Securing Client and Server Connections from Parameter Tampering Using Time-Based Hash Encryption

[1]Chandra Yogatama, [1]*Oktoberty, [1]Rita Hariningrum, [1]Sri Pramono, [1]Riyanto Wibowo
[1]Maritime Faculty, Universitas Ivet, Semarang, Central Java, Indonesia

Tjokro Hadi.
[2]Politeknik Negeri Semarang, Central Java, Indonesia

Abstract:- This paper proposes a method to prevent parameter tampering on web services. Time based hash encryption is able to give unpredictable output which changes periodically ac-cording to input. The proposed method uses different parts of function output as authenti-cation key and parameter order. The presented method has several characteristics. Each encryption and decryption needs to be able to generate the same VST. Vst used must have a periodic record so that it is pos-sible to get Vs (t-1).Each used Vst works without trouble on both time variants (tests 1-3) and invariant (tests 3-5). From the Decrypt column, we can see that, as a side effect, each pass parameter is decrypted and cut off. To evaluate proposed work, we make software to encrypt and decrypt a request. We limit the test to having no delay.

Keywords:- *Securing Client, Server Connections, Tampering, Encryption.*

I. INTRODUCTION

Online software is software that needs an internet connection. Each operation uses a request package sent from online software to the server and replied with data. This process often uses unsafe connections prone to attacks. Online software attacks require four steps [1], which are information gathering, assessing vulnerability, launching an attack, and cleaning up. Information gathering is collecting transported data in a network, TLS (Transport Layer Security) is able to avert this [2]. Unfortunately, TLS is not reliable and needs a secure endpoint [3]. Encryption can prevent assessment of vulnerability [4]–[6]. Launching an attack and cleaning up is avertable by authentication [7].

One of the known attacks is parameter tampering [8]. It works by assessing package vulnerabilities and then launching attacks by modifying or creating packages able to bypass the rules [9].

II. PARAMETER TEMPERING PREVENTION

Tampering prevention by requiring connection divides authentication into the following.

A. single line authentication

This method uses one line for requests and authentication. A season key is given after the login process. A later request will be made using the received season key, [10], prone to hijacking.

B. double line authentication

This method uses a separate connection for the request and authentication.

a) Token generator.
This method uses a separate device to generate tokens that were previously synchronized with the server before being handed to the user [7]. This method uses one line that serves as a connection between software and the server, as the token connection has already been severed when the user receives the token.

b) SIM card
With SIM cards, the system could use different communication lines to exchange token information [11]. It limits It is used in some devices that run SIM cards, such as mobile phones, and will only be usable if the SIM card is able to receive a signal.

c) RFID
An authentication token using RFID or other verification objects that can emulate RFID is also viewable. Any device that hosts a client must have an RFID reader. And because the token is always the same, if an attacker knows it, a new request from an attacker using this token will always be accepted by the server.

C. Common vulnerability

Each method presented is prone to hijacking executed on the information gathering step of an online software attack [1]. This vulnerability is caused by each data package having a common form, which means the attacker only needs to fill in the authentication parameters [8], [12] either by interjects or guessing [13]. Guessing includes repeatedly entering possible text strings, commonly known as brute force attacks [12].

In this paper, we propose a method to prevent hijacking and brute-forcing by making the package's format randomly generated. It will serve as encryption, which will be decrypted when the package arrives. A time hash is a time-input hash encryption capable of generating random numbers on a regular basis [14], [15]. A key and package format will be generated using the output [9]. The server will be able to understand it if it knows the exact time it was encrypted. A time zone was chosen and implemented on both the online software and the server to do this. The next section will cover the design and testing of our proposed method.

III. PROPOSED WORK

This section describes our proposed method and how to put it into action on a package.

A. Design

The main objective of our method is to create dynamic keys that change from time to time. Our aim isn't fully random but di-rected randomness, so it is able to be used for authentication.

From above, we conclude that we should use hash encryption with time-based varia-bles as input. Where Cdh denotes dark hash cryptography, Vst denotes a time-based variable, and HAs denotes output..

$$HAs = Cdh(Vst) \quad (1)$$

Dark hash cryptography (Cdh) is multi-ple hashes combined into a function. A time-based variable (Vst) is a variable that changes according to time. This variable can be called a time-synch variable as this variable is needed to generate the same output if calculated at the same time in dif-ferent locations. If the same VST is used, the function's output (HAs) will always be the same.We use hash because it's irre-versible, hence the input will always be un-known. The server needs to process re-quests at the same time to use the same in-put (Vst). HAs are used as a base to decide parameter order and authentication key. Pa-rameter order is decided by the LSB (less significant bit) and the authentication key by the MSB (most significant bit). This pro-cedure encrypts the package before it is re-quested by the server, also calculates HAs to decrypt received packages.

