This paper proposes a method for protecting the access tokens in client-server data exchange without saving the state based on the formation of the signature of the request using a temporary secret. The devised method allows one not to transfer access tokens with each request, which would make it possible for the attacker to authenticate as a valid user when compromising the connection, for example, when using a «person in the middle» attack.

Two variants of the method have been proposed and substantiated – simplified and improved, the scope of which depends on the needs for protection and technical capabilities of their implementation. The robustness of both variants is ensured by the practical inability to select the initial input data of the hash function used to form the signature. The improved version also makes it possible to protect access tokens at the stage of receiving them and provides protection against the attack of the recurrence of the request. Initial user authentication protection is achieved by using the Diffie–Hellman protocol to exchange a secret and access token. Using query IDs and time labels prevents the query from being reused.

Advanced security for access tokens is important because having an attacker's access token gives the attacker full control over the user account. The use of SSL/TLS may not produce the desired level of protection for such important data.

It was established that the use of the proposed method does not add significant time costs. The SHA-256 hash function example shows that the relationship between message size and extra time to send and receive a message is linear. When using the proposed method in the browser, the absolute value of additional time spent for messages from 100 bytes to 2,048 KB ranges from 0.4 ms to 142 ms. Given this, the proposed method could be used without significant impact on the experience of use

Keywords: access token protection, client-server message signature, authentication, session security

UDC 004.056.53
DOI: 10.15587/1729-4061.2022.251570

DEVELOPING THE ALGORITHM AND SOFTWARE FOR ACCESS TOKEN PROTECTION USING REQUEST SIGNING WITH TEMPORARY SECRET

Vasyl Bukovetskyi
Corresponding author
Postgraduate Student*
E-mail: bukovetsky@outlook.com
Vasyl Rizak
Doctor of Physical and Mathematical Sciences, Professor*
*Department of Solid State Electronics and Information Security
State Institution of Higher Education «Uzhhorod National University»
Narodna sq., 3, Uzhhorod, Ukraine, 88000

Received date 27.12.2021
Accepted date 07.02.2022
Published date 28.02.2022

How to Cite: *Bukovetskyi, V., Rizak, V. (2022). Developing the algorithm and software for access token protection using request signing with temporary secret. Eastern-European Journal of Enterprise Technologies, 1 (9 (115)), 56–62. doi: https://doi.org/10.15587/1729-4061.2022.251570*

1. Introduction

The rapid development of the Internet and related technologies has opened the way to the creation of dynamic web resources and client-server applications that can provide the user with personalized services. Such functionality is achieved by constantly exchanging data of the client application with the server. The most common data transfer protocol on the Internet is HTTP, which is a protocol without saving the state. Using the protocol without saving the state requires the constant transfer of identifying user data in each request. One of the most common methods of identification in such communication is to attach a certain token of access to each new request. Examples of such tokens are session ID and JSON Web Token.

The number of client-server applications used in everyday life is growing day by day. Bank branches are closed, instead, they develop applications that make it possible to receive all the services directly from a smartphone. Many states have also taken a course on digitalization, with many public services becoming available online. Now one can launch a business from a computer or smartphone, view medical information, view information about the existing property, and even transfer ownership of it. This gives attackers more and more motives to carry out attacks. Previously, the maximum profit of the attacker could be a profile on the forum; now their goals are accounts in banking and government systems.

Data transmission over the network opens up many opportunities for attacks, the main goal of which can be the access token. Obtaining this information would allow the attacker to be represented by the user even without knowing his basic data to log in to the web service (usually a username and password). Such web services can be online banking, a public service portal, smart home system management, etc.

The most common method of protection is to use SSL/TLS encryption protocols. SSL/TLS are cryptographic protocols that ensure that a secure connection between the client and the server is established. According to Google, as of August 27, 2021, from 79 % to 98 % (depending on the platform) pages were loaded via HTTPS (based on SSL/TLS cryptographic protocols) [1].

It is these cryptographic protocols that the data privacy protection feature relies on in most modern web applications.

However, the HTTPS and SSL/TLS protocols have drawbacks and may not always be used due to technical limitations.

Thus, studies on the development of improved methods of data protection in data transmission systems without preserving the state are relevant.

2. Literature review and problem statement

The danger of a MITM attack is becoming increasingly urgent every year. As already mentioned, most modern web applications put the function of protecting access tokens that are transmitted with each request to the HTTPS protocol.

