УДК 004.49

DOI: 10.18372/2073-4751.71.17000

Олійник Я.О.,
orcid.org/0000-0002-1245-2704,
Мельниченко П.І.,
orcid.org/0000-0003-3746-1241

ОГЛЯД МЕТОДІВ ЗАХИСТУ WEB-ЗАСТОСУНКІВ ВІД ВРАЗЛИВОСТЕЙ ТИПУ CSRF (МІЖСАЙТОВА ПІДРОБКА ЗАПИТІВ)

Національний авіаційний університет

1304521@stud.nau.edu.ua

Вступ

На сьогоднішній день у сфері інформаційної безпеки спостерігається стійке зростання кількості комп'ютерних атак, які, у свою чергу, призводять до зниження рівня захищеності *Web*-застосунків. Основними причинами реалізації комп'ютерних атак зазвичай є наявність вразливостей у програмному забезпеченні, відсутність засобів захисту, а також недостатня захищеність взаємодії між клієнтом та сервером у *Web*-застосунках.

Враховуючи динамічний розвиток інформаційних систем та їхніх вразливостей, створення нових методів атак, сучасні засоби інформаційного захисту *Web*-застосунків зазвичай не можуть забезпечити необхідний рівень безпеки. Оскільки увага розробників *Web*-застосунків, у більшості випадків, зосереджена на реалізації методів та засоби захисту від атак типу *SQL/RCE*-ін'єкцій або міжсайтового скриптингу (*XSS*), дані факти дозволяють спрогнозувати найбільш популярні атаки на *Web*-застосунки, одним з основних є *CSRF*.

Аналіз літературних джерел

Великі компанії, наприклад, *Microsoft,* створюють перелік рекомендацій [1] для розробників щодо захисту від вразливостей *Web*-застосунків, але ці рекомендації не є повними. Авторами [2] розроблено автоматичний сканер вразливостей, але він працює лише з застосунками, що написані на мові *PHP*, та вихідний код яких є доступним для динамічного аналізу. Автори [3] проаналізували якість документації веб-фреймворків, та зазначили що 61,4 % не надають достатніх рекомендацій щодо захисту від *CSRF*. Загалом дослідження [3] показує, що хоча *Web*-фреймворки надають розробникам інструменти для захисту *CSRF,* їх правильна та безпечна реалізація часто залежить від обізнаності розробників і різноманітного досвіду щодо *CSRF*-атак, яким потрібно подолати відсутню та неповну документацію, а також небезпечні параметри *Web*-середовища за замовчуванням. Авторами [4] проведено порівняльний аналіз *JavaScript*-фреймворків без врахування дій розробників щодо захисту від *CSRF*-вразливостей. Автори [5] пропонують використовувати для захисту від *CSRF*-атак лише атрибут *SameSite*.

Проведений аналіз літературних джерел показав, що методи захисту від *CSRF*-атак не мають чіткого алгоритму дій для забезпечення повної безпеки інтерфейсу (*client-side*) *Web*-застосунків, а кількість вразливостей згідно Національної бази даних вразливостей Національного інституту стандартів і технологій (США) щороку зростає [6].

Мета

Метою статті є дослідження методів захисту *Web*-застосунків від вразливостей типу міжсайтова підробка запитів.

Аналіз вразливостей типу CSRF

CSRF (англ. *Cross Site Request Forgery* – Підробка міжсайтових запитів) – це атака, яка змушує кінцевого користувача виконувати небажані дії у *Web*-застосунку, в якому від на даний час автентифікований [7,8].

Відомо, що *CSRF* є видом атаки на користувачів *Web*-застосунків, при якому

браузер жертви, під час перегляду спеціально сформованої зловмисником сторінки, відправляє *HTTP*-запити на вразливу сторінку *Web*-застосунку, дані якого є ціллю атаки. Атаки *CSRF* спеціально націлені на запити, що змінюють стан, а не на викрадання даних (проте, важливо зазначити, що існують відносно нерозповсюджені *CSRF*-атаки, при яких кінцевою ціллю є саме викрадення даних, коли зміна стану дозволяє це реалізувати), оскільки зловмисник не має можливості переглянути відповідь на підроблений запит. За допомогою соціальної інженерії (наприклад, відправку посилання через електронну пошту, месенджер, соціальні мережі, тощо) зловмисник може примусити користувачів *Web*-застосунків виконати певні дії, що обрані зловмисником. Якщо жертва є звичайним користувачем, успішна *CSRF*-атака може змусити її виконати запити, наприклад, такі як передача коштів, зміна адреси електронної пошти тощо. Якщо жертвою є обліковий запис адміністратора, *CSRF* може скомпрометувати весь *Web*-застосунок. Тобто користувач повинен відвідати певну сторінку спеціально сформованого зловмисником *Web*-застосунку зі шкідливим кодом, після чого *HTTP*-запит на зміну стану буде відправлено автоматично, без відома жертви.

