Methods of information systems protection

UDC 004.05 **doi: 10.20998/2522-9052.2020.1.18**

V. Pevnev, O. Popovichenko, Ya. Tsokota

National Aerospace University "Kharkiv Aviation Institute", Kharkiv, Ukraine

WEB APPLICATION PROTECTION TECHNOLOGIES

Abstract. The **subject matter** of the article is the vulnerabilities that there are in web applications. The **goal** is to analyze the problem of violation of information security of web applications. The **tasks** to be solved are: view statistics on web attacks on web applications; identify the main prerequisites for cyber-attacks; considered the most common types of vulnerabilities; suggest ways to create a secure application. The **methods** used are: analytical method, literature analysis, description. The following **results** were obtained: For each given type of vulnerability, a scenario of a possible attack by an attacker was considered. There were also suggested ways for developers to use these vulnerabilities and develop a secure web application. **Conclusions**. Keep in mind that the best protection for web applications is writing safe code. Developers who implement applications should be aware in advance of the existence of common types of attacks and how they work in order to protect applications and prevent possible cyber-attacks. It is best to use security methods comprehensively to protect your web application as much as possible.

Keywords: web application; vulnerability; attack; web applications protection; secure application.

Introduction

Today, information technology (IT) is becoming more and more common in society. They are used in a wide variety of industries: social networks, email, news and government portals, electronic commerce, forums, blogs and other websites. Along with the development of IT, information security (IS) is becoming increasingly important. If we consider the historical perspective of the development of IS, it turns out that since 1996, when the Constitution of our state was adopted, until the moment of writing this work, the number of people who feel protected in the information field constantly falling. The point is not only in the avalanche-like growth in the use of IT, and most likely in the violations that the state makes in the information sphere. The growing requirements for providing IS are determined, first of all, by the development of IT. Potential violators have new opportunities for destructive actions in all areas of our lives. The most effective of these actions occur in cyberspace. Based on the foregoing, we can conclude the need to protect web applications.

Formulation of the problem. Depending on where the web application is used, information of different access levels and values can be processed in it. For example, applications can use information about bank cards, personal data of users, passwords and other identification data. No user is immune from the fact that his data can be copied or subjected to a number of influences of an accidental and malicious nature in the process of its processing, transmission and storage. Therefore, it is important for developers to provide IS for users.

All these listed resources can be implemented using web applications, which are a "client-server" architecture. In this concept (see Fig. 1), the client is the user's browser and the server is the web server. At the heart of the "client-server" concept, information is exchanged over the network, the client starts interacting and data is stored primarily on the server.

Basically, web applications have a distributed structure. The main advantages of this structure:

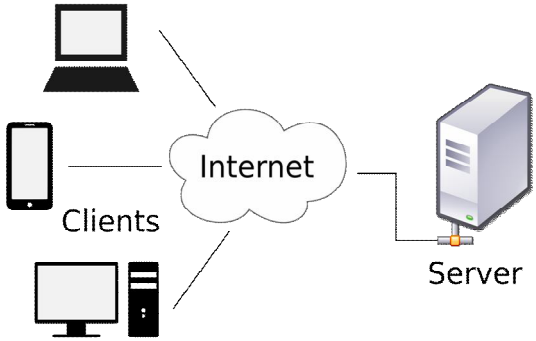

Fig. 1. The concept of "client-server"

– good scalability – the ability to increase functionality without changing the structure;

– the availability of the ability to manage the load of the application – redirecting user request flows to less loaded servers [1].

The typical architecture of these applications can be represented in three levels: the client part (web browser), web server, data (DB) (Fig. 2).

Analysis of recent research. Every day, any website may be exposed to cyber attacks. Typically, most attacks are targeted. This means that the intruder is not limited to a single attempt to obtain the necessary data in an unauthorized way, since he does not know exactly which vulnerabilities are in the code. All attempts to hack a site add up to a series of events that occur over a certain period of time.

