Global Journal of Computer Sciences: Theory and Research

Volume 12, Issue 1, (2022) 23-31 www.gjcs.eu

Session hijacking vulnerabilities and prevention algorithms in the use of internet.

Elira Hoxha [a1] Department of Statistics and Applied Informatics, Faculty of Economy, University of Tirana, Albania.

Igli Tafa [b], Polytechnic University of Tirana, Faculty of Information and Technology, Albania.

Kristi Ndoni[c], Polytechnic University of Tirana, Faculty of Information and Technology, Albania.

Islam Tahira[d], Polytechnic University of Tirana, Faculty of Information and Technology, Albania.

Andrea Muco[e], Polytechnic University of Tirana, Faculty of Information and Technology, Albania.

Suggested Citation:

Hoxha, E., Tafa, I., Ndoni, K., Tahira, I., Muco, A. (2022). Session hijacking vulnerabilities and prevention algorithms in the use of internet. *Global Journal of ComputerSciences: Theory and Research*. *12*(1), 23-31. https://doi.org/10.18844/gjcs.v12i1.7449

Received from January 15, 2022; revised from Fabruary 15, 2022; accepted from April 01, 2022. Selection and peer-review under responsibility of Assist. Prof. Dr. Ezgi Pelin Yıldız, Kafkas University,

Abstract

The concept of Internet security is studied by computer science as a safe medium for exchanging data while minimizing the likelihood of online threats. The extensive use of advanced web-based software in different industries such as education, retail, medical care, and payment systems, represents a security challenge for the programmers and an opportunity for the hackers to attack through session hijacking. This paper aims to present vulnerability with the respective control mechanisms and to propose an approach for avoiding hijacking threats by using one-time cookies along with other prevention strategies. The study uses a la review of literature, by analyzing resources from existing literature. Based on recent OWASP guidelines, session hijacking of attack is indeed one of the most frequent attacks that happens lately. Session hijacking happens as a result of poorly designed websites and a lack of security mechanisms, where the user's identity and session data are exposed.

Keywords: Cookies**;** internet; security; session hijacking, vulnerability.

***** ADDRESS FOR CORRESPONDENCE: Elira Hoxha*, Department of Statistics and Applied Informatics, Faculty of Economy, University of Tirana, Albania.
Email address: elira.hoxha@unitir.edu.al

1. Introduction

Nowadays, both internet and web technologies are being used widely to access most of the applications that previously ran on desktops and offline [1]. As a result, websites that once were made of static HTML, now have evolved into interactive Web apps, or full of content services, available everywhere on the Internet. Commercial transfers and mail sharing are only two of the utilities provided by web apps. The financial sector, education organizations and other institutions related to it, health care and wellness facilities, various corporate groups, and state institutions, are some of the most influential industries in which web-based technologies are the main tool used for optimizing everyday operational efficiency or used for updating legacy systems. Users acquire information from web servers, as well as database servers, via those applications that are accessed through the internet. These technologies and the applications built through them may have serious security vulnerabilities that can cause basic hacks to occur.

Session bugs are one of the most common risks within web-based applications. This kind of risk is attributed to the web application's poor session control. Attackers take advantage of incorrectly designed websites and hijack user's sessions, which leads to identity theft. Session states store sensitive data, which is an important objective for attackers [2]. Whenever a user has to log in to a protected website, personal information must be filled in to verify the user's legitimacy, including a username, a password, or even a birthdate and other sensitive information, which enables them to verify their identity and also to access information related to their profile. To reduce the complexity of re-authentication, session control requires the web-based application to build a session, for the user to not have to go through this process any time they interact with the application. Session control checks that the user connecting to the web and accessing information from the application is the same user that logged in the first time by completing the personal data. As a result, attackers impersonate existent users and can obtain access to data with no need for authentication. User identifiers are the most popular method of session control.

A session begins whenever a user accesses or logs in to a certain website or application via their device, and ends once the user closes the website (or the application), or logs out of the device. While connected, a session will also briefly store data related to the actions of the user. The primary purpose of the session is to keep track of the user's authentication information and through this working session, the user is allowed to enter the program and use it. A session key, known as SID, would be a name-value combination. The value seems to be a sequence of alphanumeric characters that corresponds to a web session. Every submitted query will have the SID added to it and it can be used as an identity provider inside the program. As a result, the SID must be created and stored safely. Data breaches and session management attacks are also one of the most significant web application security concerns, according to the Open Web Application Security Project (OWASP) [3].

