МАГІСТЕРСЬКА РОБОТА
МР.ІПм – 02.00.00.000 ПЗ

Група ІПм-22-5

Гавриш Микола

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Гавриш Микола Васильович

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

МАГІСТЕРСЬКА РОБОТА

"Моделі та методи автоматизації процесів інсталяції програмного забезпечення згідно вимог концепцій кібербезпеки"

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Гавриш М.В.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Піх Володимир Ярославович к.т.н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

В.о. завідувача кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Рецензент

доц.

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
В.о. зав. кафедрою **ІПЗ**

доц. В.В. Бандура

“ 04 вересня 2023 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Гавришу Миколі Васильовичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “ Моделі та методи автоматизації процесів інсталяції програмного забезпечення згідно вимог концепцій кібербезпеки”

керівник проекту (роботи) Піх Володимир Ярославович к.т.н., доцент,

затверджені наказом закладу вищої освіти від “ 18 ” грудня 2023 р. № 738/7

2. Строк подання студентом проекту (роботи) 15 січня 2024 р.

3. Вихідні дані до проекту (роботи) Теоретичні концепції та формальні моделі побудови та функціонування інформаційних та програмних технологій певного класу

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Теоретичні основи автоматизації безпечних інсталяцій

2. Реалізація автоматизованих методів інсталяції програмного забезпечення

3. Автоматизація встановлення ПЗ та оцінка надійності онлайн-сховища

3. Аналіз отриманих результатів автоматизації інсталяції

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1 Спрощена ілюстрація CRATE. (рис. 1.1)

2. Типова послідовність атаки в SVED (рис. 1.2)

3. Скріншот розділу графіка атак СВІЗу (рис. 1.4)

4. Внутрішнє сховище з використанням Sonatype Nexus 3 (рис 2.2)

5. Статистика результатів встановлення (рис. 2.4)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Нормоконтроль	доц., к.т.н. Вовк Р.Б.	
Перевірка на плагіат	доц., к.т.н. Вовк Р.Б.	

7. Дата видачі завдання 04 вересня 2023 р.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури	01.10.2023	виконано
2	Теоретичні основи автоматизації безпечних інсталяцій	25.10.2023	виконано
3	Реалізація автоматизованих методів інсталяції програмного забезпечення	10.11.2023	виконано
4	Теоретичні основи автоматизації безпечних інсталяцій	01.12.2023	виконано
5	Реалізація функціональності запропонованої інформаційної технології	20.12.2023	виконано
6	Затвердження пояснювальної записки роботи завідувачем кафедри	15.01.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 82 с., 10 рис., 14 табл., 52 джерела,

Тема: Моделі та методи автоматизації процесів інсталяції програмного забезпечення згідно вимог концепцій кібербезпеки

Об’єкт дослідження: процес автоматизації безпечних інсталяцій програмного забезпечення. Дослідження спрямоване на вивчення теоретичних основ, методів та інструментів, які використовуються для забезпечення безпечності та ефективності при встановленні програм на комп'ютерні системи.

Предмет дослідження: процес автоматизації безпечних інсталяцій програмного забезпечення.

Мета магістерської роботи: дослідження теоретичних основ автоматизації безпечних інсталяцій програмного забезпечення, а також реалізація автоматизованих методів встановлення програм з фокусом на забезпеченні безпеки.

Висновок

Пропоноване дослідження сприяє розширенню знань у галузі безпечної автоматизації інсталяцій програм та надає основу для подальших досліджень та вдосконалення практичних використань автоматизованих методів в області процесів інсталяції програмного забезпечення згідно вимог концепцій кібербезпеки.

КІБЕРБЕЗПЕКА, ПАКУВАЛЬНИК МЕТАСПЛОЙТИ, БАЗА ДАНИХ, АВТОМАТИЗАЦІЯ ВСТАНОВЛЕННЯ ПЗ, ЕКСПЛОЙТ, СХОВИЩЕ, МЕТОД, ТЕСТУВАННЯ URL-АДРЕС, ЕФЕКТИВНІСТЬ ІНСТАЛЯЦІЇ

ANNOTATION

Master's thesis: 82 pages, 10 figures, 14 tables, 52 sources,

Topic: Models and methods of automation of software installation processes according to the requirements of cyber security concepts

The object of research: the process of automating secure software installations. The research is aimed at studying the theoretical foundations, methods and tools used to ensure safety and efficiency when installing programs on computer systems.

The subject of research: the process of automating secure software installations.

The purpose of the master's work: the study of the theoretical foundations of the automation of secure software installations, as well as the implementation of automated methods of installing programs with a focus on ensuring security.

Conclusion

The proposed study contributes to the expansion of knowledge in the field of safe automation of program installations and provides a basis for further research and improvement of practical uses of automated methods in the field of software installation processes according to the requirements of cyber security concepts.

CYBERSECURITY, METASPLOYTA PACKER, DATABASE, AUTOMATE SOFTWARE INSTALLATION, EXPLOIT, STORAGE, METHOD, URL TESTING, INSTALLATION PERFORMANCE

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

PDF - Portable Document Format

SCADA - Supervisory Control And Data Acquisition

API - Application Programming Interfac

SSH - Secure SHell

VPN - virtual private network

CRATE - Cyber Range And Training Environment

ВСТУП

Актуальність роботи.

Оскільки сучасне суспільство підключене до Інтернету в більш широкому діапазоні, ніж будь-коли раніше, також зростає загроза кібератак [1, 2]. Системи, які раніше не були підключені до Інтернету, такі як блоки диспетчерського контролю та збору даних (SCADA), все частіше підключаються, щоб задовольнити попит на дистанційне керування [3], а такі концепції, як розумні міста [4], створюють ширшу поверхню для атаки [5]. Протягом останнього року попит на можливість працювати вдома зріс через пандемію COVID-19 [6], яка вводить ще одну поверхню атаки [7]. Щоб протистояти цій загрозі, в усьому світі реалізуються численні проекти, спрямовані на підвищення як освіти, так і досліджень у сфері кібербезпеки [8,9,10,11]. Існують також зусилля для підвищення стійкості проти атак на базові протоколи, які з'єднують промислові пристрої [12]. Кібердіапазони чудово підходять для проведення контрольованих досліджень і практики, оскільки вони ізольовані від Інтернету та мають потенціал для забезпечення необхідних ресурсів, а також гнучкості для різних сценаріїв [13].

Актуальність дослідження також зумовлена тим, що під час війни різко збільшилась кількість кібератак на інформаційно-комунікаційні системи (ІКС) ОКІІ та приватного сектору. Причиною успішного проведення таких кібератак є наявність вразливостей в ІКС та технологічних мережах ОКІ, розробка нових інструментів для кібератак хактивістами, швидка зміна тактик (методів) кібератак, недостатня кваліфікація фахівців з кіберзахисту, низький рівень кібергігієни співробітників. У рамках існуючої парадигми інформаційної безпеки (забезпечення конфіденційності, цілісності та доступності) захист інформації та кіберзахист повинен включати управління ризиками, менеджмент вразливостей, заходи з кібероборони.

Управління вразливостями є одним з найважливіших процесів забезпечення кібербезпеки ІКС та АСУ ТП. Водночас дані підходи лише

вказують набори кроків для побудови процесу управління вразливостями, не пояснюючи особливостей їх реалізації в ІКС та АСУ ТП критичної інфраструктури. Як наслідок процес управління вразливостями реалізується несистематично і має досить сумнівну якість його результатів. З огляду на зазначене, доцільним є розробити метод автоматизації управління вразливостей на основі технології SCAP.

Пропонована робота є актуальною і важливою, оскільки відповідає сучасним викликам у галузі безпеки інформаційних технологій та сприяє подальшому розвитку автоматизованих методів встановлення програмного забезпечення. Зростання кількості кіберзагроз, Об'єм та різноманітність програмного забезпечення, Необхідність ефективного управління ресурсами, Швидкі темпи розвитку технологій, Кількісне та якісне зростання інтернет-застосунків, вона є вкрай актуальною в контексті сучасних викликів та вимог інформаційного суспільства, сприяє забезпеченню безпеки, ефективності та стабільності інформаційних систем.

Мета і задачі дослідження

Метою роботи є дослідження теоретичних основ автоматизації безпечних інсталяцій програмного забезпечення, а також реалізація автоматизованих методів встановлення програм з фокусом на забезпеченні безпеки.

Відповідно до мети необхідно вирішити наступні задачі:

- Вивчення теоретичних аспектів;

- Реалізація автоматизованих методів встановлення програмного забезпечення;

- Автоматизація встановлення програмного забезпечення та оцінка надійності онлайн-сховища;

- Представлення контролю виходів з Ansible та URL-адрес;

- Аналіз отриманих результатів.

Об’єктом дослідження є процес автоматизації безпечних інсталяцій програмного забезпечення. Дослідження спрямоване на вивчення теоретичних

основ, методів та інструментів, які використовуються для забезпечення безпечності та ефективності при встановленні програм на комп'ютерні системи.

Предметом дослідження є процес автоматизації безпечних інсталяцій програмного забезпечення.

У контексті конкретного тексту, предметом дослідження є теоретичні основи та реалізація автоматизованих методів встановлення програмного забезпечення з підвищеним акцентом на безпеку. Дослідження охоплює аналіз кіберполігону, менеджерів пакетів, засобів автоматизації, інструментів налаштування віртуальних машин, сканування уразливостей, використання експлойтів, візуалізацію та інші аспекти, що стосуються безпечного встановлення програм.

Методи дослідження

Для досягнення поставлених цілей та вирішення завдань дослідження використовувалися наступні методи дослідження: літературний аналіз, експертні опитування, емпіричні дослідження, аналіз відгуків та випробувань користувачів, моделювання процесу, методи тестування безпеки.

. **Наукова новизна отриманих результатів** полягає в їхньому унікальному внеску у розвиток та вдосконалення автоматизованих методів встановлення програмного забезпечення з урахуванням сучасних викликів у сфері інформаційної безпеки.Практичне значення одержаних результатів.- Розробка та оптимізація методів, Інтеграція безпеки у процес, Тестування та валідація, Аналіз ефективності та продуктивності, Інноваційні технології, Комплексне управління ризиками, Аналіз впливу на загальну безпеку систем:

Практичне значення отриманих результатів

Практичне значення отриманих результатів у галузі "Автоматизація безпечних інсталяцій програмного забезпечення" визначається конкретним внеском у покращення безпеки, ефективності та управління програмами в інформаційних системах.

Особистий внесок студента

Особистий внесок студента полягає у вивченні, дослідженні та виборі методів, практичній реалізації, аналізі результатів та ефективній комунікації отриманих знань для автоматизації процесів інсталяції програмного забезпечення згідно вимог концепцій кібербезпеки.

Структура та обсяг магістерської роботи

Магістерська робота викладена на 82 сторінках друкованого тексту, який складається із вступу, чотирьох розділів, висновків, списку використаних джерел (52 найменування). Робота містить 14 таблиць, 10 рисунків.

РОЗДІЛ 1.
ТЕОРЕТИЧНІ ОСНОВИ АВТОМАТИЗАЦІЇ БЕЗПЕЧНИХ ІНСТАЛЯЦІЙ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Опис структури налаштовуваного кібер середовища

Cyber Range And Training Environment (CRATE) — це кібер-діапазон, який дозволяє налаштувати велику кількість віртуальних машин для імітації великомасштабної мережі з кількома віртуальними організаціями та промисловими системами. Він складається з приблизно 800 фізичних серверів, розроблений і використовується FOI для використання в демонстраціях, тренуваннях та інших навчаннях [14].

Віртуальні машини в CRATE розділені на ізольовану ігрову мережу, де розташовані всі вразливі хости та керовані зловмисниками машини, і мережу адміністрування, яка підключає ігрову мережу до зовнішнього світу та керує станом і конфігурацією всіх віртуальних машини, розташовані всередині ігрової мережі. Мережу адміністрування можна підключати за допомогою набору інтерфейсів прикладного програмування (API) протоколу передачі гіпертексту (HTTP), які можуть допомогти автоматизувати налаштування великих мереж. Спрощену ілюстрацію з відповідною інформацією для цього проекту можна побачити на рисунку 1.1. Кожна відповідна частина, показана на малюнку, обговорюється в наступних розділах і главах.

На рисунку мережа адміністрування називається площиною керування, а ігрова мережа — площиною подій. Найпростіший спосіб отримати доступ до рівня керування — за допомогою Secure SHell (SSH) через віртуальну приватну мережу (VPN), а до рівня подій — за допомогою SSH або протоколу віддаленого робочого стола (RDP), що тунелюється через фізичні вузли (які самі є доступ через площину керування). Робочі станції, які використовуються для розробки, розташовані за межами цієї мережі з доступом як до Інтернету, так і до CRATE. Вони показані у верхньому лівому куті рисунка 1.1.

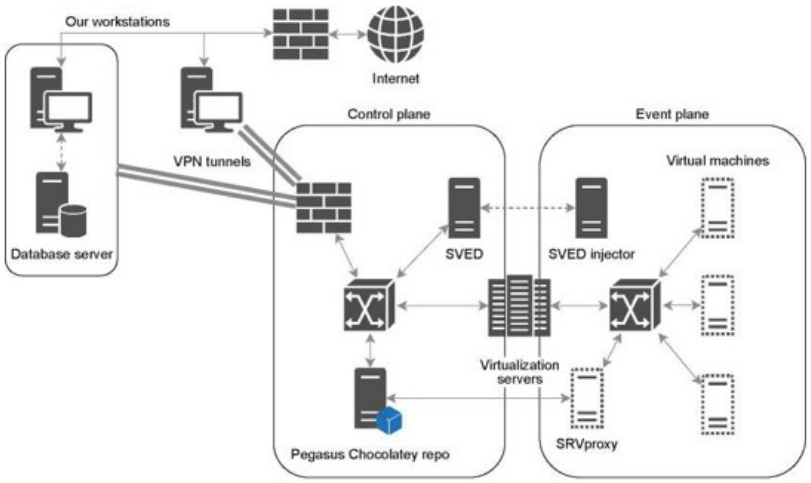

Рис. 1.1. Спрощена ілюстрація середовища CRATE

1.2. Менеджери пакетів для Windows

У той час як менеджери пакунків, які можуть встановлювати програмне забезпечення без взаємодії з користувачем, є рекомендованим способом придбання програмного забезпечення в дистрибутивах Linux, таких як Debian [16], у системах Windows воно завантажується безпосередньо від постачальника або через магазин Microsoft [17]. Незважаючи на те, що магазин Microsoft надає уніфікований метод інсталяції програмного забезпечення від кількох постачальників, автоматизація інсталяції пакетів із нього не здається простою d 1 . Деяке програмне забезпечення можна розповсюджувати у форматі, який дозволяє автоматичне або тихе встановлення, наприклад, формат інсталятора Microsoft Windows (MSI), який має опцію /quiet для виконання встановлення без відображення інтерфейсу користувача [18]. Інші додатки можуть розповсюджуватися у форматах, які використовують нестандартизований спосіб виконання тихої інсталяції або можуть взагалі не мати опції тихої інсталяції [19].

Оскільки автоматичне встановлення програм є бажаним для багатьох системних адміністраторів [20], з’явилися рішення, спрямовані на подолання цих проблем за допомогою спеціальних процедур встановлення та сценаріїв, адаптованих до конкретного способу виконання тихої інсталяції кожної

програми. Деякі з цих рішень обговорюються в наступних розділах. Розглядаються лише безкоштовні рішення, оскільки багато платних версій цих програм ліцензуються або на кожного користувача, або на машину, на якій розгорнуто програмне забезпечення, з яких останнє було б надзвичайно дорогим для FOI через CRATE, що містить до тисяч віртуальних машин. Модель ліцензування на основі користувача може бути більш здійсненною, але також може бути дуже дорогою та важкою в управлінні, якщо кожен користувач, який хоче створити віртуальні машини в CRATE із спеціальним набором програмного забезпечення, вимагатиме власної ліцензії.

Середовище надає набір інструментів, які можна використовувати для упаковки автономних двійкових файлів, сценаріїв або інсталяторів в уніфікований формат, який потім можна встановити, видалити або оновити за допомогою утиліти командного рядка [21]. Також існує офіційно підтримуваний репозитор у 3 створених спільнотою пакетів, доступних для багатьох поширених настільних програм і служб, таких як Adobe Acrobat Reader, Google Chrome, медіаплеєр VLC, Python 3, MySQL Server та інші. Станом на 08.02.2021 репозиторій містить 8 173 пакети [22]. Для багатьох із них також доступні старіші версії, що цікаво, оскільки багато експлойтів застосовні лише до певних версій до виправлення вразливостей, які вони використовують. Наприклад, найстарішою версією Mozilla Firefox, доступною в репозиторії, є версія 15.0 від 2012 року. Однак через проблеми з правами на розповсюдження репозиторій Chocolatey часто не може містити двійкові файли інсталятора, натомість отримує їх безпосередньо від постачальника програми під час інсталяції [23]. Таким чином, немає гарантії, що ці старі версії все ще можна встановити. Якщо ручне пакування є варіантом, пакунки, які включають ці інсталятори, можна створити та розмістити в локальному сховищі для внутрішнього використання.

Керування пакетами Windows r 4 (також відомий як WinGet) — це інструмент від Microsoft, який наразі доступний як бета-версія. Як і інші менеджери пакетів, він здатний зупиняти пакети через інтерфейс командного

рядка та отримувати програми для встановлення безпосередньо від постачальника [24]. Кілька поширених настільних програм, таких як Adobe Acrobat Reader, Google Chrome, медіаплеєр VLC і Python 3, доступні в репозиторії спільноти у 5 , який станом на 2021-02-08 містить 1249 пакетів [25].

Scoop у своєму репозиторії доступно близько 2013 пакетів із поєднанням орієнтованих на розробників інструментів командного рядка та програм із графічним інтерфейсом. Доступно багато поширених програм, таких як веб-браузери та медіаплеєри. Йому не вистачає належної підтримки для старіших версій, оскільки всі програми мають по одному файлу метаданих, що запобігає існуванню двох версій з однаковою назвою програми. Хоча доступне сховище різних версій 7 зі 129 пакетами, він в основному містить бета-версії та основні випуски інтерпретаторів, таких як PHP: Hypertext Preprocessor (PHP), Python і Node.js, де нова версія може бути несумісною зі старим кодом або вносити інші великі зміни. Назви версій тут також трохи скорочено, тому пакет із назвою "python39" може встановити Python 3.9.2. Scoop також інсталює пакети лише для локального користувача за замовчуванням, що може створити проблеми для експлойтів, які припускають, що програми інстальовано у вказаному місці, але пакети, здається, також можна інсталювати глобально, надаючи параметр командного рядка.

Також доступно кілька інших менеджерів пакунків, але багато з них або не націлені на програмне забезпечення Windows, або мають занадто обмежений вибір доступних пакунків.