HTTPS is an extension of the HTTP protocol, where transport is performed on top of the SSL (TLS) protocol. The protocol provides authentication and encryption of transmitted data.

The use of a secure HTTPS connection is actively promoted by developers of modern browsers, so some of the new features and APIs are available only in a protected context [2].

The literature describes in detail the different methods of attacks and vulnerabilities of these widely used protocols.

Paper [3] considers possible backdoors in many popular implementations of the Diffie Hellman algorithm in TLS, especially when used for HTTP transport by browsers.

Internet users depend daily on the HTTPS protocol for secure communication with the sites they intend to visit. Over time, many attacks on HTTPS and the certificate trust model it uses have been implemented and/or improved. Meanwhile, the number of certification authorities trusted by browsers (respectively, trusted by their actual users) has increased while proper analysis of the procedure for issuing basic certificates has decreased. Study [4] investigates and classifies visible HTTPS security issues, conducts a systematic analysis of previous and current calls, and creates a context for future actions. It also offers a comparative assessment of current proposals to improve the infrastructure of certificates used in practice.

The increased implementation of the HTTPS protocol has allowed the creation of an Internet network in which most of the transmitted data is encrypted, but these security achievements often stand in the way of a government that seeks to see and control user communications.

In 2019, the Government of Kazakhstan carried out an unprecedented large-scale HTTPS interception attack, forcing users to trust their own root certificate. The authors of work [5] were able to detect the interception and track its scale and evolution using measurements from vantage points in the country and remote measurement methods. The attack was aimed at connecting up to 37 unique domains, with a focus on social media and communication services indicating a motive – surveillance. The attack affected much of the connections running through the country's largest Internet service provider, Kazakhtelecom. Continuous real-time measurements showed the interception system was shut down after 21 days. Subsequently, the two main browsers (Mozilla Firefox and Google Chrome) completely blocked the use of the root certificate of Kazakhstan. However, this incident sets a dangerous precedent not only for Kazakhstan but also for other countries that may be trying to bypass encryption on the Internet.

In [6], it is empirically assessed whether browser security warnings are effective enough to clearly inform the user of the possible dangers associated with their current connection. The study used telemetry from Mozilla Firefox and Google Chrome browsers, which allowed the authors to analyze more than 25 million warning impressions. During the study, users missed a tenth of Mozilla Firefox malware and phishing warnings, a quarter of Google Chrome malware and phishing warnings, and a third of Mozilla Firefox's SSL warnings. This demonstrates that safety warnings can be effective in practice; security experts and system architects should not reject the purpose of transmitting security information to end-users. The authors also found that user behavior depends on warnings. Unlike other warnings, users continued to miss 70.2 % of Google Chrome's SSL warnings. This indicates that the experience of using the warning can have a significant impact on user behavior.

Consequently, statistics on skipping SSL error warnings call into question the possibility of full use of HTTPS to protect the privacy of access tokens in its current implementation.

In [7], a study of user behavior was conducted to assess whether improved browser security indicators and increased awareness of phishing led to improved user ability to defend themselves against phishing attacks. Participants were shown a number of websites and asked to identify which ones were phishing. To obtain objective quantitative data, eye tracking was used to determine which visual cues attract users' attention when they determine the authenticity of websites. The results show that users have successfully detected only 53 % of phishing websites, even when they have been prepared to identify them, and that they usually spend very little time looking at security indicators compared to website content during evaluation. Interestingly, the general technical knowledge of users does not correlate with improved detection rates.

Paper [8] describes the method of attacking MITM on HTTPS using public and free tools.

From works [3–8] we can conclude that the use of the HTTPS protocol does not guarantee the complete security of the transmitted data. More and more methods of bypassing protection appear each time, which enables attackers, during a known but not yet corrected vulnerability, to carry out a MITM attack.

Because of the nature of the HTTP protocol where each request is completely isolated from the previous ones, an identification system is applied using an access token that can be transmitted in Cookies, in headers, or in the body of a message. An access token can be considered more valuable than other data that are transferred because owning it makes it possible to receive temporary or permanent access to the information system, even without knowing the initial data for authentication.

The problem of stealing the data of the session has already been considered in work [9], which described possible techniques of attacks, as well as methods of counteraction. The proposed methods of counteraction are the use of protocols such as HTTPS, constant reauthorization of the user, encryption of access tokens at the application level. Paper [10] describes session hijacking protection based on linking the access to the client's IP address and web browser data.