Step	Output
0	www.website.com/api.php?service=01&user=0001&pass=3342...E
1	www.website.com/api.php?1=01&2=0001&3=3342...E
2	www.website.com/api.php?1=01&2=0001&33342...E &4=DDDD…54321
3	www.website.com/api.php?1=01&2=0001&3=3342...E&4=FFFF...0&5=DDDD... 54321
4	www.website.com/api.php?1=01&2=0001&3=3342&4= FFFF &5= DDDD
5	www.website.com/api.php?1=DDDD&2=FFFF&3=3342&4=0001&5=01

Table 1: Test result

Here are the 5 steps of the proposed en-cryption.

- Change the parameter name
- Add HAs as a new parameter
- Add fake parameter
- Reduce the LSB of parameters that aren't as important.
- Shuffle parameters using the LSB of Has

Step 1 is used to disguise the purpose of each parameter. Step 2 is adding the authentication key as a parameter for authentication. Step 3 is used to make each operation have the same number of parameters to camouflage the type of service. The parameter added is randomly generated as it is not needed later and must be inserted between the required parameter and authentication key. Step 4 is used to equate less important parameters with fake parameters by likening their length.

The designer must understand which pa-rameters are not allowed to be cut. Step 5 is used to change the parameter order periodi-cally.

Here is the required step to decrypt the encryption above.

- Calculate HAs
- Validation
- Extract parameters
- Try to process it
- If there is an error, go back to step 1 and use Vs (t-1)

Step 1's goal is to get the parameter's or-der and authentication key. Step 2 is check-ing the authentication key before com-municating with the server for further package processing. Step 3 is to get the oth-er parameters from the received package. Step 4 is to try to execute the request using variables extracted from the package. Step 5 is to accommodate the delay in transportation time.

B. Testing

In this section, we will apply the above method to secure a simple get request. The encryption begins with the requested form and proceeds for every 5 steps of encryp-tion, which is decrypted later. Each encryption step output is shown on Table 1 with explanations given below.

- Step 0 is a simple Get request serving as encryption target.
- Step 1 is output after changing the pa-rameter name
- Step 2 is after adding HAs which become parameter 4. HAs calculated using GMT hour-minute (HH:MM) as Vst and MD5 is used as Cdh.
- Step 3 is adding fake parameter between real parameter and HAs. parameter 4 be-come fake parameter and HAs is moved to parameter 5.
- Step 4 is removing the LSB of other pa-rameters so they have the same length as the most important parameter, which is "us-er" or parameter 2.
- Step 5 is reordering parameters using the LSB of HAs. Then, this request is sent to the server to be decrypted later.

We follow decryption design to process encrypted request. Which is the output of step 5.

- Step 1 calculates HAs using the current time, which is DDDD... 54321. As the au-thentication key, take the MSB part by the mode of length of the parameter while the order takes the LSB part. So, authentication key = DDDD and order = 54321. Authenti-cation keys are on the highest number of orders where service is the lowest, followed by user and pass.
- Step 2 is validation. We can see that the highest number is on the first digit, so the authentication key is on the first parameter. The request is valid if we both have the same key. Step 3 is extraction. We get 3 parameters: service = 01, user = 0001, and pass = 3342.
- Step 4 is executing the parameters. Any service that uses cut parameters must modi-fy its process so it is capable of using that parameter.
- If a process is unable to execute using cut parameters, then those parameters must not be cut on step 4.
- Step 5, if there is a problem faced on step 2, 3, or 4, repeat from step 1 with Vs (t-1) accommodating a possible delay re-sulting in wrong HAs.

We can see from the table that our meth-od is able to encrypt a simple GET request and decrypt it to successfully receive each parameter with one parameter received cut.

IV. RESULTS AND DISCUSSION

To evaluate proposed work, we make software to encrypt and decrypt a request. We limit the test to having no delay. The results are given in Table 2.

From table 2, our proposed work suc-cessfully encrypts and decrypts on any Vst, which in this case means GMT current time. Each used Vst works without trouble on both time variants (tests 1-3) and invari-ant (tests 3-5). From the Decrypt column, we can see that, as a side effect, each pass parameter is decrypted and cut off. To ac-commodate this, we need to modify the service on the server side to be able to work with it.

Test	Request	Encrypted		Decrypt		V_{st} (GMT)
	service=01	1=01	2=3a13	s=01		
1	user=0001	3=d321	4=0001	u=0001	Key=a3d3	20:00
	pass=d321aa	5=a3d3		pass=d321		
	service=01	1=27ac	2=01	s=01		
2	user=0001	3=d321	4=0001	u=0001	Key=9e34	20:01
	pass=d321aa	5=9e34		pass=d321		
	service=01	1=01	2=341e	s=01		
3	user=0001	3=d321	4=0001	u=0001	Key=7871	20:03
	pass=d321aa	5=7871		pass=d321		
	service=01	1=01	2=c2a8	s=01		
4	user=0001	3=d321	4=0001	u=0001	Key=7871	20:03
	pass=d321aa	5=7871		pass=d321		
	service=01	1=01	2=17ea	s=01		
5	user=0001	3=d321	4=0001	u=0001	Key=7871	20:03
	pass=d321aa	5=7871		pass=d321		

Table 2: Experimental results

V. CONCLUSION

This paper proposes a method to prevent parameter tampering on web services. Time based hash encryption is able to give un-predictable output which changes periodi-cally according to input. The proposed method uses different parts of function output as authentication key and parameter order. Unpredictable authentication keys and parameter orders that are only active for a limited time will prevent hijacking and brute force attacks, effectively preventing parameter tampering.