Нініт e8 пропонує вибір із близько 90 програм [28], які можна встановити, вибравши, які програми потрібно встановити, а потім завантаживши одну програму, яка встановлює їх із мінімальною взаємодією користувача. Однак цей вибір потрібно зробити на веб-сайті Ninite, і отриманий інсталятор не можна буде запустити тихо без придбання ліцензії на комп’ютер. Програма встановлення завжди встановлює останню версію кожної вибраної програми, навіть якщо саму програму встановлення було

завантажено давно, що робить це рішення погано придатним для цілей відтворення на основі мотивацій і цілей.

AppGe t 9 містить близько 1400 пакетів, але, як і Scoop, він також не підтримує старіші версії через той самий архітектурний вибір, що має лише один файл метаданих на програму. 01.08.2020 служба була закрита назавжди через те, що Microsoft випустила WinGet [29]. Оскільки він має відкритий вихідний код, а репозиторій пакунків усе ще доступний для повторного використання в інших проектах, було б можливим (з певною роботою) інсталювати пакети з AppGet, але порівняно з альтернативами, які активно підтримуються та пропонують старіші версії, це не може бути хорошим кандидатом на цю тезу.

NuGe t 10 це ще один менеджер пакетів, який дозволяє автоматично встановлювати пакети, а також це те, на чому побудований Chocolatey. На відміну від Chocolatey, його репозитор у 11 в основному стосується бібліотек для розробки .NET [30], і тому обмежено використовується для звичайних користувачів. Здається, доступна версія Firefox, але існує лише кілька версій, найновішій з яких два роки тому.

1.3. Опис засобів автоматизації

Існує кілька інструментів для автоматичного встановлення програмного забезпечення та керування великою кількістю машин. Деякі з найпоширеніших з них детально описано в цьому розділі разом із деякими прикладами синтаксису конфігурації для кожного з них. Хоча більшість із цих прикладів розроблено для систем на базі Linux, їх можна також адаптувати для роботи в Windows.

Ansible - це інструмент автоматизації, який зараз підтримує Red Hat. Він призначений для автоматизації розгортання системи, встановлення програмного забезпечення та іншої конфігурації простим у використанні способом, не вимагаючи встановлення спеціалізованого програмного

забезпечення агента на керованих машинах [32]. Користувальницькі дії можна реалізувати в Python, якщо включений вибір модулів 3365 s 13 виявляється недостатнім. Формат YAML не є мовою розмітки (YAML) використовується для визначення того, які дії слід виконувати на наборі машин. Файл, що містить набір таких дій, структурованих у список завдань, називається playbook. Приклад короткого підручника, що містить два завдання, можна побачити в лістингу 1.1.

```
- ім'я : налаштувати хости
  веб-серверів : веб-сервери

  завдання :
  - ім'я : встановіть останню версію Apache
    apt :
      назва :
      apache2 стан
      : останній
  - ім'я : Оновити шаблон конфігураційного
    файлу Apache :
      src : files/apache_web.conf dest
```

Лістинг 1.1. Приклад Ansible YAML

Chef 14 або Chef Infra — ще один інструмент автоматизації, розроблений Progress. Він зосереджений на вирішенні проблеми керування кількома машинами в організації. Він розділений на три частини: робоча станція, з якої адміністратор може ініціювати зміни та адмініструвати мережу, сервер Chef, який збирає кулінарні книги та перекладає інструкції з робочої станції в код і політики та розповсюджує їх серед клієнтів. Клієнти Chef — це вузли, комп'ютери, сервери або віртуальні машини, якими слід керувати.

Це означає, що Chef використовує модель головного агента, де адміністратор повинен завантажити інструкції, які називаються рецептами (або кулінарну книгу, якщо є кілька рецептів), на сервер, який, у свою чергу, може перетворити інструкції в код, який надсилається на кожну машину. Кожна машина має агента, який отримує інструкції та виконує їх. Інструкції, які використовуються для побудови рецептів, написані мовою Ruby Domain-

Specific Language (DSL), яка може дозволити більш складний набір інструкцій, ніж YAML, але також є складнішими для налаштування та сприйняття для користувачів, які не знайомі з Ruby. За своєю основою він є відкритим вихідним кодом, але компанія також пропонує корпоративну версію, де надається серверний хостинг і додаткова підтримка за певну плату [34]. У лістингу 1.2 показано приклад рецепту налаштування сервера Apache за допомогою Ruby.

```
вузол . default [ 'main' ][ 'doc_root' ] = "/vagrant/web"

виконайте "apt-get update" виконайте команду "apt-
get update" end

apt_package "apache2" виконати дію : встановити
кінець

служба "apache2" виконати дію [ :enable , :start ]
end

вузол каталогу [ 'main' ][ 'doc_root' ] do owner
'www-data' group 'www-data' mode '0644' action
:create end

file cookbook #{ node [ 'main' ][ 'doc_root' ] }
/index.html" do source 'index.html' owner 'www-data' group
'www-data' action :create end

шаблон "/etc/apache2/sites-available/000-default.conf" зробити
  джерело "vhost.erb"
  variables({ :doc_root => node [ 'main' ][ 'doc_root' ] })
  action :create
  сповіщає :restart , resources( :service => "apache2"
) end
```

Лістинг 1.2. приклад Chef Ruby [35]

Puppet також є хорошим кандидатом для автоматизації. Головне завдання маріонетки — вирішити проблему керування кількома серверами та комп’ютерами, подібно до Ansible і Chef. Він також має відкритий код і написаний на Ruby. Подібно до Chef, він використовує агента для кожного клієнта, який отримує та виконує інструкції. Інструкції, які виконуються, записуються в Puppet's DSL, також званому Puppet Code. Зразок синтаксису див. у лістингу 1.3. Цей код використовується для опису бажаного стану системи, а не для того, як досягти цього стану [36]. Потім Puppet застосовує

необхідні кроки для досягнення цього стану за допомогою агента. Однак, оскільки метою цього проекту є лише налаштування певного середовища, а не підтримка його стану, деякі функції Puppet можуть бути неактуальними для цієї дипломної роботи.

```
case $operatingsystem {
   centos , redhat : { $service_name = 'httpd' }
   debian , ubuntu : { $service_name = 'apache2'
   }
}

package { 'apache2' :
   secure => установлено
   ,
}

служба { 'apache2' :
   ім'я      =>  $service_name,
   забезпечити         =>       працює ,
   enable    => true ,
   підписка => Файл [ 'apache2.conf' ],
}

файл { 'apache2.conf' :
   path => '/etc/apache2/apache2.conf' ,
   забезпечити => файл ,
   вимагати => пакет [ 'apache2' ],
   джерело => "puppet:///modules/apache2/apache2.conf" ,
   # Цей вихідний файл буде розміщено на основному сервері Puppet за
   адресою
   # /etc/puppetlabs/code/modules/apache2/files/apache2.conf
}
```

Лістинг 2.3: Приклад Puppet на основі офіційної документації [37]

Puppet Bolt це новіше рішення від розробників Puppet, яке не працює без агентів і може використовувати SSH або протокол віддаленого керування Windows (WinRM) для підключення до машин у мережі [38]. Це проект з відкритим вихідним кодом, який спрямований на виконання необхідної оркестровки безпосередньо з локальної робочої станції. Скрипти можна виконувати безпосередньо з терміналу або структурувати у файли YAML подібно до того, як це робить Ansible. Дивіться лістинг 1.4 для прикладу такого файлу. Це може бути потенційним кандидатом, оскільки він не має агентів і його легко інтегрувати в інфраструктуру.

```
                                    параметри :
                                      цілі :

  тип : Кроки TargetSpec :
- ім'я : завдання
  install apache :
  цілі пакета :
  параметри $targets :
    дія : ім'я
    встановлення :
    apache2
  description : "Встановити Apache за допомогою завдання пакетів"
```

Лістинг 1.4. Приклад Puppet Bolt [38]

Salt або Saltstack, це інструмент автоматизації з відкритим кодом, який тепер належить VMware і заснований на Python. Це потужний і масштабований інструмент автоматизації, який використовує ZeroMQ для швидкого та гнучкого зв’язку з керованими вузлами [39]. Існує також можливість для вузла діяти як проксі для іншого вузла, щоб також можна було керувати пропрієтарними системами або машинами з обмеженими ресурсами [40].

Salt має структуру з майстром Salt і кількома міньйонами Salt, де майстер Salt керує міньйонами Salt, які розташовані на кожній машині як агент. Оскільки він використовує код YAML для структурування своїх інструкцій, він дуже схожий на інші альтернативи. Приклад можна побачити в лістингу 1.5. VMware також пропонує Salt SSH, який є безагентною версією Salt, яка виконує команди за допомогою протоколу SSH. Однак це негативно впливає на швидкість Salt, оскільки він втрачає переваги ZeroMQ. Це було б прийнятною втратою, але, на жаль, Salt SSH не підтримується в Windows, якщо ви не придбаєте корпоративну версію, яка також використовує WinRM замість SSH [41]. Ця корпоративна версія, здається, є частиною пропозиції vRealize Automation від VMware, яка коштує близько 1400 шведських крон щороку за керований вузол [42]. Це було б дуже дорого з сотнями машин у CRATE, тому це не варіант.

```
apache2 :
  pkg.installed

Служба apache2 :
  service.running
  :
    - назва : apache2
    - enable : Правда
    - вимагати :
      - pkg : apache2

Вимкніть KeepAlive
  : file.replace :
    - ім'я : /etc/apache2/apache2.conf
    - шаблон : «KeepAlive On»
    - repl : 'KeepAlive Off'
    - show_changes : Правда
    - вимагати :
      - pkg : apache2

/etc/apache2/conf-available/tune_apache.conf :
  file.managed :
    - Джерело : salt://files/tune apache.conf
    - вимагати :
      - pkg : apache2
```

Лістинг 1.5. Приклад Salt

1.4. Інструменти налаштування віртуальної машини

На додаток до більш загальних інструментів автоматизації, представлених досі, існують також рішення, які спрямовані на надання методу налаштування віртуальної машини з певною конфігурацією або набором програм. Незважаючи на те, що вони чудово підходять для випадку використання, вони мають деякі обмеження, які можуть створити проблему для майбутнього розширення або архітектурних змін.

Boxstarte r18 , створений компанією Chocolatey Software, має на меті спростити розгортання машин Windows шляхом автоматичної інсталяції певного набору пакетів Chocolatey на них. Він може встановлювати пакети на віддаленій системі та вирішувати кілька поширених проблем, які можуть перешкоджати встановленню пакетів, наприклад, незавершені перезавантаження, інші одночасні процеси встановлення, одночасне встановлення оновлень Windows або ввімкнення шифрування диска [44]. Однак одна складність полягає в тому, що він повинен працювати в системі Windows і може керувати лише іншими системами Windows. Оскільки фізичні

вузли в CRATE базуються на Linux, це означає, що Boxstarter потрібно запускати локально на кожній віртуальній машині (VM) і керувати ним через Ansible. Також неможливо буде використовувати його для керування системами на базі Linux/Unix у майбутньому .

CRATE спрямований на автоматизацію створення образів віртуальних машин для різних платформ і може, як частина цього процесу, також запускати інші засоби автоматизації, щоб інсталювати програмне забезпечення або вносити інші зміни в конфігурацію. Хоча це, можливо, було б хорошим рішенням для автоматизації розгортання віртуальних машин у CRATE, це також потребувало б заміни великих частин існуючої інфраструктури, пов’язаної з їх підготовкою. З цієї причини це може бути хорошим варіантом розглянути, чи процес розгортання буде перероблено в майбутньому, але для цієї тези це, ймовірно, призведе до несумісності та проблем, якщо створений інструмент інсталяції буде здатний інсталювати пакети на віртуальні машини, які вже налаштовано наявними інструментами FOI. Перерахування загальних платформ (CPE) — це метод представлення програмних продуктів і пакетів структурованим способом. Кожен запис CPE містить тип продукту (наприклад, операційну систему або програму), постачальника, назву продукту та версію. Поле версії можна опустити або замінити зірочкою, якщо не вказано конкретну версію продукту. У CPE також може міститися інформація про конкретну апаратну платформу, випуск, мову тощо, яку за бажанням також можна опустити.

1.5. Загальні вразливості та ризики. Метасплойти та інструменти візуалізації

Загальні вразливості та викриття (CVE) — це спосіб присвоєння унікального ідентифікатора опублікованій уразливості в програмному забезпеченні. Цей ідентифікатор зазвичай має форму CVE-YYYY-NNNNNN, де YYYY — це рік, коли вразливість стала загальновідомою (або коли було

надіслано запит на CVE, якщо раніше) та унікальний номер NNNNNN із 4 або більше цифр. Кожен запис CVE містить опис уразливості та посилання на відповідну інформацію, таку як поради щодо безпеки або веб-сайти постачальників [48]. Національна база даних про вразливості США (NVD) також підтримує базу даних із додатковою інформацією для кожного CVE, а саме оцінку загальної системи оцінки вразливості (CVSS), що вказує на серйозність уразливості, ідентифікатор загального переліку слабких місць (CWE), який вказує на категорію недоліки у вихідному коді, які спричинили вразливість, і вражені CPE.

Metasploi t20 це платформа тестування на проникнення, розроблена Rapid7, яка містить багато модулів для сканування мереж, експлуатації, обфускації тощо. Ці модулі написані на Ruby та відповідають стандартизованій специфікації, яка полегшує запуск модулів у рамках автоматизованого процесу. Станом на зараз в категорії «експлойти» доступно 2102 модулі, які всі спрямовані на використання мережевих або локальних уразливостей у програмах або вбудованих системах. Експлойти далі поділяються на групи залежно від способу доставки або типу послуги, на яку вони спрямовані. Віддалені експлойти запускаються проти мережевої служби без взаємодії з користувачем, тоді як експлойти браузера намагаються використати веб-браузер користувача. Експлойти формату файлів постачаються як файл, який користувач повинен відкрити за допомогою вразливої програми.

Є також 592 корисних навантаження, які можна доставити через експлойт для виконання певної операції. Існують корисні навантаження для багатьох операційних систем і мов програмування, які можна вибрати залежно від мети експлойту, наприклад, служби операційної системи або веб-програми. Багато з цих платформ також мають кілька корисних навантажень для різних дій, таких як створення оболонки, яка підключається до зловмисника, створення сеансу віддаленого робочого столу, додавання облікового запису користувача, перезавантаження машини, форматування

жорсткого диска або відтворення повідомлення через динаміки за допомогою перетворення тексту в мовлення.

Сканування, виявлення вразливостей, експлойтів і виявлення (SVED) — це структура для розробки та виконання ланцюжків атак проти будь-якої кількості хостів автоматизованим способом. Він розроблений FOI та використовується в CRATE для автоматизації атак за допомогою інфраструктури Metasploit. Атака в SVED може складатися з кількох дій і керування потоком, що визначає, яку дію слід виконати залежно від результату попередньої. Наприклад, атака на мережу може початися зі сканування портів певної машини. Якщо порт 80 (використовується для трафіку HTTP) відкритий і повернута версія відповідає очікуваному значенню, експлойт може бути виконано проти цієї машини. У разі успіху наступна дія може бути налаштована на збір паролів із скомпрометованої машини, які потім використовуються для атаки на сервер бази даних тощо. Ці ланцюги атак можуть також включати надмірність для невдалих атак, наприклад, якщо цільовою є інша служба замість HTTP, якщо веб-сервер не запущено, або інший комп’ютер може бути обраний як ціль. Ці ланцюжки подій встановлюються вручну, але FOI також створив прототип іншого фреймворку, який використовує штучний інтелект для автоматичного вибору найкращого курсу дій для кожної ситуації. Це повинно забезпечити більш автоматизовану експлуатацію мереж у CRATE, але поки це непридатне для використання на практиці. Для цілей цієї роботи основними частинами SVED, які мають відношення, є адміністративний інтерфейс та інжектори. Адміністративний інтерфейс містить API і графічний інтерфейс користувача (GUI) для створення послідовностей дій, а інжектори — це віртуальні машини, які можна автоматично розмістити в правильному сегменті мережі всередині площини подій CRATE з метою фактичного запуску атак, які керував адміністративною частиною СВЕД. Інструмент візуалізації SVED (SVIZ) праву панель. Піктограма головоломки, позначена червоним фоном, є початковою точкою графіка атаки. Залежно від результату дії, виконання може продовжуватися

або за зеленою стрілкою для успішної дії, або за червоною для невдалої. Наведений приклад показує невеликий сегмент більшого тесту атаки, де дотримується типова послідовність подій, як описано на рисунку 1.2.

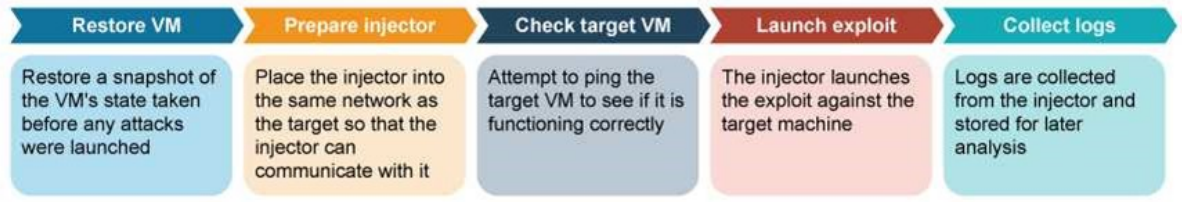

Рис. 1.2. Типова послідовність атаки в SVED

Віртуальна машина спочатку відновлюється з попередньо зробленого знімка, потім інжектор розміщується в правильному сегменті мережі, після чого він перевіряє, чи ціль відповідає, і, нарешті, запускає експлойт. Якщо це вдається, встановлений сеанс реєструється як успішний, а потім знищується, а потім (або якщо будь-який крок не вдається) віртуальна машина знову відновлюється зі знімка для підготовки до наступної атаки.

Для аналізу журналів, створених із SVED, використовується інструмент візуалізації SVED (SVIZ), щоб отримати розуміння того, які дії та атаки були успішними, а які невдалими. SVIZ був розроблений в FOI і може обробляти файли журналу, створені SVED, для відображення результатів по відношенню до їх відповідних дій у SVED.

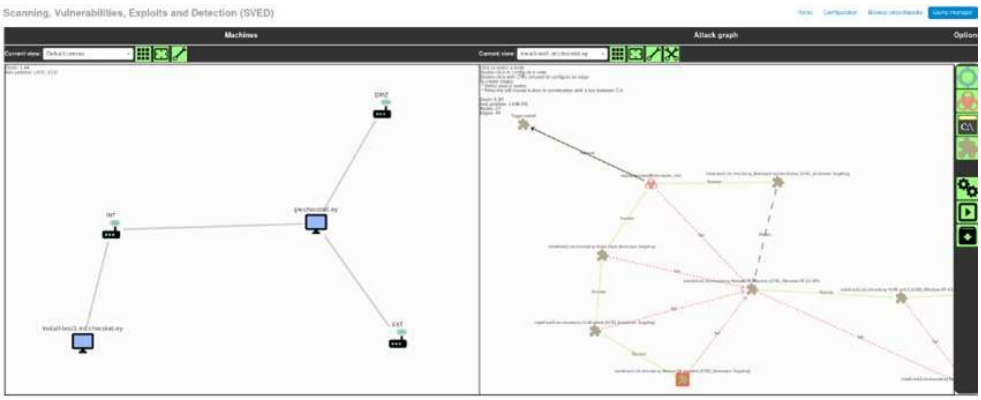

Рис. 1.3. Скріншот творця графіка атак у SVED

Приклад інтерфейсу показано на рис. 1.4. Експлойти представлені трикутниками, а допоміжні дії, такі як скидання віртуальних машин, представлені шестернями. Успішно виконані дії відображаються зеленим кольором, очікувані – жовтим, а невдалі – червоним. За допомогою цього інструменту значно легше виявити цікаві події у більшому наборі атак, особливо коли декілька машин працюють паралельно.