3. The aim and objectives of the study

The purpose of this work is to devise a user identification algorithm in applications that use the HTTP or HTTPS protocol, in which access tokens will not be sent with each new request. This would reduce the possible damage from

the attack when compromising the methods of protection of the highest level.

At the same time, the proposed method should be easy to implement, understandable, and not create additional strong load on the server and client.

To accomplish the aim, the following tasks have been set:

– to construct an algorithm for exchanging requests without constant transfer of access markers in open form, as well as its implementation;

– to analyze possible methods of bypassing the algorithm and build its improved version;

– to determine the influence of the proposed algorithm on the speed of data exchange.

4. The study materials and methods

To construct the algorithm, we have selected the JavaScript programming language, which is the de facto primary language for developing the client part of web applications. PHP is selected to develop the server part.

The choice of PHP as a language for the development of the server part is due to its widespread use in the development of web applications; it is used by 78.3 % of websites [11]. Using JavaScript to develop a client part is obvious – it is the primary language for developing the client side of websites, and can also be used to develop full-fledged applications.

Servers were used to execute the server part of the code: Supermicro X9SCI (Taiwan production) with Intel® Xeon® E3-1230 processor (release – second quarter of 2011, frequency 3.20 GHz, 4 core/8 streams), 12 GB of DDR3 RAM with a frequency of 1,600 MHz and two Seagate ST4000NM033-3ZM hard drives (manufactured in China) combined into a Level 1 RAID array. Ubuntu 20.04 operating system with PHP version 7.2; server with Intel ® Xeon® E-2176G (release – third quarter of 2018, frequency 4.70 GHz, 6 cores/12 streams), 64 GB of DDR4 RAM with a frequency of 2,666 MHz, and two hard drives HGST HUH721008AL (made in Thailand) are combined into a Level 1 RAID array. Ubuntu 20.04 operating system with PHP version 7.2.

To build the client part, the Google Chrome browser version 93 was used, running on a computer with the Windows 10 operating system version 20H2, on an AMD FX-8300 processor (3.3 GHz, 8 cores, 8 streams), 16 GB of DDR3 memory, and an SSD with a Kingston SHFS37A120G disk.

5. Results of studying methods for protecting access tokens based on the signature of requests with a temporary secret

5. 1. Results of constructing an algorithm for exchanging requests without constant transmission of access tokens in an open form

The most common procedure of data exchange between the client and the server has been considered. The client initiates a new session by sending his username and password in the body of the first request. The server may respond negatively (if the login information is incorrect) or positively. In the case of a positive response, the server sends access tokens. With each subsequent request, the client sends the received access tokens in the body of the request, in a separate header, or in a cookie. According to the received access token, the server would look for the user corresponding to it in the database, and, in relation to this, it would form its request.

Fig. 1 shows a screenshot of Charles Web Debugging Proxy, which makes it possible to intercept and analyze traffic. Fig. 1 demonstrates the access token that is transmitted in the cookie header. In the case of compromise of encryption, the attacker could have access to it.

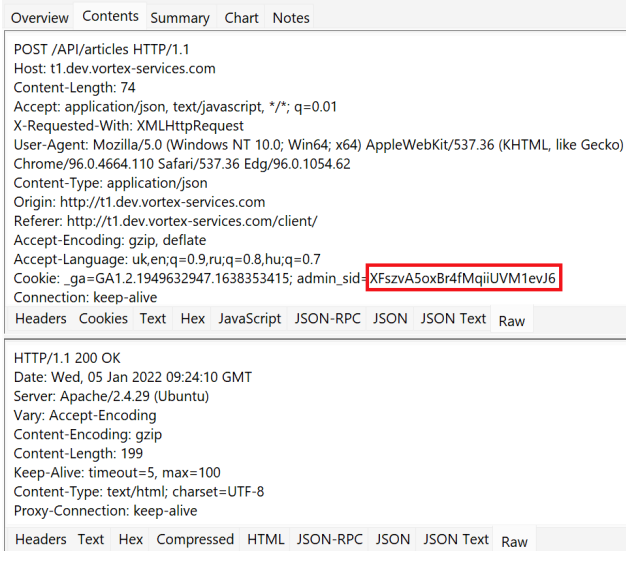

Fig. 1. Unsecured request to the server in which the access token is transmitted in the cookie header, in the sid field (circled in red)

To solve the problem with the constant transfer of the token, one needs to make some changes to this procedure. With initial successful authorization, instead of an access token, the server would send the token ID and secret. At the same time, the server must remember the affination of the user – the identifier of the token – the secret. The received token ID and the secret the client remembers by the end of the session.