На даний час було розроблено багато недостатніх дій, направлених на захист від *CSRF*-атак, таких як: використання файлів *cookie* з атрибутом *secure* (необхідно зазначити, що цей атрибут, у даному контексті, забезпечує захист лише від сніфінгу (перехопленню) заголовків *HTTP*-запитів *Cookie* та *Set-Cookie*), приймання лише *POST*-запитів (*Web*-застосунки можуть бути розроблені так, щоб приймати лише *POST*-запити. Помилкове уявлення полягає в тому, що оскільки зловмисник не може створити шкідливе посилання, то атака *CSRF* не може бути реалізована. Це не вірне твердження, оскільки існує багато методів, за допомогою яких зловмисник зможе заставити користувача надіслати підроблений *POST*-запит, наприклад, проста форма із прихованими значеннями, що розміщена на сторінці *Web*-застосунку зловмисника. Ця форма може бути автоматично запущена за допомоги *JavaScript*, або може бути запущена самою жертвою), багатоетапні транзакції (поки зловмисник може передбачити або детектувати кожен крок завершеної транзакції, *CSRF* можливий), *HTTPS* (сам по собі *HTTPS* не захищає від *CSRF*-атак, однак його слід розглядати як передумову надійності будь-яких профілактичних заходів).

Алгоритм реалізації *CSRF*-атаки можна представити у наступному вигляді:

1. Користувач повинен бути авторизований у *Web*-застосунку (мати авторизаційну сесію);

2. Зловмисник досліджує *Web*-застосунок на наявність *CSRF*-вразливості;

3. У разі знаходження *CSRF*-вразливості зловмисник створює спеціальну сторінку, де розміщує сформований код *CSRF*-атаки;

4. Зловмисник за допомогою соціальної інженерії передає до користувача посилання на спеціально сформовану сторінку;

5. Користувач відвідує шкідливу сторінку;

6. *Web*-браузер, використовуючи авторизаційну сесію користувача, надсилає *HTTP*-запит (що містить команду на певну дію, яка необхідна зловмиснику) на сервер *Web*-застосунку.

Автори зауважують, що *Web*-застосунок повинен мати певні компенсуючі елементи управління функціональністю, що можуть попередити користувача про можливість *CSRF*-атаки. Якщо *Web*-застосунок приймає правильно сформований *HTTP*-запит, що відповідає певній бізнес-логіці застосунку, то можемо вважати, що *CSRF*-атака буде працювати.

Під час пошуку *CSRF*-вразливостей, необхідно визначити, чи додаються будь-які унікальні ідентифікатори до *HTTP*-запитів, які відправляє браузер автентифікованого користувача до *Web*-застосунку. Якщо немає унікального ідентифікатора, що стосується кожного *HTTP*-запиту, щоб прив'язати *HTTP*-запит до користувача, то *Web*-застосунок має вразливість. Одного

ідентифікатора сесії недостатньо, оскільки він буде надісланий у будь-якому випадку, якщо користувач натисне на шкідливе посилання, оскільки користувач вже автентифікований.

Розглянемо приклад *HTTP*-запиту з використанням методу *POST,* що умовно змінює дані (ім'я та прізвище) у персональному кабінеті користувача *Web*-застосунку.

POST /account/action/changename HTTP/2
Host: www.name.tld
accept-encoding: deflate, gzip
*accept: */**
content-type: application/x-www-form-urlencoded; charset=UTF-8
cookie:
user_session=V954qKt4GEzkB3hs
origin: https://www.name.tld
referer:
https://www.name.tls/profile/account
content-length: 22

first=Ivan&last=Ivanov

Виходячи з HTTP-запиту, припустимо, що єдиним методом перевірки достовірності отриманих даних та ідентифікації і автентифікації користувача є порівняння отриманих cookie «user_session» з сесією, що зберігається на сервері. Тобто ми маємо потенційну наявність CSRF-вразливості з усіма можливими наслідками. Для недопущення цієї проблеми авторами запропоновано використати цілий ряд додаткових методів захисту (як окремо, так і разом).