1.1. Purpose of study

This paper presents an analysis of different session-related weaknesses as well as detection and prevention strategies.

2. Materials and Method

The study is a literature review study. The study analyses different session-related weaknesses as well as detection and prevention strategies, making use of existing literature. The data for this research was therefore secondary.

3. Results

Session hijacking is a frequent type of threat. The ease with which attackers can connect directly to the session makes this type of attack very dangerous [4]. Whenever a user is about to log in to a system and a connection with the server has been formed, an intruder can hijack the session while pretending to be the intended user. Once the intruder gets control of the server, he does not need to break the log-in key because he has already been verified to get the connection. Client hijacking is where a hacker gets the user's session key and has complete ownership of the machine, although the session is also running [5]. Session hijacking can be of two types: active or passive.

Active session hijacking occurs when an attacker assumes control of an operating data channel. The intruder silences the user's device and takes over the communication channel between the user and the server [6]. In this case, the intruder typically attacks a valid session and causes a denial of service (DoS). Passive session hijacking is similar to the active one, but rather than removing the user from the active session, the intruder monitors the communication between the server and the user's device. Inside a passive link, the attacker does not block the user out of the session but tracks his details and the ongoing communication exchange to record everything in a personal database for attacking purposes. The intruders generally start their activity through passive session hijacking [7].

The introduction of cookies as session authentication tokens in the mid-90s created privacy issues. Many other studies have shown that web authentication schemes have many flaws, like the susceptibility to session hijacking threats [8;9]. Browsers might have a variety of bugs and webpages are susceptible to cyberattacks very often. It is important to understand this type of threat and address it. Session-related bugs are well-known risks compared to others. Several experts as well as academics already suggested numerous identification and avoidance approaches. That is why security analysts suggest improvements to strengthen authentication cookies' reliability. Jackson et al. [10] proposed cookie systems that use cryptographic approaches to have greater security and credibility. Furthermore, cookie expiry dates are being used to mitigate the effects of session hijacking threats. Most systems, though, utilize longer termination periods to prevent compromising the user's experience, which reduces the feasibility of such a strategy [11;12].

Cache cookies, which are various types of permanent states within the web, have been found helpful to cookies in preserving users' data and session identities. Cache cookies, although immune to phishing threats, require HTTPS security to avoid malicious activities. Felten and Schneider [13] showed how a server can detect the presence of an image file in the cache of a browser, to use cached images as cache cookies. Their techniques are based on timing analysis, so they are relatively difficult to be implemented.

Juels et al. [9] revealed an easier way to exploit cache cookies by using the browser history. For example, Cascading Style Sheets (CSS), a web technology feature for displaying Web Pages allows a server to include code in a website that identifies if a browser has a specific URL in the history page. Ogundele et al. [14] investigate potential consequences of cached data or associated computer capabilities, which offer a good perspective of the site-to-site domain tracing risks for the users. Users

often discover new aspects of cached data, like object identifiers. Dacosta et al. [15] suggest using web applications that impose uniform security standards along with a variety of cross-domain monitoring mechanisms. Session monitoring keeps a record of the user's behavior through several connections to web pages. A common application of monitoring is mostly the sign-in process. A special code is assigned to each user like a connection identifier or a connection token. Several methods can be used to enforce tokens: They are stored in cookies and submitted through secret sectors generated from particular system types; Then the tokens get attached to every connection that the user taps.

For connection monitoring, some programs utilize HTTP verification. Search engines might use HTTP requests instead of the system's Website script, for submitting login information. Other apps utilize session-less mechanisms. They transmit the user's information between every system contact without using tokens. Typically, cryptographic algorithms are being utilized in conjunction with this mechanism [16].

