УДК 004.056.5

Владислав РУЩАК, Степан ІВАСЬЄВ

Західноукраїнський національний університет

ДОСЛІДЖЕННЯ ВРАЗЛИВОСТІ БІБЛІОТЕКИ CLICKBAR/DOT-DIVER

Вступ. Бібліотека @clickbar/dot-diver, реалізована на TypeScript (відкритий вихідний код), надає зручний API для зчитування значення поля об'єкта (функція getByPath) та запису значення у поле об'єкта (функція setByPath), має широке застосування. В бібліотеці було виявлено вразливість типу prototype pollution, що критично для безпеки вебзастосунків на TypeScript, що її використовують.

Мета: дослідити шляхи реалізації вразливості prototype pollution бібліотеки dot-diver.

1. Аналіз вразливості функції `setByPath`

Уразливість виявлено у функції setByPath, яка приймає три аргументи:

- object - об'єкт, у властивість якого встановлюється значення;
- path - шаблон шляху до властивості, що підлягає зміні;
- value - значення, яке має бути записане у вказане поле.

При виклику setByPath відбувається рекурсивний обхід елементів у шаблоні шляху. Після знаходження цільового поля йому присвоюється нове значення. Нижче зазначено, що у вихідному повідомленні наводилося загальне пояснення механіки роботи setByPath.

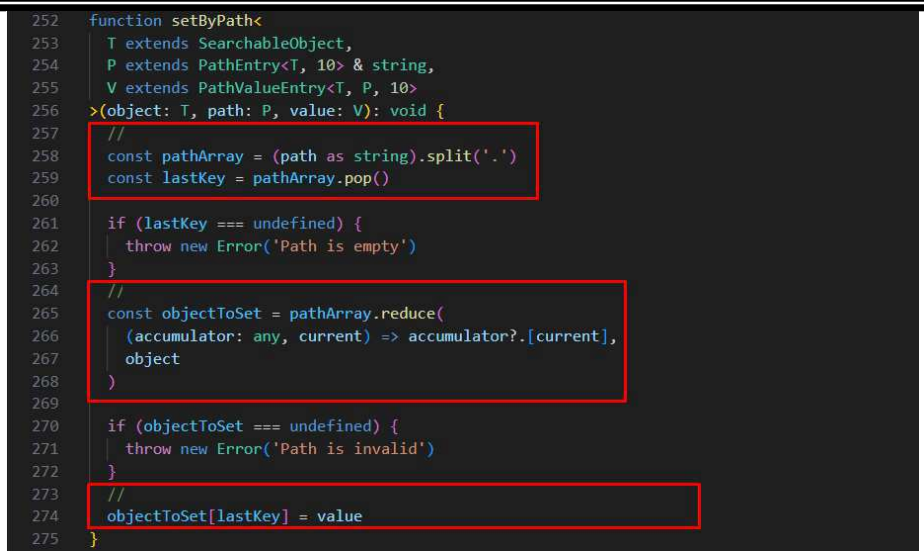
```
1  import { getByPath, setByPath } from '@clickbar/dot-diver'
2
3  // Define a sample object with nested properties
4  const object = {
5    a: 'hello',
6    b: {
7      c: 42,
8      d: {
9        e: 'world',
10     },
11   },
12   f: [{ g: 'array-item-1' }, { g: 'array-item-2' }],
13 }
14 // Example 2: Set a value by path
15 setByPath(object, 'a', 'new hello')
16 console.log(object.a) // Output: 'new hello'
17
18 setByPath(object, 'f.1.g', 'new array-item-2')
19 console.log(object.f[1].g) // Output: 'new array-item-2'
```

Рисунок 1 - Механізм роботи setByPath

Основна проблема застосування цієї функції полягає в тому, що якщо значення шляху містить посилання на прототип, з'являється можливість встановити властивості прототипу, що може призвести до забруднення глобального прототипу.

