UDC 004.056 **doi: 10.20998/2522-9052.2019.4.15**

I. R. Ragimova[1], F. A. Gubadova[1], B. A. Asker-zade[1], S. S. Pohasii[2]

[1] Azerbaijan Technical University, Baku, Azerbaijan
[2] Simon Kuznets Kharkiv National University of Economics, Kharkiv, Ukraine

JAVASCRIPT SECURITY USING CRYPTOGRAPHIC HASH FUNCTIONS

Abstract. The subject of this research is the development of methods of protection against attacks on third-party JavaScript and the document object model (DOM). **The purpose of the article** is to develop an algorithm and determine the effectiveness of using cryptographic hash functions as one of the methods of protection against attacks on third-party JavaScript resources. The third-party JavaScript code download chain may consist of three or four third-party websites. From a security point of view, this creates a risk of attack on a third-party resource. If the attacker compromises one of the third-party resources, this will affect the entire chain using this resource. Based on these conditions, it is indispensable to solve the following **tasks**: to develop a secure algorithm for hash functions for protecting applications in JavaScript, which will constantly monitor changes that occur on a web page; determine the advantages and disadvantages of the method in real operating conditions. In the process of the study, the following **results** were obtained: the problems of writing safe code in JS were considered, the algorithm for using cryptographic hash functions was proposed, the essence of which is that the hash is calculated at the first moment of loading a third-party resource. Each time a third-party resource is loaded, the algorithm calculates its hash and compares it with the value of the first hash. It is established that cryptographic hash functions on the example of sha384 have the property of an avalanche effect. It is recommended to use this method for web pages with mission-critical operations, such as payment pages, registration, password reset or account login. Their strengths and weaknesses were also revealed in the process of improving the JavaScript protection method.

Keywords: JavaScript security; cryptographic hash functions; method of protection against attacks; document object model.

Introduction

Recently, considerable efforts have been expended and some successes have been achieved in the process of developing cryptographic methods for analyzing the hash function. In this article, we propose a method of protection against attacks on third-party JavaScript resources using cryptographic hash functions. Hash functions are widely used in cryptography, especially in the construction of digital signature systems and various cryptographic protocols.

The purpose of the article is to develop an algorithm and determine the effectiveness of using cryptographic hash functions as one of the methods of protection against attacks on third-party JavaScript resources.

Cryptographic requirements for key and keyless hash functions arise directly from the conditions of their use in practice. The main requirement for hash functions is the uniform distribution of their values with a random choice of argument values. For cryptographic hash functions, it is also important that with the slightest change in the argument, the value of the function changes strongly (avalanche effect). Despite its simplicity, the algorithm allows timely prevention of the use of compromised third-party resources. However, the method creates additional administrative functions to support the site, as most modern web sites will include hundreds of third-party resources.

Literature review. The OWASP TOP 10 Project document presents a list of the most significant and critical risks of web applications. The decision to include vulnerabilities in this list is based on the expert opinion of cybersecurity experts from around the world.

Implementation of Applications function related to authentication and session management are often incorrectly implemented, allowing attackers to compromise passwords, keys or session tokens, as well as exploit other implementation errors to temporarily or permanently intercept user accounts.

The disadvantages of authentication. Many web applications and APIs have poor protection for critical data. Confidential data requires additional security measures, for example, their encryption during storage or transmission, as well as special precautions when working with the browser.

XML External Entities (XXE). Older or poorly configured XML processors process references to external entities within documents. These entities can be used to access internal files through file URIs, shared folders, port scans, remote code execution, and denial of service.

Disadvantages of access control. The actions allowed by authenticated users are often incorrectly controlled. Attackers can take advantage of these shortcomings and gain unauthorized access to other users' accounts or confidential information, as well as change user data or access rights.

Incorrect security settings occur due to the use of standard security settings, incomplete or specific settings, open cloud storage, incorrect HTTP headers and detailed error messages containing critical data.

Cross-site scripting (XSS) occurs when an application adds unverified data to a new web page without checking or converting it, or when it refreshes an open page through a browser API using user-provided data that contains HTML or JavaScript code. Using XSS, attackers can execute scripts in the victim's browser, allowing them to intercept user sessions, swap site pages, or redirect users to malicious sites.

Insecure deserialization often leads to remote code execution. Deserialization errors that do not lead to remote code execution can be used for attacks with replay, injection, and privilege escalation [7].

Formulation of the problem. To ensure the safety of JavaScript, it is not easy to track what is happening when running scripts on JS, but, armed with suitable tools, you can find and rewrite problem areas even in the most confusing code.

JavaScript does not stand still: new and new features appear in it, it is often used to write applications (both mobile and desktop), and is also increasingly found on servers (and not only) thanks to Node.js. And this means that the art of catching bugs must be taken to a new level.