According to statistics for 2018, the largest number of attacks on one web application falls to the websites of financial organizations, transport companies and service companies (see Table 1) [2].

The choice of attack method depends on the features of the web application. Suppose, if the application does not provide the ability to enter user data, then the attacker will not conduct attacks aimed at changing the logic of the application through user input. The most popular attacks include: injecting SQL code, going outside the directory and cross-site scripting (Table 2).

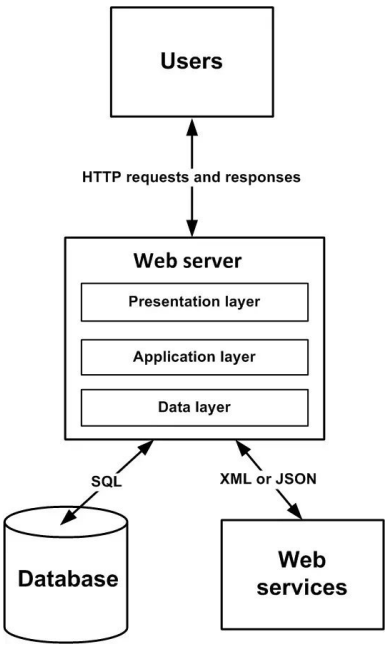

Fig. 2. Distributed web application architecture

Table 1 – **Average number of targeted attacks per day per web application**

The scope of the site	Number of attacks
Financial organizations	151
Transport companies	135
Services sector	114
IT companies	87
State institutions	86
Education and science	57

Table 2 – **Most popular attacks**

Type of attack	Percentage ratio
SQL Injection	27 %
Path Traversal	17 %
Cross-Site Scripting	14 %
Local File Inclusion	11 %
Information Leakage	8 %
OS Commanding	7 %
Brute Force	5 %
Remote Code Execution	2 %
Denial of Service	2 %
Server-Side Template Injection	1 %
Other	6 %

The purpose of the article is to consider the most common vulnerabilities in web applications and develop possible ways to create secure web applications.

Research results

Most often, web applications are attacked because vulnerabilities are present in the application code [3]. Consider the most popular types of vulnerabilities present in web applications and find out what may be attacks on such applications.

The first category includes code problems associated with an unverified conversion and redirect. In such cases, an attacker tricks users into a dangerous site by manipulating the URL of a real site. Redirection typically uses the query string parameter returnUrl [4]. *Example of an attack:* first, the attacker assures the user to click on the link to the page of the source site for authorization, while he adds the value of the query string returnUrl to the URL.

Consider the web application on test.com with the login page http://test.com/Log-in?returnUrl=/Home/. After manipulating the value of the query string, the user clicks on the second URL test1.com, not test.com. Successfully logs in to your account. The site redirects the user to http://test1.com/Log-in (a malicious site that looks like a real one). The user logs in again, providing the malicious site with their credentials, and is redirected again to the real site. With such actions, the user might think that he was unable to log in the first time, not suspecting that his credentials were compromised. *To protect the application* from this attack it is necessary: during development, assume that all provided user data does not inspire confidence. If the web application contains functions that redirect users based on the contents of the URL, then redirection can only be implemented locally in the application or strictly behind a known URL, and not to any address in the query string. The second type of attack is the interworking of the request. In this case, the attacker exploits the flaws of the HTTP protocol. When a browser opens a page, malicious code is executed. It forces the user to send a specific request to the attacker's server. As a result, some actions are performed that are necessary for the attacker. *An example of such an attack:* suppose there is a fictional bank "Some bank". It has a page for sending cash payments to a specific account number. The request can look like this:

http://www.somebank.com/bank/transfer.aspx?creditAccount=1001160141&transferAmount=1000.

If the attacker found this link, then he can send a letter to the user, where he will first post the necessary link to his site (Fig. 3) [5]. For example, the letter will look like:

"Good afternoon, Some bank user!

Recently, we have implemented on our server several security improvements that require confirmation of your account. Use the following *link*."