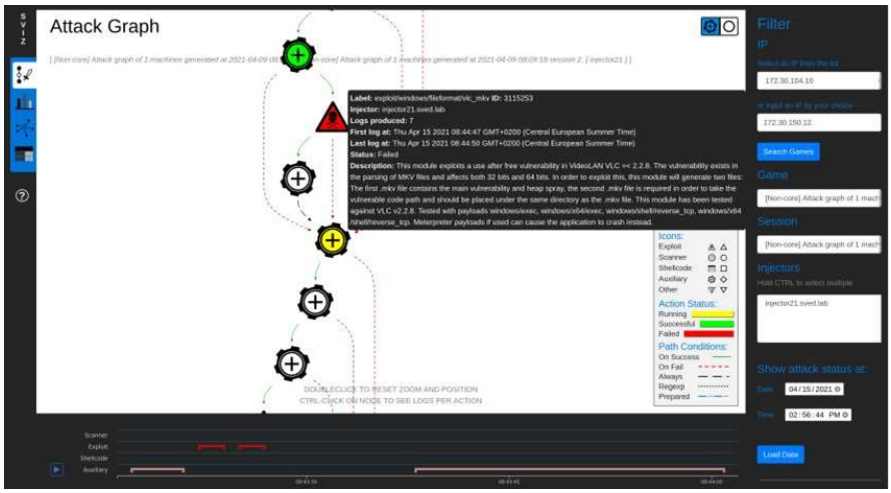

Рис. 1.4. Скріншот розділу графіка атак СВІЗу

Висновки до розділу

В даному розділі наведено та досліджено теоретичні основи автоматизації безпечних інсталяцій програмного забезпечення. Здійснено опис структури налаштовуваного кібер середовища, описані менеджери пакетів для Windows, засоби автоматизації та досліджені інструменти налаштування віртуальної машини.

РОЗДІЛ 2.
РЕАЛІЗАЦІЯ АВТОМАТИЗОВАНИХ МЕТОДІВ ІНСТАЛЯЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Автоматизація встановлення програмного забезпечення

Зміст цього розділу в основному служить методом відповіді на RQ2, але він також містить інформацію, яка стосується інших питань дослідження. Інформаційний потік, на який спрямована ця робота, можна побачити на рисунку 2.1, де адміністратор указує у форматі JavaScript Object Notation (JSON), які програми має встановити кожна машина. Приклад показано в лістингу 2.1.

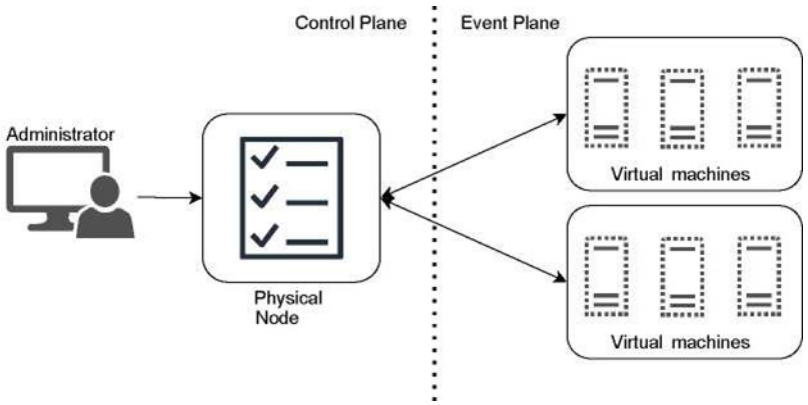

Рис. 2.1. Інформаційний потік для автоматичного встановлення програмного забезпечення

Потім це слід обробити, щоб бути сумісним із засобом автоматизації, який керує процесом встановлення. Оскільки CRATE використовує VirtualBox для розміщення віртуальних машин, було б краще, якби можна було дозволити зв’язку проходити через API VirtualBox, оскільки це зменшило б обсяг мережевого трафіку, який може заважати вправам. Якщо API VirtualBox не працює, можна використовувати SSH або WinRM. Потім засіб автоматизації дає вказівки менеджеру пакунків на віртуальній машині, які

програми слід інсталювати. Потім журнали інсталяції слід передати назад адміністратору у форматі JSON.

Щоб вирішити, який менеджер пакетів і інструмент автоматизації підійдуть для цього проекту.

```
{
    "vm_name" : "win10-ent2004-x64-
    off2013.src" , "username" :
    "Адміністратор" ,
    "password" : "пароль_адміністратора" ,
    "пакети" :  [
        {
            "ім'я" : "vlc" ,
            "version" : "3.0.14" ,
            "платформа" : "win_chocolatey"
        },
        {
            "ім'я" : "firefox" ,
            "версія" : "89.0" ,
            "платформа" : "win_chocolatey"
        },
        {
            "ім'я" : "git" ,
            "версія" : "2.31.1" ,
            "платформа" : "win_chocolatey"
        }
    ]
}
```

Лістинг 2.1. Список пакетів у форматі JSON

2.2. Вибір менеджерів пакунків. Вибір засобів автоматизації

Порівняння останніх доступних версій та їхнього віку для кількох програм наведено в таблиці 2.1. Деякі менеджери пакунків мають пакети, які посилаються на інсталятори або уніфіковані покажчики ресурсів (URL-адреси), які вказують на останню версію, доступну від постачальника програмного забезпечення, вони позначені «найновішою» у таблиці. Ці пакунки не потрібно оновлювати їхньому автору щоразу, коли випускається нова версія запакованої програми, але стає важко визначити, яку версію буде встановлено, оскільки версія, зазначена в описі або метаданих пакунка, буде неточною.

Хоча наявність останніх версій програмного забезпечення є важливою для багатьох користувачів, доступність старіших версій становить інтерес для питань, які розглядаються в цій дипломній роботі. Оскільки необхідно перевірити якомога більше експлойтів, існує потреба у великому наборі старіших версій, щоб збільшити ймовірність того, що буде знайдена вразлива

версія, яка відповідає експлойту. Порівняння доступності старіших версій кількох поширених програм показано в таблиці 2.2. Зауважте, що Google Chrome доступний лише як пакет автоматичного оновлення від Chocolatey і WinGet, а це означає, що неможливо встановити старіші версії, оскільки всі старіші версії

В таблиця 2.1 показано останні версії та їх вік, доступні з різних менеджери пакетіву сховищах все ще завантажуйте останню версію Chrome під час процесу встановлення.

Таблиця 2.1.

Останні версії та їх вік

Менеджер пакетів	Google Chrome		Mozilla Firefox		LibreOffice		VLC	
	Версія	Вік	Версія	Вік	Версія	Вік	Версія	Вік
Шоколадний	останній	1	85.0.2	14	7.1.0	20	3.0.12	37
WinGet	останній	1	84,0	70	7.0.1	179	3.0.11	253
Совок	87.0.4280.88	83	85.0.2	14	7.1.0.3	17	3.0.12	37
AppGet	останній	1	останні	0	6.4.2.2	334	3.0.11	253
Нініт	88.0.4324.190	1	85.0.2	14	7.1.0	20	3.0.12	37

З таблиці 2.2 видно, що Chocolatey пропонує найбільший вибір версій, а отже, він є найбільш релевантним для використання в цьому проекті. Хоча наявність старіших версій у сховищі пакунків не є гарантією, що всі вони працюють, оскільки деякі програми могли бути видалені постачальником або іншим чином зламані з часом, мати можливість спробувати встановити старішу версію краще, ніж просто можливість інсталювати останню версію.

Як згадувалося в попередньому розділі наявність старіших версій, які можуть містити вразливі - здібності, зовсім не означає, що певний менеджер пакунків уразливіший або гірший за інший. Зазвичай звичайний користувач повинен завжди встановлювати останню доступну стабільну версію кожного пакета, яка є типовою для всіх перевірених менеджерів пакетів. Наявність

старіших версій не становить загрози безпеці для звичайних користувачів цих менеджерів.

У Scoop, AppGet і Ninite немає можливості вказати версію під час встановлення пакета, що обмежує кількість доступних версій до однієї. Встановлення старіших версій за допомогою Scoop або AppGet можна здійснити шляхом вилучення старіших версій метаданих пакета із систем контролю версій, які використовуються цими інструментами для керування сховищами програмного забезпечення. Однак це потребує додаткової роботи, якщо дві програми з різних моментів часу потрібно інсталювати одночасно, а також ускладнює процес пошуку правильного пакета за бажаною назвою та версією програмного забезпечення.

Таблиця 2.2.

Кількість версій, доступних у різних менеджерах пакетів

Менеджер пакетів	Гугл хром	Mozilla Firefox	LibreOffice	VLC
	Версії	Версії	Версії	Версії
Шоколадний	1	192	42	47
WinGet	1	18	9	2
Совок	1	1	1	1
AppGet	1	1	1	1
Нініт	1	1	1	1

Інструменти автоматизації, представляють подібні рішення однієї проблеми: автоматичне виконання послідовності дій на наборі віддалених машин. Таким чином, вибір того, який з них використовувати, здебільшого залежить від потреби в спеціальних функціях, таких як перевага безагентним рішенням для цієї дисертації, особисті переваги щодо використовуваної мовидля визначення дій або які раніше використовувалися для інших проектів. У цьому випадку наявна робота вже була виконана у FOI з використанням Ansible у поєднанні з VirtualBox, і ми маємо певний попередній досвід роботи з Ansible і Python. Розробники Chocolatey також порівняли різні засоби автоматизації, з якими Chocolatey добре працює. Як видно з таблиці 2.3,

Ansible і Puppet підтримують усі розглянуті функції Chocolatey. Оскільки Ansible підтримує всі розглянуті функції та не працює без агентів, ці факти роблять його придатним рішенням для використання в решті дипломної роботи.

Таблиця 2.3.

Порівняння Chocolatey інтеграції засобів автоматизації

Менеджери конфігурації	Ансібль	Шеф-кухар	Маріонетка	сіль
Керувати пакетами	/	/	/	/
Встановити Chocolatey	/	/	/	/
Встановіть Chocolatey із внутрішнього	/	/	/	X
Керуйте джерелами	/	/	/	/
Керувати типом джерела	/	X	/	X
Керуйте функціями	/	X	/	X
Керувати налаштуваннями конфігурації	/	/	/	X

2.3. Опис процесу вибору інструментів та контроль списку пакунків

З попереднього розділу можна побачити, що Ansible і Chocolatey здаються придатними для використання в цій дипломній роботі, і тому вони будуть основою для решти роботи, описаної в цьому розділі. Хоча такі інструменти можна використовувати в поєднанні для автоматичного встановлення програм, FOI хотів би інструмент, який міг би створювати звіт про те, яке програмне забезпечення та версії доступні для встановлення, як пройшло встановлення, які помилки, якщо такі були, тощо в формат, який можна інтегрувати в інший інтерфейс. Щоб уникнути внесення значних змін у цю майбутню інтеграцію, також було б корисно, щоб цей інструмент діяв як уніфікований інтерфейс для кількох менеджерів пакунків, якщо одному не вистачає певних програм або потрібно підтримувати інші операційні системи. Це означає, що Ansible і Chocolatey можна використовувати для виконання важкої роботи, а наш розроблений інструмент служить оболонкою для них, яка розкриває необхідні функції в машиночитаному форматі. Якщо необхідно,

ця конструкція також дозволяє замінити Ansible/Chocolatey, якщо інший інструмент пізніше буде визначено як кращий варіант.

Хоча FOI мав список із 1615 пакетів Chocolatey з різними версіями для 149 унікальних пакетів, це була невелика підмножина з тисяч пакетів, доступних у сховищі спільноти Chocolatey. Про кожен пакет також була доступна додаткова інформація, як-от ім’я постачальника, опис, дата публікації тощо, яку було б корисно мати. На жаль, цю інформацію непросто зібрати без тисяч запитів до веб-API Chocolatey.

Завдяки моніторингу трафіку, створеного під час використання інструментів інтерфейсу командного рядка Chocolatey (CLI) для перегляду доступних пакетів, було виявлено, що він надсилає близько 700 запитів із загальним обсягом 120 МБ даних для відображення імен і останньої версії близько 5000 пакетів. На основі кількох зразків пакетів було визначено, що кожен пакет має в середньому близько 10 версій, що вимагає 2000 запитів для завантаження всіх пакетів і версій. Оскільки це було б еквівалентно виконанню трьох викликів «списку пакетів» за допомогою інструментів CLI, це було визнано достатньо неінвазивним, щоб це було можливо. Між кожним запитом була введена затримка в одну секунду, щоб зменшити вплив на сервери Chocolatey, і все потім було завантажено список пакетів. Загалом для цього знадобилося 3 118 запитів, що трохи більше, ніж очікувалося, але все ще в допустимих межах.

Важливо отримувати відгуки про встановлення на віртуальній машині, щоб побачити результати встановлення. Без них важко дізнатися, яке програмне забезпечення існує на віртуальній машині. Важливо також знати причину невдачі потенційної інсталяції. Оскільки відгуки, які Chocolatey надає від інсталяцій, є локальними для кожної віртуальної машини, інформацію потрібно передати на хост відповідним чином. Chocolatey надає зворотній зв’язок прямо до терміналу, з можливістю встановити позначку докладного тексту для отримання різної кількості інформації. Ansible має хорошу підтримку для отримання відповіді з Chocolatey, яку можна конвертувати у

файли JSON. Витягуючи інформацію з цих файлів JSON, можна відфільтрувати лише найрелевантнішу інформацію для відображення в терміналі чи десь ще. Файли JSON складаються з усіх журналів щодо Ansible і Chocolatey, того, як проходили завдання, потенційних кодів помилок тощо. Лістинг 2.2 містить приклад цього результату для інсталяції пакета, який не вдався через відсутність файлу на серверах постачальника.

Для відстеження всіх програм, включаючи інформацію про атаки, результати встановлення, версії та інше, була створена база даних. Щоб спростити розгортання цієї бази даних, її було розміщено на одній із робочих станцій, які використовувалися під час розробки (видно у верхньому лівому куті рисунка 2.1). MariaDB було обрано як базу даних, щоб мати можливість співпрацювати з оновленням колекції результатів встановлення та інших даних. База даних складається з одинадцяти стовпців: software, cpe, action, name, version, verified_working, installs_ok, result, interakcija_required, comment і result_internal. Перші три зі списку пакетів Chocolatey вище, за ними йде проаналізована назва та версія зі стовпця програмного забезпечення, а потім інформація про встановлення.

Вихідні дані процесу встановлення для кожного пакета аналізуються на загальні рядки помилок, щоб представити більш стислу версію повідомлення про помилку. Код результату, заснований на цьому, також зберігається в базі даних, щоб вказати тип помилки. Таблиця 2.4 містить список цих кодів результатів.

Таблиця 2.4.

Приклад структури бази даних

назва	версія	перевірено працюю	installs_ok	результат	взаємодія_потрібна ...
Git	2.19.0	помилковий	правда	в порядку	– ...
OpenOffice	4.1.7	помилковий	правда	в порядку	– ...
openvpn	2.2.2	помилковий	правда	installer_hangs	так ...

2.4. Відображення вразливостей. Онлайн-тести для встановлення

Щоб відповісти на запитання RQ3, буде проведено тестування використання встановлених програм. Щоб знати, який експлойт використовувати проти конкретної програми, потрібно створити відповідність між програмою та можливими експлойтами. Оскільки експлойт часто прив’язаний до певної версії чи кількох версій програми, використання кодів CPE створює уніфіковану та стандартизовану назву для кожної програми та версії. Потім кожен із них зіставляється з набором можливих модулів Metasploit, які можна використовувати для використання встановленої програми.


```
{
    "uuid" : "39d0dd24-2e12-4a75-a1f3-52b107db3ed3" ,
    "лічильник" : 25 ,
    "stdout" : "fatal: [win10-ent2004-x64-off2013.src]: ПОМИЛКА! =>
        {\"змінено\": false, \"command\": \" [...]\" " ,
    "start_line" : 26 ,
    "кінцевий рядок" : 28 ,
    "runner_ident" : "d41cedc2-1f22-4c14-8036-1bb3bc4420c1" ,
    "event" : "runner_on_failed" ,
    "pid" : 64765 ,
    "створено" : "2021-05-04T11:56:22.861142" ,
    "parent_uuid" : "9f306331-db9d-d558-fcd1-00000000000d" ,
    "event_data" :      {
        "playbook" : "playbook.yml" ,
        "playbook_uuid" : "e05eb35e-ad1c-4e3a-a992-7e8b53bb795a" ,
        "play" : "Встановити пакети Chocolatey" ,
        "play_uuid" : "9f306331-db9d-d558-fcd1-000000000006" ,
        "task" : "Установити WindowsAzureLibsForNet 2.9" ,
        "task_uuid" : "9f306331-db9d-d558-fcd1-00000000000d" ,
        "task_action" : "win_chocolatey" ,
        "task_args" : "" ,
        "task_path" : "playbook.yml:36" ,
        "хост" : "win10-ent2004-x64-off2013.src" ,
        "remote_addr" : "win10-ent2004-x64-off2013.src" ,
        "res" : {
            "змінено" : false ,
            "виклик" :      {
                "module_args" :       {


                }
            },
            "rc" : 404 ,
            "command" : "C:\\ProgramData\\chocolatey\\bin\\choco.exe встановити
            WindowsAzureLibsForNet -fail-on-unfound -yes -no-progress -limit-output -
            timeout 2700 -version 2.9",
            "stdout" : "Встановлення наступного
                пакети:\r\nWindowsAzureLibsForNet\r\n [...]" ,
            "stderr" : "" ,
            "msg" : "Помилка встановлення пакета(ів)
            WindowsAzureLibsForNet" , "stdout_lines" :      [
                "Встановлення таких пакетів:" ,
                "WindowsAzureLibsForNet v2.9 [Схвалено]" ,
                "встановлення файлів пакета windowsazurelibsfornet завершено.
                    Виконання інших кроків встановлення." ,
                «Спроба отримати заголовки для
                ^ https://download.microsoft.com/download/B/4/A/B4A8422F-C564-j
                ^ 4393-80DA-6865A8C4B32D/MicrosoftAzureLibsForNet-x64.msi ^
                помилка." ,
                "Віддалений файл або не існує, або є неавторизованим, або
                    заборонений для url [...]" ,
                "Помилка під час запуску 'C:\\ProgramData\\chocolatey\\lib\\j
                    WindowsAzureLibsForNet\\Tools\\ChocolateyInstall.ps1'."
                , "Chocolatey встановив пакети 0/1 . 1 пакетів не вдалося."
                ,
                " Подробиці дивіться в журналі
                    (C:\\ProgramData\\chocolatey\\logs\\chocolatey.log)." ,
            ],
            "stderr_lines" :   [],
        },
        "початок" : "2021-05-04T11:55:59.944377" ,
        "end" : "2021-05-04T11:56:22.861014" ,
        "тривалість" : 22.916637 ,
        "ignore_errors" : правда ,
        "uuid" : "39d0dd24-2e12-4a75-a1f3-52b107db3ed3"
    }
}
```


Лістинг 2.2. приклад JSON від Ansible

Таблиця 2.5.