Statement of the main material

Developing secure JavaScript applications is a tricky task due to JavaScript features, due to the following problems of the complexity of writing safe JS code.

1. **The compiler will not help** since JavaScript is an interpreted language. It will not be refusing to work and pushing you to correct errors and optimize the code.

2. **The dynamic essence of JavaScript.** JavaScript is dynamic, weakly typed, and asynchronous, which confirms that security problems are inevitable.

3. **Intricate features of JS.** JavaScript has prototypes, first class functions, and closures. They make the language even more dynamic, and writing safe code is more difficult.

4. **Close interaction between JavaScript and the document object model (DOM).** Thanks to the DOM, web pages loaded into the browser can be updated in stages during the loading of data from the server. However, this convenience also has a flip side: the code fragments that are responsible for the dynamic interaction between the JS and the DOM are especially error prone.

5. Complex event interactions. JavaScript is an event-driven language. Most events are triggered by user actions. There are those that can be triggered without it, for example, time events and asynchronous calls. Each event can be echoed throughout the DOM tree and activate several "listeners" at once. Sometimes tracking all of this is a rather non-standard task.

Problem: A web page consists of many different third-party JavaScript scripts.

The following is an example HTML page with embedded JavaScript code on external resources (see Fig. 1).

```
<html>
<head >
<title>My Site</title>
<script type="text/JavaScript"
src="//se.monetate.net/js/2/a-ec87f2d7entry.js">-
</script>
<script
src="//nexus.ensighten.com/Bootstrap.js"></script>
</head>
<body>
My test website
</body>
</html>
```

Fig. 1. Algorithm 1

As can be seen, in addition to standard HTML attributes such as head, body, the Script attribute is also present. This attribute allows to load third-party code into an HTML page.

In this example, two third-party JavaScript codes are loaded into the page.

As can be seen from the example, the third-party code itself is missing, and instead links to external resources are indicated. The original code is downloaded directly from a third-party resource.

Modern web applications use hundreds of links to third-party JavaScript code.

The Fig. 2 shows a graph that displays the number of third-party JavaScript resources using a popular website as an example.

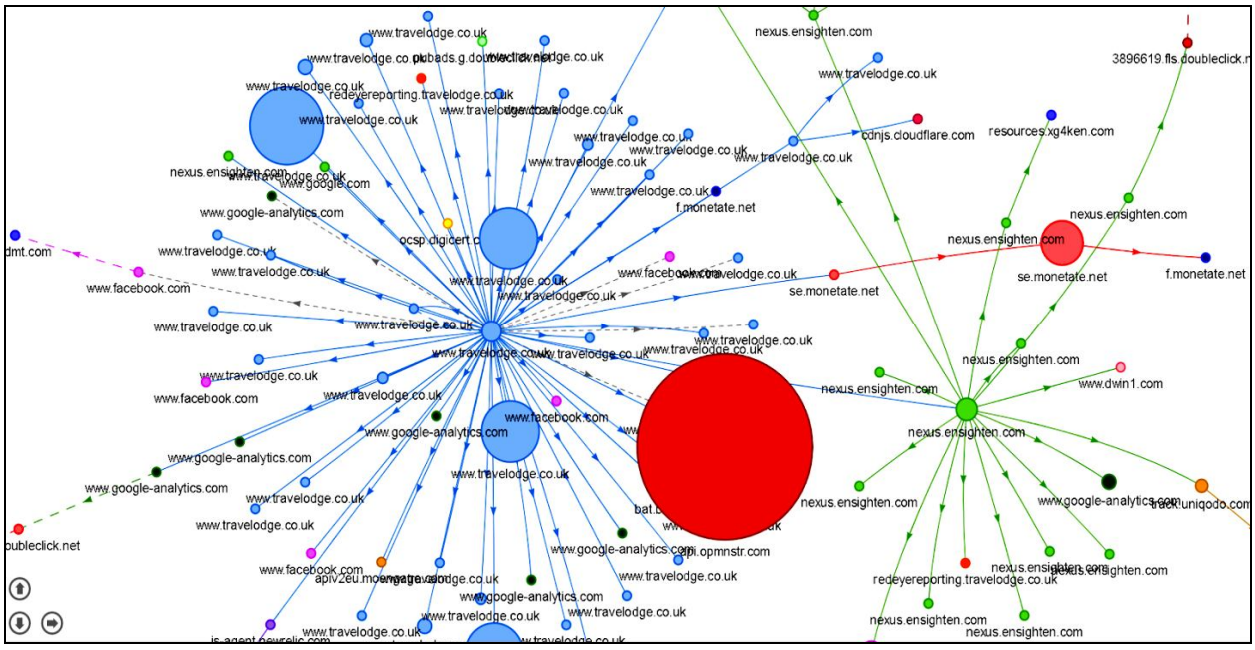

Fig. 2. The hash is calculated at the first moment of loading a third-party resource

The central vertex of the graph represents the main website, and the vertices coming from it represent third-party JavaScript resources.

