УДК 004.056

С. В. ХРУЩАК, О. М. ТКАЧЕНКО, О. Р. БОЙКО, О. О. КОШМЕЛЮК

АНАЛІЗ ВИКОРИСТАННЯ ЙМОВІРНІСНИХ ФІЛЬТРІВ ДЛЯ ІНВАЛІДАЦІЇ ТОКЕНІВ АВТЕНТИФІКАЦІЇ У РОЗПОДІЛЕНИХ СИСТЕМАХ

Вінницький національний аграрний університет, 21012, вул. Сонячна, 3, м. Вінниця, Україна
e-mail: sergey.khruschak@gmail.com, Вінницький національний технічний університет,
ТОВ "Він Інтерактив", Вінниця, Україна

Анотація. У статті розглянуто проблему централізованої автентифікації користувачів у складних розподілених системах з використанням криптографічних токенів на основі JWT (JSON Web Token). Такі системи дозволяють зменшити час обробки запитів, порівняно зі звичайними централізованими системами автентифікації за рахунок можливості оффлайн перевірки токенів доступу. Однак це ж і створює проблеми з їх інвалідацією у разі коли токен скомпроментовано чи заблоковано. Традиційних підхід, що застосовується в таких протоколах, як OAuth2, перекладає цю проблему на сторону клієнтських додатків, що робить API складнішим для використання. У статті розглянуто використання підходу, що дозволяє виконувати всі необхідні перевірки токенів на стороні сервера без внесення значних змін у саму систему з використанням списків блокування. Запропоновано використання ймовірнісних фільтрів для передачі оновлень про заблоковані токени. Такі фільтри дозволяють з певною точністю перевірити входження елементу в наборі, використовуючи значно менше пам'яті ніж було б необхідно для зберігання всього набору елементів. Їх зазвичай використовують для уникнення повільних операцій, таких як доступ до диску чи мережі. В результаті це значно зменшує використання пам'яті на кінцевих сервісах та обсяги трафіку між компонентами системи. Обрано критерії оцінювання роботи ймовірнісних фільтрів відповідно до задачі періодичного оновлення списків заблокованих ідентифікаторів токенів доступу та проведено аналіз методів різних реалізацій ймовірнісних фільтрів. Відповідно до критеріїв надано рекомендації з застосування конкретних реалізацій ймовірнісних фільтрів та їх параметрів для розподілених систем різного розміру.

Ключові слова: автентифікація, авторизація, інформаційні системи, розподілені системи, OpenID, OAuth2, JWT, ймовірнісні фільтри.

Abstract. The article investigates the problem of centralized user authentication in complex distributed systems using cryptographic tokens based on JWT (JSON Web Token). Such systems allow decreasing request processing times comparable with conventional centralized authentication systems by allowing offline token verification. However, this creates problems with revoking of compromised or blocked tokens. The traditional approach used in such protocols as OAuth2, shifts this problem to the client side, complicating the client side and making the API more difficult to use. The article discusses the use of an approach that allows developers to keep all costs on the validation token validation on the server side without making significant changes to the system by blocklists. It is suggested to use probabilistic filters to transmit updates about blocked tokens. Such filters at the cost of losing some precision in checking if the entry belongs to the set of elements, using significantly less memory than would be necessary to store all the elements of the set. They are usually used to avoid slow operations such as disk or network access. As a result, it significantly reduces the memory usage on the services end and decreases the traffic volumes between the system components. The criteria for evaluating the performance of probabilistic filters were discussed for the task of periodically updating the lists of blocked identifiers of access tokens. Also various implementations of probabilistic filters were analyzed according to criteria. At the end recommendations for the application of specific probabilistic filters implementations and their parameters for distributed systems of various sizes are provided.

Key words: authentication, authorization, information systems, distributed systems, OpenID, OAuth2, JWT, probability filters.