Можливі коди результатів встановлення

Результат	опис
в порядку	Встановлення успішно завершено.
не знайдено	Один або кілька файлів, на які посилаються в пакеті, не вдалося - завантажити з Інтернету.
можливо зламаний	Chocolatey позначив пакет як можливо зламаний через невдалі автоматизовані тести.
not_in_repo	Пакет не знайдено в репозиторії Chocolatey. Причиною цього можуть бути помилки, але при використанні існуючого списку пакунків найімовірнішою причиною є те, що пакет було видалено зі сховища.
невідповідність контрольних сум	Контрольна сума отриманого файлу не відповідає вказаній у пакеті.
installer_hangs	Процес інсталяції так і не завершився, і його довелося скасувати. Це може статися, коли інсталятор вимагає введення користувача, що спричиняє проблеми під час автоматичного встановлення.
інша_помилка	Сталася помилка іншого типу, яка спричинила збій встановлення.

Це робиться за допомогою сценарію, наданого FOI, який використовує логіку зіставлення рядків на основі вхідних даних, у цьому випадку назви програми, постачальника та версії. Результатом є список значень, розділених комами (CSV), із назвою та версією програмного забезпечення, кодом CPE, рівнем достовірності відповідності між програмою та кодом CPE, можливими експлойтами для цього коду CPE та рівнем достовірності відповідності. між CPE і експлойтом. Див. таблицю 2.6 для прикладу запропонованого CPE та відповідного експлойту для кількох програм. Однак текстове зіставлення не завжди забезпечує правильний CPE або відповідність експлойту, що важливо для успішного експлойту. Бо навіть якщо програма є вразливою, атака за допомогою неправильного експлойту не вдасться. Оптимізація цього алгоритму виходить за рамки цієї дисертації, але вдосконалення можна зробити, не змінюючи сам алгоритм, регулюючи кількість даних, які надаються для кожного пакета, щоб він міг забезпечити гарну відповідність. Однак надання занадто великої кількості інформації може дати йому можливість порівняти з нерелевантною інформацією, тому слід бути

обережним, вирішуючи, які вхідні дані йому надати. Результат цього відображення також зберігається в базі даних програм.

Таблиця 2.6.

Приклад пакетів Chocolatey із CPE та діями

програмне забезпечення	CPE	Дія
php-manager 2.3.0	cpe:/a:php:php:4.3	exploit/multi/http/activecollab_chat
IcoFx 1.6.4.02	cpe:/a:icofx:icofx:1.6.4	exploit/windows/fileformat/icofx_bof
freeSSHD 1.2.6 mirc 7.34	cpe:/a:freesshd:freesshd:1.2.6	exploit/windows/ssh/freesshd_authbypass
	cpe:/a:mirc:mirc:6.34	exploit/windows/misc/mirc_privmsg_server

Хоча цей процес працює досить добре, коли можна знайти точні збіги або лише кілька CPE, які відповідають даній програмі, він часто може запропонувати зіставлення з високою впевненістю, навіть якщо є невеликі відмінності, які можуть бути дуже важливими під час пошуку вразлива версія. У першому рядку прикладу в таблиці 2.6 пакет php-manager 2.3.0 зіставлено з CPE, що відповідає PHP 4.3. Це абсолютно неправильно, оскільки php-manager використовується для керування кількома встановленнями PHP і не буде вразливим до атак, спрямованих на сам PHP. Другий ряд більш точний, де є лише невелика різниця в номерах версій. Третій рядок – це точна відповідність, і пов’язаний експлойт, сподіваємось, працюватиме з цим програмним забезпеченням. Останній рядок містить правильну назву програми, але версія 7.34 зіставлена з CPE із версією 6.34. Зіставник на основі рядків припускає це з високою впевненістю, оскільки лише один символ відрізняється, але різниця між версіями 6.34 і 7.34, швидше за все, занадто велика, щоб змусити експлойт, націлений на 6.34, працювати на версії 7.34.

CPE також містить поле з назвою постачальника програми, яке іноді відсутнє або збігається з назвою програми, але якщо воно є, його можна порівняти за допомогою інструментів, створених FOI. Однак постачальник зазвичай відсутній у назві пакетів Chocolatey, а пакет, що містить Mozilla Firefox, просто називається firefox. Це зменшує -можливості, які має

відповідник для отримання точних результатів, і таким чином це буде також корисно мати назву постачальника для кожного пакунка в репозиторії Chocolatey . Завдяки списку пакетів, отриманому з Chocolatey, ця інформація тепер доступна для кожної програми у сховищі.

Щоб перевірити, які пакети Chocolatey працюють і можуть правильно встановити свою програму, було проведено автоматизовані тести. Локальні віртуальні машини (за межами CRATE, щоб вони мали доступ до Інтернету) під керуванням Windows 10 використовувалися як тестовий стенд для встановлення різних програм. За допомогою створеного інструменту набір неперевірених програм було обрано випадковим чином, щоб на одній машині тестувалася лише одна версія програми. Можна створити кілька наборів для паралельного тестування кількох версій, але для цього потрібні додаткові віртуальні машини, оскільки встановлення двох версій одного пакета на одній машині призведе до конфліктів. На жаль, API VirtualBox може бути небезпечним для потоків, оскільки одночасний запуск іноді не вдається через помилки, пов’язані з VirtualBox. Це означає, що навіть із кількома віртуальними машинами інсталяційні тести не можна виконувати на одному комп’ютері, не заважаючи один одному. Одним із рішень було б використання кількох комп’ютерів, що для цього проекту обмежено двома фізичними робочими станціями та чотирма фізичними вузлами в CRATE.

2.5. Обмеження швидкості та надмірне використання внутрішнього сховища

Chocolatey реалізував систему обмеження швидкості, щоб пом’якшити зловживання їхнім API, де, якщо певна кількість запитів буде зроблено протягом певного періоду часу, вони тимчасово заборонять запитувану адресу Інтернет-протоколу (IP) на одну годину. Не всі виклики API вважаються однаковими, наприклад, завантаження самого Chocolatey обмежено п’ятьма повтореннями на хвилину на IP-адресу, а завантаження інших пакетів

обмежено 20 спробами на хвилину на IP-адресу. Окремі користувачі, ймовірно, ніколи не досягнуть цього обмеження, але організації, які працюють за проксі-сервером або спільно використовують IP-адресу через трансляцію мережевих адрес (NAT), цілком можуть досягти цього обмеження. Вони також мають інший API для отримання інформації про кожен пакет Chocolatey, який інструмент командного рядка Chocolatey використовує для переліку доступних пакетів. Цей API не такий чутливий до великих обсягів трафіку, як інші, оскільки команда переліку Chocolatey робить кілька сотень запитів протягом невеликого періоду часу. Однак, хоча він може бути не таким чутливим, важливо не зловживати ним.

Chocolatey також має більш жорсткий механізм захисту себе та спільноти від надмірного використання, результатом якого є тимчасова (але не автоматично скасована) заборона IP-адреси. Точне обмеження не є загальнодоступним, але вони надають приблизні вказівки щодо того, як мають поводитися клієнти, які використовують їхній API. Надмірне використання може бути, наприклад, якщо потрібно встановлювати в середньому 100 пакетів на годину протягом певного періоду часу. Оскільки Chocolatey очікує, що це станеться через неправильну конфігурацію, а не через зловмисний намір, до них можна зв’язатися, щоб вирішити проблему.

Простим рішенням, щоб уникнути надмірного навантаження на послуги Chocolatey, було б просто застосувати затримку між завантаженням кожного пакета, але це буде лише ефективний на дуже обмеженій кількості машин і взагалі погано масштабується під час налаштування цілих мереж і кількох машин. Кращим рішенням, яке також рекомендує Chocolatey, було б замість цього розмістити внутрішній репозиторій пакетів, який також може кешувати завантажені пакети з Інтернету, щоб зменшити навантаження на їхні сервери, якщо пакет завантажується кілька разів.

Внутрішнє сховище. Цей розділ стосується RQ1 і буде зосереджено на тому, як пакети Chocolatey можна надійно зберігати у внутрішньому сховищі.

Як згадувалося, то Chocolatey рекомендує організаціям розміщувати власне внутрішнє сховище з кількох причин. Це не тільки зменшує навантаження на їхні сервери, але й забезпечує стабільність і надійність, коли залежності можуть зберігатися локально. Це також зменшує час встановлення для великих пакетів, оскільки отримання файлів із внутрішнього сервера швидше за все буде швидшим, ніж завантаження даних безпосередньо з Інтернету Існують обмеження щодо того, що Chocolatey може мати у своїх загальнодоступних сховищах через права на розповсюдження. Їх послуга автоматичного пакування також неможлива через їх тарифний план, який коштуватиме понад 150 000 шведських крон на рік, якщо припустити, що всіма віртуальними машинами в CRATE потрібно буде керувати за допомогою цієї системи.

У попередньому дослідженні FOI було зроблено доказ концепції щодо того, як Chocolatey можна використовувати в CRATE. У цьому дослідженні було зроблено висновок, що Chocolatey є хорошим кандидатом для CRATE щодо встановлення програм Windows, але потрібно ще більше роботи. Іншим результатом цього дослідження стало внутрішнє сховище на основі Sonatype Nexus Repository Manager OS S2 , яке було заповнено кількома пакетами Chocolatey. Але щоб репозиторій був корисним, пакунки Chocolatey потрібно завантажити разом із усіма залежними від них файлами, а потім перепакувати, щоб пакунки могли знайти все, що їм потрібно, із цього сховища. Просте використання репозиторію кешування не спрацює, оскільки воно може кешувати лише завантаження пакетів, а не фактичні інсталятори, які зазвичай витягуються з Інтернету під час встановлення пакета.

Щоб вирішити проблему, коли пакети не є автономними, було створено інструмент, який автоматично завантажує та перепаковує (інтерналізує) існуючі пакети. Насамперед потрібна була інформація про пакети. Сценарій Python було створено для отримання інформації про всі доступні пакунки Chocolatey, зокрема назву, версії, URL-адреси та інформацію про перевірку. Були вжиті запобіжні заходи, щоб гарантувати, що API не

зазнає занадто великого стресу. Ця інформація була структурована у форматі розширюваної мови розмітки (XML), що дозволило легко отримувати відповідну інформацію. Інформація перевірки є результатом автоматичних перевірочних тестів Chocolatey.. Вони спочатку перевіряють, чи пакет містить необхідні метадані відповідно до встановлених специфікацій Chocolatey, а потім, чи можна встановити та видалити в тихому режимі без взаємодії з користувачем. Ці тести проводяться періодично як на нових, так і на вже існуючих пакетах. Проте все ще є деякі старі версії, які все ще можуть бути зламані, навіть якщо в інформації про перевірку зазначено інше, тому інформації про перевірку та перевірку не можна сліпо довіряти. Але за допомогою цієї інформації відомі зламані пакети можна видалити. XML-файл також використовується для отримання інформації про постачальника програм.Ця інформація покращує зіставлення між програмою та кодами CPE.

Друга частина полягає в тому, щоб завантажити пакети, їхні залежності та перепакувати їх перед завантаженням у репозиторій. Це робиться за допомогою сценарію Python, який запитує пакет, використовуючи його назву та версію, щоб завантажити вміст у тимчасовий каталог. Завантажений пакет є пакетом NuGet, який потім відкривається та змінюється. Файли, які потрібно змінити, це файл специфікації, у якому зазначено, які залежності має пакет, і сценарії Powershell, які містять URL-адреси, які стосуються встановлення пакета. Ця частина виконується рекурсивно, щоб усі залежності також були завантажені та переписані. Для кожного файлу в пакеті здійснюється пошук URL-адрес за допомогою регулярного виразу. Оскільки ці URL-адреси можуть з’являтися в коментарях або посилатися на документацію замість двійкового файлу, відповідні URL-адреси фільтруються таким чином, що завантажуються ті, що містять одне з розширень у лістингу 2.3 .

```
[ . exe, . msi, . zip, .7 z, . дьоготь . gz, . ключ, . pgp, .
gpg, . зростання, . sig, . xpi, . sha256, . mcy, . iso, . rar, .
```

Лістинг 2.3. Список розширень файлів, які завантажуються інтерналізатором

Це не вичерпний список, оскільки важко врахувати кожен тип файлу, який може знадобитися завантажити програмі. Наведений вище список було створено ітераційно шляхом спроб інтерналізувати набір пакунків, визначення файлів, які не вдалося завантажити, додавання їхніх розширень до цього списку та повторення. Щоб врахувати невідомі типи файлів, URL-адреси, які з’являються як частина призначення змінної змінній, що містить підрядок "url", завантажуються, навіть якщо вони не містять жодного з перелічених вище розширень. Це тому, що пакети Chocolatey часто містять змінну з іменем url, url64, urlbase або подібну, яка завантажується як частина процесу встановлення. Оскільки деякі постачальники використовують такі URL-адреси, як https://example.com/download?version=latest , які не мають розширення, яке б вказувало, який тип файлу буде завантажено, це може допомогти успішно інтерналізувати більше пакетів, уникаючи URL-адрес, які вказують на документацію. або інша інформація, яка не має значення для автоматизованого процесу встановлення. Потім кожен завантажений файл завантажується у внутрішнє сховище, а URL-адреса, яка вказує на нього, переписується таким чином, щоб вона вказувала на внутрішнє сховище, а не на Інтернет.

Щоб покращити співвідношення пакунків, які можна успішно інтерналізувати, сценарій також можна запускати в напівручному режимі, де користувачеві надається можливість ігнорувати відсутні файли (якщо URL-адреса була неправильно ідентифікована або не потрібна для успішної інсталяції) і запит на заміну значення для змінних, які з’являються всередині URL-адрес. Оскільки деякі супроводжувачі пакетів підтримують кілька мов або просто хочуть зменшити зусилля з оновлення пакета щоразу, коли постачальник випускає нову версію програми, URL-адреси можуть містити змінні, які оцінюються та замінюються під час встановлення. Це створює проблему для інтерналізатора, оскільки URL-адреса недійсна під час вилучення з файлу. Для простіших типів посилань на змінні, де змінну можна замінити простою операцією пошуку та заміни, значення, надане

користувачем, можна використовувати для видалення посилання на змінну та заміни його статичним значенням. Деякі пакунки використовують змінні для таких речей, як мова/локаль або поточна версія пакунка, які користувач сценарію може визначити та ввести вручну. Це може обмежити деякі функціональні можливості таким чином, що можна встановити лише одну мову, але це краще, ніж пакет повністю виходить з ладу.

Також реалізовано допоміжний ручний режим, у якому користувачеві пропонується вручну редагувати кожен сценарій у пакеті, що може бути корисним для більш складних замін змінних . Потім сценарій піклується про отримання URL-адрес, переписує їх і натискає змінений пакет у внутрішній репозиторій. Це економить час у порівнянні з повністю ручним пакуванням і забезпечує велику гнучкість для пакунків, які вважаються обов'язковими.

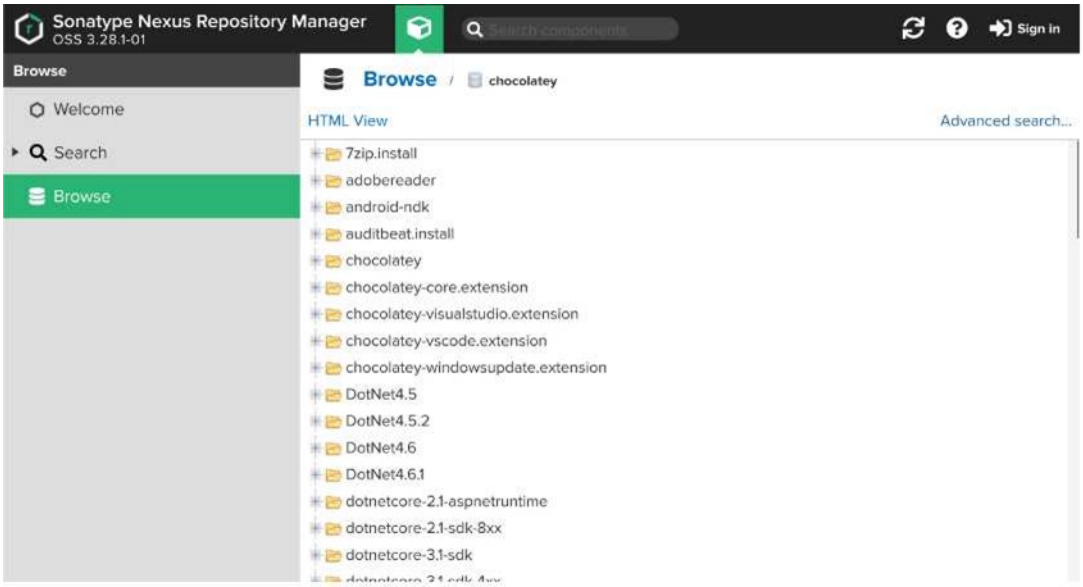

Рис. 2.2. Внутрішнє сховище з використанням Sonatype Nexus 3

Коли пакет було змінено, його можна завантажити у внутрішній репозиторій, що робиться за допомогою API. На рисунку 2.2 показано, як виглядає інтерфейс менеджера сховища після завантаження кількох пакетів. Щоб використовувати внутрішній репозиторій замість публічного, потрібно просто змінити джерело для Chocolatey, що можна зробити за допомогою Ansible.

2.6. Тести встановлення в автономному режимі та вибір експлойтів для оцінки

Після інтерналізації пакунків, які можна було успішно інсталювати з доступом до Інтернету, їх також протестували в автономному режимі, налаштувавши Chocolatey отримувати пакети з внутрішнього репозиторію замість загальнодоступного сховища спільноти. Це служить для перевірки того, чи пакунки все ще працюють належним чином після того, як їх було перепаковано, або чи залишилися залежні від Інтернету частини, які інтерналізатор не зміг переписати. Ці автономні тести проводяться в CRATE, що означає відсутність доступу до Інтернету та обмежені способи доступу до самої віртуальної машини. Трафік до внутрішнього сховища маршрутизується через проксі-сервер, який дозволяє отримати доступ до сховища зсередини ізольованої площини подій. Щоб мінімізувати мережевий трафік і журнали, створювані на машинах усередині CRATE, кращим способом керування віртуальними машинами є VirtualBox AP I 3 . Цей API також має створений спільнотою Python bindin g 4 який FOI використав для створення плагіна для Ansible, який дозволяє йому спілкуватися через цей API замість мережі через WinRM. Лише з невеликою модифікацією цей плагін можна було б використовувати для виконання підручників із фізичного вузла з інструкціями про те, які програми встановити, а Chocolatey встановить їх у віртуальну машину.

