. Однак надійний, підписаний контейнер не обов’язково буде безпечним для запуску, оскільки уразливості можуть бути виявлені в деякому програмному забезпеченні всередині контейнера після його підписання. З цієї причини Docker та інші пропонують рішення щодо сканування контейнерів на наявність вразливостей, такі рішення можуть сповіщати адміністраторів про наявні проблеми в інфраструктурі. Компанія із захисту контейнерів під назвою Polyverse використовує той факт, що контейнери можна запускати за частки секунди для того, щоб кожні кілька секунд перезапускати контейнерні програми у відомому робочому стані, щоб мінімізувати час, який хакер може використати для запуску експлойту до програми що виконується у контейнері. Інша компанія - Twistlock пропонує програмне забезпечення, яке аналізує очікувану поведінку контейнера та створює "білі списки" процесів, мережевої активності (наприклад, IP-адреси відправників та одержувачів їх порти) та навіть певні практики зберігання даних програмним забезпеченням що використовує контейнер, таким чином будь-яка шкідлива чи несподівана поведінка може бути помічена.

2.2 Аналіз та порівняння наявних рішень для розгортки програмних додатків в контейнерному середовищі.

2.2.1 Docker

Docker - це програмне забезпечення для контейнеризації, яке виконує віртуалізацію на рівні операційної системи.

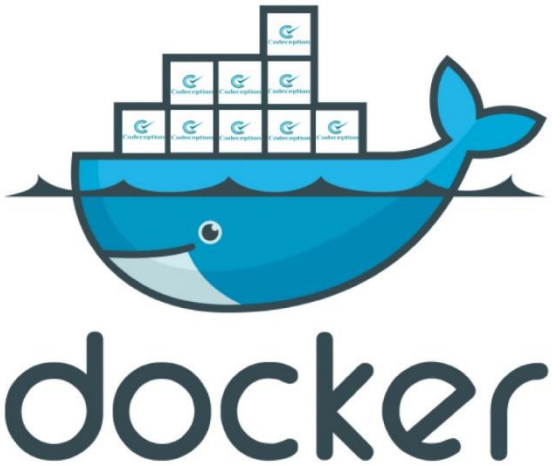


Рис. 2.3 Логотип Docker

Розробник цього програмного забезпечення - Docker, Inc. Перший випуск цього програмного забезпечення відбувся в 2013 році. Написаний на мові програмування „Go”. Це безкоштовне програмне забезпечення як послуга та має ліцензію Apache License 2.0 як ліцензію вихідного коду.

Docker може упакувати програму та її залежності у віртуальний контейнер, який може працювати на будь-якому сервері. Це дозволяє застосунку працювати в різних місцях, наприклад, локально, у відкритій хмарі та / або у приватній хмарі. Docker використовує функції ізоляції ресурсів ядра Linux (таких як cgroups та простори імен ядра) та файлову систему, яка підтримує об'єднання (наприклад, OverlayFS), щоб дозволити контейнери запускатися в межах одного екземпляра Linux, уникаючи накладних витрат на запуск та обслуговування віртуальних машин.

Оскільки контейнери Docker не потребують багато ресурсів, на одному сервері або віртуальній машині можна одночасно запускати кілька контейнерів. Аналіз 2018 року показав, що типовий випадок використання Docker

передбачає запуск восьми контейнерів на хост і що чверть аналізованих організацій запускає 18 і більше на хост.

Підтримка простору імен ядра Linux в основному ізолює операційне середовище від програми. Починаючи з версії 0.9, Docker включає власний компонент (званий "libcontainer") для безпосереднього використання засобів віртуалізації, що надаються ядром Linux, на додаток до використання абстрактних інтерфейсів віртуалізації через libvirt, LXC та systemd-nspawn (див. Рис. 2.4).

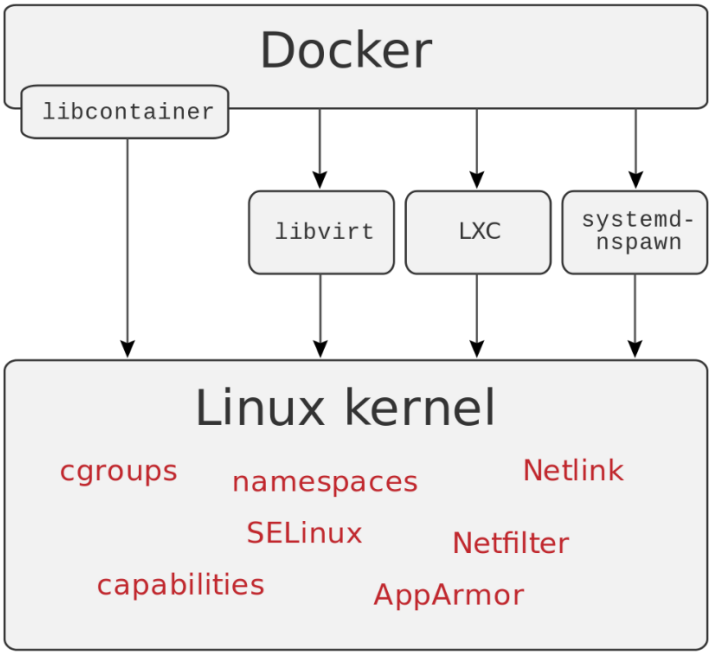


Рис. 2.4 Docker може використовувати різні інтерфейси для доступу до функцій віртуалізації ядра Linux

Плюси:

- Дуже добре підходить для CI / CD систем.
- Економить місце.
- Існує багато готових образів докера для популярного ПЗ.
- Заощаджує час на доставку патчів та простої в порівнянні з віртуалізацією.

- Працюючи в команді, вам не потрібно турбуватися про те, що різні учасники мають різні версії мови програмування, бібліотеки тощо.
- Відкритий вихідний код.
- Для покращення його функцій доступно безліч плагінів.
- Можна запускати на Linux, Windows та Mac OS.

Мінуси:

- Досить важко побудувати контейнер з нуля.
- На вивчення цього інструменту потрібно досить багато часу.
- Створення постійного сховища вимагає великих зусиль.
- Не має графічного інтерфейсу користувача.

2.2.2 LXC (Linux Containers)

LXC забезпечує віртуалізацію на рівні операційної системи за допомогою віртуального середовища, яке має власний простір процесів та мережевий простір, замість створення повноцінної віртуальної машини. LXC покладається на функціонал cgroups ядра Linux, випущений у версії 2.6.24. Він також покладається на інші види ізоляції простору імен, які були розроблені та інтегровані в основне ядро Linux. LXC подібний до інших технологій віртуалізації на рівні ОС в Linux, таких як OpenVZ та Linux-VServer, а також до подібних технологій в інших операційних системах, наприклад FreeBSD jails, AIX Workload Partitions та контейнери Solaris. На відміну від OpenVZ, LXC працює у чистому ядрі Linux, не вимагаючи додаткових патчів ядра. LXC написаний на мовах C, Python, Shell та Lua.

Рис. 2.5 Логотип LXC

Постає питання, чим LXC відрізняється від Docker'а та іншого програмного забезпечення для контейнеризації? LXC - це технологія контейнерів, яка дає легкі Linux-контейнери, а Docker - це механізм віртуалізації одного додатку, заснований на контейнерах. Вони можуть здаватися схожими, але по суті є абсолютно різними.

На відміну від контейнерів LXC, контейнери Docker не поводяться як легкі віртуальні машини і не можуть використовуватись таким чином. Конструкція контейнерів Docker обмежена лише одним додатком. Можна увійти до свого контейнера LXC, поводитися з ним як з ОС, встановлюючи програми та служби, і вони будуть працювати, як очікувалося. Цього не можна цього зробити в контейнері Docker. Шаблон базової ОС Docker зведений до середовища для єдиної програми і не підтримує такі речі, як служби, демони, системний журнал, cron або запуск декількох додатків.

Це підводить нас до думки, що технологія LXC досить сильно схожа на технології віртуалізації. Так, але технологія віртуалізації несе набагато більше витрат в плані продуктивності системи і є менш гнучкою. Контейнери не

імітують апаратний рівень і використовують групи та простори імен у ядрі Linux для створення легких віртуалізованих середовищ ОС із швидкістю роботи близької до «рідної» операційної системи без віртуалізації. Оскільки ви не віртуалізуєте сховище, контейнеру байдуже на тип диску чи файлові системи, він просто працює там, де його розміщено. Це принципово змінює спосіб віртуалізації робочих навантажень та додатків, оскільки контейнери просто швидші, портативніші та можуть масштабуватися ефективніше, ніж апаратна віртуалізація, за винятком робочих навантажень, які потребують ОС, відмінної від Linux, або певної версії ядра Linux.

Плюси:

- Відкритий вихідний код.
- Швидше і дешевше, ніж віртуалізація.
- Можливість розгортання великої кількості контейнерів.

Мінуси:

- Порівняно менш безпечний, ніж інші методи віртуалізації на рівні ОС.
- Під LXC можна виконувати лише контейнери Linux. Немає підтримки Windows, Mac або інших ОС.

2.3 Сканування вразливостей програмного забезпечення.

Маючи час, ресурси та мотивацію, зловмисник може проникнути майже у будь-яку систему. Зрештою, всі доступні в даний час процедури та технології безпеки не можуть гарантувати, що система повністю захищена від проникнення. Маршрутизатори допомагають захистити шлюзи до мережі Інтернету. Брандмауери допомагають захистити край мережі. Віртуальні приватні мережі дозволяють передавати дані в зашифрованому потоці. Системи виявлення

вторгнень попереджають вас про зловмисну діяльність. Однак успіх кожної з цих технологій залежить від ряду змінних, зокрема:

- Досвіду персоналу, відповідального за налаштування, моніторинг та підтримку технологій.
- Можливості швидко та ефективно виправляти та оновлювати служби та ядра.
- Здатності відповідальних осіб підтримувати постійну пильність щодо того що відбувається у мережі.

З огляду на динамічний стан інформаційних систем і технологій, забезпечення захищеності корпоративних ресурсів може бути досить складним завданням. Через цю складність часто важко знайти експертів для управління всіма системами. Незважаючи на те, що можливо мати висококваліфікований персонал у багатьох сферах інформаційної безпеки, важко утримати співробітників, які є експертами у кількох предметних областях. Це відбувається головним чином тому, що кожна предметна область інформаційної безпеки вимагає постійної уваги та зосередженості. Сфера інформаційної безпека не стоїть на місці.

Припустимо, що ми адмініструємо корпоративну мережу. Такі мережі зазвичай складаються з операційних систем, додатків, серверів, моніторів мережі, брандмауерів, систем виявлення вторгнень тощо. Враховуючи складність сучасного програмного забезпечення та мережевого середовища, вразливості та помилки у роботі системи виникнуть із великою імовірністю. Тримати темп виправлення та оновлення всієї мережі являє собою нетривіальну задачу у великій організації з неоднорідними системами.

Поєднуючи вимоги до знань із завданням підтримувати поточний стан мережі неминуче трапляються несприятливі інциденти, зловмисники проникають у системи, пошкоджуються дані та виникають відмови в обслуговуванні. Профілактична оцінка вразливостей власних систем та

мережевих ресурсів може виявити потенційні проблеми, які можна вирішити до того, як зловмисник зможе їх використати.