DOI: 10.31649/1681-7893-2024-47-1-34-41

ВСТУП

Зі зростанням обсягів даних, які доводиться обробляти для вирішення багатьох сучасних задач стає необхідним використання складних розподілених сервісно-орієнтованих архітектур заснованих на взаємодії багатьох окремих компонентів. Зазвичай такі розподілені архітектури систем дозволяють ефективно управляти великими обсягами даних, забезпечують високопродуктивні обчислення на них та безперебійну роботу, навіть у разі збоїв окремих компонентів [1]. Завдяки можливості горизонтального масштабування та розподілу навантаження, розподілені архітектури дозволяють досягти високої продуктивності та надійності, що є необхідністю для більшості сучасних технологій та бізнес-процесів. Одним з важливих аспектів сучасних розподілених систем є процес автентифікації та авторизації користувачів. Складність його для розподілених систем полягає в тому, що система складається з багатьох індивідуальних сервісів, кожен з певним набором API, які відповідно можуть незалежно вимагати автентифікації та авторизації користувача, однак вимоги до безпеки зазвичай вимагають складних систем моніторингу та централізованого управління доступом, які дуже складно реалізувати для кожного сервісу окремо [2].

Метою статті є аналіз використання ймовірнісних фільтрів для відізвання токенів доступу при авторизації в розподілених системах та розробка рекомендацій щодо їх використання. Для досягнення мети необхідно вирішити такі задачі:

1. Вибрати критерії для оцінки реалізацій ймовірнісних фільтрів відповідно до задачі авторизації.
2. Провести експериментальні дослідження та вибрати оптимальні реалізації фільтрів для задачі.

ПІДХОДИ ДО АУТЕНТИФІКАЦІЇ В РОЗПОДІЛЕНИХ СИСТЕМАХ

Зазвичай в розподілених системах поширеним є відокремлення автентифікації від коду самих сервісів [3], що дозволяє вносити зміни в логіку автентифікації з мінімальним впливом на інші частини системи. Тому на практиці зазвичай застосовуються сервіси автентифікації, засновані на протоколі OAuth2 та OpenID Connect, як, наприклад, Keycloak, Auth0, Okta та інші. Такі сервіси дозволяють автентифікувати користувачів створювати та валідувати токени автентифікації та, якщо необхідно, проводити авторизацію дій користувачів.

Одним з перспективних підходів є використання в таких системах асиметричної криптографії при генерації токенів доступу. Це дозволяє проводити їх валідацію практично оффлайн за допомогою попередньо збереженого публічного ключа, тим самим дозволяючи значно зменшити навантаження на центральний сервер автентифікації та мінімізувати затримку при валідації запиту [3]. Схема такого підходу зображена на рис 1.

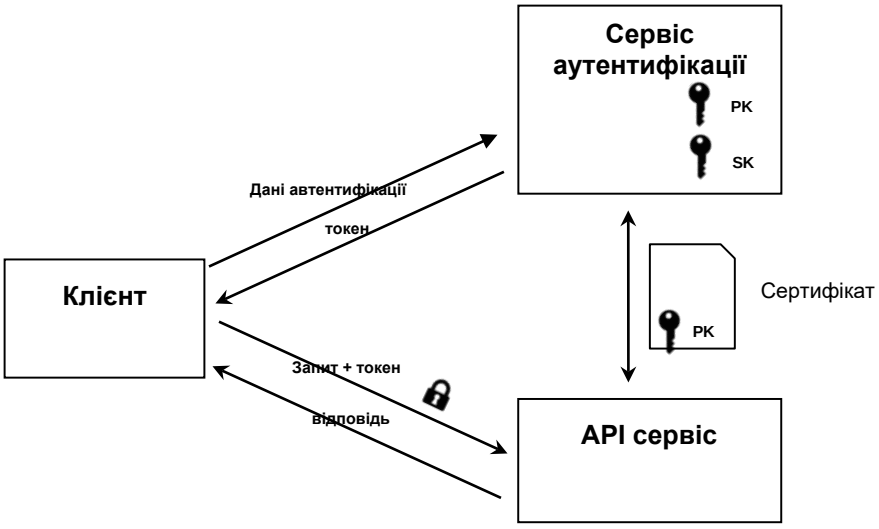

Рисунок 1 – Офлайн валідація токенів з використанням асиметричної криптографії

При такому підході під час логіну користувач отримує криптографічно підписаний токен доступу, зазвичай це JWT (JSON Web Token), який може містити переважну більшість необхідних даних про користувача та його дозволи та має обмежений термін дії. Токен підписується приватним ключем,

що зберігається на сервісі автентифікації, або ж у апаратному модулі HSM (Hardware Secure Module). Даний токен передається при кожному запиті клієнта та зазвичай є достатнім для того, щоб його автентифікувати та авторизувати на кінцевому сервісі. Для перевірки валідності токену, API сервіс може запитати сертифікат з публічним ключем у сервісу автентифікації та використовувати його для валідації всіх наступних запитів [4]. Таким чином сервіс авторизації задіяний тільки при генерації токенів та періодичному оновленні сертифікатів, що дозволяє краще масштабувати всю систему.