In this case, when the user clicks on the link, he goes to the site and receives, for example, a message stating that an amount of $1000 was transferred to account 1001160141.

When the user goes to this page, no action is required from he, since the form is automatically submitted when the URL is loaded.

There is the following way to protect the application using the referer header. Since most browsers tell the server which page the request was sent from, it is possible to reject the request on the server if the referer does not match the host domain name (Fig. 4).

Another popular type of vulnerability is the lack of access control. This means that high-level functionality is hidden from low-level functionality instead of making changes using access control [6]. Therefore, an attacker who acts as a low-level user can gain access, for example, to the web application administration interface. *An example of an attack would be this:* an application uses data in an SQL call that is not verified.

```
<html>
<body onLoad="document.getElementById('transferForm').submit()">

<form id="transferForm" action="http://www.somebank.com/bank/transfer.aspx" method="post">

<input type="hidden" name="creditAccount" value="1001160141">
<input type="hidden" name="transferAmount" value="10">

</form>

</body>
```

Fig. 3. Example code of an attacker page

```
If(request.getHeaders("referer") != null
&& request.getHeaders("referer").indexOf(
    "http://www.somebank.com") != 0){
    throw new Exception("Invalid referer");
        }
```

Fig. 4. Example use of referer

Calling access to user account information:

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

The attacker changes the "acct" parameter to send the desired account number:

```
http://example.com/accountInfo?acct=notmyacc
```

Without the necessary verification, an attacker could gain access to an account.

To prevent such an attack, you must first deny access by default, except for open resources. Access control features should be implemented and cross-domain resource usage should be minimized. And also limit the frequency of access to the API and controllers.

Unprotected direct object references are also a common drawback of web applications. This allows an attacker to obtain data from a server by manipulating file names. *An example of an attack:* imagine that there is an image that the hacker cannot access on the server, but who wants to hack and which is published via a URL similar to this https://example.net/photos/774.jpg. A generic HTTP request has the form:

```
GET /photos/774.jpg HTTP / 1.1
Host: example.net
```

After logging into your account, the attacker can edit his personal images with a special URL in combination with the session cookie, for example:

```
https://example.net/api/edit/?image=48.jpg
HTTP request generated:
GET /api/edit/?image=48.jpg
Host: example.net
Cookie: authtoken = HRCALAGJEOWRGTMW
```

In this example, authtoken is a session cookie that tells the server that it is a user and that he is authenticated. But in the case that the server only checks the authtoken and does not check the name of the image on the account, then it is indeed allowed to edit this image. Then the attacker can reproduce the necessary request with the forbidden image of the file name in it, for example:

```
https://example.net/api/edit/?image=774.jpg
HTTP request:
GET /api/edit/?image=774.jpg
Host: example.net
Cookie: authtoken = HRCALAGJEOWRGTMW
```

In this case, it is necessary to check access to the use of direct object links from an unknown source, and it is also necessary to use only one user or one session for indirect references to objects [7].

One of the most popular types of vulnerabilities is cross-site scripting. This application defect allows attackers to insert JavaScript code on the pages of real sites. By doing so, they can replace the entire contents of the website in order to gain unauthorized access to user credentials. *Example of a possible scenario:* the developed application uses unverified data when generating HTML code without converting it. Let's say in this line of code:

```
(String) page += "" + request.getParameter("CC") + "";
An attacker can change the 'CC' parameter to:
<script>
    document.location = 'http://www.attacker.com/cgi-
    bin/cookie.cgi?
foo='+document.cookie
</script>
```

As a result, the user session identifier is sent to the attacker's site, allowing the attacker to intercept the victim's current session [8].

To prevent such attacks, you need to separate unverified data from the active content of the browser. First, you need to use frameworks with automatic data conversion (such as ReactJS or Vue.js). Secondly, it is necessary to apply context coding when changing a document in the user's browser to prevent cross-site scripting on the DOM.