Оцінка вразливості - це процес виявлення, кількісної оцінки та встановлення пріоритетів (або ранжування) вразливостей у системі. Приклади систем, для яких проводиться оцінка вразливості, включають(але не обмежуються) такими системами: системи інформаційних технологій, системи енергопостачання, системи водопостачання, транспортні системи та системи зв'язку. Такі оцінки можуть проводитись від імені цілого ряду різних організацій, від малого бізнесу до великої регіональної інфраструктури.

Оцінка вразливості має багато спільного з оцінкою ризику. Оцінки зазвичай проводяться згідно з наступною схемою:

1. Каталогізація активів та ресурсів у системі.
2. Призначення кількісного значення (або принаймні порядку ранжування) та важливості цих ресурсів
3. Виявлення вразливостей або потенційних загроз для кожного ресурсу
4. Пом'якшення або усунення найсерйозніших вразливостей для найцінніших ресурсів

У сфері інформаційних технологій є своя специфіка, але все працює приблизно за тією ж самою схемою. Від так, під оцінкою вразливості в IT розуміють процес визначення, виявлення та класифікації дір у безпеці в системах інформаційних технологій.

Регулярна оцінка вразливостей - це спосіб оцінки вразливості інфраструктури компанії. У такий спосіб компанії отримують інформацію про наявні ризики, що дає змогу їх оцінити та вжити відповідних заходів.

Під час сканування мережі можна оцінити мережу на наявність відомих вразливостей програмного забезпечення. Цей процес дозволяє виявити всі системи в мережі, визначає, які мережеві служби використовуються, а потім

аналізує ці служби на предмет можливих вразливостей. Цей процес не вимагає змін конфігурації в системах, що оцінюються. Оцінка мережі, коли поставлена «на потік» вимагає невеликих обчислювальних витрат та зусиль.

Також варто відмітити, що не зважаючи на те, що процес сканування вразливостей програмного забезпечення має багато спільного з тестуванням на проникнення, це два різні методи що відрізняються кількома критичними параметрами (див. табл. 2.1).

Таблиця 2.1

Відмінності між скануванням вразливостей ПЗ та тестуванням на проникнення

Параметри	Сканування вразливостей ПЗ	Тест на проникнення
Як часто проводиться	Постійно, особливо після введення в експлуатацію нового обладнання	Раз на рік
Звіти	Всеосяжний звіт що включає всі виявлені вразливості, та зміни в порівнянні з останнім звітом	Коротко і по суті, визначає, які дані насправді були скомпрометовані
Метрики	Перелічує відомі вразливості програмного забезпечення, які можуть бути використані зловмисником	Виявляє невідомі вразливості бізнес процесів
Ким виконується	Співробітниками компанії	Незалежною зовнішньою службою
Цінність	Реактивний контроль, який використовується для виявлення скомпрометованого обладнання	Превентивний контроль, що використовується для знаходження потенційних та реальних вразливостей

Отже сканування мережі є критичною складовою аналізу вразливостей мережі. Саме у процесі сканування здобуваються усі необхідні дані задля подальшої оцінки стану мережевої інфраструктури в цілому, та її конкретних компонентів.

Сканер вразливості починає сканування як агент, який збирає інформацію для подальшої атаки. База даних нових та старих загроз безпеці зберігається та

регулярно оновлюється, щоб програмне забезпечення могло перевірити та порівняти деталі атаки з базою даних. Сканер перевіряє точки входу, через які хакери можуть отримати доступ до мережі, а саме програми, служби, порти та помилки в побудові інфраструктури. Розглянемо детальніше типовий сценарій сканування мережі:

- Крок перший: Пошук «живих» хостів у мережі

Сканер перевіряє чи цільова система онлайн. Для цього використовуються різні методи розглянуті в розділі 1.3 цієї роботи. У цей же час сканер намагається виявити системи сховані за фаерволом.

- Крок другий: Виявлення фаерволів

На цьому кроці сканер намагається визначити чи дійсно системи помічені як сховані за фаерволом у першому кроці знаходяться офлайн, чи цільова система захищена за допомогою брандмауера, системою IDS або IPS, проте все ще відкрита для атаки.

- Крок третій: сканування портів TCP та UDP

Сканер намагається визначити відкриті порти та сервіси цільового хоста. В залежності від заданої інтенсивності сканування можуть бути апробовані яка усі 65535 наявні порти, так і лише деякі з них. При типовому сценарії оцінки ризиків гарною практикою є використання списку відомих портів для підвищення швидкості сканування та зменшення навантаження на мережу.

- Крок четвертий: Виявлення ОС та сервісів
- Крок п'ятий: Оцінка сервісів та їх версій, пошук відомих вразливостей

Спираючись на інформацію про версію операційної системи та її сервісів сканер запускає пошук відомих вразливостей та експлойтів для цих сервісів.

- Крок шостий: Генерація звіту

Беручи до уваги усе згадане вище, надійна процедура аналізу вразливостей повинна мати такі характеристики:

1. Сканування кожного хоста мережі: потрібно знати про всі активи у середовищі. Простіше виявити вразливі місця маючи повні знання про всі системи у мережевому середовищі.
2. Регулярність сканування: зі збільшенням кількості виявлених в останні роки вразливостей, розумно проводити сканування як мінімум раз на тиждень.
3. Налагоджений процес виправлення вразливостей: виявленим вразливостям потрібно виставити пріоритет та у першу чергу виправляти критичні вразливості з найбільшим впливом на систему.
4. Генерація та зберігання звітів про запуск сканування: документування результатів сканування та процесу виправлення вразливостей допоможе у процесі усунення неполадок та у проведенні аудиту мережі.

2.4 Огляд існуючих систем сканування мережі на наявність вразливого програмного забезпечення.

2.4.1 Nessus

Проект Nessus був започаткований Рено Дерайсоном у 1998 році, щоб надати Інтернет-спільноті безкоштовний віддалений сканер безпеки. 5 жовтня 2005 року Tenable Network Security, співзасновником якої є компанія Renaud Deraison, змінила Nessus 3 на власну (закриту) ліцензію. Nessus 2 і незначна частина плагінів залишилась під ліцензією GPL, що призвело до розгалуження проекту з відкритим кодом на базі Nessus, OpenVAS. Сьогодні продукт все ще існує у двох форматах; обмежена, безкоштовна версія та повнофункціональна доступна по платній підписці. Nessus доступний для Linux, Windows та Mac OS X.

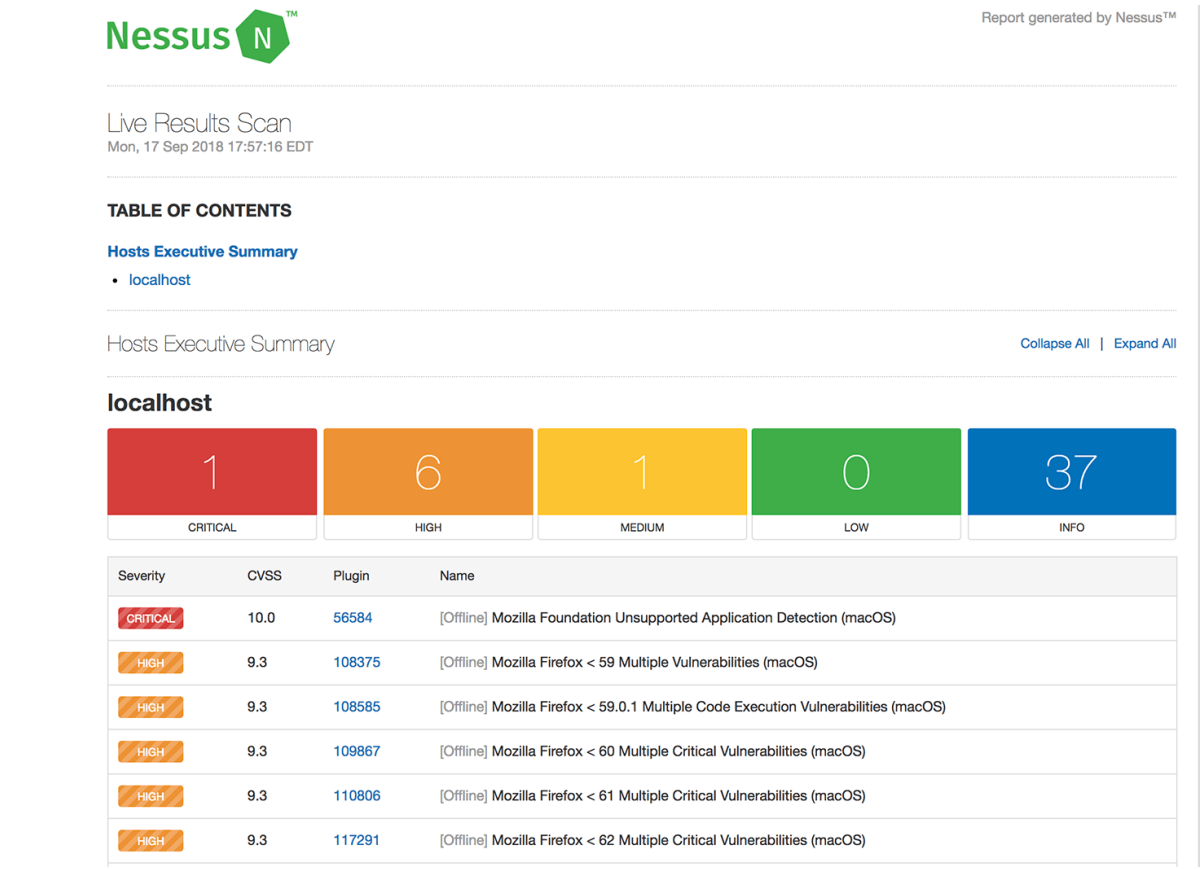
Рис. 2.6 Логотип Nessus

Приклади вразливостей які Nessus може просканувати, включають:

- Уразливості, які можуть дозволити несанкціонований контроль або доступ до конфіденційних даних у системі.
- Неправильна конфігурація (наприклад, відсутні патчі тощо).
- Паролі за замовчуванням, кілька загальних паролів та порожні / відсутні паролі для деяких системних облікових записів. Nessus також може використовувати Hydra (зовнішній інструмент) для запуску атаки по словнику.
- Вразливості відмови у обслуговуванні.

Сканування за допомогою Nessus покриває широкий спектр технологій, включаючи операційні системи, мережеві пристрої, гіпервізори, бази даних, веб-сервери та критичну інфраструктуру.