Однак значним недоліком такого підходу є те, що він не підтримує можливість відізвати існуючий токен. Наприклад, у випадках коли користувач виходить з додатку, його профіль було заблоковано, або ж конкретний токен скомпроментовано – він повинен бути заблокованим. Однак при зазначеному підході, токен буде валідним до закінчення його терміну дії, який може складати декілька днів, що в більшості випадків є неприпустимим з міркувань безпеки й, в результаті, змушує повернутись до онлайн перевірки токена на централізованому сервісі для кожного запиту, фактично нівелюючи переваги офлайн схеми.

Поширеним методом подолання даного недоліку, який пропонується в протоколі OAuth2, є зменшення часу життя токену доступу до мінімуму (зазвичай до кількох годин) та використання окремого виду токенів – токенів оновлення з більшим часом життя [5]. Обидва токени видаються сервісом автентифікації при логіні користувача, однак при запиті до сервісів використовується тільки токен доступу. За рахунок того, що токени доступу мають обмежений час дії, вони періодично вимагають від клієнта звертатись до сервіса автентифікації для їх оновлення з використанням токену оновлення. Це дозволяє не запитувати постійно логін та пароль у користувача і одночасно проводити періодичні централізовані перевірки прав доступу на стороні сервера. Діаграму послідовності для даного підходу наведено на рис. 2.

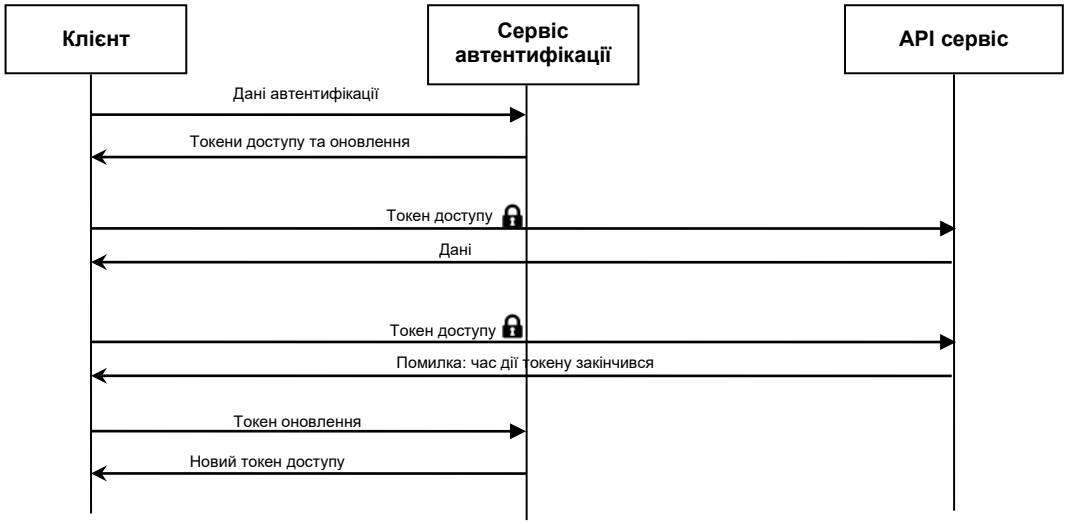

Рисунок 2 – Діаграма послідовності для автентифікації з використанням токенів оновлення

Недоліком цього підходу є те, що це перекладає проблему на клієнтські додатки, оскільки вони тепер змушені реалізовувати додаткову схему періодичного отримання токенів доступу та отримують додаткові вимоги на безпечне зберігання токенів оновлення. Також, в даному випадку, відізваний токен все ще працюватиме деякий час, хоч і значно менший.

ОФЛАЙН ВАЛІДАЦІЯ ТОКЕНІВ З ВИКОРИСТАННЯМ БЛОК-СПИСКІВ

В даній статті досліджується один з альтернативних методів покращення наведеної схеми з чорних списків заблокованих токенів, які періодично оновлюються з центрального сервісу, при цьому локальні копії списку зберігаються у пам’яті кожного API-сервісу [6]. Описану схему наведено на рис. 3.