A very common vulnerability in many web applications is authentication weaknesses. In this case, the attackers have access to combinations of names and passwords for attacks on accounts, lists of standard administrator credentials and dictionary attacks. For example, password authentication is not a reliable way to protect personal data, as users tend to choose simple passwords and the same passwords in different systems. In this case, the developer should not allow users to create simple passwords, do not allow passwords to be transmitted over an insecure HTTP connection or in the address bar, and should not allow session tokens to be transmitted over an insecure HTTP connection or in the URL bar. And also, allow users to change the password and notify them about changing the password, use secure hash functions to store passwords, and require reauthentication after important actions, such as changing the password, changing confidential information [9].

Finally, the most popular type of attack is injection. Injections allow hackers to change the server command request through unauthorized user input. Such implementations can lead to data loss or data corruption, and can also be used for the interests of third parties. Such consequences occur if the data entered by the user is not checked, not filtered or cleared. If non-parameterized calls without contextual screening are directly used in the interpreter. *Injection example*: an application uses untrusted data when creating the next vulnerable SQL call. For instance:

```
String query = "SELECT * FROM accounts
WHERE custID='" + re-
quest.getParameter("id") + "'".
```

In this case, the attacker changes the value of the "id" parameter in his browser to send 'or' 1 '=' 1. For instance:

```
http://example.com/app/accountView?id=' or '1'='1
```

Modifying the query allows you to get all the entries from the credential table. More serious attacks allow you to modify or delete data, as well as call stored procedures [10].

In order to detect vulnerabilities in the code and understand how the system will respond to the attack – the application needs to be tested. The testing process is as similar as possible to the hacking process that an attacker conducts. The purpose of such actions is to determine how vulnerable the web application is [12].

According to the source [13], the most popular testing methodologies are:

– the Open Source Security Testing Methodology Manual;

– the National Institute of Standards and Technology (NIST) Special Publication 800-115;

– OWASP Testing Guide;

– Penetration Testing Execution Standard;

– Information Systems Security Assessment Framework.

To test the security of web applications, it is more appropriate to use the OWASP methodology. This methodology is based on the black box method – information about the tested application is limited or absent at all [14].

Software security covers a very wide area of subjects. To have a secure software application you have to consider many things. Here is a little diagram (Fig. 5)

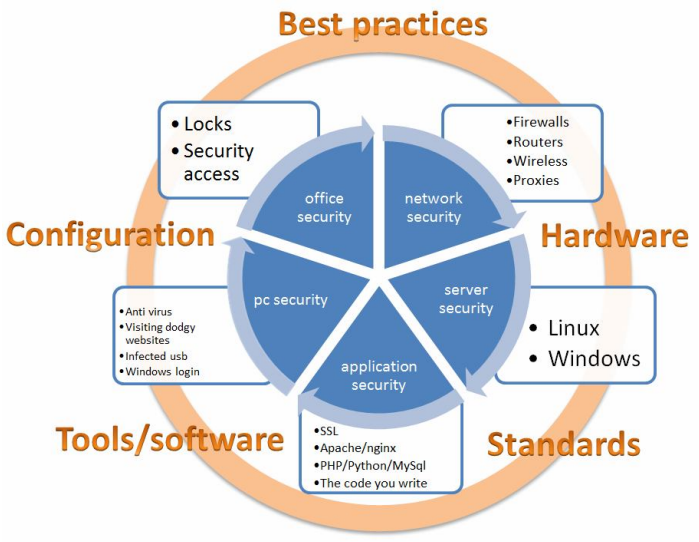

Fig. 5. Web security layered approach

what illustrate different areas of security, what they cover, and what needs to be considered [15-17].

Conclusions

This article discusses some types of vulnerabilities in web applications and possible attacks using them. The proposed defense methods are based on creating secure code. Developers who implement applications should use their best efforts to study proven types of attacks. In addition, as preventive measures to ensure the IS of the developed applications, it is necessary to predict likely attacks on the web application. In addition, it is necessary to use an integrated approach to the creation of an information security system, which should combine measures to ensure confidentiality, accessibility and integrity of information.