Результати сканування можуть бути представлені у різних форматах, таких як звичайний текст, XML, HTML та LaTeX. Результати також можна зберегти у базі знань для налагодження. Результати сканування легко кастомізуються під потребі користувача (див. рис. 2.7, рис. 2.8)



Nessus™

Report generated by Nessus™

Live Results Scan
Mon, 17 Sep 2018 17:57:16 EDT

TABLE OF CONTENTS

Hosts Executive Summary
- localhost

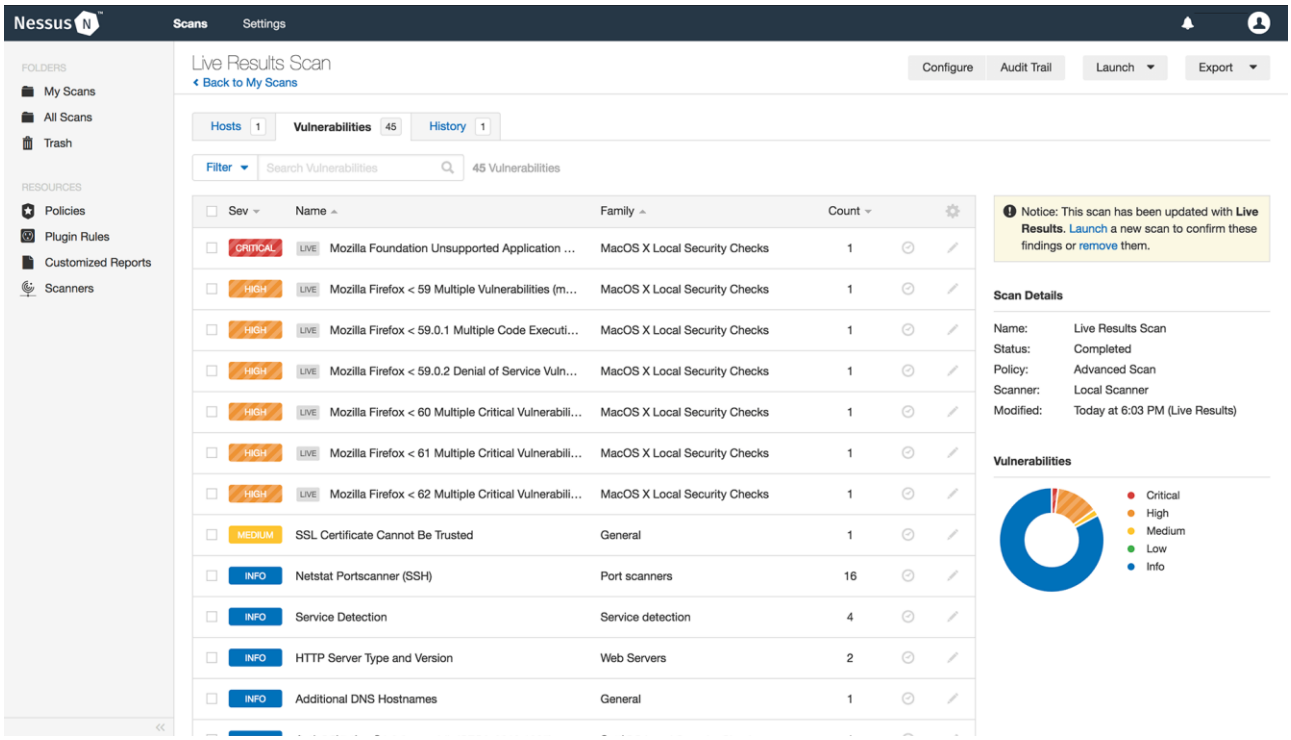
Hosts Executive Summary — Collapse All | Expand All

localhost

1	6	1	0	37
CRITICAL	HIGH	MEDIUM	LOW	INFO

Severity	CVSS	Plugin	Name
CRITICAL	10.0	56584	[Offline] Mozilla Foundation Unsupported Application Detection (macOS)
HIGH	9.3	108375	[Offline] Mozilla Firefox < 59 Multiple Vulnerabilities (macOS)
HIGH	9.3	108585	[Offline] Mozilla Firefox < 59.0.1 Multiple Code Execution Vulnerabilities (macOS)
HIGH	9.3	109867	[Offline] Mozilla Firefox < 60 Multiple Critical Vulnerabilities (macOS)
HIGH	9.3	110806	[Offline] Mozilla Firefox < 61 Multiple Critical Vulnerabilities (macOS)
HIGH	9.3	117291	[Offline] Mozilla Firefox < 62 Multiple Critical Vulnerabilities (macOS)

Рис. 2.7 Приклад згенерованого репорту після сканування за допомогою Nessus



Nessus — Scans — Settings

FOLDERS
- My Scans
- All Scans
- Trash

RESOURCES
- Policies
- Plugin Rules
- Customized Reports
- Scanners

Live Results Scan
‹ Back to My Scans

Configure | Audit Trail | Launch | Export

Hosts 1 | Vulnerabilities 45 | History 1

Filter | Search Vulnerabilities | 45 Vulnerabilities

Sev	Name	Family	Count
CRITICAL	Mozilla Foundation Unsupported Application ...	MacOS X Local Security Checks	1
HIGH	Mozilla Firefox < 59 Multiple Vulnerabilities (m...	MacOS X Local Security Checks	1
HIGH	Mozilla Firefox < 59.0.1 Multiple Code Executi...	MacOS X Local Security Checks	1
HIGH	Mozilla Firefox < 59.0.2 Denial of Service Vuln...	MacOS X Local Security Checks	1
HIGH	Mozilla Firefox < 60 Multiple Critical Vulnerabili...	MacOS X Local Security Checks	1
HIGH	Mozilla Firefox < 61 Multiple Critical Vulnerabili...	MacOS X Local Security Checks	1
HIGH	Mozilla Firefox < 62 Multiple Critical Vulnerabili...	MacOS X Local Security Checks	1
MEDIUM	SSL Certificate Cannot Be Trusted	General	1
INFO	Netstat Portscanner (SSH)	Port scanners	16
INFO	Service Detection	Service detection	4
INFO	HTTP Server Type and Version	Web Servers	2
INFO	Additional DNS Hostnames	General	1

Notice: This scan has been updated with **Live Results**. Launch a new scan to confirm these findings or remove them.

Scan Details

Name:	Live Results Scan
Status:	Completed
Policy:	Advanced Scan
Scanner:	Local Scanner
Modified:	Today at 6:03 PM (Live Results)

Vulnerabilities

Critical
High
Medium
Low
Info

Рис. 2.8 Приклад згрупованого по категоріям результату сканування

У UNIX сканування можна автоматизувати за допомогою командного рядка. Для управління окремими або розподіленими сканерами Nessus існує багато різних комерційних, безкоштовних та відкритих інструментів як для UNIX, так і для Windows.

Nessus надає додаткову функціональність, крім тестування на відомі вразливості мережі. Наприклад, він може використовувати облікові дані Windows для перевірки рівнів виправлень на комп'ютерах під управлінням операційної системи Windows. Nessus також може підтримувати аудит конфігурації та відповідності, аудит SCADA та відповідність PCI.

Nessus, насамперед, використовується для сканування портів і визначає сервіси, що їх використовують. Для тестування вразливостей використовуються спеціальні плагіни, написані на мові NASL (Nessus Attack Scripting Language). Наразі існує більше 70 000 таких плагінів. Також проводиться перевірка сервісів по базі вразливостей. База вразливостей оновлюється щотижня, проте для комерційних передплатників є можливість завантаження нових плагінів без семиденної затримки. При відключеній опції «safe checks» деякі тести на уразливості, що використовуються Nessus можуть призвести до порушень в роботі сканованих систем.

Ще однією перевагою Nessus є продуманий інтерфейс користувача. Розпочати своє перше сканування без попереднього досвіду роботи з програмою можна вже за хвилину після встановлення сканера (див рис. 2.9). Проте для того щоб повністю оволодіти ним знадобиться багато часу. Nessus має дуже гнучку систему налаштувань для роботи у середовищі будь-якої складності.

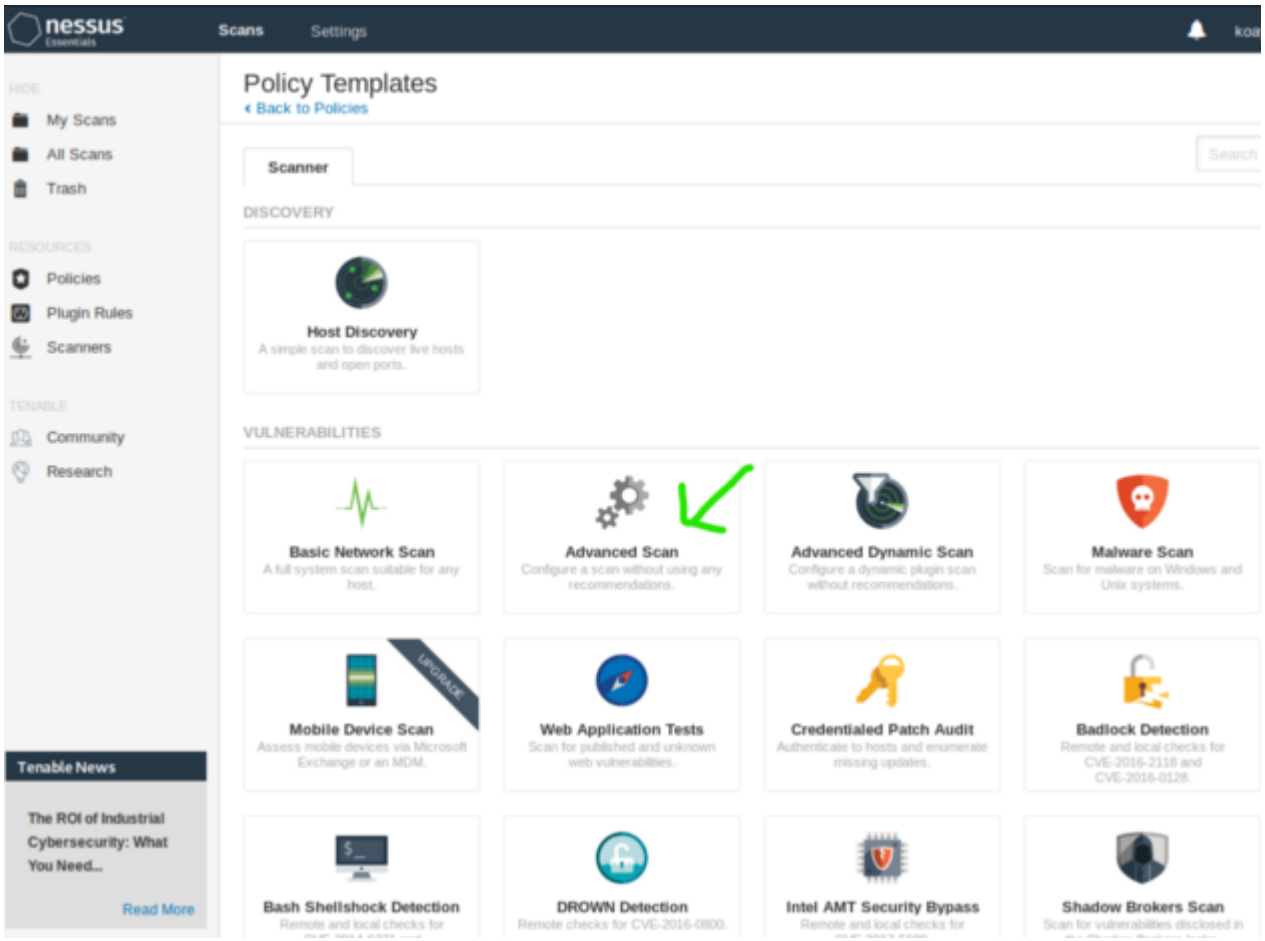


Рис. 2.9 Приклад інтерфейсу користувача сканера Nessus

2.4.2 Rapid7 Nexpose

Rapid7 була заснована в 2000 році і протягом багатьох років її діяльність зосереджувалась на безпеці даних та аналітичних технологіях, включаючи менеджмент вразливостей. Продуктовий портфель компанії включає Nexpose, один з найстаріших менеджерів вразливостей. Rapid7 також розробляє Metasploit для тестування на проникнення, AppSpider для сканування безпеки веб-додатків, UserInsight для аналізу поведінки зловмисників, Logentries для вдосконаленого управління журналами, а також різні професійні послуги на основі продуктів компанії.