The most serious vulnerabilities related to session hijacking are those connected to the process of token generation and session monitoring strategies. Token generation vulnerability allows hackers to establish a token and as a result, they can use a legitimate token. Tokens may be obtained when putting different elements of users' data, including his username and perhaps his e-mail address. Any hacker may decrypt the token as well as generate a legitimate one when the methods are adjustable. Tokens can be created as a series of alphabetic characters, with the condition that every token is generated randomly [2]. Whenever a token-generating algorithm implements one of these techniques, hackers have better chances of predicting the tokens. Encrypted variations of build tokens are implemented over standard numerical series. Weak token generation is another existing vulnerability. Since machines depend on deterministic processes, they do not seem to offer flexibility. Computers use arbitrary value producers to get around the problem (pseudorandom number generators - PRNG). Distinct input devices, including sound panel performance as well as the number of button presses, get combined to produce PRNGs [16].

Although tokens are produced correctly and are volatile, hackers may be clever enough to tackle them. Hackers will do so by taking advantage of unsecured packets including vulnerabilities within secret algorithms that websites need in producing tokens. Tokens can also be intercepted by looking for them in log data like window reports or proxy logs. Any hacker will retrieve tokens from logs when it’s included in the URL as a variable. Also, they can look for tokens inside the search engine as well as a proxy buffer that will save whole site headers and response headers. Utilizing weak token assignment systems, granting different tokens to similar users, and using fixed tokens for every user, are some of the many weaknesses exploited by the hackers. Furthermore, ineffective session closure strategies open up several cyber threats. The activity must be short and feasible to decrease the contextual timeframe of the threats. Since certain programs do not have a specific procedure for session termination, hackers will seek several different properties before the session ends. Whenever a user signs out, the system deletes the token from the user’s computer, but as the user (either a hacker) has sent any formerly accessed token, the system continues allowing him [11].

In difficult cases, the system gets zero demands during the sign-out process and the connection is not invalidated. When any intruder acquires that key, he will be able to use the session like a user that

has not signed out at all. Eventually, unless tokens are stored inside cookies, cookie variables can be vulnerable to many attacks. Because a protected label is not placed inside cookies, it gets transmitted through unsecured packets [17]. Cross-site scripting threats (XSS) can be used by hackers to capture those variables because the HTTPOnly marker has not been placed. The reach of a cookie may be used by hackers. Some weaknesses are related to incorrect HTTPS utilization. Certain programs recognize HTTPS secured sectors but also use matching tokens out of these sectors. As a result, hackers will get tokens through eavesdropping on HTTP traffic. Furthermore, several systems support HTTP protocol and hackers may persuade users to send HTTP calls so they snatch tokens. Spoofing emails, posters, and psychological manipulation are widely used in several threats. Other apps utilize HTTP for viewing static resources such as photos, scripts as well as CSS. Detecting such inquiries allows hackers to catch tokens.

Hackers will pull out threats including cross-site request forgery (CSRF), session sniffing, predicting, and session fixation. Namitha and Keerthijith [1] defined the triggering weaknesses of every threat that hackers should check before launching a thrust.

Session sniffing threats include finding communications that are not encrypted to get the session id:

In HTTP packets sniffing the HTTP streams are intercepted. Hackers should find data from the user's system or the domain of the software institution's website. Four underlying issues exist. First, non-HTTPS areas of such sites may be identified. Second, a stable marker is not fixed. Third, the service provides HTTP calls to sites that are protected by HTTPS. Consequently, before authorization, the program employs HTTP [18; 19].

In cache sniffing, the token can be obtained in every configuration, where the hacker enters a proxy cache or a search engine. Two triggering weaknesses are related to cache management of Webpage. HTTP response headers do not include instructions. Then CacheControl: private policy hardly allows the buffer to be used on the computer where the user is currently running. With distributed computers, such a scenario poses danger (e.g., in Internet cafes) [20].

In sniffing logs, tokens are obtained through reviewing logs within various structures engaged with the server-client interaction. Tokens are sent through URL variables, which can end up in logs. They can be sent with a secret sector, where GET calls are accepted rather than POST requests by the system. The client-side scripting file may implement this same demand reversal. Token gets submitted as a URL variable, resulting logs will contain it.