References

1. Markov, E. (2019), *Distributed Application Architecture* [online], available at: https://www.itweek.ru/infrastructure/article/detail.php?ID=66147
2. Ptsecurity.com (2019), *Attacks on web applications: results of 2018* [online], available at: https://www.ptsecurity.com/ru-ru/research/analytics/web-application-attacks-2019
3. Habr.com (2015), *10 attacks on web applications in action.* [online], available at: https://habr.com/ru/company/ua-hosting/blog/272205
4. Docs.microsoft.com (2017), *Prevention of open redirect attacks in ASP.NET Core* [online], available at: https://docs.microsoft.com/ru-ru/aspnet/core/security/preventing-open-redirects?view=aspnetcore3.1
5. Ionescu, P. (2014). *Prevention of falsification of cross-site requests: latent danger on browser tabs* [online], available at: https://www.ibm.com/developerworks/ru/library/se-appscan-detect-csrf-xsrf/index.html
6. Habr.com (2014), *Speedran for 13 vulnerabilities on sites. Basic concepts and means of protection* [online], available at: https://habr.com/ru/post/226321
7. Cadelta.ru (2019), *The best solutions for protecting sites and web-applications* [online], available at: https://cadelta.ru/security/id3369
8. Hackware.ru (2018), *Lesson 1. The basics of XSS and the search for sites vulnerable to XSS* [online], available at: https://hackware.ru/?p=1174
9. Habr.com (2015), *Overview of authentication methods and protocols in web applications* [online], available at: https://habr.com/ru/company/dataart/blog/262817/.

10. Nalivaiko, A. (2017). *How to protect a web application: basic tips, tools, useful links* [online], available at: https://tproger.ru/translations/webapp-security
11. Owasp.org (2017), *The ten most critical threats to the security of web applications* [online], available at: https://www.owasp.org/images/9/96/OWASP_Top_10-2017-ru.pdf
12. Kalchenko, V. (2019), "Analysis of the existing methodology for conducting computer system security audits in government agencies", *Control, navigation and communication systems,* 3(55), pp. 110-114.
13. Kalchenko, V. (2018), "An overview of penetration testing methods for assessing the security of computer systems", *Control, navigation and communication systems,* 50, pp. 109-114.
14. Owasp.org (2017), *The ten most critical threats to the security of web applications* [online], available at: https://www.owasp.org/images/9/96/OWASP_Top_10-2017-ru.pdf
15. Svyrydov, A., Kuchuk, H., Tsiapa, O. (2018), "Improving efficienty of image recognition process: Approach and case study", *Proceedings of 2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies*, DESSERT 2018, pp. 593-597, DOI: http://dx.doi.org/10.1109/DESSERT.2018.8409201
16. Mozhaev, O., Kuchuk H., Kuchuk, N., Mozhaev, M. and Lohvynenko M. (2017), "Multiservise network security metric", *IEEE Advanced information and communication technologies-2017*, Proc. of the 2th Int. Conf, Lviv, pp. 133-136, DOI: https://doi.org/10.1109/AIACT.2017.8020083
17. Nalivaiko, A. (2017). *How to protect a web application: basic tips, tools, useful links*. [online], available at: https://tproger.ru/translations/webapp-security/.

Received (Надійшла) 14.12.2019
Accepted for publication (Прийнята до друку) 12.02.2020

ВІДОМОСТІ ПРО АВТОРІВ / ABOUT THE AUTHORS

Пєвнєв Володимир Яковлевич – кандидат технічних наук, доцент, доцент кафедри комп'ютерних систем, мереж та кібербезпеки, Національний аерокосмічний університет імені М.Є. Жуковського «ХАІ», Харків, Україна;
Volodymyr Pevnev– Candidate of Technical Science, Associate Professor, Associate Professor of Computer Systems, Networks and Cyber security Department, National Aviation University "Kharkiv Aviation Institute", Kharkiv, Ukraine;
e-mail: v.pevnev@csn.khai.edu; ORCID ID: http://orcid.org/0000-0002-3949-3514.

