Міністерство освіти і науки України
Київський Національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ПОЯСНЮВАЛЬНА ЗАПИСКА
Дипломної роботи

магістра
(назва освітньо-кваліфікаційного рівня)

галузь знань *12 Інформаційні технології*
(шифр і назва галузі знань)

спеціальність *125 Кібербезпека*
(код і назва спеціальності)

освітній рівень *магістр*
(назва освітнього рівня)

освітньо-наукова програма *кібербезпека*
(код і назва кваліфікації)

На тему: ***Методи превентивного та реактивного захисту при використанні менеджерів пакетів***

Виконавець: студент ІІ курсу, групи КБм-

Вдовенко Данило Сергійович
(підпис) (прізвище ім'я по-батькові)

	Прізвище, ініціали	Оцінка	Підпис
Науковий керівник	**Лукова-Чуйко Н. В.**		
Рецензент	**Гнатюк С. О.**		
Нормоконтроль	**Фесенко А. О.**		

Київ 2022

Міністерство освіти і науки України
Київський Національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
завідувач кафедри
кібербезпеки та захисту інформації

_____ Н.В. Лукова-Чуйко

«_____» _____2021 року

ЗАВДАННЯ
на виконання дипломної роботи

спеціальності *125 Кібербезпека*
(код і назва спеціальності)

студенту ***КБм-21*** **Вдовенко Данило Сергійович**
(група) (прізвище ім'я по-батькові)

Тема дипломної роботи ***Методи превентивного та реактивного захисту при використанні менеджерів пакетів***

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Рішення засідання кафедри кібербезпеки та захисту інформації факультету інформаційних технологій протокол № 5 від 29.10.2021

2. МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень ***менеджери пакетів які надають програмне забезпечення через мережу Інтернет.***

Предмет досліджень ***розробка та опис методів захисту для безпечної роботи користувачів які використовують менеджери пакетів.***

Мета ***розробити методи та принципи покращеного захисту менеджера пакетів.***

Вихідні дані для проведення роботи

3. ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна ***розроблено методи захисту менеджера пакетів на різних, які можуть бути використані при розробці менеджерів пакетів та їх тестуванні на проникність.***

Практична цінність ***запропоновані методи та рекомендації можуть бути використані при розробці власних менеджерів пакетів та при тестуванні їх на проникність.***

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

5. ЕТАПИ ВИКОНАННЯ РОБОТИ

Найменування етапів робіт	**Строки виконання робіт (початок-кінець)**
Постановка та уточнення завдання	29.10.2021 - 30.10.2021
Аналіз літературних джерел	01.11.2021 – 15.11.2021
Формування запропонованого методу	16.11.2021 – 31.01.2022
Реалізація програмного рішення	01.02.2022 - 31.03.2022
Аналіз ефективності запропонованого рішення	01.04.2022 – 05.05.2022
Перевірка роботи на антиплагіат	06.05.2022 – 08.05.2022
Оформлення пояснювальної записки	09.05.2022 – 14.05.2022
Підготовка презентації	15.05.2022 – 16.05.2022
Підготовка до захисту роботи	17.05.2022 – 18.05.2022

6. РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект ***Запропоновані методи та рекомендації можуть бути використані при розробці власних менеджерів пакетів та при тестуванні їх на проникність.***

Соціальний ефект ***Запропоновані методи та рекомендації можуть бути використані при розробці власних менеджерів пакетів та при тестуванні їх на проникність.***

7. ДОДАТКОВІ ВИМОГИ

Завдання видав _____ Лукова-Чуйко Н. В.
(підпис) (прізвище, ініціали)

Завдання прийняв
до виконання _____ Вдовенко Д. С.
(підпис) (прізвище, ініціали)

Дата видачі завдання: _____
Термін подання дипломної роботи до ЕК _____

РЕФЕРАТ

Пояснювальна записка: 72 с., 11 рисунків, 1 таблиця, 25 джерела.

Об’єкт дослідження: менеджери пакетів які надають програмне забезпечення через мережу Інтернет.

Мета роботи: розробити методи покращеного захисту менеджера пакетів.

Методи дослідження: спостереження, системний аналіз, формалізація, синтез, експеримент.

У роботі досліджено існуючі популярні менеджери пакетів і представлено дев’ять можливих атак на них. Існують атаки, які встановлюють шкідливі пакети, відмовляють користувачам у оновленні пакетів або призводять їх збою. Також визначено три правила безпеки керування пакетами використовуючи пакетні менеджери. У роботі запропоновано методи для реактивного та превентивного захисту під час використання менеджерів пакетів як зі сторони користувача, так і зі сторони дистриб’ютора програмного забезпечення.

Наукова новизна дослідження полягає у тому, що було розроблено методи захисту менеджера пакетів на різних рівнях, поєднуючи методи конфігурації, криптографії та алгоритму вибору довіри.

Напрямки подальших досліджень: дослідження роботи алгоритму у більш складних середовищах та при їх поєднанні; покращення роботи алгоритму та архітектури розповсюдження та встановлення пакетів; модифікація алгоритму та архітектури при появі більш сучасних підходів та протоколів.

Ключові слова: менеджер пакетів, менеджер програмного забезпечення, вебсервіси, репозиторій, захист репозиторіїв, захист метаданих пакета.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

YUM – Yellow dog Updater, Modified

RPM – RPM Package Manager

APT – Advanced Packaging Tool

NPM – Node Package Manager

URI – Unique Resource Identifier

XML – Extensible Markup Language

WSDL – Web Service Description Language

SOAP – Simple Object Access Protocol

W3C – World Wide Web Consortium

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

SQL – Structured Query Language

URL – Uniform Resource Locator

DoS – Denial of Service

SOA – Service Oriented Architecture

SAML – Security Assertion Markup Language

DTD – Document Type Definition

TLS – Transport Layer Security

MLS – Message Layer Security

SSL – Security Socket Layer

BSP – Base Security Profile

QA – Quality Assurance

MITM – Man-in-the-Middle

DNS – Domain Name System

CERT – Computer Emergency Response Team

ВСТУП

Актуальність. Менеджери пакетів наразі один з найпопулярніших засобів розповсюдження програмного забезпечення для сучасних операційних систем. Менеджер пакетів об’єднує програмне забезпечення в архів, які називаються пакетами та забезпечує привілейований центральний механізм для керування програмним забезпеченням у комп’ютерній системі. Сучасні механізми встановлення пакетів інсталюють їх в кореневому контексті операційної системи (root), що порушує питання безпеки керування пакетами та робить це питання важливим для загальної безпеки комп’ютерної системи.

Сучасне програмне забезпечення, в основному, складається з сукупності пакетів з широкого спектра джерел. Такі пакети програмного забезпечення можуть бути розроблені власними силами в рамках внутрішньої системи, придбані у третіх сторін або завантажені з безкоштовних та загальнодоступних джерел. Ризики безпеки при використанні окремо кожного з цих джерел зрозумілі. При спільному ж використанні виникають нові взаємодії які можуть поставити під загрозу навіть організацію з гарним захистом.

Багато організацій використовують загальнодоступні менеджери пакетів – такі як Maven Central, npm, NuGet Gallery та Python Package Index (PyPI) – через перевагу відкритих екосистем, які вони пропонують. Як наслідок, організації досить часто не розглядають такі некеровані відкриті канали як потенційне джерело зловмисного програмного забезпечення. Проблема також полягає в тому, що проєкти, які завантажують пакети одночасно з декількох загальнодоступних і приватних каналів, можуть бути піддані вразливостям у ланцюжку завантажень, такі вразливості будуть унікальними для подібних конфігурацій залежно від компанії та її потреб.

Метою роботи є розробити методи покращеного захисту менеджера пакетів.

Для досягнення поставленої мети вирішуються такі задачі:

- аналіз існуючих менеджерів пакетів, їх архітектура та логіка роботи;
- збір та аналіз даних щодо існуючих властивостей та особливостей архітектури найбільш відомих менеджерів пакетів;
- аналіз вразливостей менеджерів пакетів та причини таких вразливостей;
- проектування та опис методів захисту від проаналізованих атак;
- перевірка та аналіз розроблених методів захисту від атак на предмет стійкості;
- опис рекомендацій щодо впровадження для реактивного та превентивного захисту користувачів.

Об’єктом дослідження є менеджери пакетів які надають програмне забезпечення через мережу Інтернет.

Предметом дослідження є архітектура та вразливості існуючих менеджерів пакетів, їх взаємодія з користувачами.

Методи дослідження, які були використані під час підготовки роботи: спостереження, системний аналіз, формалізація, синтез, експеримент.

Новизна одержаних результатів полягає в аналізі існуючих вразливостей популярних менеджерів пакетів, яким чином їх уникнути використовуючи описані методи превентивного та реактивного захисту менеджера пакетів на різних рівнях.

Практична цінність отриманих результатів полягає в тому, що запропоновані методи та рекомендації можуть бути використані при розробці власних менеджерів пакетів та при тестуванні їх на проникність. Вони вирішують більшість вразливостей які притаманні найбільш популярним менеджерам пакетів.

Апробація. Роботу було апробовано на VII міжнародній конференції «Information Technology and Implementation» (IT&I-2021).

У цій роботі увагу зосереджено на проєктуванні та представлені структури менеджера пакетів, який буде дотримується безпечних принципів проєктування. Дотримуючись описаних правил безпеки управління пакетами,

спроєктований менеджер пакетів не буде вразливий до атак, якими можуть бути вражені популярні менеджери пакетів, такі як YUM і APT.

Автор роботи наведе можливості та наслідки щодо уражень атаками для менеджерів пакетів, які ігнорують необхідні правила безпеки. Також буде описано порушення правил, які призводять до вразливостей та як можливо цих вразливостей уникнути на прикладі YUM та APT.

Також, у цій роботі автор визначить, як побудувати архітектуру безпеки з функціональними можливостями, необхідними для збереження безпеки та буде описано три принципи безпечного управління пакетами.

РОЗДІЛ 1
ВЕБРЕСУРС ЯК ОБ'ЄКТ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ

1.1 Поширені загрози для вебресурсів

Безпека вебресурсу – це технологічні та управлінські процедури, що застосовуються до системи для забезпечення конфіденційності, цілісності та доступності інформації, якою обмінюється вебресурс. У цьому розділі буде описано проблеми безпеки, характерні для вебсервісів, і проілюстровано методи розробки та тестування, необхідні для забезпечення безпеки протягом життєвого циклу розробки вебсервісів.

Вебресурс – це програма, яку можна описати, опублікувати, знайти та викликати через Інтернет [25]. Вебресурс ідентифікується за URI (Unique Resource Identifier), загальнодоступні інтерфейси та прив'язки якого визначаються та описуються за допомогою XML (Extensible Markup Language) у документі WSDL (Web Service Description Language). SOAP (Simple Object Access Protocol), специфікація W3C (World Wide Web Consortium), є найпоширенішою прив'язкою, яка використовується для передачі повідомлень між споживачами послуг (відомими як клієнти) і постачальником послуг (сервером). SOAP визначає, як дані повідомлення мають бути обгорнуті та відформатовані разом з метаданими (заголовками).

Існує багато характеристик, властивих вебресурсу, які ускладнюють їх безпеку. Численні атаки можуть поставити під загрозу конфіденційність, цілісність або доступність вебресурсу або внутрішніх систем, які може виявляти вебресурс. Деякі з цих загроз мають багато спільного зі звичайними системами вебдодатків (вебсайтами), тоді як інші стосуються вебсервісів. Однак, перш ніж заглиблюватися в конкретні проблеми безпеки вебсервісів, спочатку необхідно розглянути загальні загрози безпеки, які можуть виникнути у будь-якому вебдодатку.

Як правило, вебсайти та вебсервіси використовують спільні технології з точки зору мов програмування для розробки програми або сервісу [18, 24]. Наприклад, обидва використовують сховища даних і сервери програм на серверній частині, а на клієнтській частині зазвичай використовують вебсервер і надають доступ через HTTP (Hypertext Transfer Protocol) або HTTPS (Hypertext Transfer Protocol Secure). Така архітектурна та технологічна схожість призводить до того, що вебсервіси успадковують багато поширених загроз безпеки від вебсайтів.

SQL ін’єкції

Достатньо відома та широко застосовувана атака на базі SQL (Structured Query Language) ін’єкції. Цьому типу атаки вебсервіси та вебсайти можуть бути вражені коли оператори SQL динамічно створюються під час виконання програмного забезпечення [9]. Зловмисник може зламати базову безпеку та передати фіксовані введення в оператор SQL, у такому випадку ці введення стануть частиною оператора SQL. SQL-ін’єкції можна генерувати шляхом вставки просторових значень або символів у SOAP-запити, надсилання вебформи або підставляючи певні параметри в URL (Uniform Resource Locator) адреси. Зловмисник може використовувати даний тип атаки, щоб отримати доступ до привілейованих даних, увійти в захищені паролем області без належного входу, видалити таблиці бази даних, додати нові записи до бази даних або навіть увійти в програму з правами адміністратора.

Захоплення повідомлень та відтворення атак

Оскільки вебповідомлення передаються через мережу Інтернет, вони схильні до атак «людина посередині» [21]. Атака «людина посередині» відбувається, коли зловмисник отримує доступ до певної точки між одноранговими користувачами при обміні повідомленнями. Наприклад,

зловмисник може захопити та відтворити запит SOAP, щоб здійснити грошовий переказ, або змінити запит до того, як він досягне місця призначення, що в кінцевому підсумку спричинить серйозні втрати для однієї зі сторін в обміні повідомленнями.

Переповнення буфера

Програми можуть бути не захищеними від неперевірених розмірів вхідних даних. Якщо введені дані не перевіряються, атака переповнення буфера може відбутися віддалено через SOAP-запити або надсилання вебформ [21]. Атаки переповнення буфера відбуваються тоді, коли зловмиснику вдається вказати більше даних в одне або кілька полів і записати в буфер понад розмір пам’яті, виділеної для зберігання даних. Переповнення буфера може призвести до збоїв програми або системи, а при ретельному відтворенні вони навіть можуть дозволити зловмисникам скомпрометувати систему та отримати доступ до несанкціонованої інформації або ініціювати несанкціоновані процеси. Зловмисник може використати цю вразливість, щоб функція повернулася до функції, призначеної зловмисником, або щоб функція виконувала призначену зловмисником процедуру.

Denial of Service

Атаки відмови в обслуговуванні (Denial of Service) виконуються, щоб порушити доступність системи. Існує два способи виконання DoS-атак. Перший спосіб, коли зловмисники споживають ресурси вебпрограми до такої міри, коли інші користувачі не можуть отримати доступ до програми або використовувати її. Таку атаку можна виконати, надсилаючи запит для отримання великої кількості даних [20].

Другий спосіб може бути реалізований, коли зловмисники блокують доступ користувачів до їхніх облікових записів або навіть призводять до збою

всієї програми, перевантажуючи великою кількістю запитів. Зловмисники можуть поєднати ці два підходи з атаками на вебсервіси, щоб максимізувати шкоду.

Неправильна обробка помилок

Достатньо поширеною практикою є коли сервер програми повертає деталі, якщо сталася внутрішня помилка. Такі деталі зазвичай включають трасування стека помилки. Цей стек та такі відомості корисні під час розробки та налагодження функціоналу, але після розгортання програми важливо, щоб такі деталі не потрапляли до звичайних користувачів, оскільки деталі можуть містити інформацію про внутрішню структуру та виявити вразливості. Наприклад, повідомлення про помилку про неправильний запит SQL вказує зловмиснику на те, що його або її вхідні дані використовуються для створення запитів до бази даних, що, можливо, виявляє вразливість SQL ін’єкції [14, 20, 24].

Можливо навести також інший приклад. Запит, який містить неправильне ім’я користувача або пароль, не повинен відправляти помилку чи є ім’я користувача дійсним чи ні – це полегшить зловмиснику ідентифікацію дійсних імен користувачів, а потім використовувати їх для вгадування паролів.