For each new server request, the client prepares the following data: query body, token ID, salt and query signature. To form a signature, the client combines the body of the request, secret, and salt, and applies the hash function; its result would be the signature of the current request. After that, the client can send the relevant data through the relevant transport protocol, for example, HTTP. In this case, the token ID, salt, and signature of the request would be sent via HTTP headers. Fig. 2 shows the scheme of operation of such an algorithm.

The algorithm does not involve the use of any particular hash function but the main part of the defense mechanism depends on it. It can be argued that as long as the source data cannot be retrieved from the result of the hash function, the corresponding access tokens cannot be obtained from the query signature.

Accordingly, the algorithm should use a hash function, which is cryptographically robust, and, accordingly, for which no methods for selecting collisions were found. Because of this, the use of such popular hash functions as MD5 [12–14], SHA1 [15, 16] is not recommended.

However, the algorithm is not tied to a specific hash function and, if a vulnerability is found, would make it possible to quickly switch to a safer alternative.

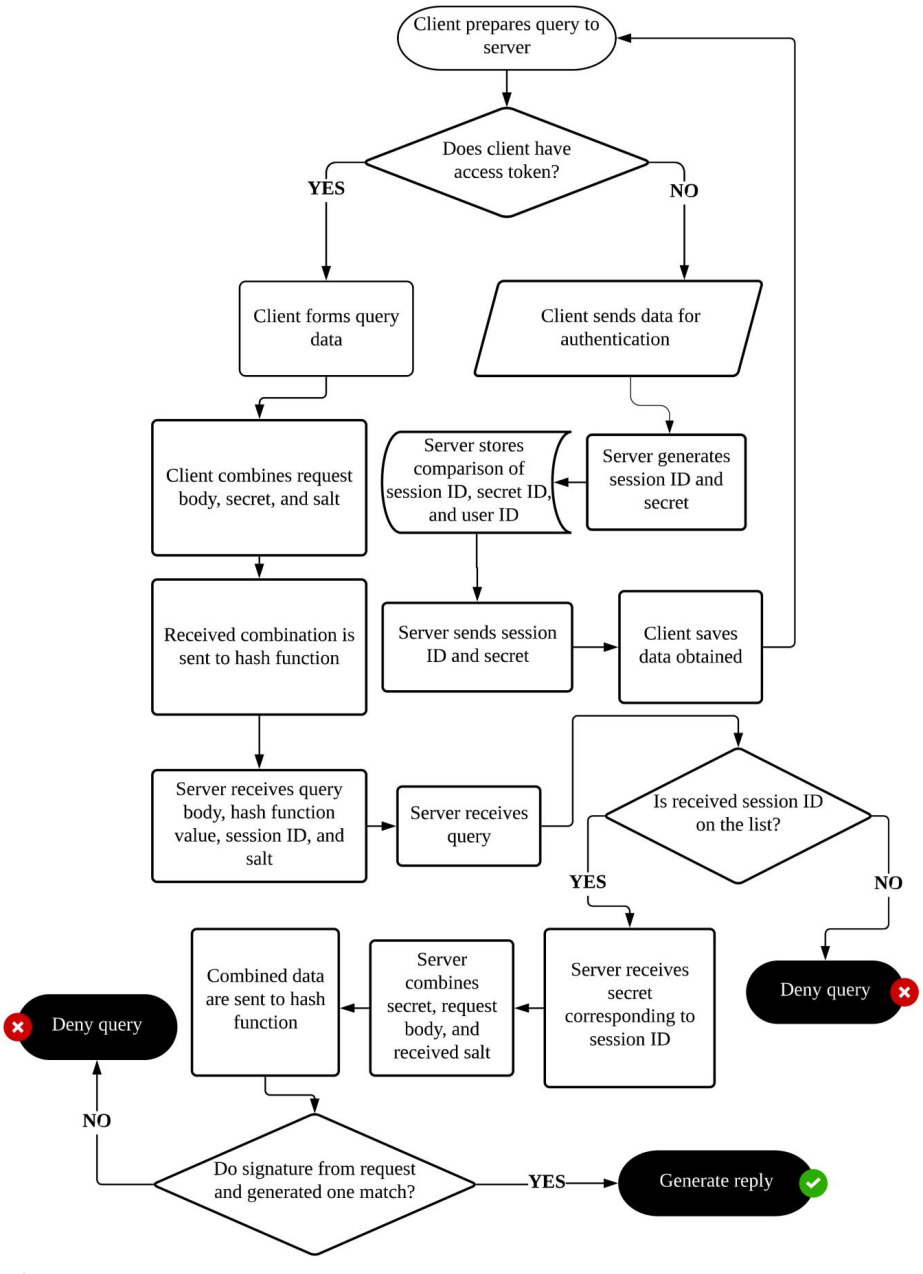

Fig. 2. Flowchart of the basic messaging algorithm without constant transfer of the access token

When using the developed algorithm, the data sent with each request would not be enough for the attacker to form a new query under the user's guide. Of course, this method of transfer will not protect the data sent from disclosure but could significantly reduce the chances of an attacker gaining full control over the user account.

5. 2. Results of analysis of possible methods of bypassing protection and development of an improved algorithm by requests without constant transmission of access tokens in an open form

It should be noted that access tokens can be intercepted during the initial secret exchange, so at this stage, it is recommended to use the Diffie-Hellman protocol (or similar asymmetric protocol) to exchange keys and subsequent encrypted secret transmission.

Obviously, a symmetrical method is not suitable for this task because the client and server cannot know in advance the key with which it would be possible to perform a secure exchange of access tokens.

In addition, this method does not protect against attack with a repeat of the request. If an attacker intercepts a communication channel and receives one of the requests sent by the user, he will be able to send it again and the server, having checked all the data, will accept this request as authentic. This can lead to significant problems because such a request can be, for example, obtaining the latest banking operations, which will make it possible to receive permanent fresh data while the current access token is valid.