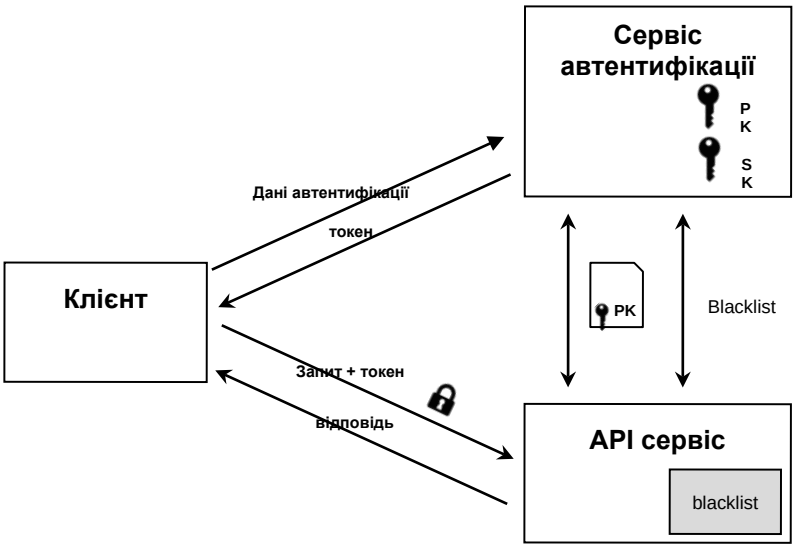

Рисунок 3 – Офлайн валідація токенів з використанням списку заблокованих токенів

Якщо токен не знайдено в кешованому списку, він вважається валідним, при цьому можлива ситуація, коли заблокований токен буде визнаний валідним, оскільки поширення списку блокування не є миттєвим, однак вона може бути виникати не так часто, як у попередніх підходах, оскільки всі оновлення відбуваються тільки на сервері. Перевагою даного підходу є те, спрощення логіки клієнтів, оскільки їм не потрібно проводити періодичне оновлення токенів доступу та більш швидке блокування відізваних токенів.

Для передачі списку заблокованих токенів можна скористатись ймовірнісними фільтрами. Такі фільтри використовуються для швидкої перевірки належності елементу до певного набору й дозволяють проводити перевірку валідності токена на стороні сервера використовуючи значно менше пам'яті ніж сам набір елементів. Ймовірнісні фільтри дозволяють однозначно встановити що елемент не входить в певний набір, однак можуть давати помилки першого роду – тобто, перевірка входження в набір може давати похибку й тому вимагає додаткової перевірки з реальним списком ідентифікаторів. В нашому випадку це передбачає відправку додаткового запиту на централізований сервер для перевірки чи дійсно токен є заблокованим.

Отже задачу дослідження можна звести до наступного: знайти структуру передачі заблокованих токенів, яка мінімізує кількість запитів для перевірки валідності токену до централізованої системи й при цьому займає найменше пам'яті та забезпечує швидку перевірку входження елементів та оновлення списку. В статті досліджено використання різних реалізацій ймовірнісних фільтрів в якості такої структури, їх переваги та недоліки при застосуванні для вирішення описаної проблеми, проведено порівняльний аналіз фільтрів між собою та з використанням звичайного списку.

АНАЛІЗ ЗАСТОСУВАННЯ РІЗНИХ РЕАЛІЗАЦІЙ ЙМОВІРНІСНИХ ФІЛЬТРІВ ДО ЗАДАЧІ АУНТЕНТИФІКАЦІЇ

Наразі використовується дві типи реалізації ймовірнісних фільтрів: Bloom-фільтри та варіанти зі збереження відбитків ключа. Алгоритмічно стандартні Bloom-фільтри представляють собою колекцію хеш-функцій h_1, h_2, …, h_n від ключа x, які використовуються, як індекси в бітовому масиві S. При додаванні ключа біти визначені хеш функціями встановлюються як 1: $S[h_1(x)] \leftarrow 1$, $S[h_2(x)] \leftarrow 1$, …, $S[h_n(x)] \leftarrow 1$. Для перевірки входження ключа в набір для нього знову обраховуються зазначені хеш функції та перевіряється чи встановлені відповідні біти в масиві [7]:

$$S[h_1(x)] \wedge S[h_2(x)] \wedge \dots S[h_n(x)] = 1 \quad (1)$$

Перевірка входження може давати хибно позитивні помилки, оскільки біти можуть бути вже встановленими попередніми ключами, однак в цьому випадку хибно негативні спрацювання – не можливі. При цьому змінюючи розмір бітового масиву S та кількість хеш функцій n можливо гарантувати різну ймовірність хибно позитивних похибок для заданої максимальної кількості елементів в