Nexpose працює у фізичному, віртуальному, хмарному та мобільному середовищах, щоб виявляти активи та шукати вразливості, після чого визначає

пріоритети ризиків, заснований на можливості використання зловмисниками цих вразливостей у середовищі організації. Продукт також надає пріоритет виправленню вразливостей і дозволяє адміністраторам планувати сканування та налаштовувати сповіщення безпеки.

Rapid7 Nexpose націлений у першу чергу на моніторинг та усунення вразливостей у великих корпоративних мережах. Через високу ціну(16000$ за 128 ip в версії Ultimate), обмеження Home версії продукту та малу кількість інформації у відкритих джерелах повноцінно протестувати цей менеджер вразливостей не вдалося.

2.4.3 Nmap

Nmap - це безкоштовний мережевий сканер із відкритим кодом, створений Гордоном Ліоном (відомий також під псевдонімом Федір Васькович). Nmap використовується для виявлення хостів та служб у комп’ютерній мережі шляхом надсилання пакетів та аналізу відповідей. Nmap надає ряд функцій для перевірки комп'ютерних мереж, включаючи виявлення хостів, сервісів та операційних систем. Ці функції можна розширити за допомогою скриптів, що забезпечують розширене виявлення сервісів, виявлення вразливостей та інші функції. Nmap може адаптуватися до мережевих умов, включаючи затримку та перевантаження під час сканування.

В контексті сканування мереж на наявність вразливого програмного забезпечення Nmap привертає увагу своїм движком NSE скриптів завдяки яким можна розширити його функціонал до рівня повноцінного сканера вразливостей.

Має графічний інтерфейс як окремий додаток Zenmap розроблений Адріано Монтейро Марком. Також існує веб інтерфейс, що дозволяє запускати сканування з браузера. Є кроссплатформеним та працює у Linux, Windows, Mac OS, Android та iOS.

Nmap розроблявся як інструмент для дослідження мереж тому він не має вбудованих опцій для запуску за розпорядком, або виводу у графічний формат як pdf або html. Його ефективне використання потребує вивчення документації яку можна знайти за посиланням nmap.org або запустивши додаток з флагом --help. Nmap надає чотири можливі формати виводу. Усі, крім інтерактивного виводу, зберігаються у файлі. Виведенням Nmap можна керувати за допомогою програмного забезпечення для обробки тексту, що дозволяє користувачеві створювати індивідуальні звіти.

- Інтерактивний: представлений та оновлюваний у реальному часі, коли користувач запускає Nmap із командного рядка. Під час сканування можна ввести різні параметри для полегшення моніторингу.
- XML: формат, який може бути додатково оброблений інструментами XML. Його можна перетворити у звіт HTML за допомогою XSLT.
- Grepable: ввід, призначений для построчних інструментів обробки, таких як grep, sed або awk.
- Звичайний: вивід, який видно під час запуску Nmap із командного рядка, але збережений у файл.
- Script kiddie: мав бути кумедним способом форматування інтерактивного виводу, замінюючи літери на їх візуально однакові цифрові подання. Наприклад, Interesting ports стає Int3rest1ng p0rtz. Це нотація відома як Leet.

2.4.4 OpenVAS

The Open Vulnerability Assessment System (OpenVAS) - це програмна база декількох служб для менеджменту вразливостей. Це безкоштовний інструмент з відкритим вихідним кодом, який Greenbone Networks підтримує з 2009 року. Створений як універсальний сканер, він працює із репозиторієм з понад 50 000 тестів на вразливість, що оновлюється щодня. Цей безкоштовний сканер вразливостей, розроблений спеціально для роботи в середовищі Linux, є гарним

варіантом для досвідчених користувачів, які хочуть виконати цільове сканування. Встановлення та використання його має значну криву навчання (див. рис. 2.11), і з цієї причини він не є відповідним інструментом для більшості адміністраторів мережі. Greenbone також пропонує платний продукт із більш регулярними оновленнями, гарантіями обслуговування та підтримкою клієнтів.

OpenVAS розпочав свою історію під назвою GNessUs як форк раніше відкритого інструменту сканування Nessus після того, як його розробники Tenable Network Security в жовтні 2005 р. змінили його ліцензію на власну (закриту). OpenVAS був спочатку запропонований пентестерами в SecuritySpace, обговорювався з пентестерами з Portcullis Computer Security, а потім Тім Браун анонсував його на Slashdot.

OpenVAS являється учасником проекту Software in the Public Interest.

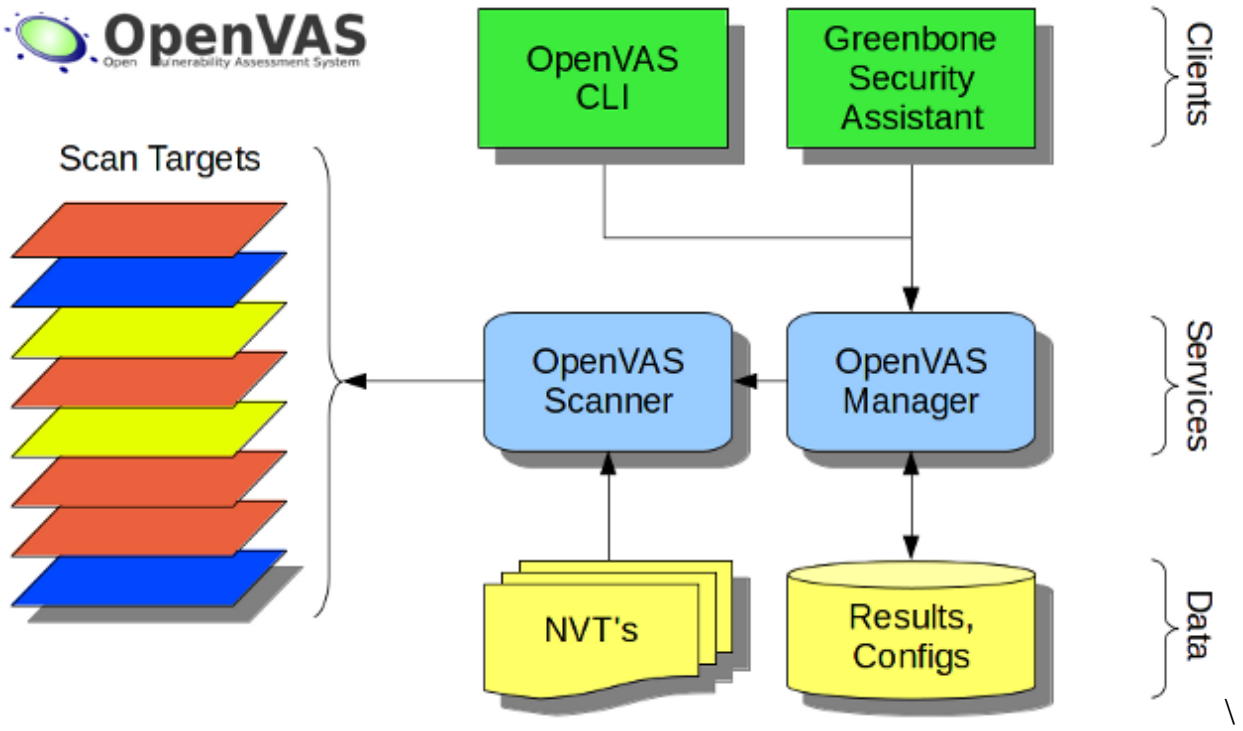


Рис. 2.10 Структура OpenVAS

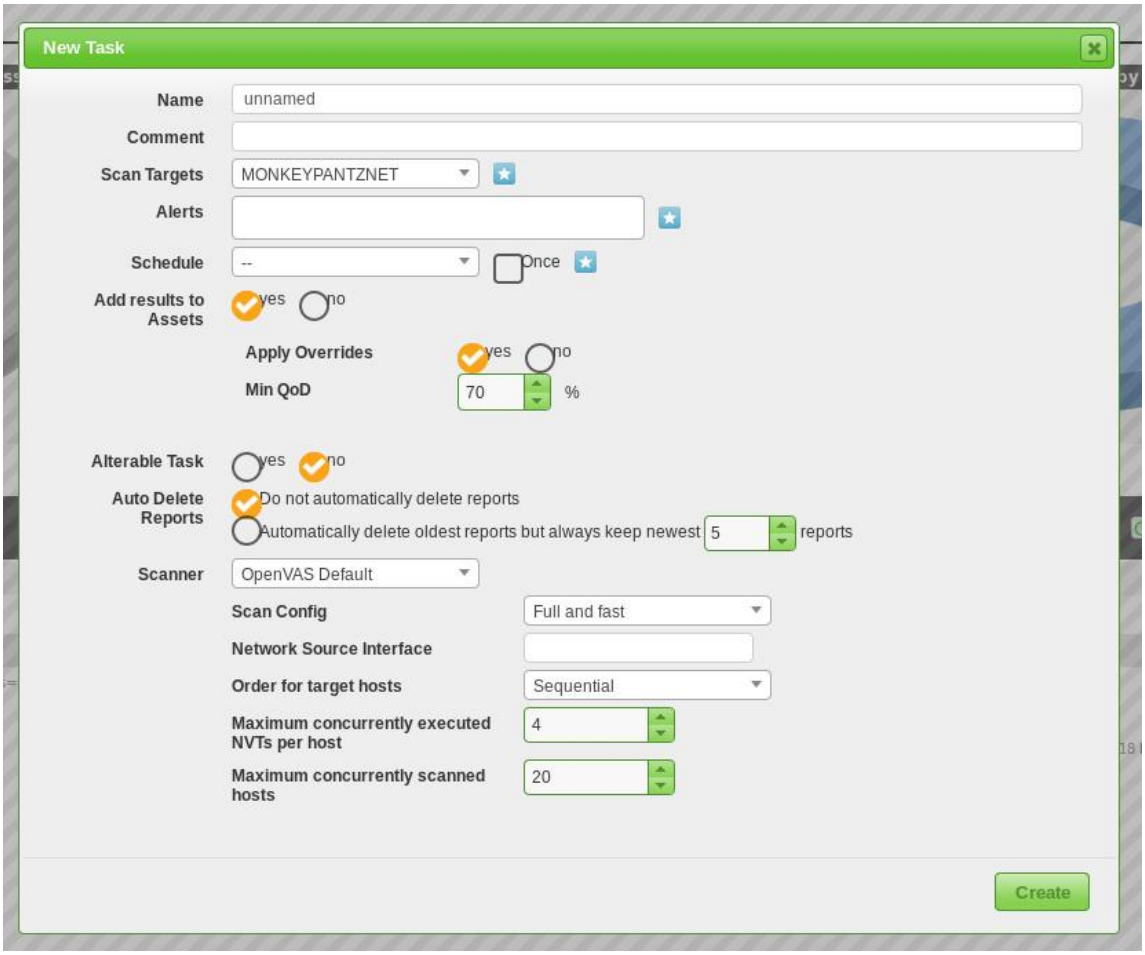


Рис. 2.11 Інтерфейс створення нової задачі OpenVAS

В плані функціоналу OpenVAS мало чим поступається розглянутому вище Nessus окрім значно менших можливостей з кастомізації звітів та інтерфейсу користувача.