To correct this shortcoming, one needs to add several adjustments to the procedure. When exchanging access tokens, the server must forward its current time. In this case, the client should remember the difference in the time and time of the server. When creating a new server request, the client must add their current time to each request, taking into consideration the difference between the server and their own.

In addition to the current time, one also needs to send a fairly long random value, which, in addition to sending in an open view, one needs to attach to the message before creating the signature value. Similarly, one must add a client timestamp to the string that will be passed to the hash function to generate the signature. It is most convenient to use a timestamp in UNIX format for this because the algorithm uses only the calculation of the absolute time difference in seconds.

The server must keep a history of random values and reject all requests if random values are repeated. Adding a timestamp to a query will make it possible for only a short-term history of random values to be kept in memory. If the difference between server time and a label in a message varies by more than the predefined storage time of random values, then such requests can be automatically rejected. Under normal circumstances, it is enough to store random values for a few seconds – enough time to initialize the connection and correct other possible delays on the network; because of this, the base of such values will not be large. The scheme of work of the improved algorithm is shown in Fig. 3.

To store random values, one can use a database of key-value type, such as Redis, Memcached, and others. This will make it possible to make only minimal delays in obtaining and storing this data, and will also make it possible to scale the system more efficiently for large loads.

There are only two possible ways to obtain an access token using a MITM attack: by finding the initial data that were used in the message signature and intercepting access tokens at the session authorization stage. The security guarantee in the first case is the irreversibility of the hash function [17], which will be used to form the signature of the message, until the result of the hash function can produce the initial data until one can get access tokens that were part of the generated signature.

Based on the practical inability to reverse the conversion of the hash function, the server can be sure of user authentication because only the correct user has one of the parts of the input data of the following function: the secret of the access token.