1.2 Характерні загрози для вебсервісів

Як було розглянуто, вебсервіси схильні до всіх видів атак, яким можуть піддаватися вебпрограми. Крім того, вебсервіси піддаються додатковому класу вразливостей, які використовують особливості XML, WSDL і SOAP – компонентів, які складають вебсервіс [23]. Масштаб цих загроз ще більше розширюється, коли підприємства встановлюють фреймворк SOA (Service Oriented Architecture), що є галузевою тенденцією для трансформації організаційної інфраструктури у слабо пов’язані вебсервіси, якими можна

краще керувати, підтримувати та організовувати для забезпечення потреб автоматизації бізнес-процесів. Покладання SOA на вебсервіси як основу системи може створити складнішу систему, підвищуючи тим самим ризики безпеки.

WSDL і сканування доступу

WSDL містить інформацію, що стосується вебслужби, таку як доступні операції, вміст повідомлень які кожна з цих операцій приймає та повертає, а також кінцеві точки, доступні для виклику операцій. Захист вебсервісу має покладатися на криптографічні заходи для повного захисту інформації, а не на неясність побудови внутрішньої структури (слід уникати захисту через неясність), оскільки інформація, яку надає WSDL, може розкрити певні архітектурні аспекти вебсервісу, які можуть полегшити здійснення атаку для неавторизованого користувача [14, 23].

Наприклад, зловмисники можуть визначити поточні формати запитів, досліджуючи схеми та описи повідомлень у WSDL, які можуть містити такі операції, як getItemByTitle, getItemById, placeOrder, getPendingOrders тощо. Знаючи цю інформацію, зловмисник може вгадати інші можливі приховані операції. Насправді розробнику достатньо легко не знати про такі вразливості до вебсервісу, оскільки WSDL здебільшого генеруються автоматично та містять повний опис доступних операцій. Пізніше розробники можуть прибрати необхідні розділи операцій, щоб приховати їх в службі, але залишити пов’язану інформацію, таку як описи повідомлень і типи схем цих операцій.

Порушення контролю доступу

Ця вразливість виникає, коли обмеження щодо того, що аутентифіковані користувачі можуть робити, не дотримуються належним чином. Зловмисники можуть використовувати ці недоліки для доступу до облікових записів інших

користувачів, перегляду конфіденційних файлів або використання несанкціонованих функцій [21]. Наприклад, вебсервіс, який вимагає цифрового підпису запитів певним сертифікатом, має відхиляти запити на всі операції вебсервісу, які повинні забезпечити виконання цієї вимоги. Під час обміну токенами аутентифікації та авторизації, такими як SAML (Security Assertion Markup Language), такі токени можуть містити складні підтвердження авторизації, які можуть бути неправильно застосовані під час реалізації, виявляючи групу вразливостей щодо цього сервісу.

Атаки зовнішніх сутностей

XML має можливість динамічно створювати дані, вказуючи на URI, де знаходяться фактичні дані, зловмисник може замінити зібрані дані шкідливими даними [20]. Наприклад, у наступному XML Entity, який може зображатися в DTD (Document Type Definition) або XML-документі, зовнішнє посилання оголошується з синтаксисом: <!ENTITY name SYSTEM "Some URI">.

Зловмисник міг надіслати запит XML з таким об’єктом, використовуючи довільний URI. Цей URI можна або вказувати на локальні файли XML у файловій системі вебсервісу, щоб змусити синтаксичний аналізатор XML читати великі обсяги даних, вкрасти конфіденційну інформацію або запустити DoS-атаки на інші сервери, якщо зламана система зображатиметься як зловмисник, вказавши URL-адреси інших серверів.

XML-бомби

DTD можуть мати рекурсивні оголошення сутностей, які при розборі та парсингу можуть швидко розгортатися в експоненційному порядку до великої кількості елементів XML [14]. Це споживає ресурси аналізатора XML, що спричиняє відмову в обслуговуванні. Наприклад:

<?xml version="1.0" ?>

```
<!DOCTYPE foobar [
  <!ENTITY x0 "BAD!">
  <!ENTITY x1 "&x0;&x0;">
  <!ENTITY x2 "&x1;&x1;">
  ...
  <!ENTITY x99 "&x98;&x98;">
  <!ENTITY x100 "&x99;&x99;">
]>
```

Якщо його обробити, DTD, наведений вище, отримає збій до серії 2^{100} «BAD!» елементів і призведе до відмови в обслуговуванні.

Велике навантаження даними

Велике навантаження даними можна використовувати для атаки на вебсервіс двома способами. У перший спосіб, вебсервіс може бути зайнятий, надіславши величезну кількість даних XML у запиті SOAP, особливо якщо запит є добре сформованим SOAP-запитом і він відповідає схемі. У другий спосіб, велике навантаження даних також може бути викликане надсиланням певних запитів, які призводять до відповідей з великою кількістю даних [14, 20].

Наприклад, зловмисник може подати запит на пошук доступних елементів, що містять загальне ключове слово в базі даних. Якщо інтерфейс запиту дозволяє це, зловмисник може надіслати запит на повернення всіх доступних елементів у великій базі даних, що споживатиме ресурси та уможливлює DoS-атаку на вебсервіс.

Інший стиль атаки може бути здійснений проти вебсервісів зі збереженням стану шляхом зловживання відкритими операціями, щоб поставити в чергу велику кількість подій або зберегти велику кількість елементів у поточному сеансі. Це виконується шляхом циклічності певних операцій, а потім запиту результату інших операцій, щоб отримати великі

відповіді від сервера. Ці фіктивні запити та відповіді вичерпують ресурси вебслужби та призводять до відмови в обслуговуванні.

Шкідливі SOAP вкладення

Вебсервіси, які приймають вкладення, можуть використовуватися як засіб передачі вірусу або шкідливого вмісту в іншу систему. Вкладення, які доставляють до вебслужби, потім можуть оброблятися іншими програмами, які можуть бути відкритими до вразливостей. Або, що ще гірше, якщо вкладення є виконуваним файлом або пакетом, який містить виконуваний файл, то можна заразити це вкладення вірусом, щоб заразити машину, яка виконує файл.

XPath ін’єкції

XPath ін’єкції подібні до SQL ін’єкцій тим, що обидві є специфічними формами атак із впровадженням коду. XPath дають змогу запитувати документи XML щодо вузлів, які відповідають певним критеріям [9]. Наприклад, XPath може бути таким простим запитом:

```
//*[local-name(.)="user"][attribute::username="foo"]/@*[local-name(.)="password"]
```

Наведений вище XPath повертає значення атрибута пароля для імені користувача «foo». Якщо такий запит створюється динамічно в коді програми (з конкатенацією рядків) з використанням недійсних введених даних, то зловмисник може ввести запити XPath для отримання неавторизованих даних.

1.3 Найкращі практики захисту вебсервісів

Уразливості вебсервісів можуть бути присутніми в операційній системі, мережі, базі даних, вебсервері, сервері додатків, аналізаторі XML, стеку реалізації вебсервісів, коді програми, брандмауері XML, засобі моніторингу або

управлінні вебсервісами чи майже у будь-якому іншому компонент у вашій системі вебсервісів.

Реалізація ефективної безпеки вебсервісів полягає в тому, щоб знати загрози та розуміти технічні рішення для пом’якшення цих загроз. Встановити й виконувати визначений інженерний процес, який враховує безпеку від початку та протягом усього життєвого циклу вебсервісу. Цей процес можна запровадити у такі чотири кроки:

1. Визначення відповідної архітектури безпеки вебсервісів.
2. Дотримання технологічних стандартів.
3. Дотримання ефективного процесу тестування вебсервісів.
4. Створення та підтримка багаторазових та повторних тестів.

Визначення відповідної архітектури безпеки вебсервісів

Архітектура безпеки вебсервісів залежить не тільки від необхідних заходів безпеки, але також залежить від обсягу служби та масштабу розгортання. Наприклад, механізми безпеки можуть бути забезпечені або на самому сервері додатків, або як окремий пристрій безпеки (наприклад, брандмауер XML), який може віртуалізувати службу, перебуваючи посередині між службою та її споживачами [3]. Більшість архітекторів вебсервісів рекомендують відокремити рівень безпеки від сервера додатків, щоб досягти кращої гнучкості та масштабованості. Однак використання засобу безпеки як посередника може не знадобитися для простого наскрізного розгортання вебсервісів [3, 21].

Інше архітектурне рішення, полягає в тому, чи реалізувати захист на транспортному рівні (Transport Layer Security) чи на рівні повідомлень (Message Layer Security). Transport Layer Security (TLS) – це достатньо «доросла» технологія, оскільки стандарти та інструменти вже розроблені. TLS також забезпечує хороший шлях переходу для інженерів, які трохи знайомі з безпекою на транспортному рівні, але новачки у вебсервісах. З іншого боку,

TLS має притаманні обмеження, які роблять його невідповідним для деяких ситуацій. Щоправда, Message Layer Security (MLS) надає альтернативне рішення для ситуацій, коли обмеження TLS є проблемними.

Transport Layer Security

TLS який використовується для вебслужби, може використовуватися для захисту вебслужби. Наприклад, для HTTP можна ввімкнути базову автентифікацію, дайджест-аутентифікацію або SSL (Security Socket Layer).

Основна перевага використання TLS полягає в тому, що він ґрунтується на наявному досвіді вебдодатків для забезпечення безпеки. Багато розробників знають SSL, і його легко ввімкнути на звичайних вебсерверах і серверах додатків. SSL є особливо ідеальним вибором для наскрізної інтеграції вебсервісів. SSL може забезпечити конфіденційність, цілісність, аутентифікацію та авторизацію, захищаючи таким чином вебсервіс від атак захоплення та повторного відтворення, доступу та сканування WSDL.

Недоліком SSL є те, що це протокол «все або нічого». Він не має деталізації для захисту певних частин повідомлення, а також не може використовувати різні сертифікати для різних частин повідомлення. Крім того, усі посередники на шляху повідомлення повинні мати відповідні сертифікати та ключі, щоб розшифрувати все повідомлення, щоб обробити його, а потім повторно надіслати його через SSL, що може бути складно або навіть неможливо в деяких випадках.

Message Layer Security

Технології MLS також можуть стати в нагоді, оскільки вони вирішують ті ж проблеми, що й TLS (конфіденційність, аутентифікація, цілісність повідомлення) на рівні повідомлення, а не на транспортному:

- XML Signature забезпечує механізм цифрового підпису XML-документів або частин XML-документів. Підпис не обов’язково має бути в документі, який підписується, тому використовувати підпис XML можливо також для документів, які не є XML.
- Шифрування XML забезпечує механізм шифрування частин XML-документів. Шифрувати повний документ досить легко, оскільки його можливо розглядати як текстовий документ.
- WS-Security, яка визначає стандартний спосіб захисту одного повідомлення шляхом застосування маркерів імені користувача, підписів XML та шифрування XML до SOAP.

При використанні одного або кількох стандартів безпеки рівня повідомлень важливо використовувати комбінацію, яка забезпечить необхідний захист безпеки. Наприклад, токени імені користувача самі по собі не можуть захистити повідомлення від атак захоплення та повторного відтворення, якщо вони не включають підписану позначку одноразового запису та часу, а тіло SOAP не підписано. Підпис може бути згенерований за допомогою пароля в токені або може бути незалежним від пароля. Однак, якщо цей пароль не присвоюється, як рекомендовано специфікацією, то сам токен слід зашифрувати. Як інший приклад, шифрування XML не забезпечує автентичність або цілісність повідомлення, і в цьому випадку підпис XML можна поєднати з шифруванням для забезпечення всіх трьох вимог.

Дотримання технологічних стандартів

Як і в інших сферах безпеки, дотримання стандартів є необхідною практикою для вебсервісів. Серед фахівців безпеки існує консенсус, що загальнодоступні, широко використовувані, добре проаналізовані криптографічні алгоритми є найкращим вибором просто тому, що вони вже пройшли багато досліджень і ретельної перевірки, оскільки були прийняті галуззю. Той самий принцип також стосується безпеки вебсервісів.

Наприклад, відповідність специфікації WS-Security від OASIS, ймовірно, буде безпечнішою, ніж розробка власної реалізації безпеки, оскільки вона була розроблена експертами в цій галузі з урахуванням захисту від загроз. Крім того, це скоротить час розробки, використовуючи легкодоступну реалізацію специфікації, і вебслужба зможе взаємодіяти з іншими реалізаціями того ж стандарту.

Ще одну річ яку слід згадати стосовно дотримання стандартів – це відповідність базовому профілю безпеки (Base Security Profile) від WS-I. BSP призначений для вирішення проблем сумісності, але в деяких випадках він обмежує специфікації W3C і OASIS таким чином, що сприяє більш жорстким методам безпеки.

Дотримання ефективного процесу тестування вебсервісів

Незважаючи на хороший рівень розуміння загроз безпеці, необхідно мати зрілий інженерний процес, який робить виявлення і тестування вразливостей безпеки невід’ємною частиною процесу розробки вебсервісів, щоб наслідки від загроз були максимально пом’якшені. Думка про безпеку якомога раніше протягом життєвого циклу вебсервісу є ключем до досягнення найкращих результатів найефективнішим способом.

Поширеною проблемою, з якою стикаються компанії, є спроба використовувати ті самі людські та технологічні ресурси для QA (Quality Assurance) щодо тестування вебсервісів, не впроваджуючи належного навчання, процесів і технологічних змін, які можуть успішно використовувати такі ресурси. Ресурси, що використовуються для звичайного QA та тестування, не можна використовувати для вебсервісів з таких причин:

- Тестування вебсервісів вимагає іншого набору навичок у XML, SOAP, WSDL та інших стандартах WS, не кажучи вже про досвід вирішення проблем безпеки, унікальних для вебсервісів.

- Тестування вебсервісів можна краще реалізувати за допомогою спеціалізованих інструментів, а не інструментів, розроблених для традиційного вебтестування. Функції цих інструментів повинні підтримувати стандарти вебсервісів і мати можливість розробляти тести відповідно до цих стандартів.
- Тестування безпеки вебсервісів вимагає спрощення інструментів і практики, які можуть виконувати тести для виявлення вразливостей, які є загальними як для вебдодатків, так і для вебсервісів.

Щоб виявити та запобігти вразливості безпеки у вебсервісах, комплекс інженерних заходів має виконуватися одразу на декількох напрямках. Ці дії можна узагальнити на три рівні.

Тестування першого рівня: статичний аналіз

Знання небезпечних практик написання програмного забезпечення має вирішальне значення при розробці безпечного програмного забезпечення, але виявлення таких практик може бути виснажливим і тривалим процесом, якщо цей процес не буде максимально автоматизований [18]. Інструменти статичного аналізу виявляються дуже ефективними у виявленні небезпечних викликів методів, недостатніх перевірок або низької якості коду. Хоча ручна перевірка коду може виявити деякі з цих проблем, такі проблеми можуть бути достатньо непомітними і їх важко знайти власноруч. Статичний аналіз не усуває повністю потреби в перевірках коду, але може значно скоротити час і зусилля, необхідні для їх виконання, оскільки інструменти статичного аналізу можуть сканувати весь вихідний код для виявлення небезпечних шаблонів кодування. Під час рецензії коду можливо проаналізувати ці випадки, щоб перевірити їх серйозність. Без такої автоматизації в першу чергу було б витрачено набагато більше часу на пошук небезпечних шаблонів кодування.

Наприклад, у Java рекомендується використовувати «PreparedStatement» замість простого «Statement», щоб запобігти ін’єкціям SQL. Правило

статичного аналізу, яке шукає Statement.executeQuery(), викликане за допомогою динамічного рядка, може точно визначити інженерові необхідність заміни цього оператора та забезпечити захист від проблем із впровадженням SQL. Інші поширені шаблони незахищеного коду, які можна знайти за допомогою статичного аналізу, включають ін'єкції XPath, неперехоплені винятки, які спричиняють неправильну обробку помилок, і деякі умови відмови в обслуговуванні, викликані ресурсномісткими операціями.