Щоб відфільтрувати деякі пропозиції експлойтів, які з меншою ймовірністю можуть бути використані , версію кожної програми та її відповідну версію CPE порівнювали за допомогою метрики відстані, яка більше підходить для номерів версій. Метод, використаний для цього, складався з порівняння кожної частини специфікатора версії з пунктиром між кожною парою наданих версій. Абсолютну різницю між цілими числами, що утворюють ці частини, було помножено на ваговий коефіцієнт (починаючи з 10), який зменшено на коефіцієнт 10 для кожної частини зліва направо в

номері версії. Потім ці відмінності були підсумовані для всього номера версії та повернуті як показник відстані між кожною парою. Порівняння припиняється, коли в одній із версій більше не залишається частин для порівняння. Продовжуючи припущення, що відсутні поля мають бути заповнені нулями, у деяких випадках могло б бути краще, але в пакетах Chocolatey іноді до номера версії додається дата створення пакета. Це спричинило б великі відмінності між, наприклад, 1.2.3.20210303 та 1.2.3 (у цьому випадку різниця 202103.03), навіть якщо фактичні версії програми дуже схожі або абсолютно однакові. Див. таблицю 2.7 для прикладу метрики відстані, яка обчислюється для кількох пар версій. Лістинг 2.4 також містить псевдокод обчислення метрики.

Таблиця 2.7.

Приклад відмінностей версій для пар версій програми та версій CPE

Версія програми	Версія CPE	Метрика відстані
6.6.0	6.2.0	4.0
2.4.4	2.0.1	4.3
2.2.4	2.2.8	0,4
1.3.1035	2.1	12.0

Інший етап фільтрації, який було застосовано до списку пакетів, полягав у використанні структурованої інформації про кожен пакет Chocolatey для перевірки наданих пропозицій CPE. Оскільки метадані містять окремі поля, наприклад, для назви програми та постачальника, можна розрахувати показник відстані між назвою 0 і назвою постачальника 0 для певної програми та запропонованого CPE. Наданий сценарій відображення призначений для прийому неструктурованих вхідних даних і намагатиметься зіставити будь-яку частину потенційного CPE з будь-якою частиною вхідних даних для певної програми. Фільтрування на основі подібності між полями одного типу зменшить кількість помилкових пропозицій, уникаючи необхідності повторно впроваджувати частину сценарію відображення, що пропонує експлойт.

```
def version_similarity (ver1, ver2):
    diff = 0 weight = 10

    # Об'єднайте основні версії, потім другорядні, а потім
    рівень виправлення для версії в zip (ver1 . split( '.'
    ), ver2 . split( '.' )): ver1 part =
    strip_nondigit_chars(ver[ 0 ]) ver1 part = int
    (ver1_part ) if ver1_part else 0

        ver2 part = strip_nondigit_chars(ver[ 1 ])
        ver2_part = int (ver2_part) if ver2_part else 0

        diff = diff + abs (ver1 part - ver2 part) * вага
        вага = вага / 10

    повернення дифф
```

Лістинг 2.4. Псевдокод для метрики різниці версій

Пакет Python fuzzywuzz у 5 використовується для обчислення метрики відстані між метаданими пакета та запропонованим CPE. Цей пакет використовує подібний метод обчислення цієї відстані до того, який використовується існуючим сценарієм FOI для відображення тексту довільної форми в CPE, але також включає етапи попередньої обробки, які можуть ігнорувати порядок слів або допускати часткові збіги між текстовими рядками [65]. Це може бути корисним для пакетів, у яких ім’я пакета Chocolatey може відрізнятися від зазначеного в CPE, наприклад VLC проти медіапрогравача VLC, але також може призвести до того, що Git для Windows випадково збігається з версіями самої операційної системи Windows. Можливість відокремити поле постачальника від поля імені допомагає в цьому відношенні, оскільки малоймовірно, що постачальник Git для Windows The Git Development Community відповідатиме постачальнику Windows Microsoft. Лістинг 2.5 містить опис псевдокоду обчислень метрики.

```
від fuzzywuzzy імпортувати пух

def get_similarity (постачальник, назва, версія, cpe):
    cpe_vendor, cpe_name, cpe_version = split_cpe(cpe)
    cpe_vendor = cpe_vendor . replace( " " , " " )
    cpe_name = cpe_name . замінити ( " " , " " )
    vendor_ratio = 100 - пух . token_set_ratio(vendor, cpe_vendor)
    name_ratio = 100 - fuzz . token_set_ratio(name, cpe_name)
    ver_similarity = version_similarity(version, cpe_version)
    return (vendor_ratio, name_ratio, ver_similarity)
```

Лістинг 2.5. Псевдокод для обчислення різниці між назвою та постачальником

Приклад усіх трьох показників (різниця постачальника, різниця в імені та різниця у версії) можна побачити в таблиці 2.8. Зауважте, що коефіцієнт відповідності для постачальників і імен, створених fuzzywuzzy, був інвертований, щоб відповідати поведінці поля різниці версій, де нижче значення вказує на ближчу відповідність. Таким чином, поля різниці постачальника та різниці в назві варіюються від 0 до 100, де 0 є найкращим збігом, а поле різниці версій – від 0 до 8.

Приклад однакових відображень, які порівнюються з дозволеними частковими збігами, наведено в таблиці 2.9. Python і Adobe Flash Player отримують менші відмінності, оскільки вони частково збігаються із запропонованим CPE, але інші програми з меншим перекриттям можуть залишатися з тією ж різницею. Програма dotnet-windowshosting також стає набагато кращою для операційної системи Windows NT , оскільки «Windows» присутній повністю в назві програми, що менш бажано. Для процесу вибору експлойтів використовувалися нечасткові показники, оскільки це зменшувало б кількість помилкових спрацьовувань, коли програма зіставлялася з неправильним CPE.

Таблиця 2.8.

Приклад відмінностей назви та постачальника

Продавець	застосування	Версія	CPE	Продавець дифф	застосування дифф	Версія дифф
пітон	python2	2.7.17	cpe:/a:python:python:2.6.7	0	8	2.0
imageglass	imageglass	4.0.4.15	cpe:/a:php:php:5.4.0	100	100	14.4
глинобитни	flashplayerppapi	27.0.0.183	cpe:/a:adobe:flash_player:7	0	21	200
крапка	dotnet-windowshosting	5.0.3	cpe:/o:microsoft:windows_nt	67	50	50

Таблиця 2.9.

Приклад відмінностей назви та постачальника з дозволеними частковими збігами

Продавець	застосування	Версія	CPE	Продавець дифф	застосування дифф	Версія дифф
пітон	python2	2.7.17	cpe:/a:python:python:2.6.7	0	0	2.0
imageglass	imageglass	4.0.4.15	cpe:/a:php:php:5.4.0	100	100	14.4
глинобитни	flashplayerppapi	27.0.0.183	cpe:/a:adobe:flash_player:7	0	8	200
крапка	dotnet-windowshosting	5.0.3	cpe:/o:microsoft:windows_nt	33	0	50

2.7. Критерії відбору експлойтів. Автоматичне тестування експлойтів. Використовуйте критерії відбору

Серед програм, які можна було успішно встановити і запакувати, ті, у яких різниця в назві програми та постачальника становить 15 і менше, а версія версії становить 5 і нижче були обрані як кандидати на експлуатацію. Ці значення було вибрано, щоб обмежити загальну кількість атак до контрольованого рівня, коли деякі з найбільш нерелевантних атак будуть пропущені. Оскільки Metasploit містить близько 2000 експлойтів, і вони працюють лише проти підмножини версій програмного забезпечення, на яке вони націлені, тестування більшої кількості комбінацій програмного забезпечення та версій також призведе до невдачі багатьох атак, оскільки всі доступні експлойти вже перевірені. Таким чином, якщо спочатку тестувати найкраще підібрані експлойти, буде обмежена користь від продовження тестування експлойтів після цього моменту.

Дозвіл невеликої різниці для назви та постачальника компенсував би незначні відмінності в угоді про іменування, наприклад, у випадку python2 проти python у таблиці 2.8. Максимальна різниця версій у п’ять також дозволить перевірити версії, які дещо не відповідають, якщо вразливість була присутня в кількох версіях. Таким чином, незначні відмінності все одно будуть прийняті, але ті, де основна версія відрізняється (1.x проти 2.x), будуть відхилені. Це також допомагає виявити помилки в процесі зіставлення імені та постачальника, коли програма та її кандидат на CPE можуть використовувати різні схеми версій і, отже, мати велику різницю у версіях. Наприклад, система контролю версій Git може бути зіставлена з CPE, що посилається на веб-службу GitLab , оскільки вони мають схожі назви, але оскільки остання версія пакета Git є 2.31.1 а GitLab працює на версії 13.11.3, різниця у версіях становитиме 130.2, що буде відхилено.

Як згадувалося існуючі інструменти, такі як SVED, використовуються разом із можливістю інсталювати вразливе програмне забезпечення для

виконання оцінки рівня успіху загальнодоступних експлойтів. Щоб мати можливість тестувати велику кількість експлойтів на багатьох частинах програмного забезпечення, сам процес експлуатації також має бути автоматизованим.

Необхідно підготувати набір віртуальних машин для розміщення в CRATE, щоб можна було запускати атаки за допомогою SVED. У них має бути встановлено якомога більше вразливих програм, щоб якомога більше експлойтів мали шанс на успіх. Оскільки автоматизований інструмент інсталяції програмного забезпечення може встановлювати програми безпосередньо в CRATE без доступу до Інтернету, це можна використовувати на існуючих голих образах ОС, щоб уникнути необхідності створювати нові образи з додатками, для встановлення програм, які добре збігаються з експлойтом, доступним у Metasploit, буде використано зіставлення програми-CPE з експлойтом, згадане в. Проте низка експлойтів, які були зіставлені з цими програмами, навряд чи буде успішною через відсутність передумов або через те, що деякі експлойти залежать від кількох програм. Наприклад, експлойт проти медіаплеєра QuickTime також вимагає встановлення браузера Safari, щоб експлойт був доставлений на цільову машину. Ці експлойти іноді також доступні в автономній версії, де жертва повинна відкрити надісланий їй файл. Пропоновані експлойти були відфільтровані на основі повідомленої точності пропозицій, показників, і сімейства ОС експлойтів, щоб видалити експлойти, які не працюватимуть у Windows. Таким чином, багато експлойтів можуть бути незастосовні для встановлених програм, але оскільки тестування може відбуватися протягом ночі, це не займе багато додаткового часу.

Щоб прискорити процес запуску атак, кілька інжекторів SVED будуть використовуватися для запуску кількох атак одночасно. Для цього також потрібні кілька екземплярів цільової системи, оскільки одночасний запуск кількох експлойтів на одній машині може спричинити конфлікти та призвести до нестабільності, якщо служби виходять з ладу в результаті невдалого чи ненадійного експлойту.

По-перше, цільова віртуальна машина відновлюється до моментального знімка завідомо справного стану, де встановлено всі відповідні програми, але до спроби будь-якої атаки. Це необхідно, оскільки деякі експлойти, особливо ті, які націлені на служби замість клієнтських програм, можуть впливати на цільову програму таким чином, що вона стає нестабільною або виходить з ладу після спроби експлойту. Експлойти також можуть залишати файли чи інші сліди, які можуть перешкодити подальшим спробам. Використання знімків дає кожному експлойту найкращі шанси на успіх і гарантує, що експлойти, які перевіряються пізніше в послідовності, не будуть у невигідному становищі (через більш нестабільну систему) порівняно з тими, які з’являються раніше.

По-друге, інжектор SVED, який доставлятиме експлойт, перемикається в ту саму віртуальну розширювану локальну мережу (VXLAN), що й цільова система, завдяки чому інжектор відображається в тому самому сегменті мережі, що й ціль, і, таким чином, він може з ним спілкуватися.

По-третє, інжектор намагається перевірити, чи ціль перебуває в мережі та відповідає, надсилаючи їй ехо-запит (ping) протоколу керування повідомленнями Інтернету (ICMP) і перевіряючи відповідь.

Нарешті експлойт запускається. Результат експлойту (незалежно від того, чи була створена будь-яка форма сеансу командного рядка в цільовій системі) і журнали процесу експлуатації будуть зібрані SVED і збережені для подальшого аналізу. Послідовність дій, наприклад копіювання файлів на цільову машину та відкриття їх за допомогою певних програм, уже наявна в SVED з метою запуску експлойту на стороні клієнта, який вимагає певної взаємодії з користувачем. Було створено додаткові тригери для експлойтів, які з’являлися кілька разів проти різних версій програмного забезпечення або з багатьма можливими налаштуваннями для цілі експлойту (яка ОС або версія програми використовується, SVED може спробувати всі цілі автоматично).

Для аналізу цих журналів використовувався інструмент SVIZ (щоб візуалізувати атаки та полегшити вибір відповідної інформації. Оскільки тест

було запущено з чотирма машинами, що працюють паралельно, кількість журналів була великою.

2.8. Ручне тестування експлойтів. Вразливі стани

Після автоматичного тестування експлойтів, експлойти перевірялися вручну та порівнювалися зі списком встановлених програм, щоб визначити, чи матимуть вони ймовірність успіху. Оскільки процес відображення базується на зіставленні рядків, високий показник достовірності не є гарантією того, що вказаний експлойт підходить для даної програми та версії. Ця перевірка була виконана шляхом ознайомлення з описом кожного запропонованого експлойта та перевірки, чи націлена правильна програма, чи збігається версія програми з версіями, які підтримує експлойт, і чи є якісь особливі передумови, які не виконуються за замовчуванням (наприклад, повинен бути присутній додатковий модуль, має бути створений обліковий запис користувача або таблиця бази даних).

Експлойти, які були визначені як такі, що добре відповідають відповідним програмам, були перевірені вручну, щоб визначити, чи можна їх успішно виконати за межами SVED. Мета цього полягала в тому, щоб підтвердити, що SVED може правильно використовувати виявлені вразливості, і перевірити, чи будь-яка з них буде поводитися по-іншому під час кількох запусків. Оскільки експлойти іноді не є детермінованими, це ручне тестування намагається визначити, чи деякі експлойти могли бути невдалими через природну випадковість місця завантаження об’єктів у пам’ять або інші параметри, які не можуть бути відомі заздалегідь. Це ручне тестування складається із запуску експлойтів за межами SVED і, якщо це вдається, створення послідовності експлойтів у SVED, де один експлойт запускається кілька разів проти цілі. Деякі експлойти також можуть покладатися на додаткову взаємодію, окрім того, що наразі доступно в SVED, наприклад, відхилення спливаючих вікон із запитом, чи бажає користувач змінити веб-

переглядач за замовчуванням, прийняти Ліцензійну угоду з кінцевим користувачем (EULA) або ввімкнути автоматичні оновлення.

Інший метод ручного тестування, який використовувався, полягав у використанні віртуальної машини на стороні CRATE із запущеним фреймворком Metasploit, що дозволяло вручну готувати та виконувати атаки проти жертви. Запропонований експлойт, про який йде мова, можна було б виконати з більшим контролем і з можливістю подальшого дослідження потенційних параметрів конфігурації експлойту. Цей метод також дає можливість внести необхідні корективи для точного налаштування атаки. Основним недоліком цього методу є час, який потрібно витратити на одну атаку.

Спочатку віртуальні машини були підготовлені з необхідними програмами за допомогою інсталяційних скриптів із цього проекту. Брандмауер і антивірус були вимкнені, оскільки більшість корисних навантажень від Metasploit виявляються досить швидко за замовчуванням, і намагатися уникнути виявлення заборонено. Потім було зроблено знімок, щоб можна було швидко повернутися до вихідної точки. Потім вибраний експлойт завантажувався в Metasploit із налаштуваннями за замовчуванням, за винятком IP-адреси цільової системи. Результат атаки вказує на наступні дії, якщо атака вдалася, тест закінчено. Якщо атака не вдасться, проводяться додаткові дослідження, щоб з’ясувати, що пішло не так, що може статися з різних причин. Інструмент аналізу протоколів Wireshar k 6 використовувався під час розслідування, зокрема, оскільки він може дати уявлення про мережевий трафік між зловмисником і ціллю. Це можна використати, щоб визначити, чи атака призвела до повернення будь-яких повідомлень про помилку на атакуючий хост, чи не було програми, яка прослуховувала певний порт. Якщо причину збою вважається можливою вирішити, тоді вносяться додаткові коригування, щоб зробити експлойт успішним. Одним із прикладів такого коригування може бути зміна корисного навантаження, що використовується в Metasploit, оскільки воно може відрізнятися залежно від стабільність і може

бути спрямований на різні платформи або вимагати встановлення певного інтерпретатора мови сценаріїв. Іншим прикладом є перегляд опису експлойту на наявність можливих вимог , які необхідно виконати, перш ніж експлойт зможе працювати, наприклад створення каталогу в певному місці, зміна файлу конфігурації або запуск певного процесу. Оскільки опис містить звичайний текст, важко автоматично зафіксувати вимоги та визначити, як правильно налаштувати програму. Як згадувалося, пакунки Chocolatey зазвичай стосуються лише встановлення програм, а не їх початкової конфігурації. Це може призвести до збою експлойтів проти вразливих програм через те, що служба бази даних не запущена або налаштована на прийом лише локальних з'єднань. Це не можна вважати помилкою з боку експлойта, оскільки розгортання сервера бази даних у реальному світі передбачало б, щоб служба фактично була запущена та, ймовірно, також приймала віддалені з'єднання з іншого сервера. Щоб визначити, чи є невдалий експлойт результатом ненадійного/зламаного експлойту чи просто тому, що програмне забезпечення було в неправильному стані, експлойти, протестовані, також перевіряються на параметри конфігурації та інші параметри, які перешкоджають програмі перебувати в вразливий стан. Мета цього полягає в тому, щоб визначити, чи експлойт вимагає певної конфігурації, яка може використовуватися лише в кількох випадках використання, чи програма взагалі не була налаштована після встановлення через пакет Chocolatey. Більш прості дії конфігурації, такі як запуск служби або додавання деяких фіктивних даних до бази даних, можна було б автоматизувати як частину автоматизованого засобу встановлення програмного забезпечення.