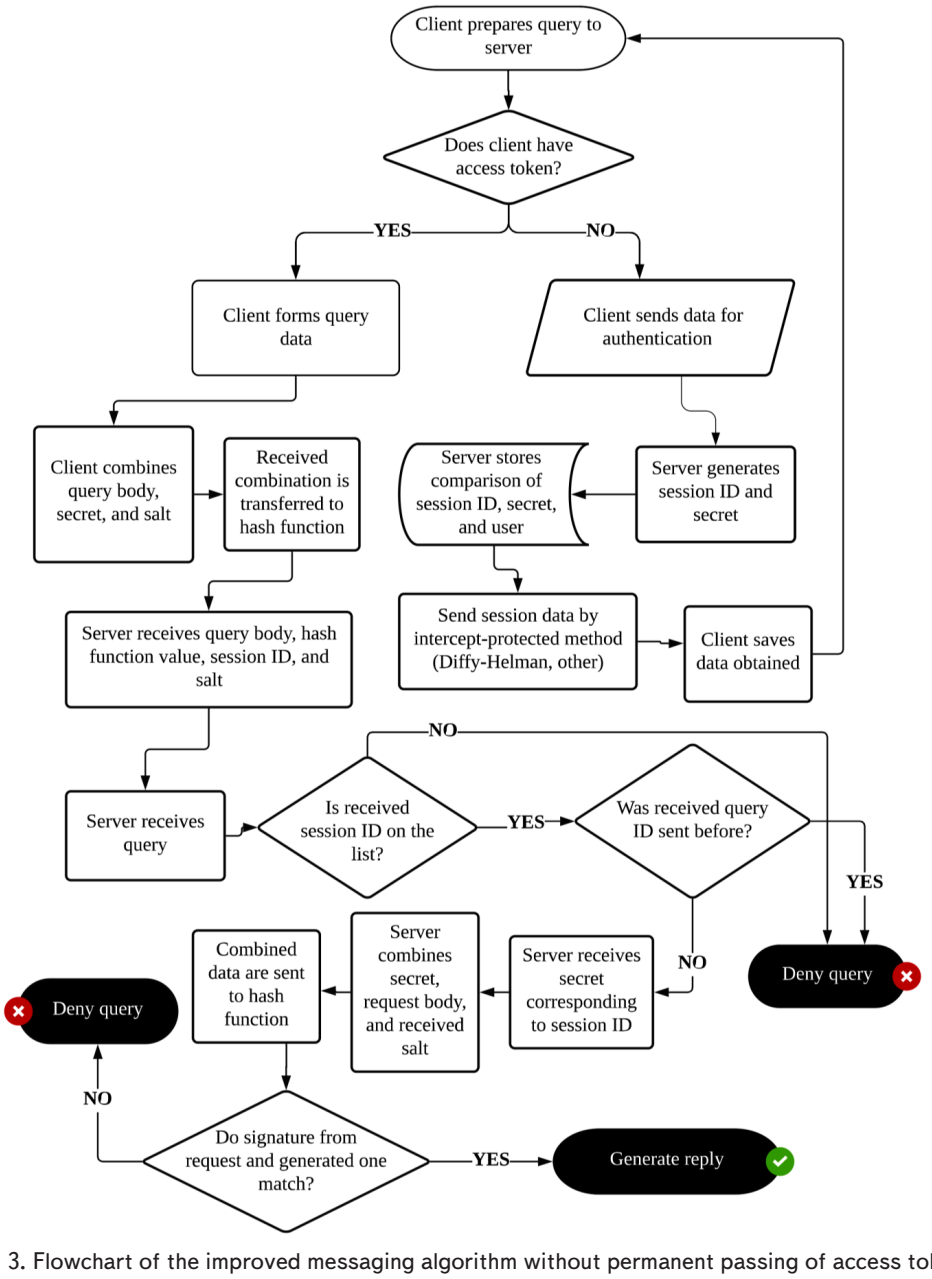

Fig. 3. Flowchart of the improved messaging algorithm without permanent passing of access token

5. 3. The impact of the proposed algorithm on the speed of data exchange

To determine the feasibility of the developed algorithm, it is necessary that the overheads are not too large and do not significantly slow down the communication between the server and client. To determine the effectiveness, the implementation of the protocol using PHP languages – the server part and JavaScript – the client part was implemented.

Measurements were carried out using the built-in JavaScript capabilities, from the beginning of the query formation to the receipt of the full server response. For each of the sizes of the message, 100 measurements were carried out. Table 1 gives the average time spent per query.