The presented method has several char-acteristics. Each encryption and decryption needs to be able to generate the same VST. Vst used must have a periodic record so that it is possible to get Vs (t-1). Each pa-rameter must be analyzed first to determine whether cutting it is possible or not. The presented method encrypts sent package parameter wise, so it is able to be used in conjunction with another package encryp-tion method that targets package.

REFERENCES

[1.] N. Hoque, M. H. Bhuyan, R. C. Baishya, D. K. Bhattacharyya, and J. K. Kalita, “Network attacks: Taxonomy, tools and systems,” *J. Netw. Comput. Appl.*, vol. 40, no. 1, pp. 307–324, 2014, doi: 10.1016/j.jnca.2013.08.001.

[2.] F. De Backere *et al.*, “Design of a security mechanism for RESTful web service communication through mobile clients,” *IEEE/IFIP NOMS 2014 - IEEE/IFIP Netw. Oper. Manag. Symp. Manag. a Softw. Defin. World*, 2014, doi: 10.1109/NOMS.2014.6838308.

[3.] E. Ronen, R. Gillham, D. Genkin, A. Shamir, D. Wong, and Y. Yarom, “The 9 Lives of Bleichenbacher’s CAT: New Cache ATtacks on TLS Implementations,” *Proc. - IEEE Symp. Secur. Priv.*, vol. 2019-May, pp. 435–452, 2019, doi: 10.1109/SP.2019.00062.

[4.] A. Solichin, M. Andika Putra, and K. Diniari, "RESTful Web Service Optimization with Compression and Encryption Algorithm," in *Proceedings - 2018 International Seminar on Application for Technology of Information and Communication: Creative Technology for Human Life, iSemantic 2018*, 2018, no. January, pp. 333–337, doi: 10.1109/ISEMANTIC.2018.8549727.

[5.] Palanivel Rajan D and Dr. S. John Alexis, "Comparative Study on Data Encryption Algorithms in Cloud Platform," *Int. J. Eng. Res.*, vol. V6, no. 10, pp. 126–129, 2017, doi: 10.17577/ijertv6is100070.

[6.] P. Princy, "a Comparison of Symmetric Key Algorithms Des , Aes , Blowfish ," *Int. J. Comput. Sci. Eng. Technol.*, vol. 6, no. 05, pp. 328–331, 2015, [Online]. Available: http://www.ijcset.com/docs/IJCSET15-06-05-055.pdf.

[7.] A. Abdellaoui, Y. I. Khamlichi, and H. Chaoui, "A Novel Strong Password Generator for Improving Cloud Authentication," *Procedia Comput. Sci.*, vol. 85, no. Cms, pp. 293–300, 2016, doi: 10.1016/j.procs.2016.05.236.

[8.] M. E. Korstanje, *Advances in Information Security, Privacy, and Ethics (AISPE)*, no. February. Argentina: University of Palermo, 2017.

[9.] Y. Liu, R. Zhang, and Y. Zhou, "Predicate encryption against master-key tampering attacks," *Cybersecurity*, vol. 2, no. 1, 2019, doi: 10.1186/s42400-019-0039-6.

[10.] T. Kivisaari and others, "Providing Secure Web Services for Mobile Applications," 2015.

[11.] M. Le, S. Clyde, and Y. W. Kwon, "Enabling multi-hop remote method invocation in device-to-device networks," *Human-centric Comput. Inf. Sci.*, vol. 9, no. 1, 2019, doi: 10.1186/s13673-019-0182-9.

[12.] S. Salamatian, W. Huleihel, A. Beirami, A. Cohen, and M. Medard, "Why botnets work: Distributed brute-force attacks need no synchronization," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 9, pp. 2288–2299, 2019, doi: 10.1109/TIFS.2019.2895955.

[13.] R. Amin, S. H. Islam, G. P. Biswas, M. K. Khan, and X. Li, "Cryptanalysis and Enhancement of Anonymity Preserving Remote User Mutual Authentication and Session Key Agreement Scheme for E-Health Care Systems," *J. Med. Syst.*, vol. 39, no. 11, 2015, doi: 10.1007/s10916-015-0318-z.

[14.] C. Yogatama, R. R. Isnanto, and A. Triwiyanto, "Aplikasi Algoritma Hash Dalam Pengacakan Pertemuan Dan Pertarungan Dinamis Pada Perangkat Lunak Permainan Dengan Sistem Operasi Android," *Transient*, vol. 3, no. 3, pp. 301–308, 2014, [Online]. Available: https://ejournal3.undip.ac.id/index.php/transient/article/view/6347.

[15.] F. Maqsood, M. Ahmed, M. Mumtaz, and M. Ali, "Cryptography: A Comparative Analysis for Modern Techniques," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 6, 2017, doi: 10.14569/ijacsa.2017.080659.