Таблиця 2.2

Порівняльна таблиця сканерів мережі на наявність вразливого ПЗ

ПЗ	Графічний інтерфейс	User-friendly	Ліцензія	Генерація звітів	Запуск сканування за розпорядком	ОС	Вимогливість до ресурсів
Nessus	+	+	+	+	+	Кроссплатформенний	+
Nexpose	+	+	+	+	+	Кроссплатформенний	+
nmap	-	-	-	-	-	Кроссплатформенний	-
OpenVAS	+	-	-	+	+	Linux	+

З таблиці 2.2 можна зробити висновок, що дружні до користувача open-source рішення для сканування мережі на наявність вразливого ПЗ з підтримкою генерації звітів та сканування за розпорядком наразі відсутні.

2.5 Висновки до розділу.

У розділі було розглянуто та проаналізовано існуючі рішення у сфері контейнеризації та сканування мережі на наявність вразливого ПЗ. Було розглянуто мотивацію використання та принципи роботи систем контейнеризації програмного забезпечення. Також було проведено порівняльний аналіз характеристик існуючих систем сканування мережі на наявність вразливого ПЗ. Та розглянуто мотивацію використання таких систем.

Підсумовуючи викладене у розділі, можна зробити наступні висновки:

1) Контейнери та технологія контейнеризації може бути використана для швидкої розгортки програмних додатків незалежно від реального середовища у якому вони використовуються. Це дозволяє легко мігрувати такі додатки між серверами та середовищами не витрачаючи час та сили на пошук проблем із середовищем виконання. Що у свою чергу зменшує час протягом сервіс буде недоступний для користувачів. Також контейнери набагато «дешевші» в плані ресурсів ніж технологія віртуалізації що дозволяє розгорнути набагато більше ізольованих сервісів при менших витратах ресурсів.
2) Основними технологіями контейнеризації що доступні користувачу є LXC та Docker-контейнери. Оскільки Docker на відміну від LXC є кроссплатформеним, він краще підходить для наших задач. LXC по суті є віртуальною машиною на реальному залізі та потребує більше специфічних знань для розгортки та управління.
3) Сканування вразливостей програмного забезпечення має бути неперервним процесом. У сучасному світі спеціалісти з безпеки майже кожного дня знаходять вразливості програмного забезпечення, отже моніторинг та менеджмент вразливостей є важливою задачею для кожної компанії.
4) З аналізу існуючих систем сканування мережі на наявність вразливого програмного забезпечення випливає, що дружні до користувача open-source рішення для сканування мережі на наявність вразливого ПЗ з підтримкою генерації звітів та сканування за розпорядком наразі відсутні.

Розділ 3. РОЗРОБКА СИСТЕМИ СКАНУВАННЯ МЕРЕЖІ НА НАЯВНІСТЬ ВРАЗЛИВОГО ПЗ. ЇЇ ТЕСТУВАННЯ.

3.1 Формулювання вимог до системи сканування мережі на наявність вразливого ПЗ.

Виходячи з аналізу наявних рішень у сфері сканування мережі на наявність вразливого програмного забезпечення можна сформулювати наступні вимоги:

1) Система має базуватися лише на програмному забезпеченні з відкритим вихідним кодом використання якого не обмежується ліцензією.

2) Система повинна мати user-friendly інтерфейс та не потребувати спеціальних знань для її впровадження та використання

3) Система повинна швидко розгортатися у будь якому середовищі, бути кросплатформеною.

4) Можливість проведення регулярних сканувань має бути вбудована у сиситему.

5) Результати сканування мають бути легкими для аналізу та містити данні що дозволяють визначити ступінь загрози кожної окремо взятої вразливості.

6) Результати мають бути оформлені у звіт в форматі pdf.

7) Окрім самої вразливості звіт має містити до якого хоста та програмного забезпечення вона відноситься.

8) Звіт повинен в автоматичному режимі розсилатись відповідальним особам.

З огляду вимог можна зробити висновок, що система потребує підтримки контейнеризації, а саме Docker контейнерів. Це дозволить легко та швидко розгорнути її у будь якому середовищі. Також система потребує засобу швидкої конфігурації в інтерактивному форматі, що дозволить використання системи без витрат часу на аналіз складної документації. Мова розробки додатку має бути

кросплатформеною, отже гарним кандидатом може стати мова програмування Python. Її стандартна бібліотека дозволить мінімізувати використання сторонніх бібліотек. Гарним кандидатом на платформу для розсилки репортів та управління скануванням може стати Telegram. Його політика у сфері конфіденційності, надійність роботи, захищеність інфраструктури та широкі можливості по інтеграції з іншими сервісами роблять його привабливою платформою для багатьох компаній.

Отже система буде написана на мові програмування Python, керування системою та розсилка звітів буде відбуватись за допомогою платформи ботів Телеграм. Система буде потребувати мінімум завчасної конфігурації та знань для розгортки. Систему можна буде розгорнути за допомогою технології контейнеризації на основі контейнерів Docker у будь-якому середовищі.

3.2 Схема роботи системи

3.2.1 Діаграма станів.

Система може знаходитись у невеликій кількості станів відображених у таблиці 3.1 та на рисунку 3.1

Таблиця 3.1

Можливі стани системи

Waiting	Це стан у якому система очікує на команди від адміністратора системи
Scanning	Це стан у якому система сканує мережу на наявність вразливого ПЗ
Generating Report	Це стан у якому система аналізує отримані данні, структурує їх та перетворює на звіт в форматі pdf
Sending Reports to subscribers	Це стан у якому система розсилає згенерований звіт аутентифікованим користувачам.

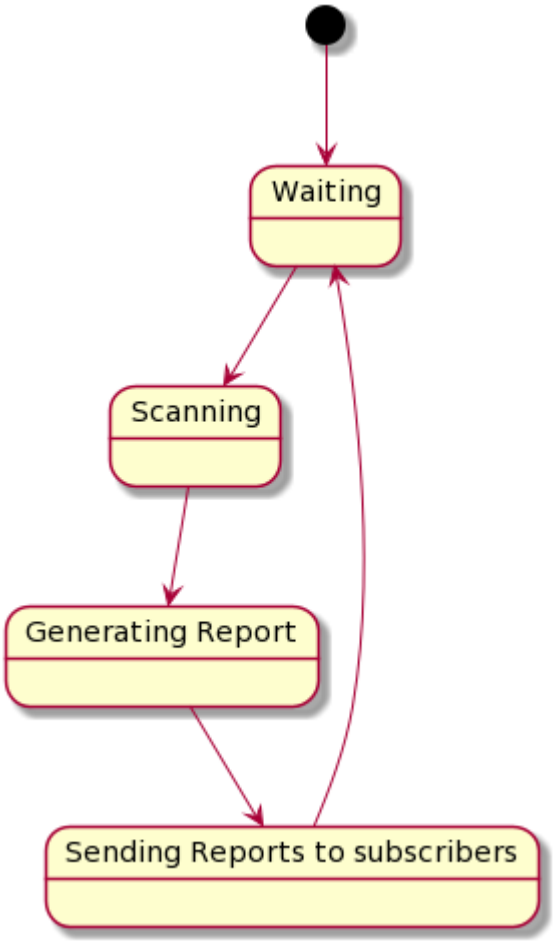


Рис. 3.1 Діаграма станів системи

3.2.2 Діаграма послідовності.

Тепер, коли ми визначилися із станами у яких перебуватиме система, розглянемо типові сценарій взаємодії підписників та адміністратора з системою.

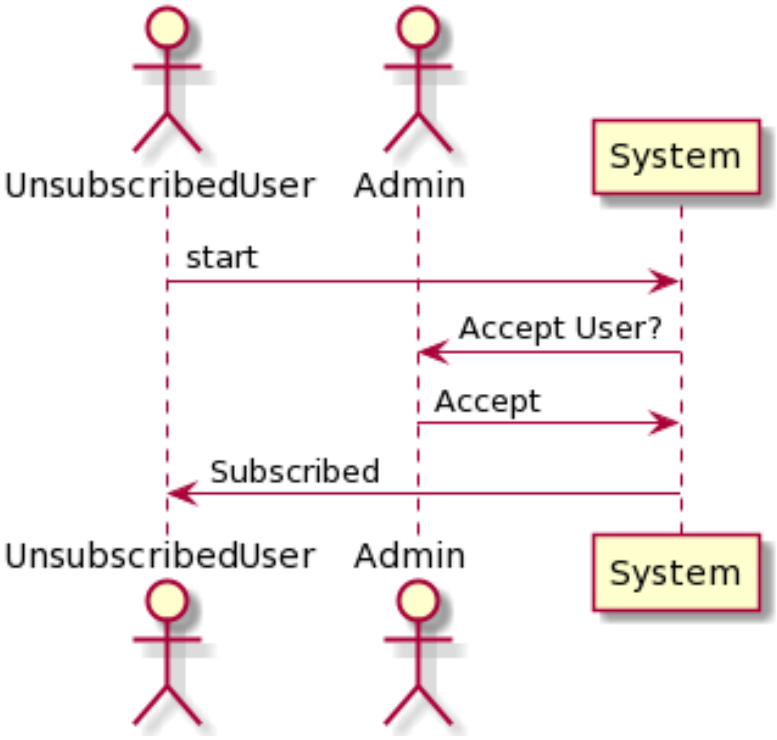


Рис. 3.2 Діаграма послідовності подій при авторизації нового підписника

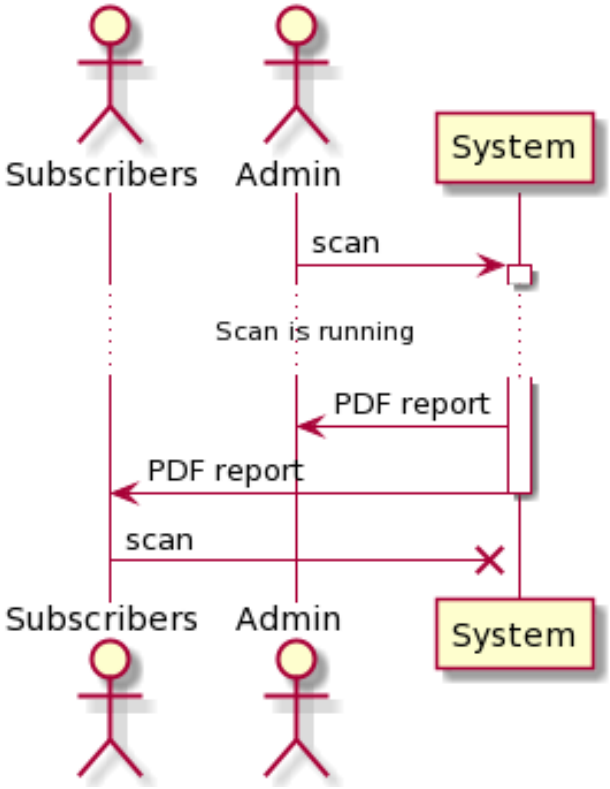


Рис. 3.3 Діаграма послідовності взаємодій підписників та адміністратора з системою.