Table 1

Time spent on different-size data transmission, protected and unprotected by the constructed algorithm

Message size	Time with unprotected connection (server with processor Intel® Xeon® E3-1230)	Query time when using simple protection (server with processor Intel® Xeon® E3-1230)	Time with unprotected connection (server with processor Intel® Xeon® E-2176G)	Query time when using simple protection (server with processor Intel® Xeon® E-2176G)
100 bytes	43.2 ms	43.6 ms	39.3 ms	39.4 ms
1 Kb	50.2 ms	51.1 ms	43.4 ms	44.5 ms
8 Kb	52.9 ms	54.2 ms	48.1 ms	50.1 ms
16 Kb	55.8 ms	57.9 ms	50.5 ms	51.3 ms
32 Kb	61.1 ms	63.8 ms	57.6 ms	59.7 ms
64 Kb	71.4 ms	76.3 ms	64.8 ms	68.3 ms
128 Kb	85.1 ms	93.1 ms	75.9 ms	81.8 ms
256 Kb	122 ms	137 ms	110.9 ms	123 ms
512 Kb	194 ms	222 ms	174.2 ms	197.4 ms
1024 Kb	380 ms	452 ms	320.1 ms	381 ms
2048 Kb	890 ms	1,032 ms	798.4 ms	903 ms

As can be seen from the measurement results, the increase in costs is almost linear. Moreover, on small messages, additional costs are 10 or more times less than the time to transmit a message by the usual method.

6. Discussion of results of the development of the access token protection algorithm

The built algorithms rely on the stability of the hash function to determine the initial input parameters. As long as this statement is true, it will not be possible to reproduce enough information to represent the user from the message being transmitted.

Unlike SSL/TLS protocols, the developed algorithms make it possible to protect access tokens not just by encrypting them but generally excluding them from transmitted messages. At the same time, the improved algorithm also excludes the open transfer of access tokens at the initial stage of connection. In combination with SSL/TLS or another encryption method, the developed algorithm will significantly improve the security of the connection.

As one can see from the data in Table 1, the use of our algorithms does not add significant time costs in the exchange of data. At the same time, the costs on small volumes are small enough to be completely invisible to the user. At the same time, the increase in time spent remains almost linear, as in the case of an unprotected connection. Using a more powerful server does not significantly improve the speed of the algorithm. The most resource-consuming part of the algorithm is the calculation of the result of the hash function, while both the client and the server perform almost identical operations. With high probability, the client will use less powerful equipment than the server, so, first of all, all future optimizations and improvements in terms of speed should be carried out in the client part of the algorithm.

Storing random values to control query duplicates does not make noticeable slowdowns in the data exchange process, so the average difference between queries that used and did not use comparisons of random values with previous ones was ~1.2 ms.

The use of the Diffie-Gelman algorithm does not affect the overall speed of operation because it is used only at the initial stage of obtaining an access token.

The largest time spent is when calculating the hash function for the signature. It should be noted that for measurements, the implementation of a hash function written in JavaScript was used, the use of more optimal tools for this task may reduce the time in the proposed algorithm.

The proposed algorithm has some fundamental limitations. The algorithm protects only access tokens from interception, not the entire message. The attacker will still be able to read the contents of the message if additional security methods are not used.

The proposed solution also has drawbacks. The solution is characterized by additional overhead costs for the calculation of the signature. Moreover, these costs increase with the size of the message. On very large messages (for example, when transferring files), such costs can become noticeable to the user. This disadvantage can be eliminated by using a more optimized algorithm for calculating the hash function.

The proposed solution can be used in practice in its current form. However, additional analysis and verification are required for real use.

7. Conclusions

1. A platform-independent algorithm and client-server messaging software have been created that allows one not to transmit access tokens with each request. Unlike security, based solely on the use of SSL/TLS encryption protocols, the developed algorithm significantly reduces the likelihood of receiving access tokens in case of interception and decryption of a message by an attacker. This is achieved by reducing the frequency of transmission of access tokens in the data exchange process. The algorithm is easy to implement because it is based on functions already available in any modern programming language. The algorithm can easily be integrated into existing systems. The proposed algorithm can be used in

pairs with HTTPS to enhance the protection of client-server communication and privacy.

2. Possible attacks on the proposed algorithm, such as the attack of re-query and interception of markers at the stage of the first exchange, have been considered. An algorithm with improved protection has been proposed. In the improved version of the algorithm, the process of initial exchange of access markers to increase the level of security has been changed. In addition, the improved algorithm was supplemented with verification for re-sending the message. The improved algorithm is also easy to implement and can be easily integrated into existing systems. Unlike the solutions proposed in [9, 10], the developed algorithms significantly improve the protection of the session from MITM attacks (man in the middle). This is achieved due to the lack of sufficient data in the messages to form a message on behalf of the user.