2.9. Автоматизація встановлення програмного забезпечення та оцінка надійності онлайн-сховища

Процес автоматизації використовує Ansible і Chocolatey разом із спеціальними скриптами Python. Ansible використовується для автоматизації

процесу, а Chocolatey — це менеджер пакетів. Інструмент може працювати у двох різних режимах: один із використанням загальнодоступного сховища Chocolatey (для якого потрібне з'єднання з Інтернетом), а другий – з використанням внутрішнього сховища. Приклад результату, що відображається користувачеві, можна побачити на рисунку 2.3

Рис. 2.3. Вихідні дані процесу встановлення програмного забезпечення

Встановлення програмного забезпечення за допомогою Ansible і Chocolatey з онлайн-доступом спрацювало добре. За допомогою сценаріїв Python список із необхідним програмним забезпеченням було перетворено на збірник ігор Ansible. Для Chocolatey назва та версія програмного забезпечення виступали унікальним ідентифікатором, щоб знайти його в репозиторій, який є єдиною необхідною інформацією для встановлення частини програмного забезпечення. Те саме стосується внутрішнього сховища.

Багато пакунків, особливо старих, так чи інакше були зламані. Найпоширенішими проблемами були помилки HTTP 404, через які відсутні файли програми. Найімовірнішою причиною цього є те, що постачальник програмного забезпечення вирішив видалити старіші версії свого програмного забезпечення, або що файли було переміщено на інший сервер/розташування, а пакет Chocolatey не було оновлено з цією зміною. Ще одна поширена помилка полягала в тому, що залежні пакети не вдалося встановити через те, що їх замінив інший пакет, або через вищезгадані проблеми. можна було успішно встановити 811 із 1354 перевірених пакетів (59,9%), які вважалися ймовірно вразливими за допомогою процесу. Таблиця 2.10 представляє

розбивку результатів встановлення з використанням категорій, визначених у попередніх розділах.

Таблиця 2.10.

Статистика результатів встановлення

Результат	% пакетів
в порядку	59.9
інша_помилка	22.1
не знайдено	8.4
невідповідність	5.5
можливо_зламаний	3.3
installer_hangs	0,7
not_in_repo	0,1

Під час розробки засобу автоматизованої інсталяції багато пакетів було встановлено як частину процесу тестування. Результати цих інсталяцій було збережено в базі даних, щоб відстежувати, які пакунки вже перевірено. Додатковий метадані, завантажені для кожного пакета, включали поле для дати створення пакета, яке можна об’єднати з результатами встановлення, щоб порівняти їх для пакетів різного віку. На рисунку 2.4 показана відносна частота кожного статусу інсталяції за шість років.

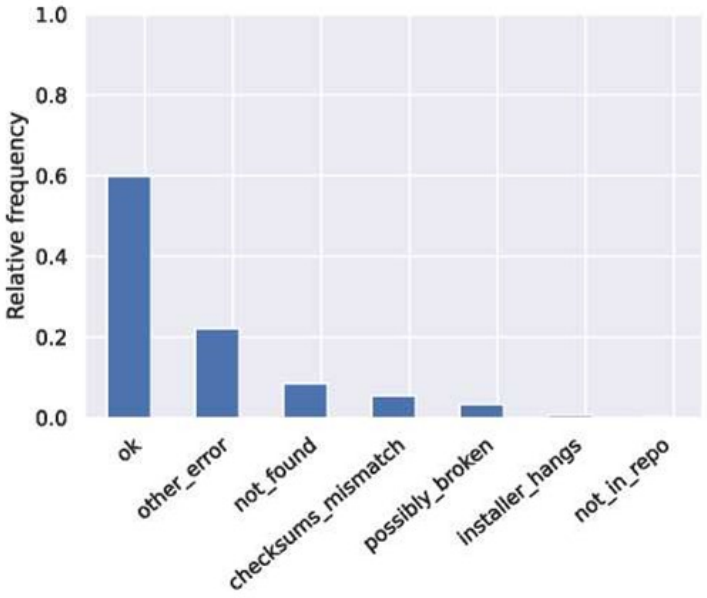

Рис. 2.4. Статистика результатів встановлення

Графіки на рисунку 2.4 показують, що надійність пакунків з онлайн-репозиторію Chocolatey у найкращому разі помірна. Навіть пакунки, створені протягом останніх чотирьох місяців, можна успішно встановити лише приблизно в 65% випадків. Ця цифра значно змінюється між місяцями, але спостерігається тенденція до зниження, якщо подивитися на попередні роки на рисунку 4.3b. Це показує, що пакети з часом ламаються, що може спричинити проблеми, коли для дослідження безпеки потрібні старіші версії програми. Рівень помилок not_found, які вказують на те, що постачальник видалив один або кілька файлів програми зі своїх серверів, особливо зростає для старіших пакетів. Ці помилки зустрічаються менше ніж у 5% пакетів, новіших ніж чотири місяці, тоді як показник стабільно зростає до 20% для пакетів приблизно 5-6 років.

Під час розробки засобу автоматизованої інсталяції багато пакетів було встановлено як частину процесу тестування. Результати цих інсталяцій було збережено в базі даних, щоб відстежувати, які пакунки вже перевірено. Додаткові метадані, завантажені для кожного пакета включали поле для дати створення пакета, яке можна об’єднати з результатами встановлення, щоб порівняти їх для пакетів різного віку. На рисунку 2.5 показана відносна частота кожного статусу інсталяції за шість років.

Графіки на рисунку 2.5 показують, що надійність пакунків з онлайн-репозиторію Chocolatey у найкращому разі помірна. Навіть пакунки, створені протягом останніх чотирьох місяців, можна успішно встановити лише приблизно в 65% випадків. Ця цифра значно змінюється між місяцями, але спостерігається тенденція до зниження, якщо подивитися на попередні роки на рисунку 2.5 b. Це показує, що пакети з часом ламаються, що може спричинити проблеми, коли для дослідження безпеки потрібні старіші версії програми. Рівень помилок not_found, які вказують на те, що постачальник видалив один або кілька файлів програми зі своїх серверів, особливо зростає для старіших пакетів. Ці помилки зустрічаються менше ніж у 5% пакетів, новіших ніж

чотири місяці, тоді як показник стабільно зростає до 20% для пакетів приблизно 5-6 років.

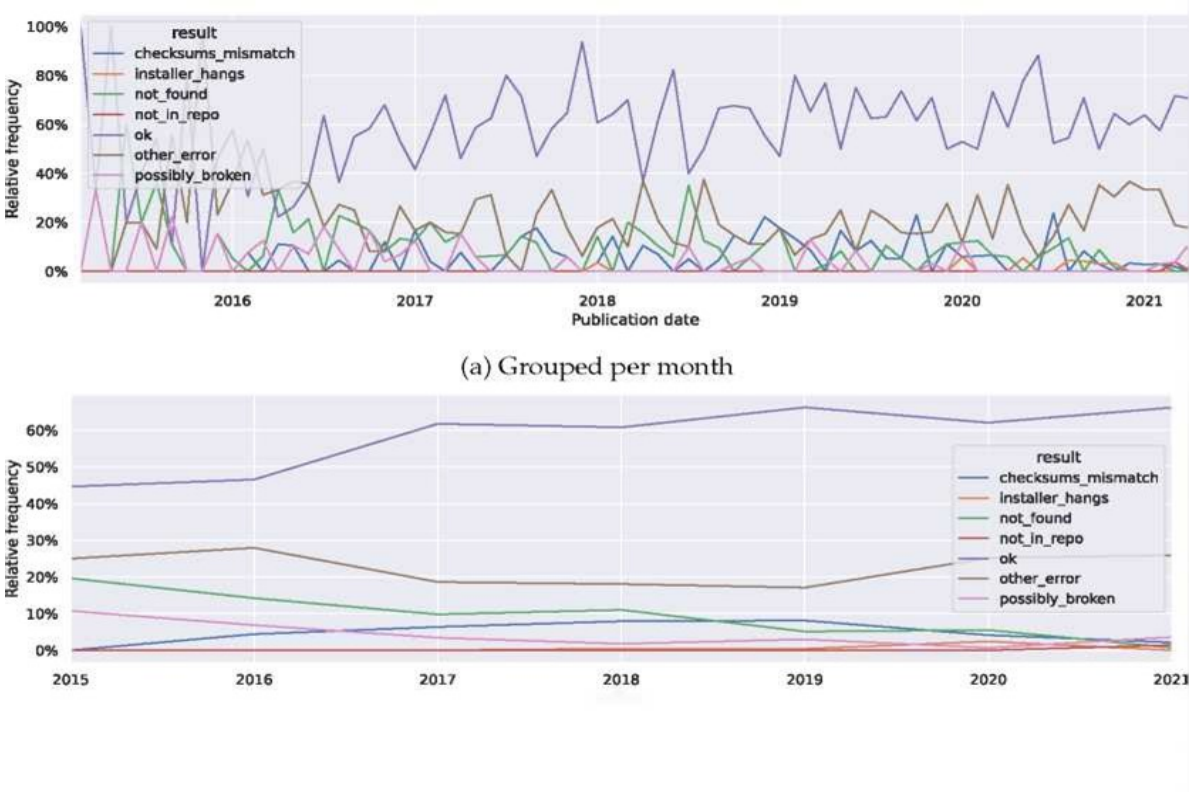

Рис. 2.5. Результати встановлення з часом

Існуюче внутрішнє сховище було заповнено пакетами Chocolatey із загальнодоступного сховища Chocolatey за допомогою спеціальних сценаріїв Python. Сценарії завантажували та змінювали пакети, які потім завантажувалися у внутрішній репозиторій. Змінивши адресу репозиторію, який Chocolatey використовує для пошуку пакетів, внутрішній репозиторій можна використовувати повністю без доступу до Інтернету, і завдяки цьому його вміст з часом буде більш надійним на відміну від публічного. Однак деякі пакунки було важче перепакувати, ніж інші. Наприклад, деякі пакунки містили лише завантажувач, який може вибірково завантажувати лише потрібні компоненти набору програмного забезпечення або надавати користувачеві зручний інтерфейс під час завантаження замість повної програми. Їх не можна перепакувати автоматично, як і файли, які будуть

завантажені визначається статично без запуску інсталятора. Інші пакунки мали URL-адреси, які обчислювалися під час виконання, тому їх не можна було статично переписати. Деякі навіть приховали свої URL-адреси. Незважаючи на те, що деякі з них можна переписати вручну, це буде трудомістким процесом, якщо потрібно змінити велику кількість пакетів, і, отже, це не життєздатний вибір, якщо немає конкретного інтересу до певного пакета.

Приблизно 85% пакунків, які можна успішно інсталювати за допомогою підключення до Інтернету, також працюють, якщо їх перепакувати за допомогою напівавтоматичного процесу, (тобто коли користувач відповідає на запитання про заміну змінних і відсутні файли, але не виконує жодних ручних редагувань файлів). Оскільки процес інтерналізації займає додатковий час на додаток до часу, необхідного для встановлення кінцевого пакета, лише підмножина робочих пакетів була інтерналізована та протестована. Загалом наразі це 378 пакетів, які можна успішно встановити в CRATE. Їх також можна буде встановлювати в майбутньому, оскільки внутрішнє сховище розміщено локально. У середньому для інтерналізації одного пакета потрібно приблизно 24 секунди, при цьому домінує 5-секундна затримка, яка вводиться після кожного запиту HTTP, щоб зменшити навантаження на сервери, на яких розміщені пакети Chocolatey і файли, на які в них посилаються.

Після отримання списку всіх пакетів Chocolatey та їх версій, , 121 379 унікальних записів було передано до сценарію зіставлення CPE з експлойтом, наданого FOI. Цей сценарій було налаштовано лише на повернення збігів із рівнем достовірності два або вище за шкалою від 0 до 4. Після обробки цих записів було повернуто 2 053 546 кандидатів на CPE та можливі експлойти. Їх було відфільтровано на основі подібності між назвою, постачальником і версією, у результаті чого залишилося 4641 запис. Вони були розподілені між чотирма віртуальними машинами під час підготовки до автоматизованого тестування. Через проблеми, коли API VirtualBox міг бути небезпечним для потоків, на кожній фізичній машині запускалася лише одна віртуальна

машина, що обмежувало кількість віртуальних машин, які можна було використовувати для тестування. Оскільки кілька версій однієї програми не можна встановити одночасно на одній машині, це також обмежує кількість версій, які можна перевірити для одного пакета, до чотирьох, по одній на VM. З цими обмеженнями лише 98 пакетів можна було встановити на віртуальних машинах для загалом 436 атак із використанням 137 унікальних експлойтів.

2.10. Автоматичне тестування експлойтів

Автоматичні тести, які виконувала SVED, проводилися чотири рази, кожен раз займав у середньому сім годин. Вони були розпочаті в другій половині дня та оцінені наступного ранку. Результати атак було візуалізовано за допомогою SVIZ, який давав огляд усіх атак із можливістю подальшого дослідження журналів. Дивіться рис. 2.6 для часткового перегляду послідовності атаки. Багато спроб атак зазнають невдачі, як показано червоними трикутниками, але у верхньому правому куті та ліворуч можна побачити кілька зелених трикутників, які вказують на успішні атаки.

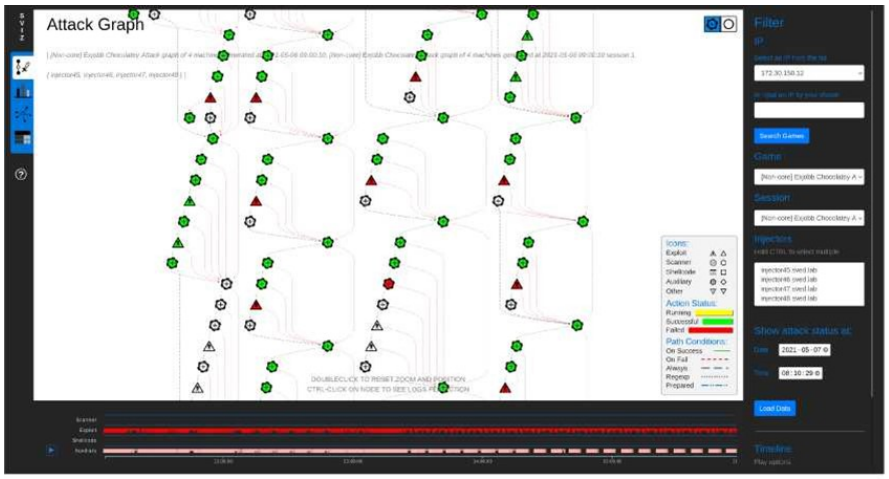

Рис. 2.6. SVIZ, що показує частину послідовності атаки

З першого і другого заходу всі атаки зазнали невдачі. Оскільки більшість атак були засновані на Flash на систему без Flash, це було очікувано. Ймовірно, ці атаки були запропоновані, оскільки в їхніх описах згадується

веб-браузер, для якого було розроблено експлойт, що призводить до того, що експлойти пропонуються в процесі пропозиції експлойтів, навіть якщо встановлено лише веб -браузер, а Flash не встановлено. Хоча деякі атаки, які мали потенціал успіху, також зазнали невдачі**.**

Таблиця 2.11.

Підсумок результатів експлойтів

Категорія	Успішні подвиги	Невдалі експлойти	Всього
формат	4	24	**28**
сервер	0	251	**251**
браузер	0	209	**209**

Після дослідження було виявлено, що причиною деяких збоїв були відсутні тригери (тобто експлойт потребував додаткової взаємодії, яка не була реалізована в SVED), що означало, що експлойти не запускалися на комп'ютері-жертві. Також виникали інші помилки, такі як помилки під час створення атаки, які пізніше були вирішені.

Коли ці проблеми були вирішені, було проведено два нових тести. Вони дали кращі результати, якщо чотири атаки були успішними. Підсумок кількості запущених експлойтів, згрупованих у експлойти на основі вразливостей аналізу формату файлу, експлойтів, націлених на серверну програму, і експлойтів, націлених на браузери, можна побачити в таблиці 2.11.

Висновки до розділу

В даному розділі виконана реалізація автоматизованих методів інсталяції програмного забезпечення та проведено вибір менеджерів пакунків та засобів автоматизації. Здійснено опис процесу вибору інструментів та контроль списку пакунків, проведені онлайн-тести для встановлення. Досліджено критерії відбору експлойтів, їх автоматичне тестування та проведена автоматизація встановлення програмного забезпечення та оцінка надійності онлайн-сховища.

РОЗДІЛ 3.
АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ ПРОЦЕСІВ ІНСТАЛЯЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗГІДНО ВИМОГ КОНЦЕПЦІЙ КІБЕРБЕЗПЕКИ

3.1. Представлення процесу першого використання внутрішнього сховища

Результати інсталяції пакетів різного віку, показані на рисунку 3.1, підкреслюють важливість наявності внутрішнього сховища, на яке не поширюються правила розповсюдження і, отже, може містити копії інсталяторів та інші бінарні файли. Заповнити внутрішнє сховище було непросто. Перепакування упаковок Chocolatey призвело до багатьох проблем, таких як обробка різних стандартів і уподобань стилю кодування між упаковками, що ускладнило автоматизацію процесу. Хоча ми могли успішно автоматизувати процес для більшості пакетів, деякі з тих, які вийшли з ладу, все одно потрібно буде обробляти вручну, якщо в них виникне потреба. Однак було вирішено, що це не стосується цього проекту, але може зацікавити певні частини програмного забезпечення, які вважаються доречними для використання в кібернетичному діапазоні.

Однією з найпоширеніших практик розробників пакунків, яка запобігала автоматичному перепакуванню, було використання змінних в URL-адресах. Лістинг 3.1 містить приклад цього, коли змінна з іменем locale вбудована в URL. Ця заміна змінної може відбуватися кількома різними способами, деякі з яких можуть оброблятися статично, а деякі вимагають більш розширеного аналізу та імітації частин інтерпретатора PowerShell. Шоколадні упаковки, написані багатьма різними авторами, здається, використовують багато з цих методів залежно від особистих уподобань людини, яка створює кожну упаковку. Таким чином, створений переписувач пакунків не може обробити деякі пакунки через те, що він не може

ідентифікувати URL-адреси, які містять складніші заміни змінних. Проте, лише 15% пакетів не вдається інтерналізувати. Це показує, що більшість пакунків або не використовують заміну змінних у своїх URL-адресах, або використовують простіший метод заміни, як-от той, що
наведено в Лістингу 3.1 , який часто можна вирішити статично.

```
$locale = 'en-US'
$locale = GetLocale -localeFile "LanguageChecksums.csv" -product $softwareName
$Url = " https://download.mozilla.org/?product=firefox-
```

Лістинг 3.1. URL із вбудованою змінною

Однією з перешкод, яка не була детально розглянута під час цієї роботи, було те, як впоратися з першим використанням програми. Щоб інструмент був більш корисним, необхідна мінімальна взаємодія з користувачем, щоб експлойти, запущені проти програм, не виходили з ладу через несподівані діалогові вікна та спливаючі вікна. Однак багато програм, які були інтерналізовані та можуть бути встановлені автоматично, все ще вимагають взаємодії з користувачем під час першого використання. Ці типи взаємодії значно відрізняються між програмами, і їх нелегко автоматизувати, оскільки їх потрібно обробляти окремо.