Підозрілі шаблони коду також можна ідентифікувати за допомогою статичного аналізу. Деякі помилки безпеки виникають через недбалість написання програмного коду. Однак більш небезпечний код може надходити від зловмисників, які приховують у своєму коді троянські програми, пасхальні яйця або бомби уповільненої дії, щоб надати доступ пізніше. Такий код часто покладається на випадкові числа або перевірку дати/часу, щоб уникнути виявлення, і він може змінити звичайні налаштування безпеки, щоб дозволити отримання прихованого доступу. Правила статичного аналізу, які знаходять усі випадкові об'єкти та об'єкти з датою часу, які називаються «тригерами», і які знаходять користувацькі завантажувачі класів і менеджери безпеки, можуть допомогти під час рецензії коду визначити та перевірити підозрілі шаблони [2].

На додаток до виявлення вразливого або підозрілого коду, важливо пам'ятати, що найкращі методи кодування відіграють значну роль у створенні безпечного коду. Існує також кілька різних типів стандартів кодування, які можна застосувати за допомогою статичного аналізу, які мають загальне, а не конкретне значення для безпеки та можуть покращити загальну безпеку програми. Наприклад, якщо знайдено код із проблемою синхронізації, така проблема впливає на безпеку, оскільки проблеми синхронізації, як правило, мають несподівані наслідки.

Тестування другого рівня: тестування на проникнення

Не всі вразливості безпеки можна знайти за допомогою статичного аналізу, тому тестування на проникнення використовується для виявлення таких проблем. Тестування на проникнення динамічно вивчає та сканує вебсервіс, розгорнутий на проміжному або робочому сервері.

Розуміння загроз безпеки дозволяє тестувальнику розробляти тести, які можуть виявити їх за допомогою відповідних інструментів. Наприклад, атаки зовнішніх об’єктів і XML-бомби можуть бути надіслані на службу, щоб перевірити, чи відмовляється служба обробляти інструкції обробки XML або DTD, повертаючи помилку SOAP. Уразливості доступу до WSDL можна виявити, спробувавши отримати WSDL без очікуваного каналу безпеки, якщо він захищений [1, 2]. Наприклад, якщо WSDL захищений клієнтським SSL на порту 443, то він не повинен бути доступним на порту 80. Коли справа доходить до запобігання загрозам сканування WSDL, важливо перевірити WSDL на наявність зайвих артефактів, таких як схеми або невикористані визначення повідомлень.

Атаки захоплення та відтворення можна моделювати, надсилаючи кілька запитів з одним і тим же ідентифікатором повідомлення, що визначає його унікальність. Наприклад, якщо ви використовуєте токени імені користувача, вам слід перевірити службу, надіславши кілька повідомлень з однаковими значеннями одноразового терміну, і переконатися, що служба відхиляє такі запити належним чином. Служба повинна реалізувати достатній, але обмежений розмір кешу для нещодавно прийнятих одноразових значень. Багато реалізацій WS-Security не враховують це за замовчуванням, що робить їх уразливими для захоплення та повторного відтворення атак.

Щоб перевірити вразливість вебсервісів до атак DoS, спричинених великими навантаженнями, такі атаки DoS слід моделювати таким чином, що підходить для вебсервісів. Ви не можете сказати, чи може служба витримати певний сценарій навантаження, якщо такий сценарій не був перевірений. Однак важливо виконувати такі тести навантаження ефективним способом [10].

Деякі інженери-випробувачі мають тенденцію виконувати навантажувальні тести з однаковим статичним запитом для створення навантаження. Хоча це життєздатний тестовий сценарій, його недостатньо, оскільки такі DoS-атаки можуть бути виявлені засобами безпеки мережі. Таким чином, моделювання DoS-атаки вебсервісу має створюватися із значеннями динамічного запиту, які є семантично дійсними і які можуть використовувати ширше охоплення коду в логіці додатків вебсервісу, щоб перевірити робочі межі вебсервісів. Такі атаки важко створити за допомогою власноруч, але вони можливі за допомогою інструментів тестування навантаження, які спеціалізуються на вебсервісах. Фактично, саме існування таких інструментів повинно попередити інженерів вебслужб, що такі атаки можуть бути легко здійснені зловмисником, якщо такі інструменти потрапили в їхні руки. Наприклад, щоб перевірити вебсервіс, який приймає токени імені користувача з мітками часу та одноразовими мітками, важливо застосувати навантаження до служби, де мітки часу та одноразові номери генеруються динамічно для кожного запиту. Інакше такі помилки, як помилки, спричинені проблемами паралельності, залишаться непоміченими. Іншим прикладом можуть бути навантажувальні тести, які надсилають підписані запити, де значення хеша та підпису мають відрізнятися від одного повідомлення до іншого.

Тести навантаження вебслужби повинні не тільки генерувати динамічні запити, але й моделювати реальні сценарії використання або моделі використання. Наприклад, сценарієм використання може бути клієнт вебслужби, який отримує токен авторизації (наприклад, твердження SAML) від центру безпеки, а потім використовує цей токен для подальших викликів вебслужб у різних службах. Щоб перевірити цей сценарій, навантажувальні тести, які постійно використовують один і той же токен авторизації, не відображають реальний сценарій, оскільки в сценарії реального використання кілька користувачів запитують і використовують кілька токенів одночасно. Виконання такого реалістичного тесту навантаження може виявити проблеми паралельності або масштабованості на ранньому етапі. Вебсервіс може

відхиляти дійсні запити або приймати неавторизовані запити під певним навантаженням, навіть якщо такі проблеми не виникають під час регулярного функціонального тестування.

Щоб виявити недійсні відповіді під час навантажувального тесту, навантажувальні тести повинні бути підкріплені достатніми валідаціями відповідей, які забезпечують виявлення регресій від правильної поведінки, оскільки важко перевірити, чи всі запити під час навантаження були виконані з правильними відповідями і чи не було виявлено регресію. Без перевірки відповідей, будуть виявлені лише проблеми з мережевим з'єднанням та помилки HTTP, що не забезпечує достатнього покриття тестом. Наприклад, відповіді можуть бути добре сформованими SOAP-повідомленнями, але з недійсними даними, або, можливо, вони містять повідомлення про помилку. Без достатньої перевірки відповіді під час навантажувального тесту такі неправильні відповіді можуть залишитися непоміченими.

Тестування третього рівня: аналіз під час виконання

Аналіз стану коду вебпрограми під час виконання необхідний для виявлення певних проблем безпеки, які неможливо виявити за допомогою попередніх двох рівнів тестування [3]. Наприклад, у програмах написаних на мовах програмування C/C++, які можуть використовуватися як вебслужби, пошкодження пам'яті (особливо пошкодження пам'яті в стеку) вказує на можливість переповнення буфера, що може спричинити серйозні проблеми з безпекою, а витік пам'яті робить програму більш вразливою до DoS-атак. Динамічний аналіз може виявити вразливості безпеки, які можуть виникнути в результаті інтеграції безпечних компонентів, оскільки він враховує аналіз потоків даних, тоді як статичний аналіз забезпечує широке охоплення коду з більш вузьким обсягом потоків даних.

Поєднання трьох рівнів

Оскільки кожен рівень тестування безпеки надає методологію, яка виявляє вразливості з унікального аспекту, поєднання двох або більше з трьох рівнів може забезпечити потужний підхід до тестування безпеки вебсервісу. Наприклад, статичний аналіз можна використовувати для визначення обсягу необхідного тестування на проникнення, рекомендуючи більш вибірковий набір можливих вразливостей на проникнення [22].

Аналіз під час виконання в поєднанні з тестуванням на проникнення дає тестеру видимість правильної роботи програми, оскільки вона працює в різних умовах. Наприклад, можна виконати аналіз часу виконання під час навантажувального тестування, щоб знайти витоки пам’яті.

Створення та підтримка багаторазових та повторних тестів

Наведені вище методи тестування можуть стати занадто дорогими для виконання, якщо до процесу тестування не буде застосовано належну автоматизацію. Багато організацій не мають ресурсів для виконання цих тестів, якщо їх потрібно виконувати вручну та повторювати для кожного етапу проекту.

Сучасні процеси розробки програмного забезпечення є ітеративними. Діяльність з розробки програмного забезпечення повинна виконуватися на повторюваній основі, а не за жорсткою односпрямованою моделлю розробки, яка тестується лише в кінці. Тестування лише в кінці циклу розробки є однією з головних причин несвоєчасних поставок і перевищення витрат проекту, і вебсервіси не є винятком.

Однак така ітеративна модель розробки може бути ефективною лише за умови належної автоматизації інженерної діяльності. Тому необхідно створити автоматизоване середовище тестування вебсервісів, яке може допомогти створювати тести, підтримувати їх, керувати ними та виконувати їх на

регулярній основі. Наприклад, щовечора в рамках існуючого «щонічного» процесу створення та тестування продукту. Альтернативою може бути запуск різних тестів вебсервісів вручну, кожного разу змінюючи запит клієнта, що є виснажливим, неефективним процесом. Тому краще зберігати та підтримувати всі створені тести вебслужби, щоб їх можна було швидко та легко запустити повторно, і щоб ви могли запускати їх усі автоматично як регресійні тести щоразу, коли вебсервіс оновлюється [3].

Після виконання тестів безпеки на трьох описаних рівнях можна виявити проблеми, які можна виправити ще на етапі розробки. Це збереже час та дасть змогу їх виправити коли це не є занадто ризикованим або занадто дорогим.

Коли проблема буде виявлена, в ідеалі тест, який виявив проблему, слід додати до наявного пулу тестів і повторити його на регулярній основі з усіма іншими тестами, щоб уникнути повторення цієї помилки.

Висновки за розділом 1

У розділі було описано як поширені загрози вебресурсів, так і шляхи превентивного та реактивного вирішення цих загроз. Захист вебсервісів є життєво важливим аспектом забезпечення успішного розгортання сервісу. У разі розгортання для використання партнерами або клієнтами лише безпечні та протестовані вебсервіси можуть забезпечити виправдане рішення для інтеграції, оскільки переваги, які вони надають, мають значно перевищувати ризики.

Для надання необхідного рівня безпеки потрібно розуміти потенційні ризики безпеки та проактивно мінімізовувати ці ризики. Використання правильного інструменту для роботи важливо як з точки зору продуктів, так і технологій. Необхідно бути переконаним у тому, що кожне рішення щодо безпеки супроводжується увагою до деталей у впровадженні та широким тестуванням, розробка вебсервісів буде менш уразлива до атак.

РОЗДІЛ 2
МЕНЕДЖЕРИ ПАКЕТІВ

2.1 Загальна інформація щодо менеджерів пакетів

У цьому розділі буде надана довідкова інформація про менеджери пакетів, яка важлива для кращого розуміння потенційних уразливостей. Більшість менеджерів пакетів можна розділити на два основні компоненти: інсталятор пакетів і засіб розпізнавання залежностей. Інсталятор пакетів – це низькорівневий компонент, який використовує спеціальні файли (так звані пакети) для керування програмним забезпеченням, встановленим на вузлі. Розпізнавач залежностей – це високорівневий компонент, який обробляє зв’язок із зовнішніми серверами, на яких розміщуються пакунки (так звані репозиторії пакетів), а також вирішення залежностей.

Формати пакетів

Пакети складаються з архіву, що містить файли та, у більшості випадків, додаткові метадані. Додаткові метадані для пакета містять інформацію про інші пакунки, з якими він повинен працювати (залежності), функціональні можливості, якими володіє пакет (що надає пакет), а також різну іншу інформацію про сам пакет.

Інсталяція пакетів

Клієнти використовують менеджер пакетів для встановлення пакетів у своїй системі [16, 19]. Менеджер пакетів збирає інформацію про пакунки, доступні в сховищах пакетів. Майже всі менеджери пакетів автоматично завантажують запитані пакети, а також будь-які додаткові пакети, які необхідні

для правильного встановлення програмного забезпечення. Цей процес називається розділенням залежностей. Наприклад, запитуваний пакет foo може залежати від libc і bar. Якщо libc вже встановлено, то libc – це залежність, яку було вирішено (необов'язково встановлювати пакет, щоб задовольнити залежність). Якщо немає встановленого пакунка, який надає bar, тоді bar є нерозв'язаною залежністю, і перед встановленням foo необхідно встановити пакунок, який надає bar. На рисунку 2.1 схематично зображено процес вирішення залежностей для пакетів foo та bar.

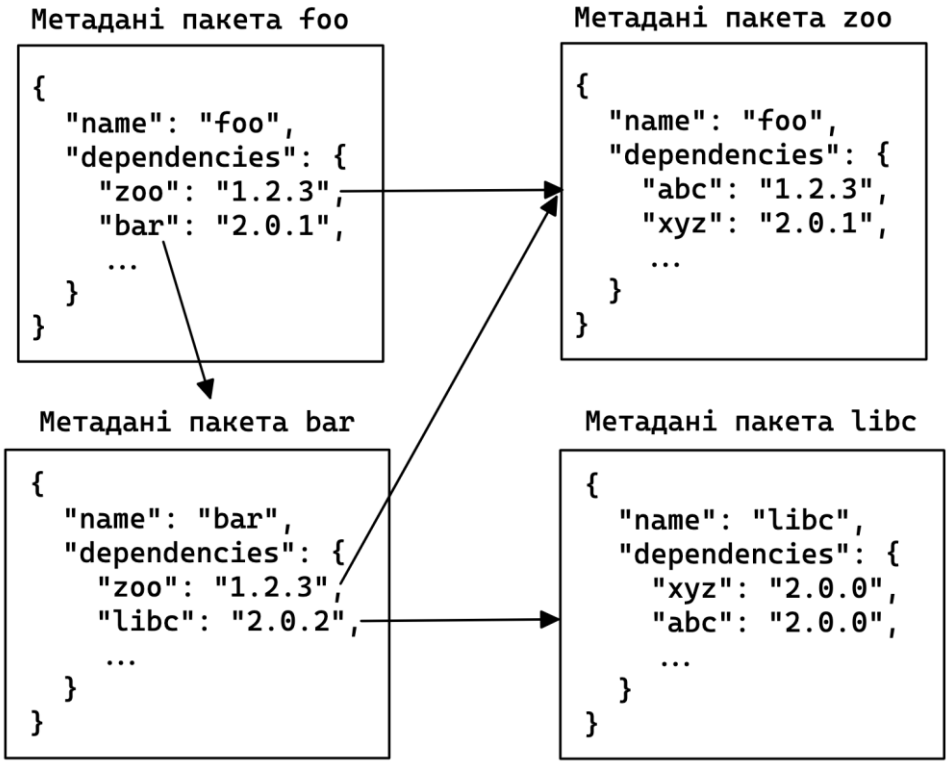

Рисунок 2.1 – Схематичне вирішення залежностей для пакетів foo та bar

Пакети, вибрані для виконання залежностей, можуть мати власні нерозв'язані залежності. Пакети постійно додаються до списку необхідних для інсталювання, доки менеджер пакетів не зможе розв'язати залежність (і не викличе помилку), або не буде вирішено всі залежності.

Фактична інсталяція виконується розпізнавачем залежностей, який викликає інсталятор пакетів. Для кількох розпізнавачів залежностей цілком нормально підтримувати один інсталятор пакетів нижнього рівня (RPM), та також цілком нормальною є ситуація коли різні користувачі одного

дистрибутива можуть використовувати різні засоби розпізнавання залежностей [16, 17, 19].

Одним з популярних засобів розпізнавання залежностей є APT. APT – це засіб розпізнавання залежностей, спочатку створений для використання інсталятора пакетів DPKG у Debian. Однак APT також був портований для використання інших інсталяторів пакетів, включаючи RPM (де він називається APT-RPM). APT автоматизує пошук, конфігурацію та встановлення програмного забезпечення, розв'язуючи залежності та обробляє зв'язок із репозиторіями. Для вирішення проблем безпеки з APT (особливо з підписами) був створений проект secure-APT [17]. Останні версії APT використовують зміни, додані для secure-APT. У цій роботі зосереджено увагу на новіших версіях APT із змінами безпечного APT та проігноровано старіші версії, оскільки відомо, що вони небезпечні.