3. It is established that the developed protection algorithms do not add significant additional time costs to the transmission of messages. On small messages, the increase in time does not exceed 10 ms, such additional costs will not be noticeable to the user. On large messages (up to 2 MB), additional time costs are up to 140 ms, which will also be practically not noticeable to the user in most scenarios of using the algorithm, while the increase in time costs occurs linearly, relative to the size of the message.

References

1. HTTPS Encryption on the Web. Google Transparency Report. Available at: https://transparencyreport.google.com/https/overview?hl=en
2. Features restricted to secure contexts. Available at: https://developer.mozilla.org/en-US/docs/Web/Security/Secure_Contexts/features_restricted_to_secure_contexts
3. Dorey, K., Chang-Fong, N., Essex, A. (2017). Indiscreet Logs: Diffie-Hellman Backdoors in TLS. Proceedings 2017 Network and Distributed System Security Symposium. doi: https://doi.org/10.14722/ndss.2017.23006
4. Clark, J., van Oorschot, P. C. (2013). SoK: SSL and HTTPS: Revisiting Past Challenges and Evaluating Certificate Trust Model Enhancements. 2013 IEEE Symposium on Security and Privacy. doi: https://doi.org/10.1109/sp.2013.41
5. Raman, R. S., Evdokimov, L., Wurstrow, E., Halderman, J. A., Ensafi, R. (2020). Investigating Large Scale HTTPS Interception in Kazakhstan. Proceedings of the ACM Internet Measurement Conference. doi: https://doi.org/10.1145/3419394.3423665
6. Akhawe, D., Felt, A. P. (2013). Alice in Warningland: A Large-Scale Field Study of Browser Security Warning Effectiveness. 22nd USENIX Security Symposium. Washington, 257–272. Available at: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/akhawe
7. Alsharnouby, M., Alaca, F., Chiasson, S. (2015). Why phishing still works: User strategies for combating phishing attacks. International Journal of Human-Computer Studies, 82, 69–82. doi: https://doi.org/10.1016/j.ijhcs.2015.05.005
8. Chordiya, A. R., Majumder, S., Javaid, A. Y. (2018). Man-in-the-Middle (MITM) Attack Based Hijacking of HTTP Traffic Using Open Source Tools. 2018 IEEE International Conference on Electro/Information Technology (EIT). doi: https://doi.org/10.1109/eit.2018.8500144
9. Kumar Baitha, A., Smitha Vinod, P. (2018). Session Hijacking and Prevention Technique. International Journal of Engineering & Technology, 7 (2.6), 193. doi: https://doi.org/10.14419/ijet.v7i2.6.10566
10. Singh, T., Meenakshi (2020). Prevention of session hijacking using token and session id reset approach. International Journal of Information Technology, 12 (3), 781–788. doi: https://doi.org/10.1007/s41870-020-00486-w
11. Historical trends in the usage statistics of server-side programming languages for websites (2021, November 1). W3Techs. Available at: https://w3techs.com/technologies/history_overview/programming_language
12. Dougherty, C. R. (2008). MD5 vulnerable to collision attacks. Vulnerability Note VU#836068. Software Engineering Institute. Carnegie Mellon University. Available at: https://www.kb.cert.org/vuls/id/836068/
13. Wang, X., Yu, H. (2005). How to Break MD5 and Other Hash Functions. Lecture Notes in Computer Science, 19–35. doi: https://doi.org/10.1007/11426639_2
14. Libed, J. M., Sison, A. M., Medina, R. P. (2018). Enhancing MD5 Collision Susceptibility. Proceedings of the 4th International Conference on Industrial and Business Engineering. doi: https://doi.org/10.1145/3288155.3288173
15. Wang, X., Yin, Y. L., Yu, H. (2005). Finding Collisions in the Full SHA-1. Lecture Notes in Computer Science, 17–36. doi: https://doi.org/10.1007/11535218_2
16. Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y. (2017). The First Collision for Full SHA-1. Lecture Notes in Computer Science, 570–596. doi: https://doi.org/10.1007/978-3-319-63688-7_19
17. Goldreich, O. (2001). Foundations of Cryptography. Volume 1: Basic Tools. Cambridge University Press. doi: https://doi.org/10.1017/cbo9780511546891