Цей же код після виправлення (версія 1.0.2) приведено на рисунку 3.

У версії 1.0.1 функція setByPath дозволяє встановлювати властивості за шляхом, що подається у вигляді рядка (наприклад, "a.b.c").



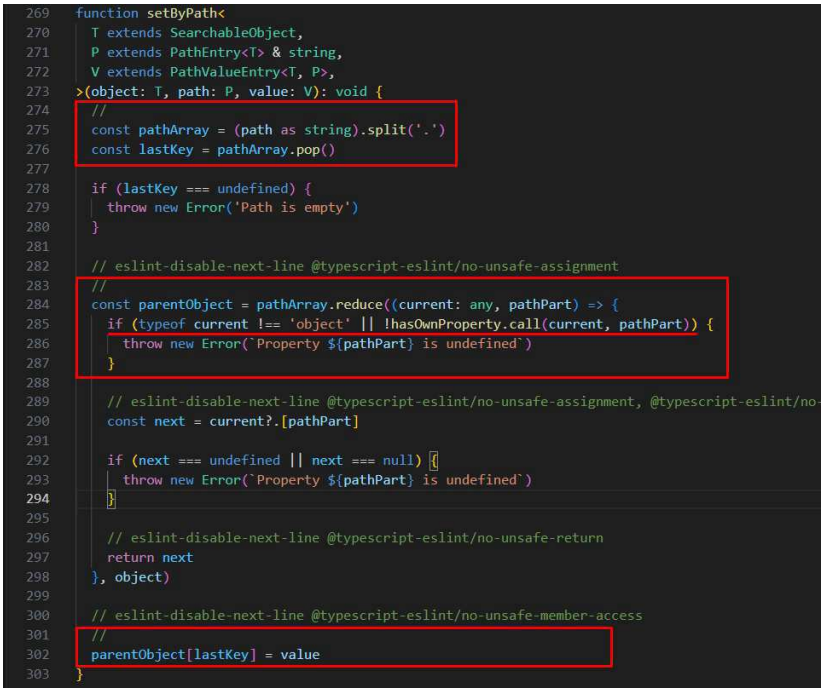
```
function setByPath<
  T extends SearchableObject,
  P extends PathEntry<T, 10> & string,
  V extends PathValueEntry<T, P, 10>
>(object: T, path: P, value: V): void {
  //
  const pathArray = (path as string).split('.')
  const lastKey = pathArray.pop()

  if (lastKey === undefined) {
    throw new Error('Path is empty')
  }
  //
  const objectToSet = pathArray.reduce(
    (accumulator: any, current) => accumulator?.[current],
    object
  )

  if (objectToSet === undefined) {
    throw new Error('Path is invalid')
  }
  //
  objectToSet[lastKey] = value
}
```

Рисунок 2 - Код функції до виправлення (версія 1.0.1)

Через відсутність перевірок спеціальних ключів (наприклад, __proto__, constructor.prototype) зловмисник може змінювати властивості прототипу - тобто відбувається prototype pollution. Це відкриває шлях до обходу механізмів контролю доступу й іншої небажаної модифікації поведінки застосунку. setByPath виконує рекурсивний або ітеративний обхід шляхового шаблона (split('.') → reduce/loop) і без додаткових перевірок записує значення у знайдене поле.