Іншим популярним розв'язувачем залежностей є YUM. YUM використовується для встановлення та видалення пакетів у системах із програмами встановлення пакетів RPM. YUM виконує автоматичне розв'язання залежностей і зв'язок із репозиторієм.

YUM і APT мають доступні механізми автоматичного оновлення. Якщо воно увімкнено, ці механізми автоматичного оновлення оновлюватимуть старі версії встановлених пакетів на нові версії пакетів, як тільки вони стають доступними, як правило, протягом 1 дня після додавання нових пакетів.

Репозиторій пакетів

Репозиторій пакетів (або сховище пакетів) зазвичай є сервером HTTP або FTP, з якого клієнти можуть отримувати пакети та метадані пакетів. Метадані пакета зазвичай є просто копією вбудованих метаданих пакета для кожного пакета у сховищі. Менеджери пакетів завантажують метадані пакета зі сховища, щоб знати, які пакети доступні з цього репозиторію. Це також надає менеджеру пакетів інформацію про залежності, необхідну для вирішення

залежностей. Щоб полегшити зручне завантаження метаданих пакетів, більшість сховищ зберігають усі метадані пакетів у невеликій кількості стиснутих файлів.

На додаток до метаданих пакету, майже всі репозиторії мають кореневий файл метаданих. Ім’я та розташування кореневого файлу метаданих відрізняються для різних форматів сховищ, але вміст подібний між собою. Кореневі метадані надають місцезнаходження та безпечні хеші файлів, які містять метадані пакета.

Схематичний макет та структуру сховища наведено на рисунку 2.2. Менеджер пакетів завантажує кореневі метадані та використовує їх для пошуку файлів, що містять метадані пакета. Потім менеджер пакетів завантажує метадані пакета. Метадані пакета використовуються для визначення доступності пакета, а також для вирішення залежностей. Потім пакети завантажуються та встановлюються. Кореневі метадані, метадані пакетів і пакети можуть бути підписані залежно від моделі безпеки менеджера пакетів. Стрілки вказують від безпечного хешу до файлу, на який він посилається.

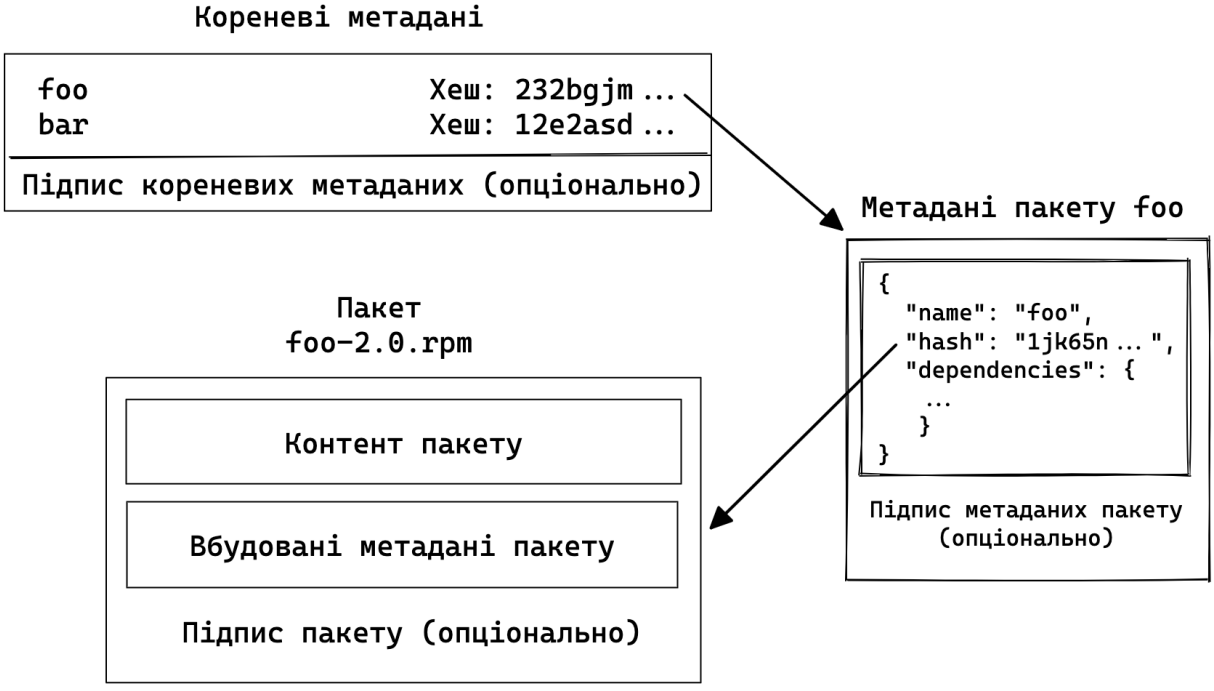

Рисунок 2.2 – Схематичне зображення структури репозиторію

Репозиторії пакетів, як правило, є просто вебсерверами, які використовуються для надання пакетів і метаданих пакетів. Метадані пакета – це метадані у форматі пакета, які витягуються та зберігаються окремо. Часто метадані для всіх пакетів зберігаються в одному tar-архіві.

Зазвичай репозиторії складаються з кількох серверів, на яких розміщені одні й ті ж дані. Ці додаткові сервери називаються дзеркалами і використовуються для розвантаження трафіку з основного сховища. Зазвичай вони містять той самий вміст, що й головне сховище, і оновлюються за допомогою rsync або подібного інструменту.

Дзеркала репозиторіїв

Зазвичай дистрибутив має більше одного сервера, з якого користувачі можуть завантажувати пакети та метадані пакетів. Існує головне сховище для дистрибутива, вміст якого копіюється багатьма окремими дзеркалами. Дзеркало зазвичай містить точно такий же вміст, що й основне сховище, і оновлюється за допомогою rsync або подібного інструменту. Дзеркало відрізняється від основного сховища тим, що дзеркало не призначене для безпосереднього додавання або видалення пакетів його адміністраторами. Пакети додаються або видаляються лише в головному сховищі, а дзеркала пізніше отримують зміни під час копіювання основного репозиторію.

Дзеркало може бути загальнодоступним (доступним для використання будь-ким) або приватним (обмеженим для певної організації). Дзеркало також може бути схвалено розповсюдженням для загального користування, як правило, коли цей дистрибутив контактує з супроводжуючими дзеркалами. Цей тип дзеркала називається офіційним дзеркалом.

Слід зазначити, що в деяких дистрибутивах не використовуються офіційні дзеркала, розміщені сторонніми організаціями. Одним із прикладів є крихітні дистрибутиви, які можуть підтримувати всіх своїх клієнтів за допомогою невеликої кількості репозиторіїв, які безпосередньо контролює

дистрибутив. Інший приклад – роздача, яка вимагає від користувачів платити за використання. Ці витрати часто використовуються для підтримки набору внутрішніх дзеркал для розповсюдження. Крім того, розподіл може дозволити або вимагати від кожної організації, яка використовує дистрибутив, налаштувати власні приватні дзеркала для власного використання всередині організації.

2.2 Модель загроз та можливі атаки

Модель загроз включає зловмисника, який може відповідати на запити, зроблені менеджером пакетів. Це може бути людина посередині (man-in-the-middle), зловмисник, який обманом змусив клієнта зв’язатися з неправильним сервером (наприклад, через підробку кешу DNS), або зловмисник, який отримав контроль над офіційним дзеркалом для розповсюдження. Модель загроз наступна:

- Зловмисник може передавати клієнту довільні файли.
- Зловмисник не знає, який пакет клієнт запросить у сервера.
- Зловмисник не має ключа, довіреного для підписання пакетів, метаданих пакетів або кореневих метаданих. Важливо звернути увагу, що дзеркала зазвичай не мають приватного ключа, який використовується для підписання файлів, вони лише копіюють раніше підписані файли з основного сховища.
- Зловмисник має доступ до застарілих пакетів, застарілих метаданих пакетів і застарілих кореневих файлів метаданих – в мережі Інтернет є багато застарілих дзеркал, де зловмисник може отримати ці файли.
- Зловмисник знає про вразливості в деяких застарілих пакетах і може використати ці вразливості – це можливо, переглянувши журнали змін та оновлення вихідних файлів програмного забезпечення або завантаживши набір інструментів використання експлойтів.

- Зловмисник не знає про вразливості в останній версії будь-якого пакета – вразливості нульового дня навряд чи знатимуть багато зловмисників.
- Якщо менеджер пакетів підтримує підписи, використовуються підписи. Якщо клієнт або дистрибутив вирішує не використовувати підписи, підтримувані його менеджером пакетів, вони так само вразливі, як якщо б вони використовували менеджер пакетів, який не підтримує підписи.
- Використовується термін придатності (expiration date) кореневих метаданих, якщо вони підтримуються, а поточні (незавершені) кореневі метадані не містять жодних уразливих версій пакетів. Кореневі метадані є одним невеликим файлом, тому головне сховище може підписати його відносно часто з коротким терміном придатності.

APT і YUM використовують різні методики для забезпечення безпеки. APT зосереджується на захисті метаданих сховища, а не на підписанні пакетів. Репозиторій APT за бажанням надає підпис для файлу Release (кореневі метадані) у файлі під назвою Release.gpg. Це дозволяє APT перевірити, що файл Release підписаний ключем репозиторію і, отже, надходить із репозиторію. Файл Release містить захищені хеші метаданих пакета в репозиторії. Метадані пакета містять захищені хеші самих пакетів.

Замість того, щоб підписувати метадані репозиторію, YUM використовує підписи на пакетах для забезпечення безпеки. Супроводжувач дистрибутива YUM підписує всі пакети в репозиторії. YUM перевіряє підписи пакетів після завантаження пакетів.

Враховуючи цю модель загроз, досліджено було наступний перелік атак, які можуть бути використані на клієнта. Вплив цих атак різний, але всі вони дозволяють зловмиснику або завершувати роботу, або контролювати комп’ютер клієнта (можливо, через використання пакета з відомою вразливістю). У таблиці 2.1 надано назву атаки, її короткий опис, яка вимога необхідна для зловмисника, її результат та правило безпеки яким було знехтувано. Правила безпеки пронумеровані наступним чином:

1. Не довіряти сховищу.
2. Довіреною сутністю з найбільшою кількістю інформації має бути та, яка підписує.
3. Попередження встановлення ненадійні пакунків.

Під метаданими пакета мається на увазі можливість змінювати метадані пакета, які користувач отримує під час запиту до репозиторія.

Таблиця 2.1

Список атак, ціль зловмисника, результат успішної атаки та правило безпеки, які порушив розробник.

Назва атаки	**Опис**	**Ціль**	**Результат**	**Правило**
Повільне завантаження	Зловмисник уповільнює зв'язок репозиторію, щоб менеджери пакетів очікували з'єднання, не отримували помилку та не зв'язувалися з іншими репозиторіями, щоб отримати оновлення пакетів.	Репозиторій	DoS	1
Нескінченний потік даних	Шкідливий репозиторій (або MITM) повертає нескінченний потік даних у відповідь на будь-який запит файлу.	Репозиторій	DoS / Збій	1
Відтворення старих метаданих	Зловмисник надає старі метадані, які правильно підписані (можливо, щоб запобігти розгляду нових пакетів).	Репозиторій	Застарілий пакет	1
Стороння залежність	Зловмисник змінює метадані для пакета, щоб вказати, що він залежить від пакета або пакетів на вибір зловмисника.	Метадані	Будь-який підписаний пакет	2
Залежність від усіх пакетів	Зловмисник змінює метадані для пакета, щоб вказати, що пакет залежить від усіх доступних пакетів.	Метадані	DoS / Збій	2
Незадовільні залежності	Зловмисник змушує менеджер пакетів ігнорувати дійсні пакети, оскільки підроблені метадані вказують на незадовільні залежності.	Метадані	DoS / Застарілі пакети	2
Встановлення довільного пакета	Зловмисник змінює метадані для пакета, щоб вказати, що він забезпечує будь-яку залежність, яку запитує користувач.	Метадані	Будь-який підписаний пакет	2
Використання відкликаних ключів	Зловмисник використовує відкликаний ключ, щоб змусити користувачів встановити пакети.	Ключ пакету	Довільний пакет	3
Ескалація привілеїв	Зловмисник скомпрометує ключ, довірений для підписання певної групи пакетів, а потім змушує користувачів прийняти підписані шкідливі версії інших пакетів.	Ключ пакету	Довільний пакет	3

У таблиці 2.1 розглядаються цілі зловмисника для запуску атак на такі популярні менеджери пакетів як APT і YUM. Загалом, вразливості які присутні у них, часто також присутні й у інших менеджерах пакетів. Є три рівні вразливості, які дозволяють проводити атаки з різнем рівнем впливу на користувачів. Перший рівень полягає в тому, що зловмисник повинен мати можливість видати себе за репозиторій, щоб запустити базову атаку. Для переходу на другий рівень вразливостей, зловмисник повинен мати можливість підписувати метадані, дозволяючи більш серйозні атаки. Якщо зловмисник може видавати себе за сховище та підписувати метадані, то на третьому рівні він повинен мати можливість підписувати пакунки, дозволяючи найсерйозніші атаки.

Крім того, замість того, щоб скомпрометувати репозиторій і ключі підпису, зловмисник може просто скомпрометувати ключ розробника, дозволяючи зловмиснику запускати атаки за допомогою звичайних механізмів оновлення пакетів у сховищі.

Видати себе за репозиторій

Атаки на менеджери пакетів вимагають, щоб зловмисник мав можливість надавати трафік від імені репозиторія. Це можна зробити трьома способами: атаки «людина посередині» (MITM), отримання контролю над репозиторієм або керування дзеркалом.

У атаках MITM зловмисник перехоплює трафік між двома користувачами, що спілкуються, і натомість надає власні дані. Ці атаки відносно прості для запуску [1, 14]. Коли використовується незахищений протокол, такий як HTTP, зловмисник MITM еквівалентний зловмиснику, який може контролювати репозиторієм. Використання безпечного протоколу, такого як HTTPS, може перешкодити зловмисникам MITM маскуватися під репозиторій, однак, HTTPS не використовується широко.

Сторонні розробники часто встановлюють власні репозиторії, щоб надавати програмне забезпечення, яке не є ядром дистрибутива [13]. Зловмисник може скористатись цією звичайною практикою, створивши власний репозиторій, який нібито надає стороннє програмне забезпечення та спробувати залучити користувачів до його використання. Однак це впливає лише на підгрупу користувачів, які використовують це програмне забезпечення.

Інший шлях для атаки – зловмисник отримати контроль над дзеркалом. Адміністратори дзеркала можуть вказати діапазон IP-адрес, до якого вони хочуть обслуговувати пакети. Насправді націлювання на підмережу означає, що користувачі цієї підмережі використовуватимуть лише це дзеркало. Це дозволяє легко націлювати атаки (на конкретну країну чи організацію) і зменшує кількість інших сторін, які споживатимуть ресурси на дзеркалі [13].

Доступ до ключа метаданих

Коли зловмисник може видати себе за репозиторій, це означає що він може створювати підроблені пакети або метадані в сховищі [6, 7, 8, 11]. Деякі репозиторії використовують приватний ключ (так званий ключ метаданих) для підписання кореневих метаданих, які публікуються репозиторієм. Якщо репозиторій зберігає ключ метаданих у автономному режимі або є дзеркалом іншого репозиторію, який виконує всі підписи, компрометація сховища може не означати компрометацію ключа підпису. MITM, який також не володіє ключем, не може підписувати метадані від імені репозиторію.

У YUM і багатьох дистрибутивах APT немає ключа метаданих для компрометування. Будь-який зловмисник, який контролює репозиторій, може запустити ці додаткові атаки без компрометації додаткових ключів. Якщо сховище підписує метадані та має ключ онлайн, то зловмисник, який зламав репозиторій, може підробити метадані. Подібним чином, якщо зловмисник контролює сховище третьої сторони і заманює користувачів додати ключ

репозиторія як довірений, користувачі стають вразливими та піддаються атакам.