Як можна побачити з діаграми уся взаємодія з системою буде відбуватись через адміністратора, без його дозволу користувач не отримає доступу до розсилки та не зможе ініціювати сканування. Термін підписник є найбільш доречним у даному контексті тому що права управління системою є тільки у адміністратора, інші ж кристувачі лише отримують звіти «за підпискою».

3.2.3 Діаграма класів.

При розробці сучасного програмного забезпечення зазвичай застосовується об’єктно-орієнтований підхід до програмування який передбачає абстрагування від імплементації, наявність рівнів абстракції та чітке розділення функцій програми між певними об’єктами-класами кожен з яких виконує одну певну задачу. Побудуємо діаграму класів програмного додатка.

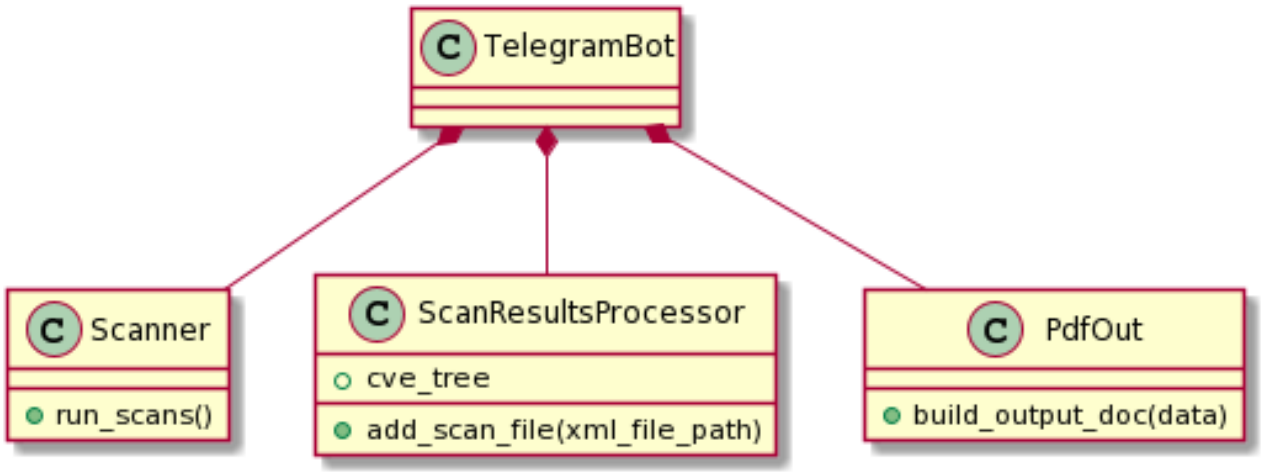


Рис. 3.3 Діаграма класів системи

Таблиця 3.2

Розбір діаграми класів

Клас	**Функціонал**
TelegramBot	Клас відповідає за комунікацію з серверами Телеграм, обробку команд від адміністратора та використовує інші класи системи для забезпечення обробки запитів від користувачів.
Scanner	Клас сканування буде відповідати за функціонал сканування мережі на наявність вразливого програмного забезпечення.
ScanResultsProcessor	Клас відповідає за обробку результатів сканування та структурує дані таким чином, щоб їх можна було зручно внести у звіт.
PdfOut	Клас відповідає за вивід попередньо організованих результатів сканування у формат pdf.

3.3 Контейнеризація системи

Контейнеризація системи перш за все потребує знання усіх залежностей системи. Приведемо список цих залежностей згрупований за їх типом:

- Python бібліотеки та інструменти
  - Pytelegrambotapi – зручна асинхронна бібліотека для комунікації з API платформи Телеграм.
  - mdpdf – інструмент конвертації Markdown документів у формат pdf.

- Системні залежності
  - Python3 – інтерпретатор мови програмування Python
  - Python3 pip – пакетний менеджер для керування залежностями для мови програмування Python.
  - Nmap – сканер мережі що завдяки системі NSE скриптів може слугувати як фреймворк для проведення сканування та аналізу мереж
  - Git - розподілена система керування версіями.

Для спрощення розгортки, залежності мови програмування Python вносять у файл requirements.txt який зазвичай залишають в корені репозиторія. Цей файл містить список залежностей який може бути швидко встановленний за допомогою пакетного менеджера для керування залежностями pip. Кожен окремий пакет пишеться з нової строки.

Маючи повний список залежностей системи можна приступити до побудови образу контейнера. Для цього в корені репозиторія проекту потрібно створити файл з іменем Dockerfile. У нашому випадку він буде виглядати так (див. рис. 3.4).

```
1  FROM ubuntu:latest
2
3  RUN apt-get update -y
4  RUN apt-get install -y python3-pip python3-dev build-essential nmap git
5  RUN git clone https://github.com/vulnersCom/nmap-vulners.git /usr/share/nmap/scripts/vulners
6  RUN nmap --script-updatedb
7  COPY . /app
8  WORKDIR /app
9  RUN pip3 install -r requirements.txt
10 ENTRYPOINT ["python3", "-u", "telegramBot.py"]
```

Рис. 3.4 Dockerfile для системи

Розберемо його построчно. На першій строчці (FROM ubuntu:latest) ми вказуємо докеру який образ потрібно взяти за основу. У нашому випадку це остання версія дистрибутиву Ubuntu Linux.

На третій строчці (RUN apt-get update -y) ми запускаємо команду для оновлення системних пакетів за допомого пакетного менеджеру apt що використовується у всіх Debian-подібних дистрибутивах Linux.

Наступним кроком за допомогою того ж самого менеджеру ми встановлюємо необхідне ПЗ, а саме інтерпретатор мові програмування Python та його пакетний менеджер pip, необхідні інструменти для зборки ПЗ, вони знадобляться нам для встановлення залежностей пакетів бібліотек що ми встановимо через pip, nmap та систему контроля версій git.

На четвертій строчці ми використовуємо git для того щоб клонувати з віддаленого репозиторія базу даних вразливостей. Наступною командою ми оновлюємо базу даних скриптів nmap.

На сьомій строчці ми копіюємо нашу систему в папку /app після чого робимо її робочою папкою за замовчуванням. Тобто виконання команд у контейнері буде відбуватись з цієї директорії.

На дев’ятій строчці ми встановлюємо усі залежності які стосуються python бібліотек.

І останньою строчкою ми вказуємо Docker що робити після запуску контейнеру, а саме – розпочати виконання з файлу telegramBot.py.

3.4 Експеримент з розгортки і використання створеної системи.

Почнемо з розгортання системи з того що отримаємо токен бота від спеціального телеграм бота призначеного для їх реєстрації (див. рис. 3.5).

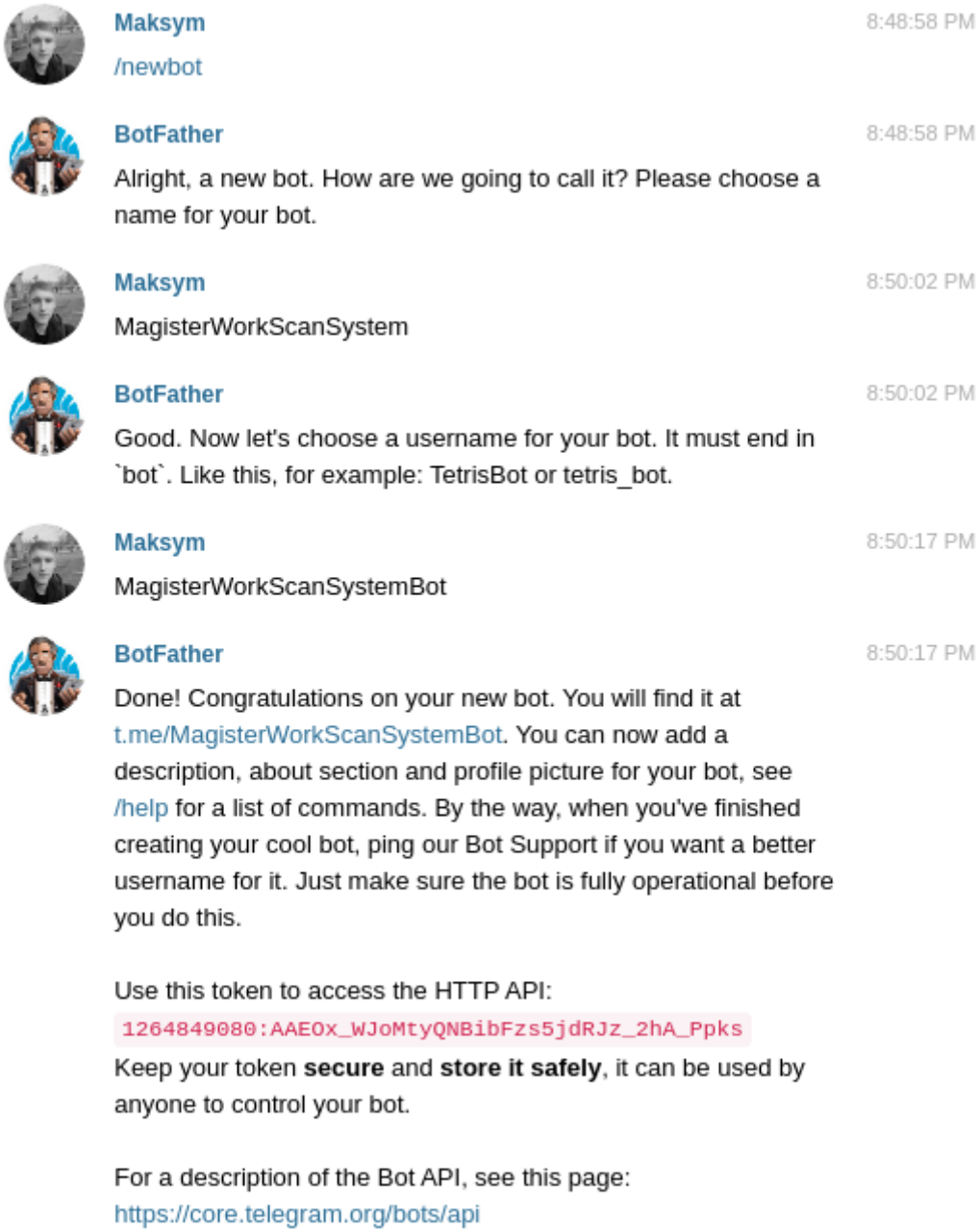


Рис. 3.5 процедура отримання токену

Після того як отримали токен авторизації бота, за допомогою git склонуємо репозиторій з системою після чого запустимо скрипт конфігурації системи та згенеруємо конфіг(див. рис. 3.6).