As can be seen, the third-party JavaScript code loading chain can consist of three or four third-party websites. From a security point of view, this creates a risk of attack on a third-party resource. If the attacker compromises one of the third-party resources, this will affect the entire chain using this resource.

Below is an example of an html page where, along with a link to a third-party JavaScript resource, the corresponding hash is also shown (see Fig. 3).

The number of third-party JavaScript resources

Each time a third-party resource is loaded, the algorithm calculates its hash and compares it with the value of the first hash.

At the slightest change in the external resource, the algorithm will calculate a new hash and detect the difference with the value of the first hash.

Cryptographic hash functions have the property of an avalanche effect. That is, a change in one bit of data will affect a complete change in the hash function. This mismatch will cause a failure to load a third-party resource.

```
<script src="https://example.com/example-framework.js"
integrity="sha384-Li9vy3DqF8tn TXuiaAJuML3ky+er1 0rcgNR/VqsV pcw+ThHm Ycwi B1pbOxEbzJr7"
crossorigin="anonymous"></script>
```

The following is an example of an avalanche effect with a hash function sha384

```
sha384(test)=
=
768412320f7b0aa5812fce428dc4706b3cae50e02a64caa16a782249bfe8efc4b7ef1ccb126255d196047dfedf17a0a9
sha384(Test)=
7B8F4654076B80EB963911F19CFAD1AAF4285ED48E826F6CDE1B01A79AA73FADB5446E667FC4F90417782C91270540F3
```

Fig. 3. Example of an html page

In the above example, we use the integrity attribute, which contains the hash value itself.

In this example, this is a hash function sha384.

Program algorithm

1. Insert a link to a third-party resource in the page.
2. Follow the external link.
3. Calculate the hash of a third-party resource.
4. Check if the hash value corresponding to this third-party resource is identical.
5. If no, value is missing, then place the calculated value into the database.
6. If the value exists, but not identical to the value just calculated, then refuse to load a third-party resource.
7. Display loading error information in the browser console.

Strengths and weaknesses of this approach

Weaknesses: This method is reactive, i.e. leaves little time for site administrators to make decisions in case of legitimate changes in third-party code.

The method will create a lot of false positives on dynamic pages, since the content in them is changes frequently.

Strengths: This method is suitable for web pages with critical operations, such as payment pages, registration, password reset or account login. Pages with these types of operations are of most interest to the attacker.

Conclusions

The use of this method will constantly monitor changes occurring on the web page. If one bit changes on any of the resources, the program will detect and warn about an attempt to change the state of the system.

Thus, the timely detection of unauthorized changes and leaving the download of a compromised third-party resource will adequately respond and classify an external threat.

REFERENCES

1. Haverbeke, Marijn (2018), *Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming Paperback*, Dec.4, 2018, available at: http://eloquentjavascript.net.
2. Kingsley-Hughes, Adrian and Kingsley-Hughes, Kathie (2008), *JavaScript 1.5 by Example 1st Edition Que Publishing*, 1 edition, 312 p.
3. Grimes R.A. (2017), *Hacking the Hacker: Learn From the Experts Who Take Down Hackers*, Hoboken: Wiley, 283 p.
4. Yaworski, P. (2015), *Web Hacking 101. Kindle Edition*, 216 p.
5. Bisson, D. (2017), *Researcher warns of 'pastejacking' hack attacks targeting users' clipboards*, available at: https://www.grahamcluley.com
6. Zaitsev, M. (2017), *Security of Java applications leaves much to be desired*, available at: https://threatpost.ru/veracode-states-that-java-apps-are-poorly-protected/22946
7. *TOP-10 OWASP – 2017* (2017), available at: https://www.owasp.org/images/9/96/OWASP_Top_10-2017-ru.pdf

Received (Надійшла) 11.10.2019
Accepted for publication (Прийнята до друку) 13.11.2019

ВІДОМОСТІ ПРО АВТОРІВ / ABOUT THE AUTHORS

Рагімова Ірада – доцент кафедри комп'ютерних систем і мереж, Азербайджанський технічний університет, Баку, Азербайджан;
Irada Rahimova – Associated Professor of Computer Systems and Networks Department, Azerbaijan Technical University, Baku, Azerbaijan;
e-mail: ika1402@mail.ru; ORCID ID: http://orcid.org/0000-0003-3158-6844

Губадова Фірангиз – доцент кафедри комп'ютерних систем і мереж, Азербайджанський технічний університет, Баку, Азербайджан;
Firangiz Qubadova – Associated Professor of Computer Systems and Networks Department, Azerbaijan Technical University, Baku, Azerbaijan;
e-mail: fira1942@hotmail.com; ORCID ID: http://orcid.org/0000- 0002-5031-9787