In CSRF the user is induced to perform acts inside an environment where they have been authenticated; a common tactic would be to deliver connections via email or instant messages. This threat has the potential to damage the user's details. Distinct from several other threats, CSRF also attempts to perform precise activities rather than gaining session access [21] .

In fixation of the session, the hacker addresses the token before one user's authorization. There are three stages to a hacker's attack:

1. Establish the session. The hacker establishes a so-called “bait session” in the computer and generates the key. Under certain instances, intruders will submit demands during routine periods to maintain the connection active, as part of the session maintenance.

2. Fixation of the session. The token is inserted into the user's machine by the hacker. Conditioned by the process of transmitting each token, various methods are used for session fixation [22]. Hackers will compel their target into tapping one connection built at once using the URL variable. Depending on the existence of a protected sector, the hacker will take advantage of an XSS weakness. Bad configuration on deployment allows session fixation. This kind of vulnerability can be addressed through a successful software product architecture [23].
3. The access to the connection. The hacker awaits around the victim so they access the session before the hacker attempts to get connected himself through two types of connection control algorithms: the rigorous systems, which permit just established predefined tokens, and the tolerant systems that welcome recent tokens, which launch the connection. Hacker offers specific tokens and also utilizes them in weak schemes. In the rigid schemes, the hacker opens one connection which gets left active during the threat.

4. Discussion

There exist different prevention approaches related to session hijacking threats [24]. An interesting suggestion would be using OTC, which are cookies that get substituted like connection authentication keys. This approach offers strong protection against unauthorized access, by making sure that they still meet the needs of dynamically dispensing systems. OTC generally keeps apart from the connection verification function and the connection monitoring one.

The attacker's target throughout this model has to gain access to connections that have been created by the users of a website. Information exchanged between the client's browser and the website is accessible to silent hackers. Data gets caught either by the system or by system log files. Inactive hackers may attempt conducting or reusing authentication tokens related to the knowledge they gained, to sabotage another user's session connection [18]. An aggressive attacker will have similar knowledge possession as an inactive hacker, but he may also selectively manipulate the queries and the replies sent between the web page and the system. A proactive attacker can construct, modify, or prohibit the messages going to the desired target. Any successful attacker may also carry out domain-level threats between the website and the app, such as session fixation, XSS, and cross-site tracing.

Malicious software attempts of capturing OTC tokens or stealing OTC permanent data from the user's device are both options for a practical challenge [25]. Here the threats are not counted under which the attacker assumes possession including his victim's account, as well as the operating system (for example, via leveraging cache overload of malicious programs), but rather threats mostly on software product architecture. Furthermore, OTC does not have security from dishonesty threats. For encrypting the authentication data during the sign-in process, OTC uses HTTPS. Then, as a result, OTC expects HTTPS to be set up properly and securely. Developers do not accept threats that compromise HTTPS's security assurances throughout successful authentication. Attackers might have extracted a user's password, which is a much more useful accreditation.

For providing a reliable and realistic solution for authorization cookies, we define some characteristics that should be present [7]. The framework should have strong user session

authorization and should be automatically protected against session hijacking. This is called connection honesty. For demand authentication, this suggested algorithm doesn't need conditions inside the website. If we consider the system load conditions, they are not different from authorization cookies. Such characteristics are important for massively scalable websites. The theoretical proposed system should be identical to cookies in terms of the user interface. There is no need for extra human interaction. Particularly, switching between the authorization cookies and OTC does not affect the user's functionality. This possible framework will provide authentication tokens fully classified and promises of honesty. Authorization tokens should not release data that undermines website confidentiality, being immune against cryptographic algorithms threats.

OTC generates another specific token. The connection key relates every token to a specific demand; therefore, the token could become reusable over several user demands [20]. Particularly, OTC tickets stored in the state details are needed for evaluating the tokens. Every other card becomes secured using a lengthy code exchanged across servers throughout the website. As a result, data contained throughout the card could become accessed by website servers. Details of the card are not visible to the recipient. Creds are keys saved inside the application, while tokens are the properties added to each requisition.

If a user needs to buy anything, they can submit another message to the server which includes the user's login details. The user would be granted an OTC upon effective authorization. This OTC is used then to verify the user for any request that he creates. When the user replies, the response is also accompanied by an OTC [18].