Цікавим є те, що оскільки кореневі метадані та підпис метаданих знаходяться в окремих файлах, вони не завантажуються та не перевіряються окремо. Це означає, що в деяких випадках підпис і кореневий файл метаданих не збігаються (наприклад, під час оновлення) навіть за відсутності зловмисників. Ці помилкові результати можуть допомогти замаскувати напади.

Також, важливо зауважити, що навіть якщо кореневі метадані підписані, зловмисник, який не має ключа метаданих, все одно може запустити атаки метаданих пакетів, якщо зможе переконати сховище розмістити шкідливо створений пакет. Це означає, що розробники можуть самі запускати ці атаки [4, 7, 8].

У випадках, коли кореневі метадані не підписані, можна стверджувати, що якщо пакети підписані, то метадані пакета захищені від підробок. Хоч YUM і не підписує кореневі метадані, більшість дистрибутивів YUM підписують самі пакети. Можна припустити, що підпис пакета використовується для перевірки метаданих пакета, але це не так. Репозиторій надійний для створення точних метаданих, які відображають вміст пакета. Ці метадані використовуються розпізнавачем залежностей, щоб вирішити, які пакунки слід завантажити. Таким чином, розпізнавач залежностей явно довіряє репозиторію який надає точну інформацію щодо залежностей. Єдиний спосіб бути впевненим, що репозиторій надав точні метадані, це використовувати метадані із завантаженого та перевіреного пакета. Це означає, що на практиці підпис пакета не використовується для перевірки метаданих пакета. У результаті метадані пакета, отримані зі сховища, (у кращому випадку) захищені лише безпечним хешем у кореневих метаданих (підписаних за допомогою ключа метаданих у деяких дистрибутивах APT), а не розробником, який упакував програмне забезпечення.

Доступ до ключа пакета

На додаток до ключів, які використовуються для підписання метаданих кореневого сховища, деякі дистрибутиви YUM підписують самі пакети [16, 17, 19]. Якщо ключ, який використовується для підписання пакетів, скомпрометований, зловмисник може підписувати довільні пакети.

Однак важливо враховувати, що відбувається, коли розпізнавач залежностей вирішує, що для вирішення залежностей необхідно встановити пакет, який не підписаний або підписаний ненадійним ключем. Якщо користувач охоче встановлює будь-які непідписані пакунки, то результат еквівалентний компрометації приватного ключа. З YUM, коли користувач намагається інсталювати пакет, підписаний ключем, якого немає в його сховищі безпечних ключів, YUM поверне помилку, якщо параметр конфігурації gpgkey не встановлено. Якщо налаштовано gpgkey, YUM шукатиме ключі GPG за вказаними URL-адресами (часто в одному репозиторії) і запитує користувача, чи хоче він встановити ці ключі, щоб перевірити пакет. Отже, якщо зловмисник може надати власний ключ, коли користувач запитує URL-адресу, зазначену в gpgkey, все, що йому потрібно зробити, це надати користувачеві пакет із підписом, який користувач не може перевірити. Користувачеві буде запропоновано додати ключ зловмисника до свого сховища безпечних ключів.

У дистрибутивах APT і YUM, які не підписують пакети, немає ключа пакета, який можна скомпрометувати. У цьому випадку компрометація ключів репозиторію та метаданих означають можливість запуску будь-яких атак, які в іншому випадку вимагали б компрометацію ключа пакету.

Ключ розробника

На додаток до розгляду доцільності атак на репозиторії, важливо розглянути, як пакунки оновлюються в репозиторіях. Якщо зловмисник може

занести шкідливий вміст у сховище через легітимні канали, він також може здійснити атаки.

Для прикладу, можна описати процес оновлення пакету Debian розробником [17]. Для того щоб оновити пакет для популярного дистрибутива APT Debian, розробник завантажує пакет, підписаний його закритим ключем, до архіву пакетів. Якщо підпис дійсний та відповідає ключу в списку розробників і пакет правильно відформатований, файл пакету поміщається в пул пакетів, які утворюють щоденний пул оновлень. Усі пакети в щоденному оновленні мають нові метадані, згенеровані для них. Кореневий файл метаданих автоматично підписується одним ключем Debian, який усі користувачі Debian мають у своєму наборі ключів. Щоденне оновлення потім передається на сотні дзеркал по всьому світу.

У цьому процесі є кілька цікавих моментів. По-перше, процес пакування та оновлення повністю автоматизовано. З опису практик безпеки на головному сайті Debian, єдине, що потрібно зробити зловмиснику – це скомпрометувати ключ розробника, щоб мати можливість завантажувати шкідливі пакети.

По-друге, будь-який розробник може завантажити оновлену версію будь-якого пакета. Це означає, що ви неявно довіряєте кожному ключу розробника Debian, навіть якщо ви не використовуєте програмне забезпечення, яке вони обслуговують. Враховуючи, що в базі даних розробників Debian перераховано понад 2448 ключів, існує значна кількість ключів, які мають бути захищеними, щоб дистрибутив залишався безпечним.

Окрім довіри до розробників, які працюють над основними дистрибутивами, важливо враховувати сторонніх розробників. Якщо користувач хоче перевірити програмне забезпечення сторонніх розробників перед встановленням, йому потрібно додати сторонній ключ до свого сховища довірених ключів. Залежно від заходів безпеки, вжитих сторонніми розробниками, цей секретний ключ може бути більш або менш безпечним, ніж закритий ключ дистрибутива користувача. Однак це завжди може бути іншим шляхом для атаки [12].

2.3 Типи атак

Після опису можливості та шляхів отримання зловмисниками необхідного доступу для запуску атак, важливо описати які саме типи атак можуть використовуватися та як. Причина ефективності зазначених атак пов’язана з порушенням трьох правил безпеки в управлінні пакетами. Причина ефективності атак пов’язана з порушенням трьох правил безпеки в управлінні безпекою пакетів.

Різні менеджери пакетів мають різну стійкість до інших атак, таких як атаки відтворення. Залежно від механізмів безпеки менеджера пакетів результатом може бути будь-який з описаних – від DoS до ескалації привілеїв.

Розглянемо більш детально зазначені дев’ять атак.

Повільне завантаження

Це доволі простий тип атаки полягає в тому, щоб дозволити клієнту відкрити з’єднання, але потім відмовитися надсилати дані або надсилати з дуже повільною швидкістю. Зловмисник утримує з’єднання відкритим якомога довше. Це не дозволяє клієнту встановлювати оновлення з інших сховищ. Такі менеджери пакетів як APT і YUM не реєструють і не друкують жодної корисної інформації, щоб допомогти адміністратору виявити, що атака відбулася. Крім «паузи», коли він намагається зв’язатися зі сховищем, немає виводу, який би вказував на проблему. Схематично вразливість зображено на рисунку 2.3.

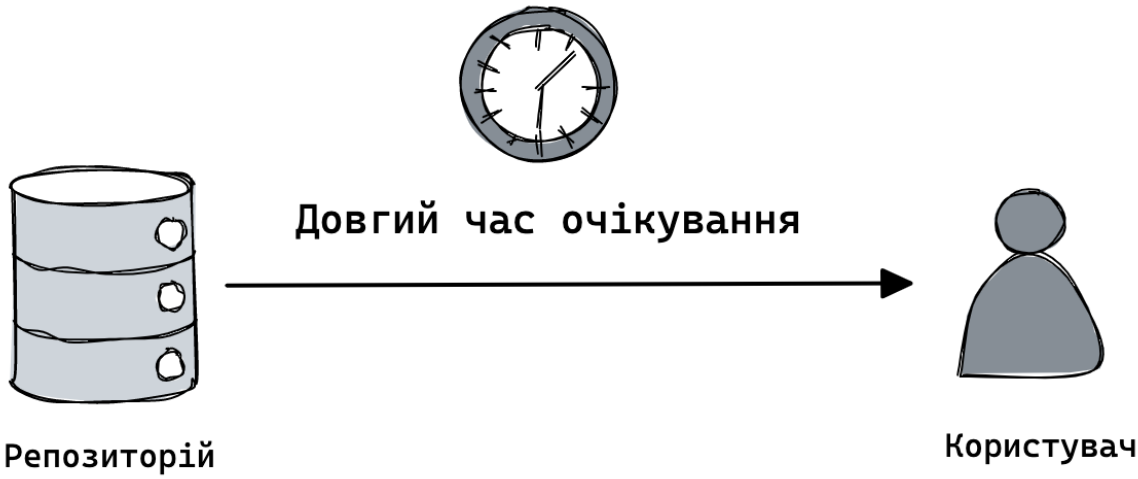

Рисунок 2.2 – Схематичне зображення реалізації повільного завантаження даних

Нескінченний потік даних

Інший тип атаки полягає у тому, щоб повернути клієнту нескінченний потік даних щоразу, коли запитуються файли. Атака з використання нескінченного потоку даних працює для запобігання клієнтам отримувати оновлення пакетів з інших сховищ. Однак ця атака також споживає велику кількість дискового простору в клієнтській системі, а також пропускну здатність мережі та ЦП. Виснаження ресурсів, особливо дискового простору, на клієнтській машині може мати катастрофічні наслідки. Наприклад, у Fedora та Ubuntu це запобігає веденню журналу, пошкоджує бази даних і зупиняє доставку пошти.

Ефективність цієї атаки можна зменшити, використовуючи окрему файлову систему для каталогу кешу. Однак ця атака все ще запобігає встановленню оновлень з інших сховищ пакетів. Схематично цей тип атаки зображено на рисунку 2.4.

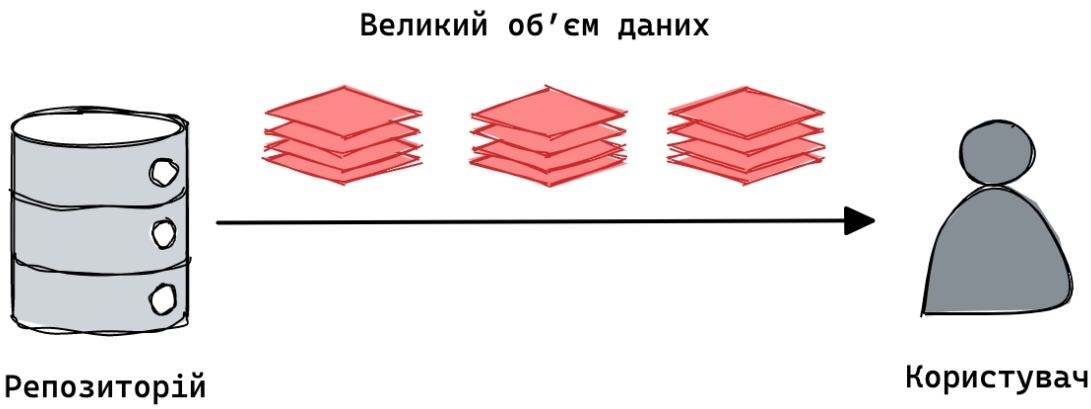

Рисунок 2.3 – Схематичне зображення реалізації нескінченого потоку даних

Відтворення старих метаданих

Деякі репозиторії APT підписують кореневі метадані, щоб запобігти маніпуляціям з боку зловмисника. Підпис не дозволяє зловмиснику, який не може підписати кореневі метадані, замінювати довільні метадані. Однак це не заважає зловмиснику відтворювати старі метадані пакетів. Зловмисник може, наприклад, захопити кореневі метадані з дати випуску вразливого пакета. Через деякий час у майбутньому, після того, як пакунок буде виправлений, зловмисник може відтворити свої захоплені метадані, змусивши клієнтів, які запитують пакет, інсталювати вразливу версію.

У APT, навіть якщо метадані підписані, щоб запобігти підробці, немає захисту від відтворення старих метаданих. APT ігнорує дату, зазначену у файлі метаданих. Ця дата може бути старішою за попередній файл або навіть мати дату в майбутньому. Фактично, APT перезаписує наявні метадані файлами, які він завантажує. Це означає, що якщо користувач не збереже старий файл метаданих вручну, у нього не буде можливості перевірити попередній стан сховища. Схематично дану атаку представлено на рисунку 2.5.

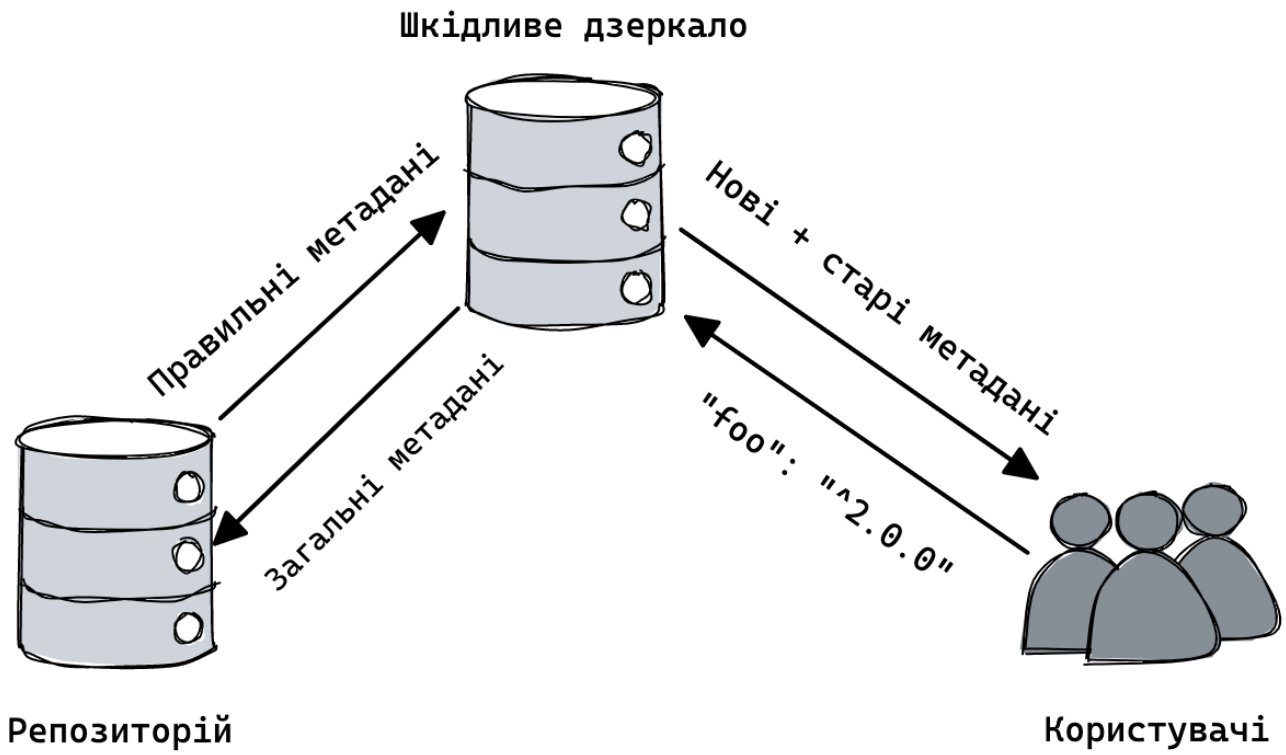

Рисунок 2.4 – Схематичне зображення відтворення старих метаданих

Сторонні залежності

Ця атака відтворюється шляхом надання неправдивої інформації про залежності для специфічного пакета. Наприклад, зловмисник може надати інформацію, що кожен пакет залежить від якогось «стороннього» пакета на вибір зловмисника.

Ця атака працює як на APT, YUM та багатьох інших менеджерах пакетів. Дивно, але ні APT, ні YUM не перевіряють, що залежності в метаданих завантажених пакетів збігаються з метаданими пакетів, які вони отримують з репозиторію. Єдине обмеження у виборі стороннього пакета полягає в тому, що якщо перевіряються підписи пакета, то пакет повинен бути правильно підписаний. Навіть якщо використовуються підписи, ця атака може призвести до встановлення нових вразливих пакетів. Атаку схематично зображено на рисунку 2.6.