Аскер-заде Бараят – доцент кафедри комп'ютерних систем і мереж, Азербайджанський технічний університет, Баку, Азербайджан;
Barayat Asker zade – Associated Professor of Computer Systems and Networks Department, Azerbaijan Technical University, Baku, Azerbaijan;
e-mail: askerzade@mail.ru; ORCID ID: http://orcid.org/0000- 0001-7630-5028

Погасій Сергій – кандидат економічних наук, доцент кафедри кібербезпеки та інформаційних технологій, Харківський національний економічний університет ім. Семена Кузнеця, Харків, Україна;
Serhii Pohasii – PhD in Economics, Associated Professor of Cybersecurity and Information Technologies Department, S. Kuznets Kharkiv National University of Economics, Kharkiv, Ukraine;
e-mail: spogasiy1978@gmail.com; ORCID ID: http://orcid.org/0000-0002-4540-3693

Використання криптографічних хеш-функцій для безпеки JavaScript

І. Рагімова, Ф. Губадова, Б. Аскер-заде, С. Погасій

Анотація. Предметом дослідження є процес розробки методів захисту від атак на сторонні JavaScript і об'єктної моделі документа. **Метою статті** є розробка алгоритму і визначення ефективності використання криптографічних хеш-функцій як одного із способів захисту від атак на сторонні JavaScript ресурси. Ланцюжок завантаження стороннього JavaScript коду може складаючись із трьох або чотирьох сторонніх веб-сайтів. З точки зору безпеки це створює ризик атаки на сторонній ресурс. Якщо атакуюча сторона скомпрометує один з сторонніх ресурсів, то це вплине на весь ланцюжок, що використовує даний ресурс. Виходячи з даних умов необхідно вирішити такі завдання: розробити безпечний алгоритм хеш-функцій захисту застосунків на JavaScript, який дозволить постійно моніторити зміни, що відбуваються на веб-сторінці; визначити переваги і недоліки методу в реальних умовах експлуатації. У процесі дослідження, були отримані наступні **результати:** розглянута проблематика складності написання безпечного коду на JavaScript, запропонований алгоритм використання криптографічних хеш-функцій, суть якого полягає в тому, що хеш обчислюється в перший момент завантаження стороннього ресурсу. При кожному завантаженні стороннього ресурсу алгоритм обчислює його хеш і порівнює зі значенням першого хеша. Встановлено, що криптографічні хеш-функції на прикладі sha384 мають властивість лавинного ефекту. Рекомендовано застосування даного методу для веб-сторінок з критично важливими операціями, такими як сторінки платежу, реєстрація, зміна пароля або вхід у обліковий запис. Також виявлено їх сильні і слабкі сторони в процесі удосконалення методу захисту JavaScript.

Ключові слова: безпека JavaScript; криптографічні хеш-функції; метод захисту від атак; об'єктна модель документа.

Використання криптографічних хеш-функцій для безпеки JavaScript

И. Рагимова, Ф. Губадова, Б. Аскер-заде, С. Погасий

Аннотация. Предметом исследования является процесс разработки методов защиты от атак на сторонние JavaScript и объектную модель документа. **Целью** статьи является разработка алгоритма и определение эффективности использования криптографических хеш-функций как одного из методов защиты от атак на сторонние JavaScript ресурсы. Цепочка загрузки стороннего JavaScript кода может состоять из трех или четырех сторонних веб-сайтов. С точки зрения безопасности это создает риск атаки на сторонний ресурс. Если атакующая сторона скомпрометирует один из сторонних ресурсов, то это повлияет на всю цепочку, использующую данный ресурс. Исходя из данных условий необходимо решить следующие з**адачи**: разработать безопасный алгоритм хеш-функций защиты приложений на JavaScript, который позволит постоянно мониторить изменения, происходящие на веб-странице; определить достоинства и недостатки метода в реальных условиях эксплуатации. В процессе исследования были получены следующие **результаты**: рассмотрена проблематика сложности написания безопасного кода на JavaScript, предложен алгоритм использования криптографических хеш-функций, суть которого заключается в том, что хеш вычисляется в первый момент загрузки стороннего ресурса. При каждой загрузке стороннего ресурса алгоритм вычисляет его хеш и сравнивает со значением первого хеша. Установлено, что криптографические хеш-функции на примере sha384 обладают свойством лавинного эффекта. Рекомендовано применение данного метода для веб-страниц с критически важными операциями, такими как страницы платежа, регистрация, смена пароля или вход в учетную запись. Также выявлены их сильные и слабые стороны в процессе усовершенствования метода защиты JavaScript.

Ключевые слова: безопасность JavaScript; криптографические хеш-функции; метод защиты от атак; объектная модель документа.