наборі. Кількість бітів *k,* необхідних для збереження одного елементу для Bloom-фільтрів визначається як:

$$k = 1.44 \cdot log_2\left(\frac{1}{\epsilon}\right) \quad (2)$$

Одним з обмежень Bloom-фільтрів є те, що вони не підтримують видалення елементів. Тому видалення токенів, у яких сплив термін дії вимагатиме перебудови всього фільтру на стороні сервісу авторизації. Для обходу цього обмеження існує також версія Bloom фільтрів з підрахунком кількості входжень, яка підтримує видалення елементів [8]. В ній замість бітового масиву використовується масив лічильників й при додаванні елементів замість простого встановлення бітів збільшуються значення відповідних лічильників. Однак, така реалізація однак вимагає в 3-4 рази більше пам'яті.

Ймовірнісні фільтри на основі відбитків ключа використовують одну хеш функцію h для перетворення ключа в фіксовану кількість бітів, які потім зберігаються та шукаються в певній структурі даних. Реалізації відрізняються між собою тим, яка хеш функція застосовується та які структури використовуються для зберігання результатів. Розглянемо більш детальніше деякі з реалізацій.

Cuckoo фільтри реалізовані як набір комірок, які зберігають відбитки ключів використовуючи компактну зозулину хеш таблицю [9] на основі двох хеш-функцій. За рахунок того, що в таблиці зберігаються тільки відбитки ключа – вона займає значно менше пам'яті ніж повний набір ключів, а сам бітовий розмір відбитку в бітах визначає ймовірність хибно позитивних похибок. Використання зозулиного хешування дозволяє отримати заповнення хеш таблиці до 95%, що дозволяє такій структурі займати менше пам'яті ніж Bloom-фільтри при значеннях ймовірності хибних спрацювань до 0.4%, а також підтримувати видалення елементів [10]. Однак при більших значення похибки, вони виявляються менш ефективними. Формулу, що задає кількість біт для опису одного елементу для Cuckoo-фільтрів наведено у виразі (3):

$$k = \left(log_2\left({}^{1}/_{\in}\right) + 3\right)/0.955 \quad (3)$$

XOR-фільтри дозволяють отримати ще більш ефективне представлення набору ключів, за рахунок того, що набір є незмінним, тобто операції модифікації набору не підтримуються. Такі фільтри використовують випадково обрану хеш-функцію для отримання відбитків ключів $F(x)$, які зберігаються в масиві S з розмірністю дещо більшою ніж повний набір ключів [11]. Для збереження елементу в наборі використовується три незалежні випадкові хеш функції h_1, h_2, h_3, які використовуються для представлення частини відбитку, таким чином, що:

$$S[h_1(x)] \oplus S[h_2(x)] \oplus \ldots \oplus S[h_n(x)] = F(x) \quad (4)$$

Такий підхід дозволяє отримати швидку перевірку на входження елементу в набір та дозволяє ефективно представляти набір в пам'яті вимагаючи *k* біт на один елемент, де *k* визначається як:

$$k = 1.23 \cdot log_2\left(\frac{1}{\epsilon}\right) \quad (5)$$

Отже кожен з наведених ймовірнісних фільтрів має як переваги так і певні обмеження, які можуть ускладнити їх використання для задачі авторизації. В той же час обсяг заблокованих токенів зазвичай не є дуже значним й майже завжди буде обмеженим десятками чи сотнями тисяч, навіть для досить великих систем, оскільки навіть при великій кількості користувачів не всі вони будуть онлайн, а також завдяки тому, що час використання токенів є обмеженим. В таких умовах незначні переваги в більш ефективному представленні даних можуть не давати практично значущої різниці.