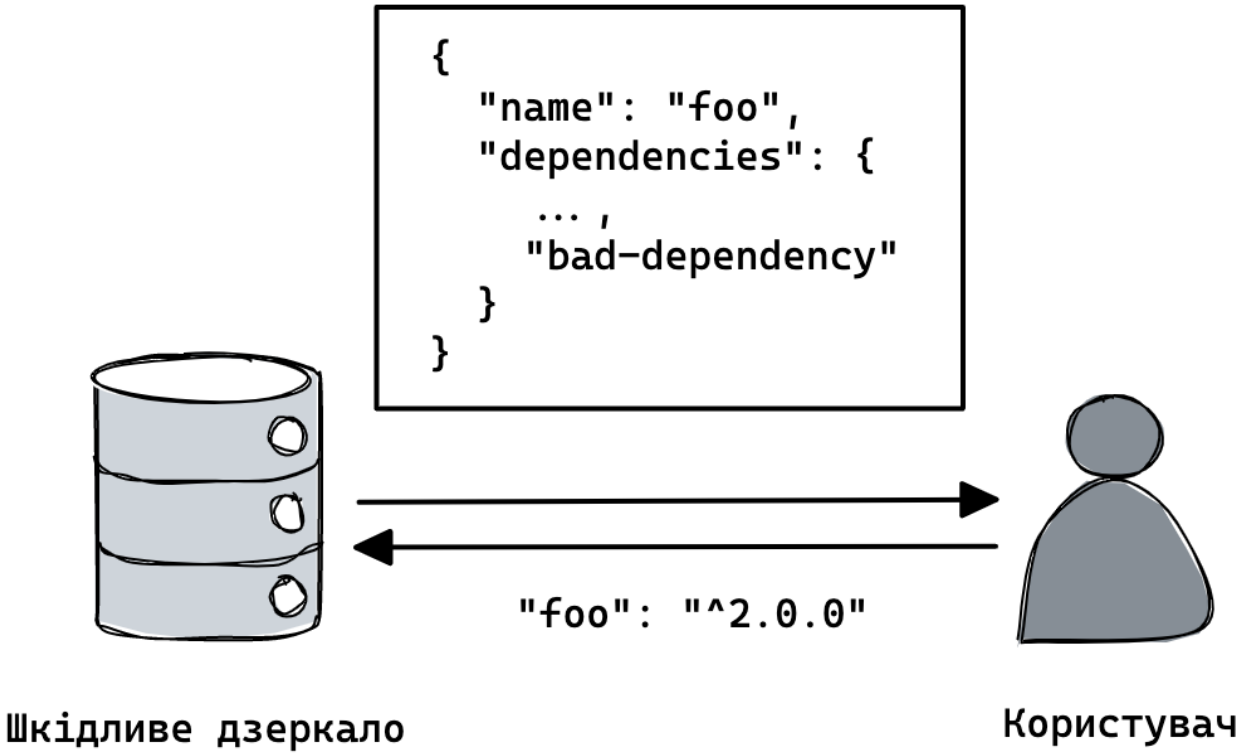

Рисунок 2.5 – Схематичне зображення надання сторонньої залежності

Залежність від усіх пакетів

Інша потенційна атака включає повернення метаданих пакета, де запитуваний пакет має величезну кількість залежностей від інших пакетів. У APT та YUM усі ці пакунки будуть завантажені перед перевіркою будь-яких підписів.

На додаток до споживання дискових і мережевих ресурсів клієнта, ця атака може використовуватися зловмисним сховищем для запуску атаки на інші репозиторії. Шкідливе сховище може надавати нову версію пакунка, яка залежить від усього набору основних пакетів розповсюдження і що в шкідливому репозиторії не розміщено жодного з основних пакетів розповсюдження. Якщо припустити, що клієнт налаштований на використання кількох репозиторіїв, він завантажить усі пакунки дистрибутива з інших репозиторіїв. Вигляд метаданих у випадку цієї атаки можна переглянути на рисунку 2.7.

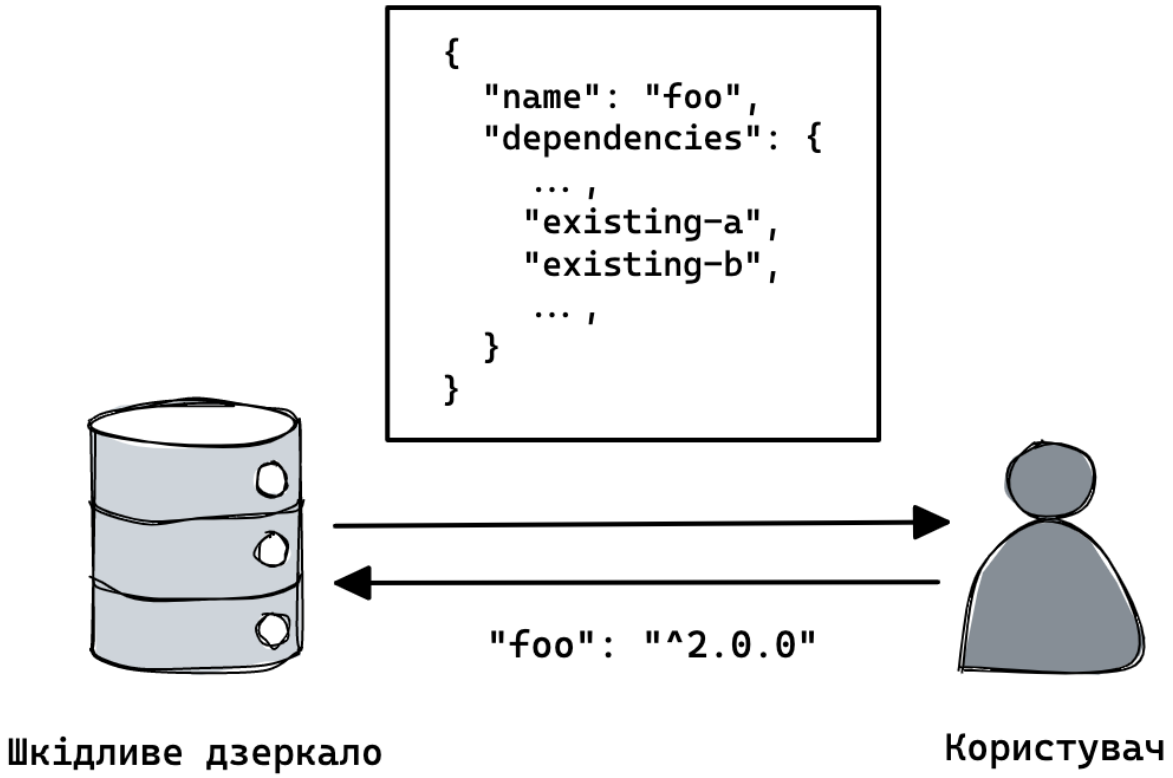

Рисунок 2.6 – Схематичне зображення залежності від усіх пакетів

Незадовільні залежності

Щоб запобігти встановленню пакунка, зловмисник може повернути список залежностей, які вказують на те, що пакет, який цікавить користувача, має нерозв’язні залежності. Це дасть змогу APT і YUM не встановити пакунок, а користувачеві буде здаватися, що репозиторій або пакет був помилковим, а не підозрюваною атакою. Якщо використовується розширення для YUM, YUM спробує встановити альтернативну версію пакета, якщо потрібна версія недоступна або не може бути вирішена механізмом розв’язування залежностей. Цей механізм може дозволити зловмиснику інсталювати певну бажану (уразливу) версію пакета.

Варто зауважити, що зловмиснику необов’язково має правильно підписати пакет який його цікавить. Оскільки пакети завантажуються лише після вирішення залежностей, клієнт не завантажуватиме пакет (або

перевірятиме підпис). На рисунку 2.8 представлено вигляд метаданих пакета коли версії не буде існувати.

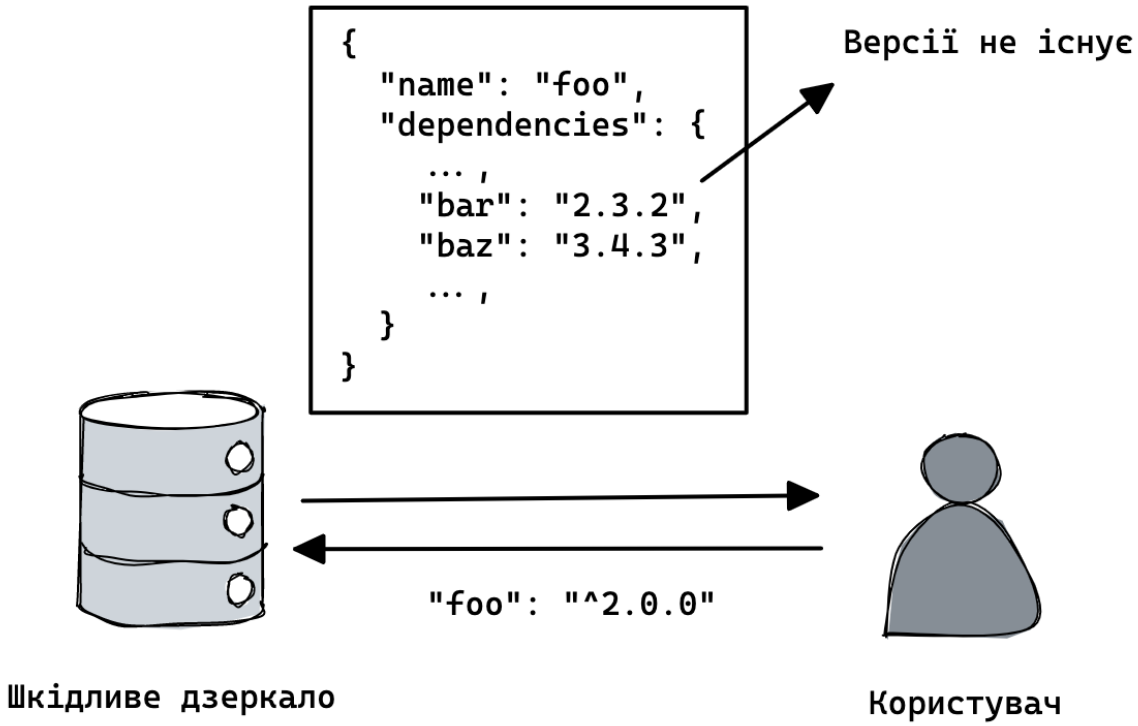

Рисунок 2.7 – Схематичне зображення незадовільної залежності

Встановлення довільного пакета

Інша потенційна атака включає повернення метаданих, які роблять вигляд, що пакет розв’язує величезну кількість віртуальних залежностей. Щоразу, коли для вирішення віртуальної залежності потрібен пакет, цей пакет буде викачано та розглянуто. Метадані цього пакета можуть бути створені, щоб спричинити його інсталяцію над іншими пакетами, які забезпечують таку саму віртуальну залежність.

Якщо існує реальний пакет з такою ж назвою, що й віртуальна залежність, перевага завжди надається справжньому пакету. Це дозволяє зловмиснику створити пакет httpd у своєму сховищі, якому буде віддано перевагу перед будь-яким із пакетів із основного дистрибутива, які забезпечують цю віртуальну залежність (наприклад, apache). Якщо є кілька пакетів, які забезпечують однакову віртуальну залежність, APT розв’язує

залежності від віртуальних залежностей, вибираючи пакунок, ім’я якого стоїть першим за алфавітом. YUM вибирає пакет, назва якого має найкоротшу довжину. Таким чином, зловмисник може створити файл з іменем a.deb або a.rpm, який буде мати перевагу перед усіма іншими пакетами.

Ця атака може бути використана зловмисниками різними способами. Якщо використовуються підписи пакетів, зловмисник може використовувати це як інший метод для встановлення сторонніх пакетів. У деяких ситуаціях це ефективніше, ніж атака сторонніх залежностей, оскільки це призведе до встановлення залежностей пакета розміщених в інших сховищах. Якщо підписи пакетів не використовуються або зловмисник зламав ключ підпису пакета, то зловмисник може використовувати це для частішого встановлення шкідливого пакета. Користувачам, які хочуть встановити інше програмне забезпечення, також буде встановлено шкідливий пакет. Цю атаку також можна використовувати для запуску атак «залежно від усього», якщо пакет, який надає все, має величезний список залежностей.

Використання відкликаних ключів

Однією з поширених потреб у будь-якій системі, яка використовує криптографію з відкритим ключем, є механізм відкликання ключів. Менеджери пакетів не є винятком. Необхідно розглянути два питання: кількість ключів у системі та спосіб відкликання.

На жаль, популярні дистрибутиви APT і YUM, які виконують підписання, використовують кілька ключів для підписання всього. У Debian та Ubuntu всі кореневі файли метаданих підписуються одним ключем. У Fedora кожен пакет підписується одним ключем. Немає способу заднім числом скасувати довіру до підписаного елемента без фактичного скасування довіри до всіх елементів, підписаних цим ключем.

Типовий механізм відкликання ключа полягає у сповіщенні користувача про те, що ключ має бути відкликаний у спеціальний спосіб. Наприклад, таке

повідомлення може мати форму повідомлення безпеки електронною поштою від такої організації, як CER. Потім користувач вручну витягує ключ із свого сховища.

Система керування пакетами призначена для швидкого і часто автоматичного оновлення пакетів. Це суперечить порівняно повільному процесу відкликання ключа вручну. Відкликання ключа створює змагання між тим, хто відкликає ключ (хто намагається позбутися довіри до скомпрометованого ключа) і сторони зловмисника (яка може намагатися використати ключ до його відкликання). Оскільки відкликання ключа є повільним ручним процесом, а оновлення пакетів відбувається швидко, це дає зловмисникам значну перевагу.

Ескалація привілеїв

Для перевірки підписів пакетів YUM використовує набір відкритих ключів, які вважаються надійними. Щоб додати новий відкритий ключ для перевірки пакета, користувачі додають ключ до свого сховища ключів. Однак перевірка того, чи є ключ підпису пакунка в сховищі ключів користувача, є звичайною булевою перевіркою – він або є, або його немає. Це означає, що користувач, який хоче перевірити пакети проекту Apache за допомогою відкритого ключа проекту Apache, також буде неявно довіряти gcc RPM, підписаному їхнім ключем.

Наявність атаки ескалації привілеїв створює проблему для адміністраторів і користувачів. Чи повинен користувач, який знає правильну версію відкритого ключа Apache, встановлювати ключ у своє сховище ключів, якщо він не планує встановлювати Apache найближчим часом? Якщо вони не додадуть ключ, їм потрібно буде підтвердити правильність ключа, якщо вони вирішать встановити Apache пізніше (адміністративний біль). Якщо ключ додано, компрометація ключа Apache може вплинути на безпеку системи, навіть якщо пакети Apache не встановлені.

Висновки за розділом 2

У цьому розділі було описано та показано кореляції між вразливими місцями у та поза репозиторієм. Якщо є спільні риси між помилками програмування та проектування, то, можливо, розробники системи можуть уникнути цих проблем. Атаки, описані в цьому розділі, можуть бути пов’язані з недотриманням трьох правил безпеки які було згадано.

Перше правило безпеки керування пакетами – не довіряти репозиторію. Це означає ретельну перевірку даних, які повертає сховище, і перевірку того, що вони відповідають правильним правилам протоколу передачі даних. Неправильна перевірка повернених даних призводить до вразливостей при відтворенні старих метаданих. Недостатня перевірка протоколу передачі призводить до вразливостей із нескінченними даними та повільним пошуком.

Довіреною особою з найбільшою кількістю інформації має бути той, хто підписує – це друге правило безпеки керування пакетами. Пакети (і метадані) повинні бути підписані суб’єктом, який володіє найбільшими знаннями. Це означає, що метадані пакету повинні бути підписані розробниками замість репозиторіїв (оскільки репозиторій не має уявлення, чи є метадані пакета правильними). Це правило порушується, оскільки метадані пакета не перевіряються за допомогою ключа підписанта пакета, що призводить до атак, пов’язаних із підробкою метаданих пакета для зміни залежностей.