2. Аналіз виправлень вразливості

Проблемою що викликала вразливість є відсутність захисту проти модифікації властивостей прототипу: не відкидаються ключі на зразок __proto__, prototype, constructor та відсутній код, щоб перевірити, чи властивість є власною (own property) об'єкта, або чи операція створює нове поле на самому об'єкті, а не змінює глобальний прототип.



```
function setByPath<
  T extends SearchableObject,
  P extends PathEntry<T> & string,
  V extends PathValueEntry<T, P>,
>(object: T, path: P, value: V): void {
  //
  const pathArray = (path as string).split('.')
  const lastKey = pathArray.pop()

  if (lastKey === undefined) {
    throw new Error('Path is empty')
  }

  // eslint-disable-next-line @typescript-eslint/no-unsafe-assignment
  //
  const parentObject = pathArray.reduce((current: any, pathPart) => {
    if (typeof current !== 'object' || !hasOwnProperty.call(current, pathPart)) {
      throw new Error(`Property ${pathPart} is undefined`)
    }

    // eslint-disable-next-line @typescript-eslint/no-unsafe-assignment, @typescript-eslint/no-
    const next = current?.[pathPart]

    if (next === undefined || next === null) {
      throw new Error(`Property ${pathPart} is undefined`)
    }

    // eslint-disable-next-line @typescript-eslint/no-unsafe-return
    return next
  }, object)

  // eslint-disable-next-line @typescript-eslint/no-unsafe-member-access
  //
  parentObject[lastKey] = value
}
```

Рисунок 3 - Код функції після виправлення(версія 1.0.2)

У виправленій версії було додано перевірку наявності власної властивості в об'єкті (за допомогою методу Object.prototype.hasOwnProperty) перед внесенням змін. Якщо потрібне поле відсутнє, викликається виняток із відповідним повідомленням.

3. Вектор атаки забруднення прототипу

Вектор атаки можна описати наступними кроками. Зловмисник подає шлях, який містить __proto__ або інший спеціальний сегмент (через користувацькі поля/JSON). setByPath проходить шлях і в кінці записує задане значення не в локальний об'єкт, а в прототип (Object.prototype або інший спільний прототип).

Після цього будь-який інший код, що покладається на наявність певної властивості або перевіряє її присутність через спадкування, може побачити змінену поведінку (наприклад, user.isAdmin === true), що призводить до ескалації привілеїв або обходу авторизації. Алгоритм реалізації вразливості приведено на рисунку 4.

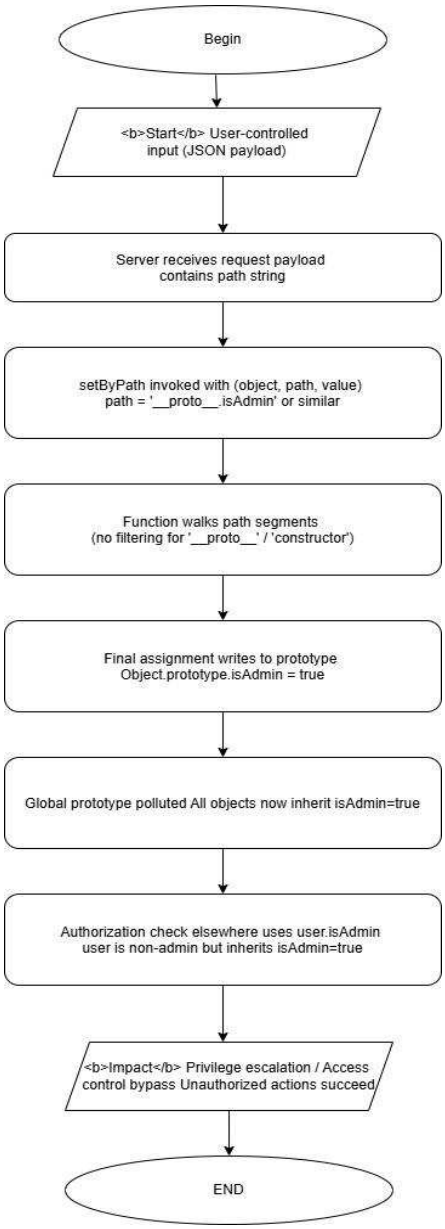


Рисунок 4 – Схема алгоритму використання вразливості setByPath

У прикладі розглядається застосунок для обліку прочитаних книг, у якому реалізовано механізм розмежування користувачів із різними ролями: user - має право додавати дані про прочитані книги та переглядати їх, admin - має право видаляти книги зі списку.