ФОРМУЛЮВАННЯ КРИТЕРІЇВ ОЦІНКИ ТА ЕКСПЕРИМЕНТАЛЬНІ РЕЗУЛЬТАТИ

Для оцінювання ефективності використання кожного з фільтрів було обрано наступні критерії, в порядку зменшення їх значущості:

- кількість хибно позитивних спрацювань повинна бути якомога меншою, оскільки кожне таке спрацювання вимагає відправки запиту на сервіс авторизації, що створює навантаження на нього і значно збільшує час обробки запиту.
- Компактність збереження набору ідентифікаторів. Оскільки кожен сервіс повинен періодично оновлювати набір з центрального сервісу, менша кількість байт для його представлення допоможе зменшити обсяги трафіку, навантаження на центральний сервіс авторизації та вимоги до пам’яті на API-сервісах.
- Підтримка оновлень набору дозволить не перебудовувати список при додаванні нових чи видаленні застарілих записів і тим самим зменшити обчислювальне навантаження на сервісі авторизації.

Для оцінювання застосування фільтрів відповідно до критеріїв було проведено симуляцію системи авторизації, яка задається такими параметрами: кількість онлайн сесій в системі, відсоток блокувань токенів, середня довжина сесії користувача, середня кількість запитів за сесію та інтервал оновлення набору заблокованих користувачів. В якості ідентифікаторів токенів застосувались універсальні унікальні ідентифікатори UUID, які зазвичай застосовуються в більшості систем автентифікації. Оскільки основний вплив на застосування фільтрів має кількість онлайн сесій,результати симуляції саме для неї наведено на рисунках 4 та 5. Використовувався розмір сесії одна доба, з відсотком блокувань користувачів 25%.

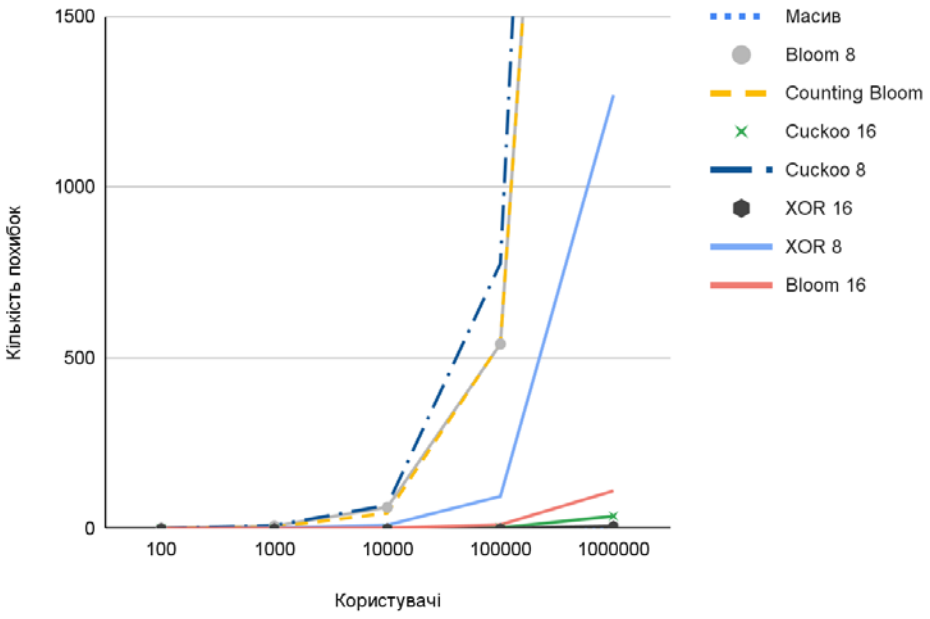

Рисунок 4 – Кількість хибно позитивних спрацювань для різної кількості онлайн сесій