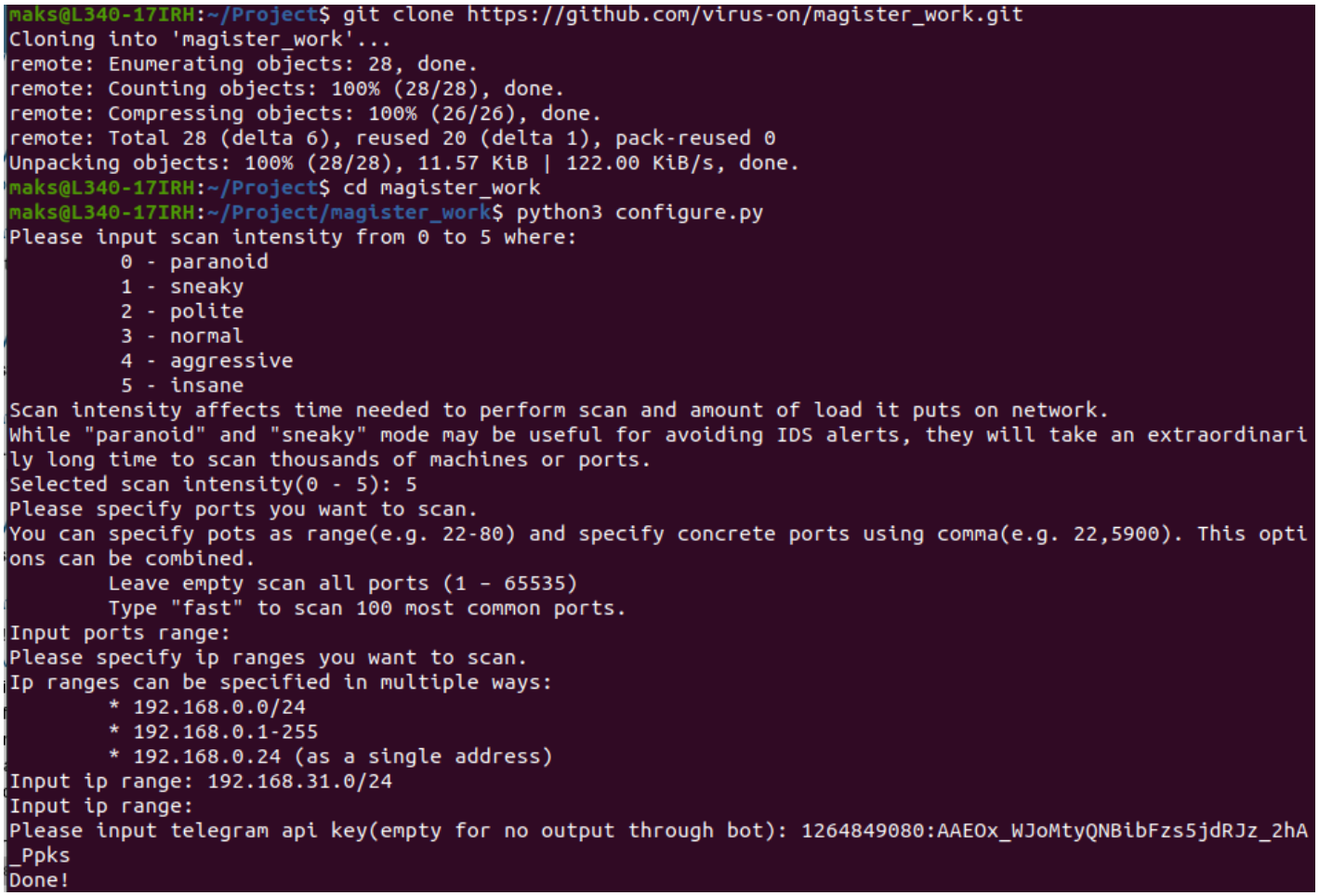


Рис. 3.6 Процедура попереднього налаштування системи

Після того як бота сконфігуровано, запустимо процес побудови образу контейнеру. Для цього виконаємо команду: docker build -t scanner_bot_magister_work:v0.1 magister_work. Приблизний результат виконання команди дивись на рисунку 3.7.

```
maks@L340-17IRH:~/Project$ docker build -t scanner_bot_magister_work:v0.1 magister_work
Sending build context to Docker daemon  143.9kB
Step 1/9 : FROM ubuntu:latest
 ---> f643c72bc252
Step 2/9 : RUN apt-get update -y
 ---> Using cache
 ---> 8815c45146a2
Step 3/9 : RUN apt-get install -y python3-pip python3-dev build-essential nmap git
 ---> Using cache
 ---> c962c7ed2d8f
Step 4/9 : RUN git clone https://github.com/vulnersCom/nmap-vulners.git /usr/share/nmap/scripts/vulners
 ---> Using cache
 ---> 754252c01696
Step 5/9 : RUN nmap --script-updatedb
 ---> Using cache
 ---> 4bbf0ef51e73
Step 6/9 : COPY . /app
 ---> Using cache
 ---> 3cc18f819d14
Step 7/9 : WORKDIR /app
 ---> Using cache
 ---> 0db875dfbd23
Step 8/9 : RUN pip3 install -r requirements.txt
 ---> Using cache
 ---> 27d26bbff1b6
Step 9/9 : ENTRYPOINT ["python3", "-u", "telegramBot.py"]
 ---> Using cache
 ---> 10371303c8fd
Successfully built 10371303c8fd
Successfully tagged scanner_bot_magister_work:v0.1
```

Рис. 3.7 Приблизний результат виконання запиту побудови контейнера

Після того як контейнер побудовано, запустимо його у фоновому режимі.

```
maks@L340-17IRH:~/Project$ docker run -d -P scanner_bot_magister_work:v0.1
38435c486fa8de82db2c76553386bbf7567cd91cea5ed88e935c6bdf49f8d969
maks@L340-17IRH:~/Project$ docker ps -a
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS
                    PORTS               NAMES
38435c486fa8        scanner_bot_magister_work:v0.1   "python3 -u telegram…"   12 seconds ago      Up 10 s
econds                                  silly_easley
54805f22f827        81835748247d                   "python3 -u telegram…"   20 hours ago        Exited
(137) 20 hours ago                      goofy_greider
6de95340f22a        81835748247d                   "python3 -u telegram…"   20 hours ago        Exited
```

Рис. 3.8 Приклад команди запуску бота

На цьому розгортка закінчена успішно, на все знадобилося менше 15 хвилин. Тепер бот працює у штатному режимі і готовий проводити сканування. Управління ботом відбувається за допомогою коротких команд. Розглянемо як можна керувати ботом за допомогою команд:

- Команда «/start» відправляє запит на підписку адміністратору бота, перший користувач що відправить цю команду боту буде позначений як адміністратор.
- «/schedule години:хвилини інтервал сканування в днях» ця команда дозволяє зареєструвати розпорядок за яким буде проводитись регулярне

сканування мережі. Прикладом такої команди: /schedule 9:30 1 що означає проводити сканування з інтервалом в один день о 9:30 ранку.

- «/schedule discard» дозволяє відмінити усі заплановані сканування.
- «/scan» негайно запустить сканування мережі на наявність вразливого ПЗ.
- «/unsubscribe_all» відпише від розсилки результатів усіх підписників\
- «/echo» Відправить повідомлення з текстом «Echo!» усім підписникам. Може використовуватися для перевірки роботи.

На наступних рисунках (3.9, 3.10) відображено як виглядає робота з системою з точки зору адміністратора:

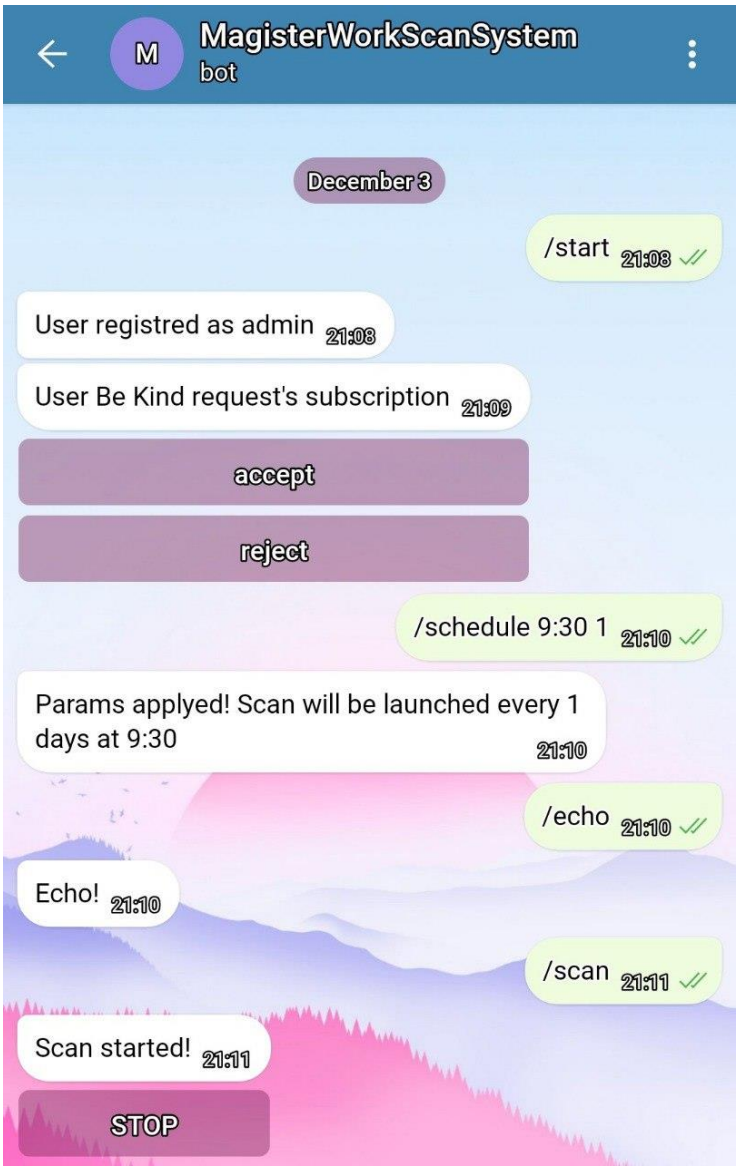


Рис. 3.9 Приклад взаємодії адміністратора з системою

Рис. 3.10 Приклад успішного проведення процедури сканування

З позиції підписника наша взаємодія з ботом є дуже лімітованою. Підписник може лише послати запит на підписку адміністратору, та якщо запит буде прийнято, почне отримувати від бота розсилку з результатами сканування.