Останнє правило безпеки керування пакетами – не встановлювати ненадійні пакунки. Це означає, що в центрі уваги має бути вбудований механізм відкликання, щоб зменшити вразливість користувачів до ключових компромісів. Іншим важливим фактором для запобігання встановленню ненадійних пакетів є запобігання розробникам одного пакета від представлення шкідливих версій іншого пакета (ескалація привілеїв).

РОЗДІЛ 3
АРХІТЕКТУРА БЕЗПЕЧНОГО МЕНЕДЖЕРА ПАКЕТІВ

3.1 Принципи проектування та методи захисту

Правила безпеки управління пакетами описують, як уникнути помилок безпеки в управлінні пакетами. Однак вони не описують, як забезпечити необхідну функціональність для забезпечення безпеки. У цьому розділі буде досліджено та розглянуто три принципи безпечного керування пакетами, які забезпечують цю функціональність. Потім буде наведено приклади того, як ці принципи слід застосовувати на практиці.

Підсумовуючи описані вразливості та атаки у другому розділі, можна описати кілька простих дій, які зменшать ефективність багатьох атак [3, 6, 13]:

1. Перевіряти зв’язок репозиторію. Перевіривши, що розміри файлів і швидкість передачі даних є розумними, можливо обмежити ефективність нескінченних даних і атак повільного пошуку.
2. Відстежувати час підпису. Необхідно відмовитися та не приймати старіші версії підписаних даних. Це обмежить ефективність атаки відтворення старих метаданих.
3. Використовувати HTTPS або інший захищений канал передачі даних. HTTPS ускладнює зловмиснику запуск будь-якої з атак, тому що атаці MITM (man-in-the-middle) буде важче маскуватися під репозиторій.
4. Захищені дзеркала. Делегування контролю над дзеркалом для розповсюдження має відбуватися з особливою обережністю. Це допоможе запобігти більшості атак, оскільки зловмиснику буде важче отримати можливість видавати себе за сховище.
5. Підписувати метадані та пакети. Підписання як метаданих, так і пакетів ускладнює зловмиснику запуск більшості типів атак.

6. Перевіряти правильність метаданих. Після того, як менеджер пакетів вирішив встановити пакет, він повинен завантажити пакет і перевірити його підпис і те, що метадані збігаються з метаданими, наданими репозиторієм. Це допоможе запобігти атакам сторонніх залежностей коли зловмисник не може правильно підписати пакет.

Ці заходи посилять труднощі при здійсненні багатьох видів атак. Проте в архітектурах більшості пакетних менеджерів немає способу виправити відкликання ключа, ескалації привілеїв, залежності на усіх пакетах, а також атаки незадовільних залежностей [17, 19]. Це означає, що архітектура безпеки більшості менеджерів пакетів, наприклад, APT та YUM, принципово не підходять для вирішення цих проблем.

З цією метою, спробуємо побудувати поверхнево архітектуру власного пакетного менеджера та розробити три принципи проєктування безпеки в управлінні пакетами. Це наступні принципи – вибіркове делегування довіри, налаштування перегляду вмісту репозиторію та базовому відношенню до репозиторію як до не довіреного.

Вибіркове делегування довіри

Вибіркове делегування довіри дозволяє користувачеві довіряти підпису іншого користувача лише для певного переліку пакетів. Це засіб, за допомогою якого користувач може запобігти атаки ескалації привілеїв, делегуючи мінімальну необхідну кількість довіри такому ж користувачу. Це ілюструє використання ієрархічної моделі, коли кінцевий користувач може довіряти керівнику проекту або розповсюджувачу пакетів, який, у свою чергу, довіряє окремим розробникам. Це принципово відрізняється від того, щоб розповсюджувач підписав пакет просто тому, що його підписав розробник. Розповсюджувач ніколи не підписує пакет, а натомість сигналізує про довіру до окремого ключа розробника. Це забезпечує природний механізм відкликання

ключа, а також усуває необхідність відкликання ключа, щоб визначити пакет як скомпрометований.

Наявність налаштованих механізмів перегляду репозиторію дає змогу кожному користувачеві «бачити» інший репозиторій, який насправді є об’єднанням метаданих пакетів у всіх репозиторіях які використовуються. Проте включаються лише метадані пакета, яким довіряє сам користувач. Метадані пакетів від ненадійних користувачів не можуть впливати на безпеку менеджера пакетів. Це дозволяє адміністраторам репозиторію дозволяти користувачам вільно додавати власні пакунки без шкоди для безпеки інших користувачів. Тоді адміністраторам репозиторія потрібно турбуватися лише про традиційні проблеми для серверів, наприклад, запобігання зловмисникам отримати root доступ в репозиторії та використання дискового простору окремих користувачів, замість того, щоб турбуватися про дійсність вмісту пакетів або метаданих пакетів.

Вибіркове делегування означає, що користувачі можуть делегувати довіреність щодо встановлення пакетів іншим користувачам дуже різними способами. Як простий приклад, якщо Аліса знає, що Боб підтримує пакет foo, Аліса може заявити, що вона буде довіряти пакетам foo, підписаним Бобом. Це означає, що Аліса не буде довіряти підпису Боба пакета bar. Схематично ця логіка зображена на рисунку 3.1.

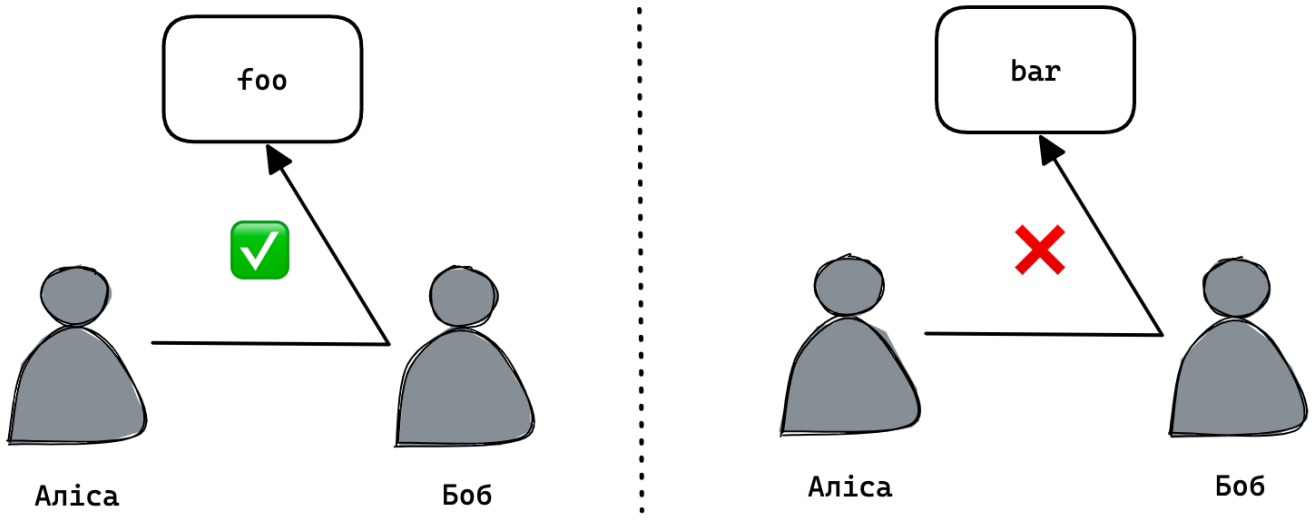

Рисунок 3.1 – Приклад схеми довіри між користувачами

Прикладом того, як це може бути використано на практиці, є те, що проект foo може мати ключ, який використовується для керування проектом. Аліса може довіряти проекту foo, щоб знати, які пакети foo дійсні. Проект foo може довіряти поточним розробникам проекту, щоб знати, які пакети foo дійсні. Розробники можуть підписувати пакети foo своїм особистим секретним ключем.

На перший погляд, логіка довіри може видатися такою самою, як і коли є єдиний ключ для підписання у проекті, щоправда є кілька важливих відмінностей. Перш за все, ключ проекту використовується лише тоді, коли членство будь-кого з розробників в проекті змінюється, і тому його можна зберігати в автономному режимі. По-друге, жодному розробнику не потрібен доступ до приватного ключа проекту. Оскільки розробники приходять і йдуть, немає необхідності змінювати ключ проекту, який використовується для підписання пакетів. По-третє, відкликати ключ розробника так само легко, як і делегувати цей ключ проекту розробнику. Кінцеві користувачі навіть не повинні знати, що членство будь-якого з розробників у проекті змінилося. По-четверте, довіру до окремих пакетів можна також скасувати без відкликання ключа підпису.

Не кожному користувачеві потрібно приймати власні рішення щодо делегування довіри. Супроводжувачі дистрибутивів, швидше за все, вибірково делегують довіру проектам та репозиторіям, які далі делегують довіру розробникам. Вони вимагають, щоб усі їхні користувачі або налаштовували довіру до пакетів власноруч, або делегували їм довіру для пакетів якими керує дистрибутив.

Іншим прикладом корисності вибіркового делегування є позначення пакета як ненадійного. Нехай існує кілька груп, які відстежують уразливості програмного забезпечення. Як тільки вразливість виявляється, вони повідомляють користувачів, що уражені пакунки не заслуговують на довіру. Замість цього вони можуть попросити користувачів довіряти їм, щоб знати, які пакети відхилити. Користувачі можуть делегувати довіру цій групі, щоб знати,

які пакети є ненадійними. У результаті, комп’ютер користувача відмовиться встановлювати пакунки, позначені як такі, що мають уразливості безпеки, навіть якщо інші користувачі позначать їх як надійні.

Спорідненим принципом із вибірковим делегуванням є налаштований режим перегляду репозиторію. Налаштовуваний режим перегляду репозиторію означає що різні користувачі, які використовують той самий репозиторій, можуть бачити різні пакети всередині репозиторія. Наприклад, Аліса довіряє Бобу який знає про існування пакетів foo, але не довіряє Чарлі. Аліса матиме пакети foo Боба під час перегляду пакетів у репозиторії, але не буде мати пакетів foo Чарлі.

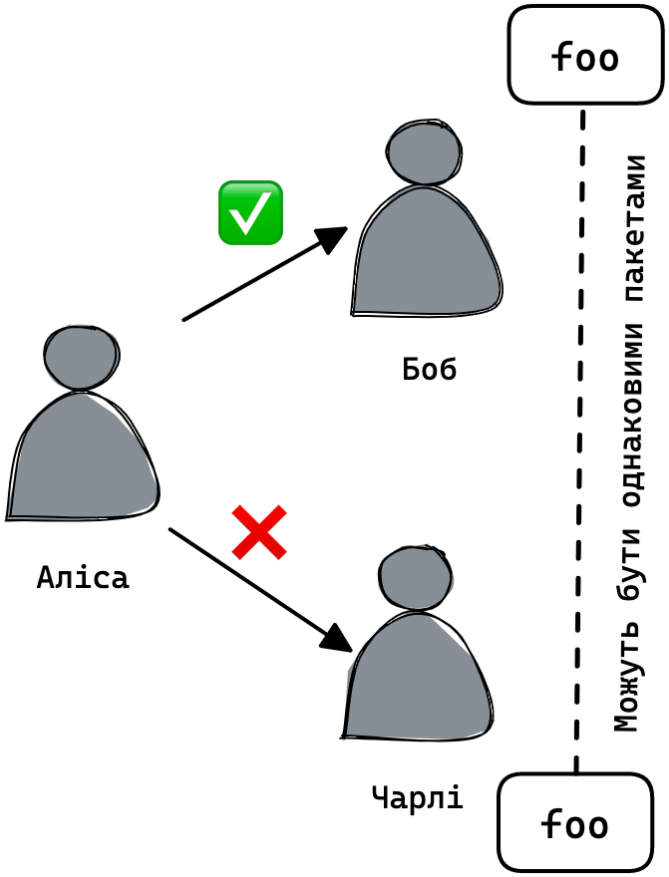

Рисунок 3.2 – Видимість пакетів при фільтрованій довірі

Налаштований режим перегляду репозиторію може поширюватися на кілька репозиторіїв одночасно. Користувач може отримати доступ до пакетів і метаданих на кількох репозиторіях. Ця інформація може бути об’єднана, щоб

утворити єдине, налаштоване представлення репозиторію лише з тими даними, яким довіряє користувач. Якщо деякі репозиторії не працюють або були зламані, користувач все одно зможе встановлювати пакети з надійних репозиторіїв та дзеркал.

Також, важливим принципом є одразу розглядати дані, що надходять із сховища, як недовірені. Це важливо, тому що вирішення залежностей виконується за допомогою метаданих пакета. Цю інформацію потрібно перевірити, щоб шкідливо модифіковані метадані не могли змінити поведінку вирішення залежностей пакетів. Крім того, клієнти отримують дані з репозиторіїв для встановлення оновлень пакетів, включаючи оновлення безпеки. Якщо клієнт використовує кілька репозиторіїв, один із репозиторіїв не може перешкодити клієнту встановлювати пакунки з інших.

3.2 Впровадження базової архітектури безпеки

Для імплементації та впровадження базової архітектури безпечного менеджера пакетів, розробимо систему управління пакетами, яка має бути призначена для усунення основних недоліків, що присутні у найбільш популярних менеджерах пакетів, таких як APT, YUM та NPM [6, 7, 8, 11].

Розглядаючи існуючі менеджери пакетів – деякі практики було взято за основу у менеджера пакетів Stork [19]. Він має кілька переваг, які не пов’язані з безпекою, порівняно з існуючими системами керування пакетами. Він забезпечує безпечний та ефективний спільний доступ до пакетів всередині віртуальної машини на одній фізичній машині [5, 15, 19]. Він також забезпечує централізоване керування пакетами, що дозволяє користувачам визначати, які пакети слід встановити на своїх клієнтах, не налаштовуючи кожного клієнта окремо. Це дозволяє кільком фізичним машинам ефективно завантажувати один і той же пакет і гарантує що оновлення пакетів поширюються на віртуальні машини своєчасно.

Довірені пакети

Цікаву архітектурну відмінність яку можна побачити у Stork – наявність нового типу файлу який містить у собі конфігурацію певної логіки щодо довіри пакетам. Новий тип файлу називається файлом довірених пакетів (trusted packages файлом або TP файлом). TP файли використовуються для надання вибіркового делегування довіри, а також для підтримки налаштованих режимів переглядів репозиторію.

Файл довірених пакетів користувача вказує, які пакети користувач вважає дійсними. TP файл не викликає встановлення цих пакетів, а натомість вказує на довіру, що пакети мають дійсний вміст і є кандидатами на встановлення. Для розповсюджувачів пакета є нормальним наявність кількох версій одного і того ж пакета, зазначених у своєму TP-файлі, щоб користувачі могли інсталювати старіші (можливо, більш стабільні) версії пакета.

TP файл дозволяє користувачеві делегувати довіру та вказувати окремі файли пакетів, яким він довіряє. Щоб довіряти пакету, ім’я пакета та хеш метаданих пакета додаються до файлу довірених пакетів. Це не те саме, що додати хеш пакета до файлу довірених пакетів, оскільки метадані тепер можна перевіряти незалежно від самого пакета. Оскільки метадані пакета містять безпечний хеш пакета, пакет все ще захищений від підробки.

Щоб делегувати довіру певному користувачеві, вказуються ім’я користувача та відкритий ключ разом із пакетами та залежностями, які їм дозволено надавати. Ця інформація додається до файлу довірених пакетів. Користувачі можуть знати, які пакети що відповідають вимогам можна встановити (за допомогою ALLOW), які пакети не встановлювати (за допомогою DENY) або використовувати дві поведінки одразу (за допомогою ANY).