Процес можна автоматизувати для окремих програм за допомогою інструментів сценаріїв, таких як AutoHotKey 1 і AutoIt 2 для створення сценаріїв, які клацають у діалогових вікнах відповідно до попередньо визначеної послідовності введення. Іншим способом може бути запис будь-яких змін у реєстрі та застосування цих змін після конкретної інсталяції, у результаті чого програма повинна поводитися так, ніби її було налаштовано за допомогою графічного інтерфейсу. Обидва ці рішення вимагають індивідуальної обробки, яка забирає час і погано масштабується для сотень програм. Ці рішення подібні до альтернатив автоматичного встановлення, оскільки як встановлення програми, так і керування її станом використовують графічні інтерфейси та включають зміни в реєстрі та файловій системі.

Використовуючи підручники Ansible, було зроблено деякі спроби підготувати програму для читання PDF-формату (Portable Document Format) Adobe Acrobat Reader таким чином, щоб вона працювала так, ніби користувач раніше з нею взаємодіяв. Acrobat Reader має приблизно 10-15 модулів Metasploit, націлених на нього, але більшість із них націлені на Windows XP і, ймовірно, не працюватимуть у Windows 10. Однак, все одно може бути доцільно підготувати його до автоматичного встановлення, якщо будуть виявлені нові атаки або історично він був звичайною мішенню для атак на формати файлів. Після першого запуску Acrobat Reader відображає послідовність Ліцензійної угоди, пропозицію зробити його програмою за замовчуванням для PDF-файлів і ознайомлювальний огляд. Ці елементи, особливо ліцензійна угода, можуть заважати експлойтам, якщо деякі частини процесу відтворення PDF не запускаються, доки діалогове вікно не буде закрито. Оскільки відкритий PDF-файл не відображається, доки принаймні не буде прийнято EULA, можна припустити, що це може мати певний вплив на експлойт, який залежить від того, що програма перебуває в певному стані.

Перевіривши реєстр Windows за допомогою такого інструменту, як RegistryChangesVie w 3 , було виявлено, що близько 800 записів було створено або змінено після простого закриття всіх діалогових вікон першого запуску. Перевірка вручну призвела до появи кількох кандидатів, які, здавалося, були відповідальними за придушення діалогових вікон під час майбутніх запусків програм, але огляд із початку роботи все одно запустився. Це показує, що можна емулювати програму, що використовується, відразу після інсталяції, але через таку кількість змін реєстру існує ризик того, що програма стане нестабільною, якщо після інсталяції буде встановлено лише частину з них. Оскільки вони також можуть містити записи, пов’язані з машиною, просте включення всіх із них також може призвести до проблем під час повторної інсталяції на іншій машині. Великий загальний обсяг записів також означає, що для розрізнення записів, специфічних для машини, неважливих записів і

записів, які контролюють поведінку першого використання, потрібна значна кількість ручної роботи.

3.2. Тестування експлойтів

З лише чотирма успішними атаками результати тесту були трохи невтішними, хоча й очікуваними. Переглядаючи запропонований експлойт для програм, багато з них були погано підібрані або просто зовсім неправильні. Наприклад, експлойти для Firefox, які також покладаються на Adobe Flash Player, не працюватимуть, якщо Flash не встановлено. Деякі експлойти покладалися на вразливості у веб-сервісах, де серверна система мала вразливості, вимагаючи також налаштування сервера з комбінацією певних внутрішніх систем. Наприклад, експлойт, націлений на PHP, також може вимагати веб-сайту, який використовує певну частину PHP, яка містить уразливість. Вважалося, що це не входить у рамки або не варто витрачати час на цю дисертацію.

Ці результати подібні до тих, де лише невелика кількість експлойтів була успішною проти великого набору віртуальних машин. Це дослідження відрізняється від їхнього тим, що експлойти на основі формату файлів і веб-переглядача перевіряються на додаток до експлойтів, націлених на серверні програми, запущені на цільовій машині, але досягає подібного висновку в тому, що дуже небагато експлойтів, які пропонуються автоматично, виявляються успішними.

Під час спроби знайти програми вручну виявилося важко знайти програми, які існували в репозиторії Chocolatey як дійсний пакет і мали надійний модуль Metasploit для цієї версії. Є лише чотири унікальні програми, які можна було успішно встановити та використовувати. На це є різні причини, одна з яких пов’язана зі способом розповсюдження пакунків Chocolatey, який з часом призводить до поломки пакунків, що ускладнює отримання потрібної версії, необхідної для роботи експлойта. Інша причина

полягає в тому, що в той час як у Metasploit є багато модулів експлойтів, готових до використання, якщо фільтрувати їх для роботи з відносно новими версіями Windows (тобто не націленими на Windows Vista або XP), а також мати програму, доступну як пакет від Chocolatey, залишилося не так багато програм. Але це не обов'язково означає, що в репозиторії Chocolatey немає інших уразливих програм. Як згадувалося в у цій дипломній роботі ми обмежуємося розглядом лише експлойтів Metasploit і Windows 10 як операційної системи.

Цільова вразливість у медіапрогравачі VLC була присутня у версії, яку було встановлено в тестовому середовищі, але експлойт не вдалося, оскільки Metasploit не мав цільового визначення для встановленої версії (тобто експлойт було розроблено для іншої версії VLC). Оскільки вразливість пов'язана з пошкодженням пам'яті, експлойт може містити посилання на абсолютні адреси певних символів у двійкових файлах програми. Коли код додається або видаляється між версіями, це може призвести до зміни цих адрес і помилки експлойту. Додати нове цільове визначення можна було б здійснити шляхом порівняння бінарних файлів між версіями та пошуку нового зсуву для кожного відповідного символу, але це виходить за межі цієї тези. Встановивши натомість VLC 2.2.8 (для якого було створено модуль Metasploit), експлойт вдалося.

Експлойт, націлений на Elasticsearch, намагався використати вбудовану функцію пошуку Elasticsearch для запуску довільного коду. Спочатку це не вдалося, незважаючи на запуск служби Elasticsearch (після перезавантаження). Під час деяких експериментів було виявлено, що додавання деяких довільних даних до сховища даних спричинило роботу експлойта.

Прочитавши опис експлойту Foxit Reader, стало зрозуміло, що експлойт мав бути доставлений через спільний мережевий ресурс Server Message Block (SMB). Згенерований PDF-файл сам по собі не міг містити шкідливого коду, але міг посилатися на файл у спільному мережевому ресурсі, який запускався щоразу, коли користувач відкривав документ. Використання

команди Metasploit msfvenom для створення програми, яка б встановлювала сеанс командного рядка назад до атакуючої машини, і розміщення цієї програми в спільному мережевому ресурсі дозволили експлойту працювати надійно.

Експлойт FreeSWITCH не працював за замовчуванням після встановлення програми. Було визначено, що це спричинено тим, що служба не запускається за замовчуванням і через те, що вона прослуховує лише локальний хост. Це зробило його недоступним через мережу, але експлойт все одно можна використовувати підвищити привілеї, якщо зловмисник раніше встановив менш привілейований сеанс до цільової машини. Щоб спростити перевірку експлойту, службу було налаштовано таким чином, щоб до неї можна було отримати доступ з будь-якої машини в локальній мережі. Після цього його можна буде успішно експлуатувати.

Після завершення перших двох автоматичних тестів було зроблено висновок, що ручне тестування може допомогти точно визначити проблему, чому деякі, здавалося б, правильні атаки зазнають невдачі. Тестування вручну проводилося, на локальних робочих станціях за допомогою VirtualBox і віртуального середовища в CRATE. Крім того, було проведено ручні тести щодо деяких додаткових програм, які не були визначені автоматично, але були доступні в Chocolatey і мали відповідні експлойти проти них. Список програм, які не вдалися, незважаючи на те, що вони здавалися вразливими, разом із спробою експлойту, встановленою версією та цільовою версією можна побачити в таблиці 3.1.

Таблиця 3.1.

Перевірені вручну експлойти

програма	Експлойт	Встановлена версія		Цільова версія
VLC	exploit/windows/fileformat/vlc_mkv	2.2.5	1.4.1	^ 2.2.8 < 1.4.3
elasticsearch-сервіс	exploit/multi/elasticsearch/search_groovy_script		9.0.1.1049	9.0.1.1049
foxitreader	exploit/windows/fileformat/foxit_reader_uaf	1.8.5		Будь-який
вільний перемикач	exploit/multi/misc/freeswitch_event_socket_cmd_exec			

Визначена причина, чому кожен тестований вручну експлойт не спрацював, і дії, необхідні для їх успішного виконання, наведені в таблиці 3.2.

Таблиця 3.2.

Перевірені вручну експлойти, причина невдачі та необхідні дії

Експлойт	Причина невдачі	Необхідна дія
exploit/windows/fileformat/vlc_mkv	Metasploit не має цілі для цієї версії.	Встановіть VLC версії 2.2.8
exploit/multi/elasticsearch/search_groovy_script	Необхідні функції не запускаються без даних у базі даних. Експлойт потрібно надавати через мережевий ресурс.	Додайте випадкові дані до бази даних (можна віддалено зловмисником)
exploit/windows/fileformat/foxit_reader_uaf	Служба прослуховує лише локальний хост	Вручну створіть спільний мережевий ресурс і помістіть туди шкідливий файл .exe. Налаштуйте прослуховування всіх мережевих інтерфейсів
exploit/multi/misc/freeswitch_event_socket_cmd_exec		

Посібники Ansible були створені, щоб перевести Elasticsearch і FreeSWITCH у вразливий стан, оскільки в цих двох програмах були виявлені помилки через відсутність змін конфігурації на стороні програми . Ці посібники можна вказати під час встановлення пакетів, що означає, що ці програми можуть стати вразливими одразу після завершення встановлення. У лістингах 3.2 і 3.3 показано ці два посібники, де посилання на acl.conf.xml і event_socket.conf.xml у лістингу 3.3 є локальними копіями конфігураційних файлів, які були змінені , щоб дозволити вхідні з’єднання з локальної мережі (LAN).

```
- ім'я :
  перезавантаж
  ити
  win_reboot :

- name : Зачекайте, поки служба
  запуститься win_wait_for :
    порт : 9200

- name : створити початковий
  фіктивний запис win_uri :
    url : http://localhost:9200/type/type1/id1
    метод : POST
    body :     '{ "some": "json" }'
```

Лістинг 3.2. Посібник для переведення Elasticsearch у вразливий стан

```
- name : Дозволити вхідні з'єднання сокетів подій з будь-якого
  місця win copy :
    src :      '{{item}}'
    dest : 'C:\Program Files\FreeSWITCH\conf\autoload_configs\'
  цикл :
    - acl.conf.xml
    - event_socket.conf.xml

- name : увімкнути службу FreeSWITCH
  win_service :
    назва : FreeSWITCH
    start_mode : авто
    стан : розпочато
```

Лістинг 3.3. Посібник для переведення FreeSWITCH у вразливий стан

3.3. Альтернативні методи виконання автоматизованого встановлення програмного забезпечення

Використання методу автоматизованої інсталяції програмного забезпечення, описаного в цій дипломній роботі, підходило для цього сценарію та вимог до CRATE. Однак існує багато інших способів автоматизувати процес налаштування віртуальної машини та інсталяції програмного забезпечення. Дослідження їх більш детально могло б бути корисним, щоб побачити, чи могли б вони бути хорошою альтернативою поточному процесу розгортання в CRATE, але вони вимагали б значної додаткової роботи, якщо поточний процес більше не можна було б використовувати. Це не входить у рамки цієї дисертації, але це міг бути варіант для тестування в меншому масштабі для порівняння з поточним рішенням.

Як згадувалося раніше, автоматизований процес тестування експлойтів спочатку не призвів до успішних атак на цільові програми. Це сталося через те, що багато експлойтів не відповідали набору встановлених програм. Витрачати більше часу на тестування більш перспективних експлойтів вручну або створення додаткових ігор для моделювання першого використання програми, можливо, було б кращою сферою, на якій варто зосередитися, замість того, щоб намагатися працювати над вирішенням проблем, що виникли. навколо процесу автоматизованого тестування. Однак можливість

виконувати автоматичне тестування за допомогою SVED програм, встановлених у віртуальних машинах, також слугує для підтвердження того, що вони перебувають у вразливому стані, що може бути корисним для забезпечення належної роботи середовищ, створених для вправи.

Джерела, використані в цій дисертації, дещо різняться за своїм походженням, причому інформація про використовувані інструменти в основному отримана з документації, написаної розробниками, або з самого вихідного коду. Для проектів програмного забезпечення це часто є найновішим і точним джерелом інформації про мету та функціональність проекту. Оскільки ми мали доступ до кібердіапазону CRATE FOI і могли запитувати людей, які беруть участь у його створенні та обслуговуванні, більшість інформації щодо CRATE можна було отримати безпосередньо. Оскільки CRATE використовується для досліджень, існує кілька наукових статей, які стосуються як тестування експлойтів, так і роботи щодо інших кібердіапазонів також використовувалися, щоб дати ширшу перспективу та знайти альтернативні рішення.

Під час проекту, коли ми намагалися встановити якомога більше програм, через деякий час ми помітили, що всі програми не вдалося встановити, і видали те саме повідомлення про помилку. Після деякого було встановлено, що це через вбудоване обмеження швидкості від Chocolatey, яке пояснюється в попередньому розділі. Але, це був лише годинний ліміт, але щоб цього більше не повторилося, ми вжили деяких запобіжних заходів. Це включало додавання невеликої затримки між завантаженням пакетів і виконання лише однієї онлайн-інсталяції на комп’ютері. З внутрішнім сховищем це не проблема.

3.4. Представлення контролю виходів з Ansible та URL-адрес

Через кілька місяців проекту ми помітили, що з базою даних щось не так. Були програми з дивними номерами версій порівняно з іншими версіями

тієї ж програми, програми з дивними описами та назвами програм, яких не було в репозиторії Chocolatey. Причиною цього було те, що деякі значення в різних стовпцях бази даних були зміщені на один рядок, в результаті чого, наприклад, програма N отримала номер версії програми N — 1. Оскільки це була незначна різниця в наборі даних із приблизно 15 000 записів, помилку було легко пропустити, і знадобився деякий час, поки її виявили. Те, що привернуло нашу увагу, полягало в тому, що деякі програми, які можна було успішно встановити вручну, в іншому екземплярі опинилися як «not_in_repo». На щастя, вихідні дані, отримані з API Chocolatey, не вплинули, що означало, що можна було створити нову таблицю та відтворити дані наших тестів із доступними виправленими даними пакета.

Під час розробки інструменту виникла дивна проблема. Журнали з Ansible за замовчуванням надсилаються на термінал одне завдання за раз, з можливістю писати детальні журнали з Chocolatey безпосередньо на термінал. Детальні журнали необхідні для визначення того, що пішло не так у разі помилки, але не потрібні в інших випадках. Однак, оскільки версії програмного забезпечення, які встановлюються, у деяких випадках мають кілька років тому, існує відносно висока ймовірність того, що інсталяція може завершитися помилкою. Журнали Chocolatey і Ansible надходять зі стандартного виводу (стандартний вихід) і stderr (стандартна помилка) і записуються безпосередньо в термінал. Використовуючи прапорець в інтерфейсі Python ansible-runner, який використовується для взаємодії з Ansible, журнал також можна записати у файл JSON. Однак існує проблема з ansible-runner у версії, яка використовується під час цієї роботи де файл JSON може бути частково пошкоджений, якщо вихідний рядок містить понад 2000 символів. Зразок такого пошкодженого коду JSON можна побачити в лістингу 3.5. Це викликало помилку під час аналізу файлу журналу, оскільки вміст більше не був дійсним даними JSON.

```
{
    "хост" : "192.168.56.106" ,
    "uuid" : "d8d8c7f5-2e57-4771-abac-48e8155201b7"
}

пп [[
    1 ; 35 м [УВАГА
    ]: Chocolatey був відсутній у цій системі, тому його було встановлено
    протяромлл[[ 0 м
лл [[
            1 ; 35 мще завдання лип.л*[[
                0 м

{
    "uuid" : "874328ec-fa17-4a0b-b105-384ed6ba2705" ,
    "лічильник" : 13 ,
    "stdout" : "\u001b[0;33mchanged:  [192.168.56.106]\u001b[0m" ,
    "start line" : 12 ,
    "кінцевий рядок" : 13 ,
    "runner ident" : "5b19763f-5169-4651-b89a-4a9a24d17397" ,
    "event" : "runner on ok" ,
    "pid" : 15214 ,
    "створено" : "2021-02-24T08:57:29.352492" ,
    "parent uuid" : "c5247a8a-752e-a6ac-0477-00000000000a"
}
```

Лістинг 3.5. Частина файлу JSON, яка була пошкоджена

Проблема не була унікальною для цього проекту, оскільки її було вирішено у гілці розробників ansible-runner. Однак зміни ще не застосовано до випускної версії Ansible, але, сподіваюся, це станеться незабаром. Тим часом було створено тимчасовий патч на основі очікуваного рішення, щоб вручну вирішити цю проблему.