Поповіченко Оксана Миколаївна – студентка кафедри комп'ютерних систем, мереж та кібербезпеки, Національний аерокосмічний університет імені М.Є. Жуковського «ХАІ», Харків, Україна;
Oksana Popovichenko – Student of Computer Systems, Networks and Cyber security Department, National Aviation University "Kharkiv Aviation Institute", Kharkiv, Ukraine;
e-mail: o.popovichenko@student.csn.khai.edu; ORCID ID: https://orcid.org/0000-0002-2083-0314.

Цокота Ярослав Віталійович – студент кафедри комп'ютерних систем, мереж та кібербезпеки, Національний аерокосмічний університет імені М.Є. Жуковського «ХАІ», Харків, Україна;
Yaroslav Tsokota – Student of Computer Systems, Networks and Cyber security Department, National Aviation University "Kharkiv Aviation Institute", Kharkiv, Ukraine;
e-mail: y.tsokota@student.csn.khai.edu; ORCID ID: https://orcid.org/0000-0001-6155-817X.

Технології захисту веб-застосунків

В. Я. Пєвнєв, О. М. Поповіченко, Я. В. Цокота

Анотація. Предметом вивчення в статті є уразливості, які присутні у веб-застосунках. **Метою** є дослідження проблеми порушення інформаційної безпеки веб-застосунків. **Завдання:** ознайомиться зі статистикою веб-атак на веб-застосунки; виявити основні передумови до кібератак; розглянути типи уразливості, які зустрічаються найчастіше; запропонувати способи створення безпечного застосунку. **Використовуваними методами** є: аналітичний метод, аналіз літератури, опис. Отримані наступні **результати**. До кожного наведеного виду уразливості був розглянутий сценарій можливої атаки з боку зловмисника. Також були запропоновані методи для розробників, які дозволяють утилізувати дані уразливості та розробити безпечний веб-застосунок. **Висновки.** Необхідно пам'ятати, що найкращий захист веб-застосунків – написання безпечного коду. Розробники, які реалізують програми, повинні бути заздалегідь поінформовані про існування поширених типів атак та про принципи їх роботи, для того щоб реалізувати захист застосунків і запобігти можливим кібератакам. Найкраще використовувати методи захисту комплексно, щоб максимально захистити веб-застосунок.

Ключові слова: веб-застосунок; вразливість; атака; захист веб-застосунків; безпечний застосунок.

Технологии защиты веб-приложений

В. Я. Певнев, О. Н. Поповиченко, Я. В. Цокота

Аннотация. Предметом изучения в статье являются уязвимости, которые присутствуют в веб-приложениях. **Целью** является исследование проблемы нарушения информационной безопасности веб-приложений. **Задачи:** ознакомиться со статистикой веб-атак на веб-приложения; выявить основные предпосылки к кибератакам; рассмотреть наиболее часто встречаемые типы уязвимостей; предложить способы создания безопасного приложения. Используемыми **методами** являются: аналитический метод, анализ литературы, описание. Получены следующие **результаты.** К каждому приведенному виду уязвимости был рассмотрен сценарий возможной атаки со стороны злоумышленника. Также были предложены способы для разработчиков, которые позволяют утилизировать данные уязвимости и разработать безопасное веб-приложение. **Выводы.** Необходимо помнить, что наилучшая защита веб-приложений – написание безопасного кода. Разработчики, реализующие приложения, должны быть заранее осведомлены об существовании распространённых типов атак и об принципах их работы, для того чтобы реализовать защиту приложений и предотвратить возможные кибератаки. Лучше всего использовать методы защиты комплексно, чтобы максимально защитить веб-приложение.

Ключевые слова: веб-приложение; уязвимость; атака; защита веб-приложений; безопасное приложение.