Методи захисту від атак типу CSRF

Беручи до уваги відсутність злагодженого алгоритму захисту *Web*-застосунків та їхню потенційну вразливість, автори пропонують наступні методи захисту від *CSRF*-атак. До них відноситься використання токенів. Токени (англ. *Token*) – спеціально сформований за певним алгоритмом приватний ключ. Сервер генерує випадковий унікальний токен для браузера користувача та перевіряє його для кожного запиту. Токени для кожного запиту є більш безпечними, ніж токени для кожної сесії, оскільки діапазон часу, протягом якого зловмисник може використати вкрадені токени, є мінімальним. Однак це може спричинити проблеми зі зручністю використання. Наприклад, якщо користувач *Web*-застосунку захоче використати кнопку браузера «*Назад*», то взаємодія з попередньою сторінкою призведе до хибно позитивної події захисту від *CSRF*-атак на сервері, оскільки попередня сторінка може містити токен, що вже не дійсний. В реалізації токена для кожної сесії, після початкового створення токена, значення зберігається у сесії та використовується для кожного наступного запиту користувача до закінчення сесії. Дану проблему можливо вирішити, якщо використовувати динамічну генерацію форми, при якому токен підставляється у момент надсилання форми.

Токени не потрібно передавати за допомогою файлів *cookie*. Токени можна додавати через приховані поля, *HTTP*-заголовки та використовувати з формами та викликами *AJAX*. Розробникам *Web*-застосунків необхідно впевнитись, що токени не потрапили в журнал сервера (логи) або в *URL*-адресу. Досить часто, при неправильному застосуванні, токени у *GET*-запитах потрапляють у декілька місць одночасно, наприклад в історію браузера, у файли журналу, до заголовків *Referer,* якщо захищений *Web*-застосунок посилається на зовнішній *Web*-застосунок (у разі, коли заголовки *Referrer-Policy* неправильно назначені).

1. *Використання CSRF-токену у тілі запиту (приховане поле)*

Цей метод передбачає генерацію на сервері унікального секретного ключа, передачу його до веб-браузера (наприклад, розміщення у прихованому полі *Web*-форми підчас генерації *HTML*-сторінки), та відправку його у тілі *HTTP*-запиту до сервера підчас виконання певної дії користувачем. У разі збігу отриманого ключа, і ключа, що має сервер - операція виконується.

Приклад додавання токена через приховане поле:

<form action="/account/action/changename " method="post">
...
<input type="hidden" name="csrf_token"
value="VGhpc193b3JrX21hZGVfYnlfT2xpaW55a19hbmRfTWVsbnljaGVua28=">
</form>

HTTP-запит матиме такий вигляд:

POST /account/action/changename HTTP/2
Host: www.name.tld
accept-encoding: deflate, gzip
*accept: */**
content-type: application/x-www-form-urlencoded; charset=UTF-8
cookie: user_session=V954qKt4GEzkB3hs
origin: https://www.name.tld
referer: https://www.name.tls/profile/account
content-length: 90

first=Ivan&last=Ivanov&csrf_token=VGhpc193b3JrX21hZGVfYnlfT2xpaW55a19hbmRfTWVsbnljaGVua28=

2. Використання CSRF-токену (у HTTP-заголовку)

Більш надійним методом захисту є використання токену у *HTTP*-заголовку, навіть якщо відбудеться витік токену, даний метод не дозволить його використати (обмеження *CORS (Cross-Origin Resource Sharing*). Аналогічно, як і в попередньому випадку, у разі збігу отриманого ключа, і ключа, що має сервер - операція виконується.

Припустимо, що дані відправляються з використанням *JavaScript API XMLHttpRequest*, тоді *HTTP*-запит матиме такий вигляд, де токен записаний у заголовок *x-csrf*, що також є секретним та генерується окремо від попереднього параметра *csrf_token*:

POST /account/action/changename HTTP/2
Host: www.name.tld
accept-encoding: deflate, gzip
*accept: */**
content-type: application/x-www-form-urlencoded; charset=UTF-8
cookie: user_session=V954qKt4GEzkB3hs
origin: https://www.name.tld
referer: https://www.name.tls/profile/account
x-csrf: Jl2GMQwvknO26Mnpd68pZDo620tv20jg
x-requested-with: XMLHttpRequest
content-length: 90

first=Ivan&last=Ivanov&csrf_token=VGhpc193b3JrX21hZGVfYnlfT2xpaW55a19hbmRfTWVsbnljaGVua28=

Отже, для захисту від *CSRF*-атак, токен повинен задовольняти такі умови: бути унікальним у межах кожної операції; бажано використовуватись лише один раз; стійкий до підбору; генеруватися криптографічно стійким генератором псевдовипадкових чисел; мати обмежений час дії.