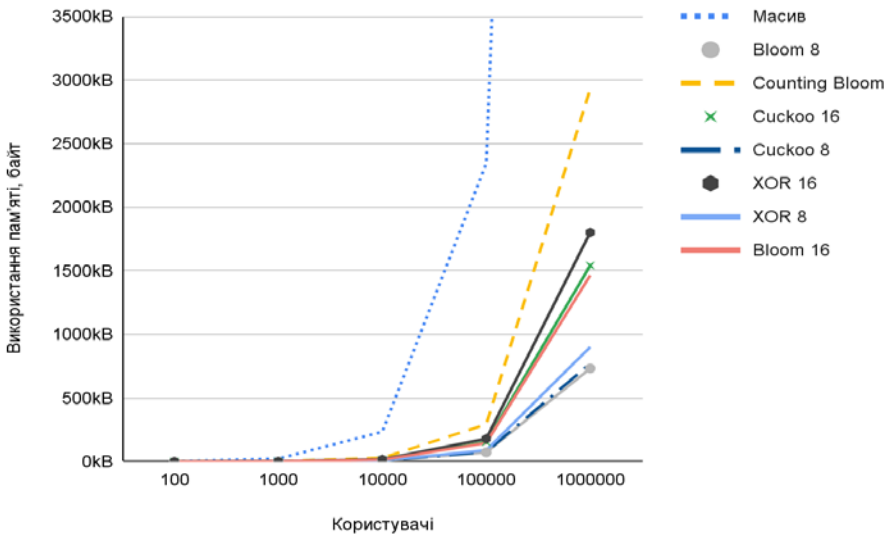

Рисунок 5 – Кількість байт необхідних для опису токенів для різної кількості користувачів

Отже результати симуляції показали що навіть при невеликій кількості блокувань використання ймовірнісних фільтрів є доцільним, оскільки дозволяє значно зменшити обсяги трафіку та отримати фактично нульову кількість хибних спрацювань. Використання фільтрів з більшою розмірністю є більш доцільним, оскільки дозволяє значно зменшити трафік до сервісу авторизації. Bloom фільтри для такої кількості даних потребують досить мало пам'яті, однак дають досить великі похибки. Також на практиці при кількості онлайн сесій до мільйона, Cuckoo та XOR-фільтри показують себе практично однаково за кількістю хибних спрацювань та використанні пам'яті. При цьому Cuckoo-фільтри мають перевагу за рахунок підтримки оновлень існуючої структури, тому більш підходять для вирішення задачі. Й відповідно переваги XOR-фільтрів найкраще себе проявляють в дуже великих системах з мільйонами одночасних сесій користувачів.

ВИСНОВКИ

Використання ймовірнісних фільтрів дозволяє у простий спосіб вирішити задачу відкликання токенів доступу в розподілених системах не перекладаючи проблему на сторону клієнтських додатків. Серед розглянутих ймовірнісних фільтрів для більшості систем найбільш підходять Cuckoo-фільтри, які забезпечують оптимальне співвідношення між щільністю представлення інформації, ймовірністю похибки та функціоналом. Для дуже великих систем доцільним є використання XOR-фільтрів.

Описане рішення може застосовуватись для побудови складних систем автентифікації та авторизації, які розраховані на високе навантаження та високу швидкодію. Подальшим напрямом досліджень є інтеграція описаного рішення у існуючі корпоративні системи автентифікації у вигляді плагінів.

СПИСОК ЛІТЕРАТУРИ

1. Hinrichs T.. Centralized vs. Distributed Authorization: the CAP theorem URL: https://www.styra.com/blog/centralized-vs-distributed-authorization-the-cap-theorem/
2. Neray G.. Best Practices for Authorization in Microservices. URL: https://www.osohq.com/post/microservices-authorization-patterns
3. Eknert A.. 4 Best Practices for Microservices Authorization. URL: https://thenewstack.io/microservices/4-best-practices-for-microservices-authorization/
4. RFC-7519: JSON Web token. URL: https://datatracker.ietf.org/doc/html/rfc7519
5. RFC-6749: The OAuth 2.0 Authorization Framework. URL: https://www.rfc-editor.org/rfc/rfc6749
6. KrakenD: Token Revocation. URL: https://www.krakend.io/docs/authorization/revoking-tokens/
7. Peter C. Dillinger and Panagiotis Manolios. 2004. Bloom Filters in Probabilistic Verification. In Formal Methods in Computer-Aided Design, *5th International Conference*, FMCAD 2004, Austin, Texas, USA,

November 15-17, 2004, Proceedings (Lecture Notes in Computer Science), Alan J. Hu and Andrew K. Martin (Eds.), Vol. 3312. Springer, 367-381. https://doi.org/10.1007/978-3-540-30494-4_26
8. Bonomi F., Mitzenmacher M., etc. An improved construction for counting bloom filters. In 14th Annual European Symposium on Algorithms, LNCS 4168, pages 684–695, 2006.
9. Paghand R., Rodler F.. Cuckoo hashing. Journal of Algorithms, 51(2): 122–144, May 2004.
10. Fan B., Andersen D. G., etc. Cuckoo Filter: Practically Better Than Bloom. *Carnegie Mellon University, Intel Labs, Harvard University.* URL:https://www.cs.cmu.edu/~binfan/papers/conext14_cuckoofilter.pdf
11. Graf T. M., Lemire D.. Xor Filters: Faster and Smaller Than Bloom and Cuckoo Filters. *Journal of Experimental Algorithmics 25 (1),* 2020. URL: https://arxiv.org/abs/1912.08258v3. DOI: https://doi.org/10.48550/arXiv.1912.08258.