Розмежування доступу реалізовано за допомогою додаткової властивості `isAdmin`, яка присутня лише в об'єкта користувача з успішною автентифікацією для ролі admin. В інших користувачів це поле відсутнє, як показано на рисунку 5.

Рисунок 5 - Механізм роботи setByPath

Фрагмент коду, який виконує обробку запиту на видалення книги за ключем title та реалізує контроль прав доступу, може виглядати так, як показано на рисунку 6.

```
app.delete('/books/:title', (req, res) => {
  const { title } = req.params;
  const user = req.user; // об'єкт користувача, отриманий після автентифікації

  // Перевірка прав доступу
  if (!user || !user.isAdmin) {
    return res.status(403).json({
      error: 'Access denied: insufficient privileges.'
    });
  }

  // Пошук книги за назвою
  const index = books.findIndex((book) => book.title === title);

  if (index === -1) {
    return res.status(404).json({
      error: `Book with title '${title}' not found.`
    });
  }

  // Видалення книги
  books.splice(index, 1);
  return res.status(200).json({
    message: `Book '${title}' was successfully deleted.`
  });
});
```

Рисунок 6 - Фрагмент коду, який виконує обробку запиту на видалення книги

У застосунку використовуються такі API-команди, як отримати список книг:

$ curl -v -X GET -H http://192.169.27.1:6000/

Оновити список книг:

$ curl -v -X PUT -H "Content-Type:application/json" --data '{"auth":{"name":"reader", "pwd":"books"},"title":"Tom Sawyer"}' http://192.169.27.1:6000/

Видалити книгу з певною назвою:

$ curl -v -X DELETE -H "Content-Type:application/json" --data '{"auth":{"name":"reader", "pwd":"books"},"title":"Tom Sawyer"}' http://192.169.27.1:6000/

Після спроби видалення книги від імені користувача reader повертається відповідь із кодом 403 та повідомленням "Access denied", що свідчить про відсутність прав на виконання цієї операції.

До запиту додається корисне навантаження для атаки:

$ curl -v -X PUT -H "Content-Type:application/json" --data '{"auth":{"name":"reader", "pwd":"books"},"title":"Tom Sawyer", "note":"__proto__.isAdmin", "text":true}' http://192.169.27.1:6000/

У відповіді на цей запит відображається результат успішного оновлення списку книг. Окрім того, у глобальному об'єкті Object з'явилося поле isAdmin, як показано на рисунку 7.

Рисунок 7 - Object з полем isAdmin

Після повторного запиту на видалення книги від імені користувача reader у відповіді повертається повідомлення про успішне видалення книги, що свідчить про реалізацію атаки та обхід механізму розмежування доступу, реалізованого в застосунку.

Висновок. Необхідно забезпечити регулярне оновлення всіх бібліотек та залежностей до останніх стабільних версій. Більшість випадків «забруднення прототипу» виникає через застарілі пакети, що містять уразливі функції для глибокого копіювання чи об'єднання об'єктів (наприклад, у lodash, jQuery, dot-prop, deepmerge). Важливу роль відіграє також ретельна перевірка та фільтрація вхідних даних, які надходять від користувачів або зовнішніх API. Усі параметри, що використовуються для побудови об'єктів або динамічних шляхів властивостей, мають проходити валідацію.

Перелік використаних джерел.

1. GitHub Security Advisory. Prototype Pollution (PP) vulnerability in setByPath (GHSA-9w5f-mw3p-pj47). [Електронний ресурс]. – Режим доступу: // GitHub – clickbar/dot-diver. – Опубл.: 02.11.2023.

2. National Vulnerability Database (NVD). CVE-2023-45827: dot-diver – Prototype Pollution у setByPath; виправлення у релізі 1.0.2 Опубл.: 06.11.2023. [Електронний ресурс]. –Режим доступу: https://nvd.nist.gov/vuln/detail/CVE-2023-45827.