3. Передача даних у альтернативному вигляді без використання MIME-типів класичних HTML-форм (без application/x-www-form-urlencoded чи multipart/form-data)

Важливо зазначити до попереднього методу, що дані у тілі запиту також можуть бути відправлені у альтернативному вигляді без використання *MIME*-типів класичних *HTML*-форм, тобто коли дані сформовані не у *application/x-www-form-urlencoded* чи *multipart/form-data,* а у будь-якому вигляді, зазвичай це *JSON* (тип даних *application/json*). Це дозволить позбутись навіть теоретичних *CSRF*-вразливостей, так як міжсайтовий запит з використанням форм можливий лише для *application/x-www-form-urlencoded* чи *multipart/form-data*. *HTTP*-запит матиме такий вигляд (параметр *csrf_token* відкинутий, так як він повністю втратив сенс):

POST /account/action/changename HTTP/2
Host: www.name.tld
accept-encoding: deflate, gzips
*accept: */**

content-type: application/json; charset=UTF-8
cookie: user_session=V954qKt4GEzkB3hs
origin: https://www.name.tld
referer: https://www.name.tls/profile/account
x-csrf: Jl2GMQwvknO26Mnpd68pZDo620tv20jg
x-requested-with: XMLHttpRequest
content-length: 34

{"first": "Ivan","last": "Ivanov"}

4. Перевірка Referer

Наступним методом захисту *Web*-застосунку від *CSRF*-атак є перевірка заголовку *Referer*. Заголовок *HTTP*-запиту *Referer* містить *URL* сторінки та дозволяє серверу дізнатися, звідки було здійснено запит [9]. Для реалізації даного методу достатньо реалізувати отримання домену (за допомоги регулярного виразу, порівняння строки та ін.) з *Referer* та порівняння з переліком дозволених (білий список), у разі не співпадіння з еталонним дозволеним *Referer*, операція зміни стану не буде виконуватись.

5. Використання SameSite Cookie

Відомо, що *cookie* надсилаються на сервер за будь-яких запитів, навіть якщо відбувається запит з чужого домена, тобто якщо відбувається міжсайтовий запит. Для того, щоб обмежити надсилання *cookie* лише з того домену, якому вони належать, необхідно використовувати атрибут *SameSite* зі значенням *Strict* [10].

У цьому разі встановлення *cookie* матиме такий вигляд:

SetCookie: user_session=V954qKt4GEzkB3hs; SameSite=Strict

Тобто у разі надсилання запиту зі стороннього домену, браузер не відправить *HTTP*-заголовок *cookie «user_session»* та, відповідно, сервер не ідентифікує користувача.

6. Підтвердження запиту для високочутливих операцій

Ще одним методом захисту *Web*-застосунків від *CSRF*-атак є підтвердження високочутливих операцій на основі взаємодії з користувачем. Для чутливих процесів, таких як переказ коштів чи зміна пароля, необхідно реалізувати підтвердження операції від користувача, тобто користувач *Web*-застосунку повинен підтвердити свої дії, наприклад, за допомогою вводу певного одноразового токена (що відправляється на *e-mail* чи *SMS*, генерується за допомогою *TOTP/HOTP (Time-based One-time Password Algorithm / HMAC-based One-time Password Algorithm)* чи їх аналогів, повторної автентифікації, вводу *CAPTCHA* та інше [11].

Висновки

Для полегшення виявлення різних вразливостей *Web*-застосунків існують різноманітні інструменти (сканери), які можуть допомогти в аналізі безпеки *Web*-застосунків та полегшити розробку захисту. Але дані інструменти у більшості можуть лише ідентифікувати проблему, а не прибрати її. Тому знання розробника є ключовим фактором у побудові безпечного *Web*-застосунку.

Для вирішення проблем безпеки *Web*-застосунків, розробникам необхідно знати усі шляхи та вектори проведення атак, щоб мати можливість розробити механізми захисту.

Проведене дослідження показало, що більшість розробників не приділяють достатньої уваги захисту *Web*-застосунків від *CSRF*-атак, тому авторами було запропоновано ряд методів для захисту *Web*-застосунків, основними з яких являються використання *CSRF*-токену у тілі запиту та у *HTTP*-заголовку, передача даних у альтернативному вигляді без використання класичних *HTML*-форм, перевірка заголовку *Referer*, використання атрибуту *SameSite* та підтвердження користувачем високочутливих операцій. Таким чином, правильно спроектований та реалізований механізм захисту дозволить *Web*-застосунку залишатись невразливим до *CSRF*-атак.