Під час роботи з інтерналізацією пакетів Chocolatey одна з проблем полягала в тому, що URL-адреси не дотримувалися стандартного формату в пакетах. Найпростішим варіантом були ті, які мали одну статичну URL-адресу для 32-бітної версії програми та одну статичну URL-адресу для 64-бітної версії. Але було також кілька «розумніших» рішень, де URL-адреси змінювалися залежно від поточної версії програми або навіть обчислювалися під час виконання на основі операційної системи, архітектури чи мови відображення, як повідомляє операційна система. Був навіть один пакет, який намагався заплутати URL-адресу, закодувавши її (див. лістинг 3.6), що призвело до труднощів із статичним читанням. Однак це можна обійти, переглянувши HTTP-запит, зроблений під час виконання (який відображається у файлах журналу), а потім замінивши обфусцовану URL-адресу вручну перед збереженням пакета в репозиторії.

```
$UrlBase = "76492d1116743f0423413b16050a5345MgB8AEQAMwBxAHUAQwBsAHUA
WgBuAHkAbABjAG0AWgBYADMAagAwADMANgBxAFEAPQA9AHwAOABiADQANABmADAAZQA4
ADgAMgBiADQAMwBhADQAYgA5ADEAMABkADcAZAAwADcANAA2ADIANAAzADgAMwA0AGUA
YQBhADEAMQBhADQAZQBjAGUAZgAxADUAZQBkADIANQB1ADAANgA0ADkAOABmAGIAYQB1
AGMAZQAxADcANQBiAGEAOAA3AGIANAA3ADgAZgA0ADAAZQA4AGIAOQAxADcAZQAwADcA
NQA1AGYAYgBhAGMAOAA4ADkANgA1ADEANAAxADMANQA5AGEAYgBmADQAZgBhAGQANwBk
ADQANQA2ADMANAAyADcANABiADgANwA4ADEAYQA4ADcAYwA2AGYAZAA0AGEANgBkADUA
MwA4ADkAMQA3ADIANQAyADUAZgA2AGUAOAA4ADEANAAyADMAMgBmADEANgBjADIAYwAx
ADIAOAAxAGQAOQA4AGIAZABhADUANQA3ADIAYgA3ADIANAAwADgAMwA4ADIANQAzAGUA
YwAxADMAZQBkAGQAMgBhAGEANABiAGEAMwBhADIAOAAwADMAMABjAGEAYwA0ADcAZQAz
ADMAZQBiADIAOQBiADgAOQAwAGIAZQAxAA=="
$UrlBase64 = "76492d1116743f0423413b16050a5345MgB8AFEANwBwAHYAYwBNAF
EAdQBpAEUARwBVAEYAYwBNADMAMwAyAHYAVABSAEEAPQA9AHwAMwA0ADcANQA5ADAAMA
BjAGUAYQAzADEANQA0ADkAMwBmAGIAYwAxADQANQAwAGQAMQBmAGQAMQA3ADYANwA2AG
IAOAAyADUAZgAzADkAZAAxADQAZgA3AGYAYwA1ADMANgA5ADIAMAA1ADAAMwAwADQAMg
B1AGYAOABhADYAYQBjAGIANQAwADMANAAwAGEANQB1ADMAMQBiAGYAYwA3ADcAZAAzAD
YAMAA4ADIAOQA0AGUAYgBhADAAZQAwAGUAOABkAGEAOABhADUANAB1AGEAMwB1AGIAZA
B1ADEAZQA5ADQAYQBmADUANwBhADYANAAxAGEAMAA2ADgAMQA5ADYAMgA0ADUAYgBmAG
EAMwAyADYAZgBkAGIAZABmAGYANwA4AGYANQAwADMAZQAxADcANwAxAGMAMwBkAGQAZg
BhADUANgAzAGYAMwAxADYAOQA3ADQAZQA0ADcAZQA0ADEAMgBkAGUAYQA4ADUAZgA3AD
QAOAAwADIAZQB1AGYAMQA4AGYANwBkAGQAOABiADkAZgA1ADIAZAA2ADYAYwAzADMAZQ
A0ADMANQBjADMANgA1ADkAYgBjADUAYwA5AA=="
$UrlBase = ConvertTo-SecureString $UrlBase -Key $UrIBase
$UrlBase64 = ConvertTo-SecureString $UrlBase64 -Key $UrIBase
[...]
$url = $UrlBase + $FileVersion + "_" + $LangCode + ".exe"
```

Лістинг 3.6. Затуманена URL-адреса

Хоча рішення, які автоматично встановлюють додатки, вже існують у кількох формах, ця дисертація спрямована на розробку програми, яка може взаємодіяти з цими існуючими рішеннями, щоб забезпечити простіший спосіб встановлення додатків у кібернетичному діапазоні. Це буде корисно для розробки вправ, оскільки можна створити більш різноманітне програмне середовище за коротший проміжок часу. Ми також надаємо статистичні дані про швидкість, з якою пакети програмного забезпечення виходять з ладу через те, що їхні постачальники видаляють файли зі своїх серверів, що підкреслює важливість збереження програмного забезпечення для майбутніх досліджень кібербезпеки.

Кібербезпека може бути предметом етичних дискусій, оскільки атаки часто потрібно здійснювати та вивчати, щоб навчитися та розробити захист від них. Експлойти, як загальнодоступні, так і створені самостійно, необхідні для розуміння вразливостей і способів їх усунення. У той час як деякі дослідження можна безпечно проводити на звичайному комп’ютері, інші

сфери, наприклад, пов’язані з мережею експлойти або зловмисне програмне забезпечення, можуть потребувати цілої мережі комп’ютерів.

Було б неетично проводити дослідження безпеки проти виробничого середовища або кінцевих користувачів без згоди. Використання кібердіапазону забезпечує простір, де такі дослідження та освіта можуть проводитись етично, не завдаючи жодного негативного впливу.

На суспільному рівні використання кібердальності покращує кібербезпеку в багатьох сферах. Уряди, компанії, університети та окремі люди можуть отримати вигоду від кібердіапазонів, оскільки вони можуть бути розроблені для різних цілей. У випадку CRATE існує мініатюрна модель міста, яка ілюструє вигадане місто. Це використовується для демонстрації деяких можливих наслідків кібератаки в міському середовищі, де більше речей підключено до Інтернету. Можна налаштувати навчальні сценарії для перевірки безпеки відкриття мостів, керування потягами та світлофорами, це лише деякі з них. Особи, які приймають рішення, які можуть не мати глибоких технічних знань, зможуть побачити наслідки в дії, не впливаючи на реальне місто, реалізація якого також буде набагато дорожчою.

3.5. Покращення зіставлення програми та версії з CPE. Покращення процесу пропозиції експлойтів

Як згадувалося в розділах раніше наданий FOI сценарій відображення для створення кандидатів на CPE із отриманих даних Chocolatey часто давав неправильні результати, оскільки він був розроблений для прийняття неструктурованих вхідних даних і, таким чином, також порівнював постачальників із назвами програм. Було проведено тест обмеженого масштабу з використанням процесу фільтрації на основі нечіткого відповідності, щоб безпосередньо запропонувати кандидатів CPE зі словника у 1 надані НВД. Іноді це могло забезпечити кращі збіги, ніж наданий сценарій, але працювало набагато повільніше та вимагало, щоб вхідні дані відповідали відомому формату з окремими полями для імені, постачальника та версії.

Оскільки запропонований CPE використовується, щоб запропонувати експлойт, який може працювати в додатку, якість цього зіставлення має важливе значення для створення відповідної пропозиції експлойту. Таким чином, майбутня робота з покращення вилучення CPE із структурованих або неструктурованих вхідних даних стане великою перевагою для процесу автоматичного пропонування відповідних експлойтів для програми.

Ще одна область, у якій є можливості для вдосконалення, — це процес пропозиції модуля Metasploit за CPE. За допомогою існуючих сценаріїв FOI це робиться за допомогою комбінації спроб зіставлення CPE з кодами CVE, а потім із модулями Metasploit, а також шляхом зіставлення CPE із заголовком, описом і цільовими визначеннями кожного модуля Metasploit. Особливо цей останній метод може не вловити значення в заяві, наприклад «впливає на версії до 1.2.3», і натомість запропонує CPE з версією 1.2.3. Використання структурованого формату вказівки версій, на які це впливає, наприклад списки CVE, доступні в NVD, які використовують CPE, щоб вказати це, може спростити підключення модуля Metasploit за допомогою його списку відповідних CVE до набору постраждалих CPE і, таким чином, набір встановлюваних програм. Це схоже на поточний процес, але додаткова робота, спрямована на визначення значення «до версії X» або «до Y включно», може допомогти зменшити кількість помилково ідентифікованих експлойтів.

Також було б доцільно мати можливість пов’язувати CVE разом із програмами, на які вони впливають. Виходячи з уразливості певного CVE, це дозволить створити список уражених програм для цього коду, якщо вони доступні у внутрішньому сховищі. Наявність цієї можливості також спростить налаштування нового сценарію, де можна перевірити вразливості. Щоб це працювало, у базі даних має бути збережено зіставлення між CVE та програмами, доступними в Chocolatey (або інших репозиторіях).

Процес інтерналізації пакета, усе ще покладається на введення користувача для вирішення ситуацій, коли в URL-адресі знайдено змінну або якщо файли не вдалося завантажити . Реалізувавши спосіб автоматичного

вирішення деяких із цих посилань на змінні, можна запакувати більше пакетів без участі користувача. Подібним чином запровадження більш просунутих методів визначення того, чи URL-адреса актуальна для функціонування пакета чи просто включена для довідки, також може допомогти усунути більше випадків, коли процес інтерналізації переривається, щоб отримати запит на введення користувача.

Висновки до розділу

В даному розділі здійснено аналіз отриманих результатів процесів інсталяції програмного забезпечення згідно вимог концепцій кібербезпеки. Проведено представлення процесу першого використання внутрішнього сховища, тестування експлойтів та запропоновані альтернативні методи виконання автоматизованого встановлення програмного забезпечення.

ВИСНОВКИ

В магістерській роботі досліджено моделі та методи автоматизації процесів інсталяції програмного забезпечення згідно вимог концепцій кібербезпеки. Відповідно мета цієї роботи полягала в тому, щоб допомогти покращити дослідження та навчання кібербезпеки шляхом вдосконалення процесу встановлення застарілого програмного забезпечення. Для цього потрібен був інструмент для автоматизації процесу інсталяції та спосіб надійного зберігання кількох версій звичайного програмного забезпечення. В роботі було вирішено питання щодо того як можна надійно зберігати застаріле, потенційно вразливе програмне забезпечення для підтримки повторюваних досліджень безпеки. Щоб вивчити це, було використано менеджер пакетів під назвою Chocolatey, який може надавати автономні пакети програмного забезпечення зі свого загальнодоступного сховища. Chocolatey також надає організаціям можливість розміщувати внутрішнє сховище, щоб мати більш надійне сховище, яке вони контролюють.

Також виконано розробку інструменту для автоматичного перепакування програмного забезпечення з загальнодоступного репозиторію у внутрішній. Він використовує вхідний файл у форматі JSON, який містить назву програмного забезпечення разом із версією. Основною причиною цього було те, що їх було видалено від самих постачальників, тому загальнодоступне сховище не таке надійне, як внутрішнє. Для автоматизації процесу встановлення ми використали Ansible, який добре інтегрується з Chocolatey. При спільному використанні процес інсталяції може бути автоматизований і масштабований для інсталяції кількох машин одночасно, якщо це необхідно.

Створено концепцію побудови інструменту, який приймає вхідний файл у форматі JSON і генерує набір ігор, які Ansible може використовувати. Потім інструмент реєструє результати кожної інсталяції, можливі помилки і тоді надає користувачеві короткий опис того, як пройшло встановлення. Також зберігаються повні журнали, якщо потрібен подальший аналіз.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Microsoft. Microsoft Digital Defense Report. 2020. URL: https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RWxPuf

2. Symantec. Звіт ISTR про загрозу безпеці в Інтернеті. 2019. URL: https://docs.broadcom.com/doc/istr-24-2019-en

3. Симон Дуке Антон, Даніель Фраунгольц, Крістоф Ліппс, Фредерік Поль, Марк Циммер Манн і Ганс Д. Шоттен. «Два десятиліття експлуатації SCADA: Коротка історія». У: 2017 IEEE Conference on Application, Information and Network Security (AINS). IEEE. 2017 р. С. 98–104.

4. Роб Кітчін. «Місто в реальному часі? Великі дані та розумне урбанізм». В: GeoJournal 79.1 (2014), С. 1–14.

5. Ловіса Мікельссон і Йохан Бенгтссон. Svenska smarta städer - En explorativ studie om förväntad nytta och potentiella sårbarheter. 2021 рік.

6. Ерік Бріньольфссон, Джон Дж. Хортон, Адам Озімек, Деніел Рок, Гаріма Шарма та Хун-Ї Тує. COVID-19 і віддалена робота: ранній погляд на дані США. техн. представник Національний Бюро економічних досліджень, 2020.

7. Крістіна Мейлі Вільямс, Рахул Чатурведі та Крішнан Чакравarті. «Ризики кібербезпеки під час пандемії». У: Journal of Medical Internet Research 22.9 (2020).

8. Кібербезпека для Європи. CyberSec4Europe - Європейська мережа компетенції з кібербезпеки. 2021-05. URL: https://cybersec4europe.eu/

9. Вільям Ньюхаус, Стефані Кіт, Бенджамін Скрібнер і Грег Вітте. «Національний Ініціатива з навчання з кібербезпеки (NICE) рамка кадрової роботи з кібербезпеки». в: Спеціальна публікація NIST 800.2017 (2017), стор. 181.

10. Еліна Суні, Юха Пійспанен, Ярмо Невала, Яні Пяйянен і Каро Сахаріне. Повідомити про існуючі кібер діапазони, вимоги. 2020 рік.

11. Джон Девіс і Шейн Маграт. Огляд кіберполігонів і стендів для випробувань. 2013. URL: https: //apps.dtic.mil/sti/citations/ADA594524.

12. Мохаммад Борхані, Мадхусанка Ліянаге, Алі Хассан Содро, Пардіп Кумар, Анка Делія Юркут і Андрій Гуртов. «Безпечні та стійкі комунікації в промисловості Інтернет». У: Посібник із стійких до катастроф комунікаційних мереж. Springer, 2020, стор. 219–242.

13. Марія Ляйтнер, Максиміліан Франк, Вольфганг Хотвагнер, Грегор Лангнер, Олівер Мауарт, Тімеа Пахі, Ленхард Ройтер, Флоріан Скопік, Пол Сміт і Мануель Варум. «AIT Cyber Range: гнучке середовище кібербезпеки для навчань, тренувань іДослідження». In: Proceedings of the European Interdisciplinary Cybersecurity Conference.EICC 2020. Ренн, Франція: Асоціація обчислювальної техніки, 2020. I SBN:9781450375993. Я: 10. 1145 / 3424954. 3424959. URL: https : // doi . org /10.1145/3424954.3424959.

14. Шведське агентство оборонних досліджень (FOI). CRATE - кіберполігон і навчальне середовище. 2020. URL: https://www.foi.se/en/foi/resources/crate-cyberrange-and-training-environment.html

15. Microsoft та інші учасники. Windows Package Manager CLI (він же крило). 2021. URL: https : // github . com / microsoft / winget - cli / дерево / 5898d09203409d08742d97f8eb54e249316695e8

16. Команда Debian Wiki. Програмне забезпечення — вікі Debian. 2021. URL: https://wiki.debian.org/Програмне забезпечення

17. Microsoft. Як інсталювати програми з онлайн-джерел на Windows 10. 2021. URL: https://support.microsoft.com/en-us/windows/how-to-install-programs from - online - sources - on - windows - 10 - a503e8b6 - e45b - fd5a - f4c5 -5a08c8bd9821

18. Microsoft та інші учасники. Стандартні параметри командного рядка інсталятора. 2018. URL: https://docs.microsoft.com/en- us/windows/win32/msi/ standard installer-command-line-options

19. Хуан Хосе Паблос. Без нагляду, система розгортання Windows: автоматична/тиха інсталяція Перемикачі для програм Windows. 2005. URL: http://unattended.sourceforge.net/installers.php

20. Ден Р. Геррік і Джон Б. Тиндалл. «Практики сталого автоматизованого розгортання програмного забезпечення». У: Матеріали 41-ї щорічної конференції ACM SIGUCCS з послуг користувача. 2013 рік,С. 189–196.

21. Програмне забезпечення Chocolatey. 2021. URL: https://chocolatey.org/whychocolatey

22. Програмне забезпечення Chocolatey. Програмне забезпечення Chocolatey | пакети. 2021. URL: https://community.chocolatey.org/packages/

23. Chocolatey Software. Застереження щодо пакетів Chocolatey.org. 2021. URL: https://docs.chocolatey.org/en- us/community- repository/community-packages застереження

24. Microsoft та інші учасники. Менеджер пакетів Windows (попередній перегляд). 2020. URL: https://docs.microsoft.com/en-us/windows/package-manager

25. Microsoft та інші учасники. Спільнота Microsoft Windows Package Manager сховище маніфестів. 2021. URL: https://github.com/microsoft/winget-pkgs

26. Люк Семпсон та інші автори. Відро наступного покоління за замовчуванням для Scoop. 2021 рік.URL: https://github.com/ScoopInstaller/Main/tree/master/bucket

27. Люк Семпсон та інші автори. Відро «Екстра» для Scoop. 2021. URL: https://github.com/lukesampson/scoop-extras

28. Захищено за проектом. Ninite - установіть або оновіть кілька програм одночасно. 2021. URL: https://ninite.com/

29. Кейван Бейгі. Репозиторій пакетів для AppGet. 2020. URL: https : // github . com /appget/appget.packages

30. Microsoft та інші учасники. Знайомство з NuGet. 2019. URL: https://docs.microsoft.com/en-us/nuget/what-is-nuget (відвідано 08.02.2021).

31. Microsoft. Галерея NuGet. 2021. URL: https://www.nuget.org/

32. Red Hat та інші учасники. Ansible документація. 2021. URL: https://docs.ansible.com/ansible/latest/index.html

33. Progress Software Corporation та інші учасники. Огляд Chef Infra. 2021 рік.URL: https://docs.chef.io/chef_overview/

34. Progress Software Corporation. Chef Enterprise Automation Stack. 2021. URL: https://www.chef.io/products/enterprise-automation-stack

35. Еріка Гайді. Керування конфігурацією 101: Написання рецептів шеф-кухаря. 2016. URL: https:/ / www. digitalocean . com / спільнота / навчальні посібники / конфігурація -management-101-writing-chef-recipes

36. Puppet. Вступ до ляльки. 2021. URL: https://puppet.com/docs/puppet/7.4/puppet_overview.html.

37. Puppet. Огляд мови Puppet. 2021. URL: https://puppet.com/docs/puppet/7.6/intro_puppet_language_and_code.html.

38. Puppet. Початок роботи з Bolt. 2021. URL: https://puppet.com/docs/bolt/latest/getting_started_with_bolt.html

39. SaltStack. Вступ до солі. 2021. URL: https://docs.saltproject.io/en/latest/topics/index.html

40. SaltStack. Міньйон Salt Proxy. 2021. URL: https://docs.saltproject.io/en/latest/topics/proxyminion/index.html

41. Шейн Лі (SaltStack). [ПОМИЛКА] Неможливо встановити salt-ssh у Windows (Коментар). 2020. URL:https://github. com / saltstack / salt / issues / 56982 # issuecomment -621333844

42. VMware. vRealize Automation. 2020. URL: https://www.vmware.com/products/vrealize-automation.html

43. Лінод. Налаштуйте Apache за допомогою Salt Stack. 2021. URL: https://blog.linode.com/docs/guides/configure-apache-with-salt-stack/

44. Дженніфер Бедхаммар і Олівер Йоханссон. «Візуалізація атак на кібербезпеку». Магістерська робота. 2020. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:дива-167144.

45. HashiCorp. Навіщо використовувати Packer? 2021. URL: https://www.packer.io/intro/why

46. Національний інститут стандартів і технологій (NIST). Офіційний словник загальної платформи (CPE). URL: https://nvd.nist.gov/products/cpe

47. Корпорація MITRE. CPE - Перелік загальної платформи: специфікації CPE. URL: https://cpe.mitre.org/specification/ (відвідано 17.06.2021).

48. Корпорація MITRE. CVE - Про записи CVE. URL: https://cve.mitre.org/cve/identifiers/

49. Національний інститут стандартів і технологій (NIST). НВД - Загальна інформація. URL:https://nvd.nist.gov/general

50. Rapid7. Metasploit Framework. URL: https://docs.rapid7.com/metasploit/
msf-overview/ (відвідано 26.02.2021).

51. Ханнес Холм і Теодор Сомместад. «Sved: сканування, уразливості, експлойти та виявлення». У: Конференція військових комунікацій IEEE MILCOM 2016-2016. IEEE. 2016, С. 976–981.

52. Томмі Густафссон і Йонас Альмрот. «Огляд автоматизації Cyber Range з а Випадкове дослідження CRATE». У: 25-та скандинавська конференція з безпечних IT-систем (Nordsec 2020). 2020-11, стор. 192–209. Я: 10.1007/978-3-030-70852-8_12.