Proxy seems to be a machine that serves as a middleman between an external system and the internet. Sometimes, rather than utilizing a proxy service on the user’s side, it is used a reverse proxy server (RPS). As a result, RPS must process any call from the user. RPS's goal is to assign OTC, address of IP, connection ID, and the website signature [25,26]. Then RPS would search through these elements with every single arriving message. When either one of the variables changes, RPS will switch to a different tab. This is the system to whom the user sends a message. That server verifies passwords, processes every user demand, and also communicates with the users.

The suggested methodology operates as follows: The user inserts login details. The call gets submitted to the RPS, which collects each user's internet protocol (IP) address, and the search engine signatures and inserts these data into a database [1,27]. The database verifies the password and executes the call. OTC is generated by the RPS, and together with the connection ID is sent to the user. The user can save the OTC that is sent or submit it through RPS within each call that they make. Every submitted call from the user is tested by the RPS. Because OTC, the IP address, the connection ID, or the application signature changes, RPS stops the connection.

5. Conclusions

Session hijacking is a significant issue that any webpage owner or company should deal with and prioritize, to protect any online data. This paper covers some of the weaknesses associated with the process of accessing a website, as well as attacks that an intruder can do to compromise sensitive data. Different ways of connection intercepting attacks are presented and how they impact web operations. Most web-based session hijacking is caused by spoofing hacks, viruses, cross-site scripts, and SQL injection attacks. Strategies towards stopping connection hijacking as well as methods employed

among attackers for committing web-based crimes are addressed. Prior research has also shown that many loopholes remain throughout online payments, necessitating an immediate implementation of such a strong amount of protection in websites that secure personal data during assaults.

Solutions of verification cookies are being suggested, although most have not been implemented yet since they cannot fulfill standards in big scalable online systems. Another drawback is that the suggested solutions need expensive operation coordination through a website, which would be a major problem with scalable applications. The OTC approach proposed is a safe solution for verification cookies. OTC may be hardly immune to session theft however that often keeps cookies convenience as well as consistency advantages. OTC within websites is an essential starting point toward ensuring connection consistency across current operating systems. For supporting OTC, technologies like Java Servlets, and Ruby on Rails must be modified. Since HTTPS is not implemented in several hosting servers, many developers go towards a connection honesty approach that could run across HTTP, while also being protected from inactive system hackers, which seems to be another interesting area of study.

References

[1] Namitha, P. and Keerthijith, P. (2018). A Survey on Session Management Vulnerabilities in Web Application. 2018 International Conference on Control, Power, Communication and Computing Technologies (ICCPCCT), pp. 528-532, doi: 10.1109/ICCPCCT.2018.8574238.

[2] Jeevitha. R, Bhuvaneswari, N. S. (2019). Malicious node detection in VANET Session Hijacking Attack. 2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), 1-6.

[3] Stock, A. V. D., Glas, B., Smithline, N., Gigler, T. (2021). OWASP Top 10:2021. Retrieved 19 May 2022, from https://owasp.org/Top10/.

[4] Czebotar, J. (2006). XSS - Stealing Cookies 101. Retrieved 08 May 2022, from http://jehiah.cz/a/xss-stealing-cookies-101

[5] Shema, M. (2010). Cross-Site Tracing (XST): The misunderstood vulnerability. Retrieved 08 May 2022, from https://deadliestwebattacks.com/appsec/2010/05/18/cross-site-tracing-xst-the-misunderstood-vulnerability.html

[6] Bortz, A., Barth, A., Czeskis, A. (2011). Origin Cookies: Session Integrity for Web Applications. Web 2.0 Security and Privacy Workshop (W2SP 2011) 7.

[7] Sathiyaseelan, A. M., Joseph, V., Srinivasaraghavan, A. (2017). A Proposed System for Preventing Session Hijacking with Modified One-Time Cookies. International Conference on Big Data Analytics and Computational Intelligence (ICBDACI).

[8] Baitha, A. K., Vinod, S. (2018). Session Hijacking and Prevention Technique. International Journal of Engineering & Technology.

[9] Juels, A., Jakobsson, M., Jagatic, T. N. (2006). Cache cookies for browser authentication. Symposium on Security and Privacy. IEEE.