Література

1. *Prevent Cross-Site Request Forgery (XSRF/CSRF) attacks in ASP.NET Core* [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/security/anti-request-forgery?view=aspnetcore-6.0

2. *G. Pellegrino, M. Johns, S. Koch, M. Backes and C. Rossow.* Deemon: Detecting CSRF with dynamic analysis and property graphs, Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security CCS, 2017. – October 30 - November 03, 2017. – P. 1757-1771.

3. *Likaj, Xhelal; Khodayari, Soheil; Pellegrino, Giancarlo.* Where We Stand (or Fall): An Analysis of CSRF Defenses in Web Frameworks. In: 24th International Symposium on Research in Attacks, Intrusions and Defenses. – 2021. – P. 370-385.

4. *Peguero, Ksenia; Cheng, Xiuzhen.* CSRF protection in JavaScript frameworks and the security of JavaScript applications. High-Confidence Computing – 2021. – P. 1.2: 100035.

5. *Compagna, Luca, et al.* A preliminary study on the adoption and effectiveness of SameSite cookies as a CSRF defence. In: 2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). IEEE. – 2021. – P. 49-59.

6. *National Vulnerability Database: CSRF statistics.* [Електронний ресурс]. – Режим доступу: https://nvd.nist.gov/vuln/search/statistics?form_type=Advanced&results_type=statistics&query=CSRF&search_type=all

7. *OWASP Cross Site Request Forgery (CSRF)* [Електронний ресурс]. – Режим доступу: https://owasp.org/www-community/attacks/csrf

8. *Reviewing Code for Cross-Site Request Forgery Issues* [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-code-review-guide/reviewing-code-for-csrf-issues

9. *Documentation for Web developers (Referer)* [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referer

10. *Documentation for Web developers (Cookies)* [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies

11. *Cross-Site Request Forgery Prevention Cheat Sheet* [Електронний ресурс]. – Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html#user-interaction-based-csrf-defense

Олійник Я.О., Мельниченко П.І.

ОГЛЯД МЕТОДІВ ЗАХИСТУ WEB-ЗАСТОСУНКІВ ВІД ВРАЗЛИВОСТЕЙ ТИПУ CSRF (МІЖСАЙТОВА ПІДРОБКА ЗАПИТІВ)

В статті представлено дослідження методів захисту Web-застосунків від вразливостей типу CSRF (міжсайтова підробка запитів). Проведене дослідження показало, що розробники Web-застосунків не приділяють достатньої уваги захисту від атак типу Міжсайтова підробка запитів, тому авторами було систематизовано та запропоновано комплекс методів захисту від CSRF-атак, та сформовано рекомендації для розробників Web-застосунків, для забезпечення комплексного захисту від CSRF-атак. Автори пропонують використовувати ряд методів, до яких відносяться: використання CSRF-токену у тілі запиту та у HTTP-заголовку, передача даних у альтернативному вигляді без використання MIME-типів класичних HTML-форм, перевірка заголовку Referer,

використання атрибуту SameSite та підтвердження користувачем високочутливих операцій.

Запропоновані методи дозволять розробникам створювати безпечні Web-застосунки, які будуть невразливі до CSRF-атак.

Ключові слова: *CSRF-атака, міжсайтова підробка запитів, захист інформації, Web-застосунок.*

Oliinyk Y.O., Melnychenko P.I.

OVERVIEW OF METHODS FOR PROTECTING WEB-APPLICATIONS FROM CSRF VULNERABILITIES (CROSS-SITE REQUEST FORGERY)

The article presents a study of methods of protecting Web-applications from CSRF vulnerabilities (Cross-Site Request Forgery). The conducted research showed that Web-developers do not pay enough attention to protection against attacks such as Cross-Site Request Forgery, the authors systematized and proposed a complex of methods of protection against CSRF-attacks, and formed recommendations for Web-application developers to ensure comprehensive protection against CSRF-attacks. The authors suggest using a number of methods, which include: using a CSRF-token in the request body and in the HTTP-header, transferring data in an alternative form without using MIME-types of HTML-forms, checking the Referer header, using the SameSite attribute and confirming sensitive operations by the user.

The proposed methods will allow developers to create secure Web-applications that are invulnerable to CSRF-attacks.

Keywords: *CSRF-attack, Cross-Site Request Forgery, data protection, Web-application.*