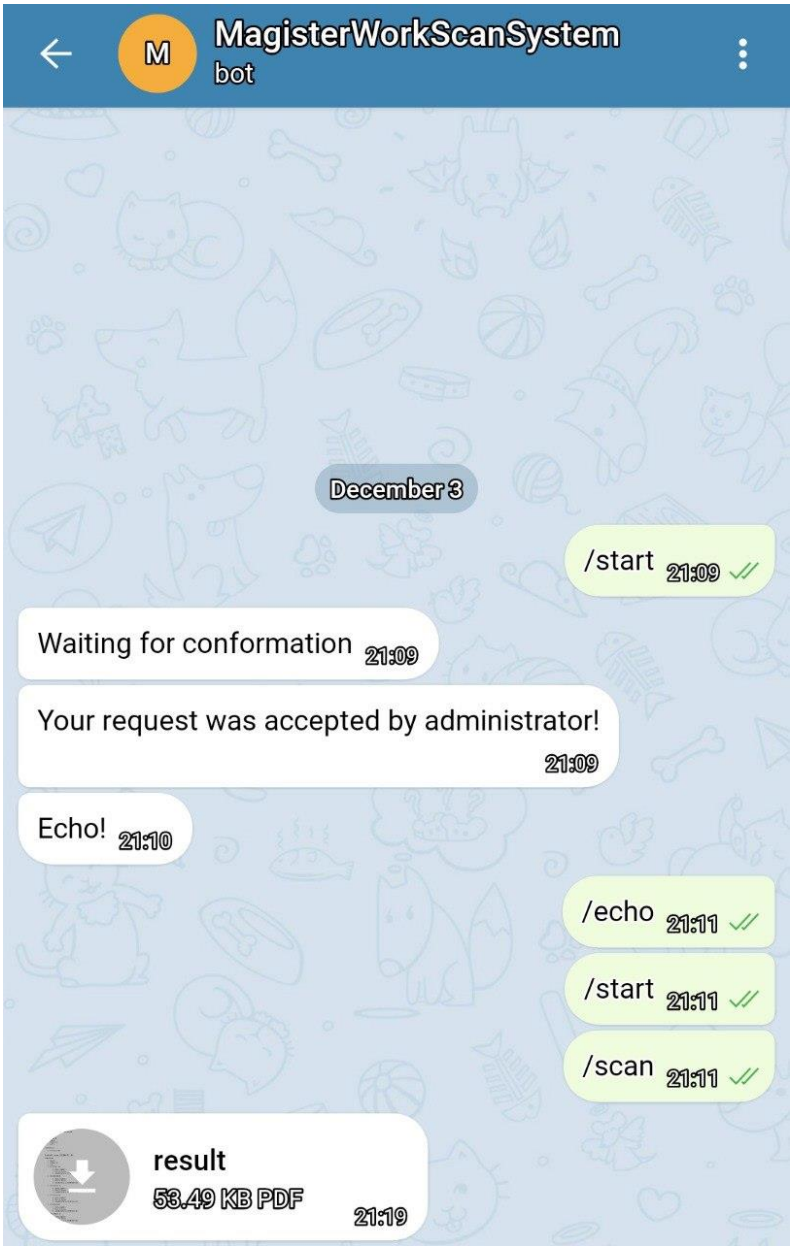


Рис. 3.11 Приклад взаємодії з системою з позиції підписника

Типовий результат сканування відображено на рисунку 3.12. Першим йде коротке резюме результатів сканування, що містить кількість знайдених вразливостей у всій мережі, після чого слідує список хостів на яких було знайдено проблеми. Після чого йде інформація по кожному із хостів окремо.

2020-11-30

Scan Report

Total number of scanned hosts: 3
Scan took: 00 hours 00 minutes 53 seconds

Issues found:

- Critical: **4**
- High: **16**
- Medium: **63**
- Low: **9**

Affected hosts:

- 192.168.31.198

Host with issues: 192.168.31.198

Issues found:

- Critical: **4**
- High: **16**
- Medium: **63**
- Low: **9**
- CVE-2010-4478
  - Rating: high[7.5]
  - Protocol: ssh
  - Affected Software: OpenSSH 4.7p1 Debian 8ubuntu1
- CVE-2008-1657
  - Rating: medium[6.5]
  - Protocol: ssh
  - Affected Software: OpenSSH 4.7p1 Debian 8ubuntu1
- CVE-2017-15906
  - Rating: medium[5.0]
  - Protocol: ssh
  - Affected Software: OpenSSH 4.7p1 Debian 8ubuntu1
- CVE-2010-5107
  - Rating: medium[5.0]
  - Protocol: ssh
  - Affected Software: OpenSSH 4.7p1 Debian 8ubuntu1
- CVE-2010-4755
  - Rating: medium[4.0]
  - Protocol: ssh
  - Affected Software: OpenSSH 4.7p1 Debian 8ubuntu1
- CVE-2012-0814
  - Rating: low[3.5]
  - Protocol: ssh
  - Affected Software: OpenSSH 4.7p1 Debian 8ubuntu1

Рис. 3.12 Приклад звіту проведеного сканування

3.5 Аналіз результатів експерименту

Експеримент показав, що система працює і вирішує поставлену задачу згідно з поставленими у розділі 3.1 вимогами у повному обсязі. Розгортка системи займає відносно невелику кількість часу та не потребує спеціальних знань. Система виконує сканування мережі відповідно до заданих параметрів та формує звіт з результатами сканування. Користувацький інтерфейс є простим та інтуїтивно зрозумілим. Звіт з результатами сканування містить необхідну інформацію для усунення вразливостей та їх ранжування за пріоритетами.

3.6 Рекомендації з подальшої доробки системи

Гарним доробком до цієї системи сканування мережі на наявність вразливого програмного забезпечення був би модуль тестування інфраструктури на наявні недоліки конфігурації сервісів та перевірка типових даних авторизації для наявних у мережі сервісів.

Також, реалізація планування кількох сканувань конкретних елементів мережевої інфраструктури з різноманітними параметрами органічно би вписалася у систему.

Можливість задання формату виводу звіту з проведеного сканування для кожного окремого підписника би стала у нагоді при спільному використанні системи у організації з великою кількістю співробітників. А надання можливості виставлення привілеїв адміністратора декільком співробітникам одночасно полегшила би роботу кожному з них.

До звіту зі сканування можна було б додати більше графічної інформації яка б полегшила емпіричну оцінку стану захищеності мережі в цілому.

Також, для полегшення ведення документації до проекту можна було б підключити базу даних яка б зберігала результати кожного окремого сканування.

3.7 Висновки по розділу.

У даному розділі було описано процес розробки та схему роботи системи сканування мережі на наявність вразливого ПЗ. Сформульовано чіткі вимоги до системи. Було розроблено, контейнеризовано та розгорнуто систему у реальному середовищі. Проведено експеримент з тестування системи та аналіз результатів експерименту.

Проведений експеримент показав що система працює та без проблем розгортається у заданому середовищі. Розгортка системи не потребує особливих знань чи навиків та не займає багато часу.

В результаті проведеної роботи було розроблено систему моніторингу мережі на наявність вразливого програмного забезпечення що відповідає усім сформульованим у розділі 3.1 вимогам.

ВИСНОВКИ

У сучасному світі все більше компаній пропонує співробітникам можливість віддаленої роботи. Разом з тим це робить проблему захищеності мережевої інфраструктури компаній гострою як ніколи, бо можливість віддаленої роботи відкриває зловмисникам більше векторів атаки на такі компанії.

Забезпечення високого рівня захищеності мережі від атак потребує постійної перевірки мережі на наявність вразливостей якими можуть скористатися зловмисники. Від так потреба у надійних та легких у використанні системах для сканування мережі на наявність вразливого програмного забезпечення також є високою.

За 2020 рік індустрія забезпечення захищеності мережі від атак зібрала рекордну кількість інвестицій що є сигналом наявності нагальної потреби у таких рішеннях.

У даній роботі проаналізовано існуючі системи сканування мережі на наявність вразливого програмного забезпечення, проведено їх порівняння та розглянуто принципи їх роботи. Сформульовано вимоги до такої системи та описано усі етапи її розробки.

Реалізовано систему сканування мережі на наявність вразливого програмного забезпечення, що завдяки технології контейнеризації може бути швидко розгорнута у будь якому середовищі. Система має зручний інтерфейс користувача, вміє генерувати звіти, розсилати їх авторизованим користувачам та проводити заплановані сканування.

Основними перевагами розробленої системи є швидкість розгортки та простота використання, та висока ефективність.

Розроблена система є готовим продуктом що може використовуватися як рядовими користувачами, так і комерційними організаціями з метою підвищення рівня захищеності мережевої інфраструктури.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Анализ новых рисков перехода на удаленную работу [Електронний ресурс]. URL: https://www.mcafee.com/enterprise/ru-ru/lp/working-from-home.html

2. Browse Vulnerabilities By Date [Електронний ресурс]. URL: https://www.cvedetails.com/browse-by-date.php

3. Новый «слив» данных SoftServe [Електронний ресурс]. URL: https://ain.ua/2020/09/16/softserve-utechka-2/

4. Buffer Overflow [Електронний ресурс]. URL: https://www.sciencedirect.com/topics/computer-science/buffer-overflow

5. Dynamic memory management [Електронний ресурс]. URL: https://en.cppreference.com/w/c/memory/free

6. Common Vulnerability Scoring System v3.1: Specification Document [Електронний ресурс]. URL: https://www.first.org/cvss/specification-document

7. RFC2828 [Електронний ресурс]. URL: https://tools.ietf.org/html/rfc2828

8. Port Scanning Techniques [Електронний ресурс]. URL: https://nmap.org/book/man-port-scanning-techniques.html

9. Port Scanning Techniques [Електронний ресурс]. URL: https://www.icterra.com/what-is-idle-scan/

10. Service and Application Version Detection [Електронний ресурс]. URL: https://nmap.org/book/vscan.html

11. OS Detection [Електронний ресурс]. URL: https://nmap.org/book/man-os-detection.html

12. Nmap Scripting Engine (NSE) [Електронний ресурс]. URL: https://nmap.org/book/man-nse.html

13. ZMap: The Internet Scanner [Електронний ресурс]. URL: https://github.com/zmap/zmap

14. multiple ports support? [Електронний ресурс]. URL: https://github.com/zmap/zmap/issues/44

15. MASSCAN: Mass IP port scanner [Електронний ресурс]. URL: https://github.com/robertdavidgraham/masscan

16. Nmap Copyright and Licensing [Електронний ресурс]. URL: https://nmap.org/book/man-legal.html

17. Memory Layout of C Programs [Електронний ресурс]. URL: https://www.geeksforgeeks.org/memory-layout-of-c-program/

18. Path Traversal [Електронний ресурс]. URL: https://owasp.org/www-community/attacks/Path_Traversal

19. CVE Abstraction Content Decisions: Rationale and Application [Електронний ресурс]. URL: https://cve.mitre.org/cve/editorial_policies/cd_abstraction.html

20. Vulnerability Metrics [Електронний ресурс]. URL: https://nvd.nist.gov/vuln-metrics/cvss

21. Полное руководство по общему стандарту оценки уязвимостей [Електронний ресурс]. URL: https://www.securitylab.ru/analytics/355336.php

22. Сканер портів [Електронний ресурс]. URL: https://uk.wikipedia.org/wiki/Сканер_портів

23. Эксплутация уязвимостей в функциях для работы с файлами в PHP [Електронний ресурс]. URL: https://xakep.ru/2012/10/01/php-vulnerabilities/

24. TRANSMISSION CONTROL PROTOCOL [Електронний ресурс]. URL: https://tools.ietf.org/html/rfc793

25. Registered Port [Електронний ресурс]. URL: https://www.sciencedirect.com/topics/computer-science/registered-port