REFERENCES

1. Hinrichs T.. Centralized vs. Distributed Authorization: the CAP theorem URL: https://www.styra.com/blog/centralized-vs-distributed-authorization-the-cap-theorem/
2. Neray G.. Best Practices for Authorization in Microservices. URL: https://www.osohq.com/post/microservices-authorization-patterns
3. Eknert A.. 4 Best Practices for Microservices Authorization. URL: https://thenewstack.io/microservices/4-best-practices-for-microservices-authorization/
4. RFC-7519: JSON Web token. URL: https://datatracker.ietf.org/doc/html/rfc7519
5. RFC-6749: The OAuth 2.0 Authorization Framework. URL: https://www.rfc-editor.org/rfc/rfc6749
6. KrakenD: Token Revocation. URL: https://www.krakend.io/docs/authorization/revoking-tokens/
7. Peter C. Dillinger and Panagiotis Manolios. 2004. Bloom Filters in Probabilistic Verification. In Formal Methods in Computer-Aided Design, *5th International Conference*, FMCAD 2004, Austin, Texas, USA, November 15-17, 2004, Proceedings (Lecture Notes in Computer Science), Alan J. Hu and Andrew K. Martin (Eds.), Vol. 3312. Springer, 367-381. https://doi.org/10.1007/978-3-540-30494-4_26
8. Bonomi F., Mitzenmacher M., etc. An improved construction for counting bloom filters. In 14th Annual European Symposium on Algorithms, LNCS 4168, pages 684–695, 2006.
9. Paghand R., Rodler F.. Cuckoo hashing. Journal of Algorithms, 51(2): 122–144, May 2004.
10. Fan B., Andersen D. G., etc. Cuckoo Filter: Practically Better Than Bloom. *Carnegie Mellon University, Intel Labs, Harvard University.* URL:https://www.cs.cmu.edu/~binfan/papers/conext14_cuckoofilter.pdf
11. Graf T. M., Lemire D.. Xor Filters: Faster and Smaller Than Bloom and Cuckoo Filters. *Journal of Experimental Algorithmics 25 (1),* 2020. URL: https://arxiv.org/abs/1912.08258v3. DOI: https://doi.org/10.48550/arXiv.1912.08258.

Надійшла до редакції 24.04.2024 р.

ХРУЩАК СЕРГІЙ ВІКТОРОВИЧ – канд. тех. наук, старший викладач кафедри комп'ютерних наук, Вінницький національний аграрний університет, Вінниця, Україна, ***e-mail: sergey.khruschak@gmail.com***

ТКАЧЕНКО ОЛЕКСАНДР МИКОЛАЙОВИЧ – канд. тех. наук, доцент кафедри програмного забезпечення, Вінницький національний технічний університет, Вінниця, Україна, ***e-mail: alextk1960@gmail.com***

БОЙКО ОЛЕКСІЙ РОМАНОВИЧ – канд. тех. наук, старший викладач кафедри комп'ютерних наук, Вінницький національний аграрний університет, Вінниця, Україна, ***e-mail: boyko.aleksey@gmail.com***

КОШМЕЛЮК ОЛЕГ ОЛЕКСАНДРОВИЧ – технічний керівник проектів в ТОВ "Він Інтерактив", Вінниця, Україна, ***e-mail: dealla@vntu.edu.ua***

S.V. KHRUSCHAK, O.M. TKACHENKO, O.R. BOYKO, O.O. KOSHMELYUK

ANALYSIS OF THE PROBABALISTIC FILTERS USAGE FOR AUTHENTICATION TOKENS INVALIDATION IN DISTRIBUTED SYSTEMS

Вінницький національний аграрний університет, Вінниця, Україна
Vinnytsia National Technical University
ТОВ "Він Інтерактив", Вінниця, Україна