[10]Jackson, C., Bortz, A., Boneh, D., & Mitchell, J. (2006). Web privacy attacks on a unified same-origin browser. In 15th Intl. World Wide Web Conference.

[11]Fu, K., Sit, E., Smith, K. and Feamster, N. (2001). Dos and don'ts of client authentication on the web. USENIX Security Symposium.

[12]Park, J. S. and Sandhu, R. (2000). Secure Cookies on the Web. IEEE Internet Computing, 4:36–44.

[13]Felten, E. W. and Schneider, M. A. (2000). Timing attacks on Web privacy. Proceedings of the ACM Conference on Computer and Communications Security, 25-32.

[14]Ogundele, I., Akinade, A. and Alakiri, H. (2020). Detection and Prevention of Session Hijacking in Web Application Management. International Journal of Advanced Research in Computer and Communication Engineering.

[15]Dacosta, I., Chakradeo, S., Ahamad, M., and Traynor, P. (2012). One-Time Cookies: Preventing Session Hijacking Attacks with Stateless Authentication Tokens. Georgia Institute of Technology.

[16]Silva, K., Vanajakshi, J., Manjunath, K. N., Prabhu, S. (2017). An Effective Method for Preventing SQL Injection Attack and Session Hijacking. 2nd IEEE International Conference on Recent Trends in Electronics Information & Communication Technology (RTEICT).

[17]Jonas, M., Hossain, M., Islam, R., Narman, H., Atiquzzaman, M. (2018). An Intelligent System for Preventing SSL Stripping-based Session Hijacking Attacks. MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM), 2019, pp. 1-6, doi: 10.1109/MILCOM47813.2019.9021026.

[18]Thakkar, N. and Vaghela, R. (2018). Secure Model for Session Hijacking using Hashing Algorithm. Computer Science & Engineering, NSIT, Jetalpur, Ahmedabad.

[19]Hu, Q., and Hancke, G. P. (2017). A session hijacking attack on physical layer key generation agreement, 2017 IEEE International Conference on Industrial Technology (ICIT), pp. 1418-1423, doi: 10.1109/ICIT.2017.7915573.

[20]Prapty, R. T., Azmin Md, S., Hossain, S. and Narman, H. S. (2020). Preventing Session Hijacking using Encrypted One-Time-Cookies, 2020 Wireless Telecommunications Symposium (WTS), pp. 1-6, doi: 10.1109/WTS48268.2020.9198717.

[21]Gill, R., Smith, J., Clark, A. (2006). Experiences in Passively Detecting Session Hijacking Attacks in IEEE 802.11 Networks. 54. 10.1145/1151828.1151854.

[22]Calzavara, S., Rabitti, A., Bugliesi, M. (2018). Sub-session hijacking on the web: Root causes and prevention. Journal of Computer Security. 27. 1-25. 10.3233/JCS-181149.

[23]Visaggio, C. (2010). Session Management Vulnerabilities in Today's Web. IEEE Security & Privacy, vol. 8, no. 05, pp. 48-56. doi: 10.1109/MSP.2010.114.

[24]Baitha, A. K., & Vinod, S. (2018). Session hijacking and prevention technique. *International Journal of Engineering & Technology*, *7*(2.6), 193-198. https://www.researchgate.net/profile/Anuj-Baitha-2/publication/325117343_Session_Hijacking_and_Prevention_Technique/links/5c1a0e8c458515a4c7e9028f/Session-Hijacking-and-Prevention-Technique.pdf

[25]Modi, N. (2020). Comparative Analysis of Session Features in Session Hijacking and Performance Improvement using OTC. https://ijsret.com/wp-content/uploads/2020/04/IJSRET_V6_issue2_309.pdf

[26]Hasan, J., Zeki, A. M. (2019). Evaluation of Web Application Session Security. 2nd Smart Cities Symposium (SCS 2019), pp. 1-4, doi: 10.1049/cp.2019.0178.

[27]Letsoalo, E. and Ojo, S. (2017). Session Hijacking Attacks in Wireless Networks: A Review of Existing Mitigation Techniques. IST-Africa. Africa: Paul Cunningham and Miriam Cunningham.