Рядки у файлі довірених пакетів обробляються в тому порядку, в якому вони відображаються у файлі. Перше правило, якому відповідає пакет, класифікує пакунок як доступний для встановлення (дозволено) або вилучає

його з розгляду (відмовлено). Будь-які пакунки, які не мають відповідності, автоматично відхиляються (в кінці є неявний <FILE PATTERN="*" ACTION="DENY"/>). Таким чином, порядок рядків у файлі TP має значення при визначенні того, які пакети можуть бути кандидатами для встановлення.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<!— Початок опису —>
<TRUSTEDPACKAGES>

<!— Відхиляти усі пакети яким не довіряє юзер ALICE —>
<USER PATTERN="*" USERNAME="ALICE" PUBLICKEY="MFw ... AQ" ACTION="DENY"/>

<!— Визначення яким певним пакетам довіряти для встановлення —>
<FILE PATTERN="emacs-2.2-5.i386.rpm" HASH="aed49599 ... " ACTION="ALLOW"/>
<FILE PATTERN="foobar-1.01.i386.rpm" HASH="16b6d223 ... " ACTION="ALLOW"/>
<FILE PATTERN="foobar-1.0.i386.rpm" HASH="3945fd42 ... " ACTION="ALLOW"/>
<FILE PATTERN="simple-1.0.tar.gz" HASH="2343485 ... " ACTION="ALLOW"/>

</TRUSTEDPACKAGES>
```

Рисунок 3.3 – Схематичний вигляд TP файлу

На рисунку 3.3 показано TP файл без обгортки підпису. Цей TP файл відхиляє будь-які пакунки, які відхиляє користувач ALICE. Зокрема, він дозволяє встановлювати emacs-2.2-5.i386.rpm, декілька версій foobar і customapp-1.0.tar.gz, якщо їхні метадані збігаються із зазначеним захищеним хешем.

Обгортки підписів

Обгортки підписів використовуються для підтримки налаштованих режимів перегляду репозиторію. Обгортки підписів захищають TP файли і кореневі метадані репозиторію від підробки, повторного відтворення старих файлів і ситуації, коли будь-яка сторона постійно повертає одну й ту саму версію файлу. Обгортки підпису містять позначку часу, час закінчення дії, підпис і хеш відкритого ключа, який використовувався для створення підпису. Позначка часу запобігає розгляду старих версій файлів порівняно з новими

версіями. Час закінчення терміну дії зупиняє використання старих файлів на невизначений термін. Підпис захищає файл від підробки.

Позначка часу вказує, коли файл був створений. За замовчуванням мітка часу генерується з часу в секундах. Клієнти відстежують останню версію кожного файлу, який вони бачили, і ніколи не приймуть старішу версію, таким чином запобігаючи відтворенню старих файлів.

Файли також можна створювати з від'ємною міткою часу. Якщо файл має негативну позначку часу, це означає, що ключ підпису слід вважати недійсним. Будь-який клієнт із файлом, який має негативну позначку часу, відхилить будь-який файл, підписаний тим самим ключем, завантажений після цього, незалежно від часової позначки. Це ефективний спосіб для того щоб вказати, що ключ був скомпрометований і що підпису ключа не слід довіряти.

Обгортка підпису також має термін придатності. Це перевіряється з часом у локальній системі, де використовується файл. Якщо поточна кількість секунд більше, ніж час закінчення, то файл вважається недійсним.

Порядок порівняння позначки часу та часу закінчення може бути доречним. Наприклад, припустимо, що є два файли: старий файл, термін дії якого минув у майбутньому, і новий файл, термін дії якого вже минув. Якщо спочатку порівняти термін дії, то буде використано старий файл, оскільки термін дії нового файлу закінчився. Якщо порівняння закінчення терміну дії виконується після порівняння позначки часу, жоден файл не є дійсним, оскільки остання версія має час закінчення терміну дії в минулому.

Механізм закінчення терміну дії вимагає, щоб клієнти мали синхронізовані годинники. Клієнт може вирішити перевірити підпис і зберегти впорядкування позначок часу, але вимкнути перевірку закінчення терміну дії. Це дозволяє користувачам обміняти зручність на безпеку (оскільки немає захисту від зловмисника, який постійно повертає ту саму версію підписаного файлу).

Підпис використовується для захисту файлу від підробки. Він охоплює час закінчення терміну дії та позначку часу, а також вбудований вміст.

Комунікація з репозиторіями

Замість того, щоб просто підключатися до репозиторію для завантаження файлів і чекати завершення завантаження, необхідно також відстежувати з’єднання. Якщо з’єднання не передає дані вище встановленої користувачем мінімальної швидкості, репозиторій вважається мертвим, і з’єднання розривається. Також необхідно обмежувати обсяг даних, що завантажуються зі сховища. Якщо репозиторій відповідає з обсягом даних більше, ніж очікуваний, або більше ніж максимально настроєний користувачем, менеджер пакетів має розривати з’єднання.

Безпечний репозиторій

Debian і Ubuntu використовують офіційне сховище, яке надає пакети з оновлення безпеки . Це запобігає запуску дзеркалом атаки повторного відтворення, оскільки менеджер пакетів використовуватиме останню версію пакета, яка буде доступна з безпечного репозиторію. Однак цей захист не поширюється на зловмисника, що знаходиться в середині, оскільки репозиторії не підтримують HTTPS.

І Debian, і Ubuntu використовують кілька дзеркал поряд із безпечним репозиторієм. У випадку, коли безпечний репозиторій не працює, зловмисник може використовувати дзеркало для обслуговування застарілого вмісту, який спочатку був із безпечного репозиторію, щоб здійснити атаку повторного відтворення.

Самосертифіковані імена шляхів

Дуже цікаву практику використовує менеджер пакетів Stork. Він використовує самосертифіковані імена шляхів для виявлення фальсифікацій [19]. Ці ідентифікатори присутні в URL-адресах або назвах файлів TP файлів, пакетів і метаданих пакетів. Це дозволяє об’єкту, такому як репозиторій, мати можливість перевірити дійсність файлу.

Пакет або метадані пакета зберігаються в сховищі за URL-адресою, що містить його захищений хеш. Усі пакети та метадані пакетів вважаються незмінними. Будь-який користувач може переконатися, що файл не був підроблений, перевіривши захищений хеш отриманих даних. Це дозволяє репозиторію перевірити, чи завантажені пакети та метадані пакета є дійсними. Це також використовується клієнтом для перевірки правильності інформації, отриманої з репозиторію.

Натомість TP файли мають відкриті ключі, вбудовані в їхні імена. Використовуючи вбудований відкритий ключ, репозиторій може перевірити, чи файл не змінений, перевіривши, що TP файл правильно підписаний і чи відкритий ключ у захищеному ідентифікаторі відповідає приватному ключу, який підписав файл.

Оскільки TP файли можуть бути змінені користувачем, можуть бути випадки, коли кілька справжніх копій TP файлу завантажуються до репозиторію. У цьому випадку репозиторій зберігає версію з останньою міткою часу. Якщо в будь-якому з файлів зазначено, що ключ має бути відкликаний (через від’ємну позначку часу або іншу конфігурацію), то файл із негативною міткою часу буде збережено.

3.3 Захист від вразливостей

Дотримуючись принципів безпечного управління пакетами, ми можемо пом’якшити вразливості, які наявні в найпопулярніших менеджерах пакетів. Розглянемо згадані вразливості та як ми можемо архітектурно спланувати менеджер пакетів щоб цих вразливостей уникнути [7, 13, 15].

Повільне завантаження

Моніторинг зв’язку з боку клієнта запобігає повільному зв’язку від «паузи» менеджера пакетів. Якщо з’єднання занадто повільне, передача припиняється, а сховище розглядається як непрацездатне.

Нескінченний потік даних

У деяких випадках клієнт може не знати правильний розмір даних, які він завантажує. Така ситуація можлива у двох випадках – коли зловмисник контролює репозиторій (і може довільно встановити розмір) або коли користувач отримує початкові кореневі метадані. Мають існувати та бути зазначеними максимальні розміри файлів для кожного типу завантажених файлів, а також максимальний розмір для репозиторію в цілому.

У випадку скомпрометованого репозиторію всі файли, крім кореневих метаданих, мають відомий розмір. Менеджер пакетів має перевіряти, що файли, які він завантажує, не перевищують цей розмір.

Відтворення старих метаданих

Оскільки кореневі метадані та TP файли мають обгортку підпису, зловмисник не може відтворити старіші метадані, ніж ті які бачить користувач. Крім того, час закінчення не дозволяє використовувати файли необмежено. Насправді метадані репозиторію будуть використовуватися лише протягом короткого періоду часу до закінчення терміну його дії. Аналогічно, TP файли користувача мають термін дії, визначені розробником, що не дозволяє їм використовуватися необмежений час.

Стороння залежність

Ця атака не може бути виконана зловмисником, який скомпрометує сховище, оскільки підпис розробника захищає метадані пакета. Ключова відмінність нашого побудованого клієнта від існуючих менеджерів пакетів також буде полягати в тому, що ключ репозиторію не захищає метадані пакета, за це відповідає ключ проекту або розробника.

Залежність від усіх пакетів

Подібно до сторонніх залежностей, ця атака не може бути виконана зловмисником, який скомпрометує репозиторій, оскільки підпис розробника захищає метадані пакета.

Незадовільні залежності

Подібно до попередньої атаки, її також неможливо виконати, оскільки підпис розробника захищає метадані пакета. Компрометація репозиторію не передбачає можливості підробити метадані пакета.

Встановлення довільного пакета

Ця атака не може бути виконана зловмисником, який скомпрометує репозиторій, оскільки підпис розробника захищає метадані пакета. Також, інший підхід до цієї вразливості може полягати в тому, чи може зловмисник спричинити встановлення свого пакета, якщо його прямо не запитують. Менеджер пакетів має давати змогу користувачеві обмежувати встановлення та викачування, яке буде дозволено метаданими пакету, підписаними певним користувачем.

Використання відкликаних ключів

Делегування довіри за допомогою TP файлу забезпечує механізм скасування довіри. Якщо припустити, що ключі проекту використовуються лише для зміни списку розробників, яким довірено підписувати файли проекту, найбільш вірогідним шляхом для компрометування пакету є індивідуальний ключ розробника. Кожен з цих ключів може бути відкликаний окремо ключем проекту без залучення інших користувачів. Якщо ключ проекту компрометовано, можна створити файл довірених пакетів із негативною міткою часу, щоб користувачі не продовжували довіряти скомпрометованому ключу та пакетам які ним підписані.

Ескалація привілеїв

TP файли дозволяють делегувати іншим користувачам конфігурацію рішень щодо довіри та конфігурувати їх дуже детально. Це означає, що якщо користувач довіряє Apache щодо дійсності та актуальності пакетів apache, але ніколи не встановлює пакет apache, компрометація ключа Apache не загрожує безпеці користувача. Це усуває дилему адміністратора щодо того, чи додавати ключі, які вони можуть перевірити, коли пакет може не знадобитися. Проста та безпечна відповідь для користувачів полягає в тому, щоб довіряти всім ключам проекту, про які користувач знає, що є дійсними, і довіряти їм лише для пакетів, наданих цим проектом. Використовуючи вибіркове делегування довіри, тільки системи, які встановлюють пакети даного проекту, піддаються ризику, якщо ключ цього проекту буде скомпрометований.

Висновки за розділом 3

У цьому розділі було визначено принципи проектування та методи захисту які можливо реалізувати у безпечному менеджері пакетів. Було продемонстровано механізми безпеки, які можливо використати щоб запобігти атакам які було згадано у другому розділі. На жаль, багато недоліків, які роблять ці атаки життєздатними, мають архітектурний характер, тому їх неможливо легко виправити для існуючих менеджерів пакетів. У цьому розділі було представлено архітектуру безпеки, яка демонструє як уникнути цих вразливостей. Вже існуючий достатньо захищений менеджер пакетів Stork був побудований з використанням схожих принципів.

Крім менеджерів пакетів, існує безліч способів оновлення та встановлення програмного забезпечення на сучасних операційних системах. Існують системи оновлення програмного забезпечення, системи, що забезпечують автентичність та цілісність програмного забезпечення (включаючи SFS-RO, SUNDR, Deployme, та Self-Signed Executables), системи що встановлюють програмне забезпечення та системи сертифікації та підпису програмного коду. Ці системи не підтримують вибіркове делегування або захист метаданих пакетів. Вони призначені для випадків, коли кожен користувач знає організацію або дистрибутора, які мають розповсюджувати програмне забезпечення, і немає програмних залежностей – цілком малореальний випадок для багатьох сценаріїв.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 2021 CWE top 25 most dangerous software weaknesses. Common Weakness Enumeration. URL: https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html.
2. 7 crucial activities to test the security of mobile apps. TestingXperts. URL: https://www.testingxperts.com/blog/7-Crucial-Activities-to-Test-the-Security-of-your-Mobile-Applications.
3. Best practices for a secure software supply chain. Microsoft Docs. URL: https://docs.microsoft.com/en-us/nuget/concepts/security-best-practices.
4. Braun M. A confusing dependency. Autsoft Zrt. URL: https://blog.autsoft.hu/a-confusing-dependency/.
5. Containerizing npm and package managers for security - SuperGeekery. SuperGeekery. URL: https://supergeekery.com/blog/containerizing-npm-and-package-managers-for-security.
6. Decan A., Mens T., Constantinou E. On the impact of security vulnerabilities in the npm package dependency network. ICSE '18: 40th international conference on software engineering, м. Gothenburg Sweden. New York, NY, USA, 2018. URL: https://doi.org/10.1145/3196398.3196401.
7. Detecting suspicious package updates / K. Garrett та ін. 2019 IEEE/ACM 41st international conference on software engineering: new ideas and emerging results (ICSE-NIER), м. Montreal, QC, Canada, 25—31 трав. 2019 р. 2019. URL: https://doi.org/10.1109/icse-nier.2019.00012.
8. Detecting malicious campaigns with machine learning / M. Weber та ін. Unit42. URL: https://unit42.paloaltonetworks.com/unit42-detecting-malicious-campaigns-machine-learning/.
9. Fonseca J., Vieira M., Madeira H. Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks. IEEE Xplore. URL: https://ieeexplore.ieee.org/document/4459684.

10. Fuzzing. Open Source Foundation for Application Security. URL: https://owasp.org/www-community/Fuzzing.
11. Garrood H. Malicious code in the PureScript npm installer. URL: https://harry.garrood.me/blog/malicious-code-in-purescript-npm-installer/.
12. Hunting malicious npm packages. Decipher. URL: https://duo.com/decipher/hunting-malicious-npm-packages.
13. Measuring and preventing supply chain attacks on package managers / R. Duan та ін. URL: https://github.com/osssanitizer/maloss/tree/master/config.
14. OWASP top ten web application security risks. Open Source Foundation for Application Security. URL: https://owasp.org/www-project-top-ten/.
15. Patch-level verification for bundler. Rubysec. URL: https://github.com/rubysec/bundler-audit.
16. RPM package manager. rpm.org. URL: http://rpm.org/.
17. SecureApt. Debian Wiki. URL: https://wiki.debian.org/SecureApt.
18. Source code security analyzers. NIST. URL: https://www.nist.gov/itl/ssd/software-quality-group/source-code-security-analyzers.
19. Stork: package management for distributed VM environments. USENIX | The Advanced Computing Systems Association. URL: https://www.usenix.org/legacy/events/lisa07/tech/full_papers/cappos/cappos_html/index.html.
20. The ten most critical web application security risks. OWASP Top 10 - 2017. URL: https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_(en).pdf.pdf.
21. Top vulnerabilities and how to prevent them. Bright Security. URL: https://brightsec.com/blog/top-7-soap-api-vulnerabilities/.
22. Wang F., Shoshitaishvili Y. Angr - the next generation of binary analysis. 2017 IEEE Cybersecurity Development (SecDev), м. Cambridge, MA, USA, 24—26 верес. 2017 р. 2017. URL: https://doi.org/10.1109/secdev.2017.14.

23. Web application scanners. NIST. URL: https://www.nist.gov/itl/ssd/software-quality-group/web-application-scanners.
24. Web application security fundamentals / J. Meier та ін. Microsoft Docs. URL: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff648636(v=pandp.10).
25. Web resource. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Web_resource.