Indonesian Journal of Electrical Engineering and Computer Science
Vol. 10, No. 1, April 2018, pp. 348~353
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v10.i1.pp348-353

Data in Transit Validation for Cloud Computing Using Cloud-Based Algorithm Detection of Injected Objects

Rashidah Funke Olanrewaju, Thouhedul Islam, Othman O. Khalifa, Fawwaz Eniola Fajingbesi
Department of Electrical and Computer Engineering, International Islamic University Malaysia

Article Info

Article history:

Received Jan 15, 2018
Revised Mar 15, 2018
Accepted Mar 29, 2018

Keywords:

Cloud Computing
Data Security
Data in Transit
Data Injection

ABSTRACT

The recent paradigm shift in the IT sector leading to cloud computing however innovative had brought along numerous data security concerns. One major such security laps is that referred to as the Man in the Middle (MITM) attack where external data are injected to either hijack a data in transit or to manipulate the files and object by posing as a floating cloud base. Fresh algorithms' for cloud data protection do exist however, they are still prone to attack especially in real-time data transmissions due to employed mechanism. Hence, a validation protocol algorithm based on hash function labelling provides a one-time security header for transferable files that protects data in transit against any unauthorized injection. The labelling header technique allows for a two-way data binding; DOM based communication between local and cloud computing that triggers automated acknowledgment immediately after file modification. A two layer encryption functions in PHP was designed for detecting injected object; bcrypt methods in Laravel and MD5 that generate 32 random keys. A sum total of 1600 different file types were used during training then evaluation of the proposed algorithm, where 87% of the injected objects were correctly detected.

Corresponding Author:

Rashidah Funke Olanrewaju,
Department of Electrical and Computer Engineering,
International Islamic University Malaysia,
Kuala Lumpur, Malaysia.
Email: frashidah@iium.edu.my

1. INTRODUCTION

The 21st-century paradigm shift to information technology is centred on cloud computing (CC). Its enormous benefits such as ease of contents, services and instant resource sharing show no limitation to the interest of both private and public establishments. In due time a total shift, it would be achievable due to the crowd development accessibility, elasticity and virtualisation of cloud computing [1]-[4]. The bright future for CC might not come to fruition in good time if major data security vulnerability in areas such as data at rest and data in motion is not adequately curbed [5]-[6]. A data siting on a computer, a machine, a server or somewhere in a cloud is referred to as data at rest while the process of exchanging information, moving an object from point a to b between a local and a remote computer or transferring information between severs are considered as data in transit or data in motion [7]-[8]. Data in transit are more susceptible to attacks than their counterparts at rest as Packets may be cached on intermediate systems, or temporary files may be created at either endpoint. External object injection at different stages is difficult for administrators to identify the primary source of the application due to the boundary limit access of data in motion [8]. As illustrated in Figure 1, the most dangerous threat to data in motion occurs when attacker gain unauthorised access during data transmission via object injection [9]-[10].

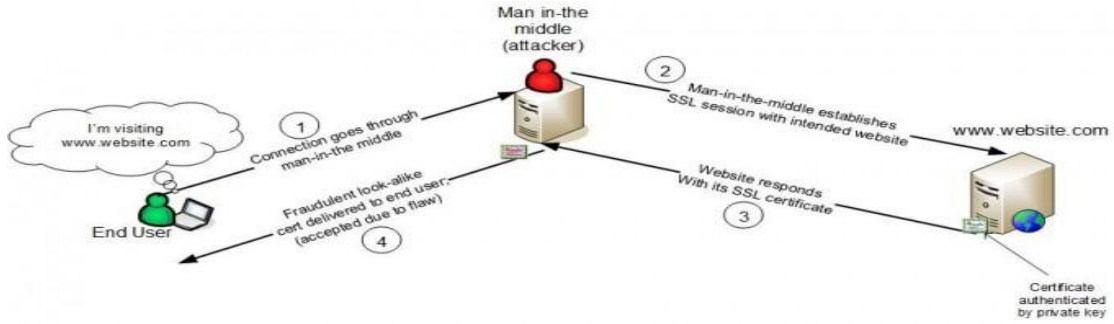

Figure 1. Man in the Middle (MITM) attack on Data in Motion

Algorithms are created to combat CC data security concerns such as MITM. Majority of such algorithms tend to enhance data authorization scheme across access points; however, these attack vulnerabilities persist as access from a different location and device with multi-tendency accessing opportunities exist [6]. Hence, this research proposes an algorithm for securing and validating data in motion by auto detection of injected objects using advanced encryption standard and hash functionality algorithm.

2. RESEARCH METHOD

The algorithm design in this project was based on PHP and VueJS hence factory design pattern shown in Figure 2 was used. Since Vuex design pattern is used for real-time communication via TCP protocol between sender and receiver and Vuex design offers the best matching for state management of transferring data hence used for the front end design.

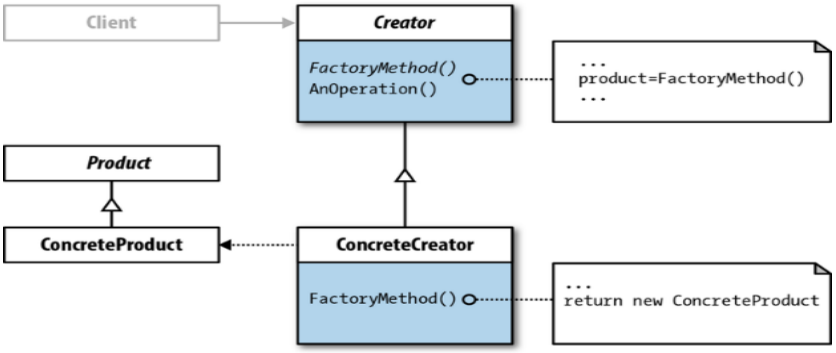

Figure 2. Factory Design Pattern

2.1. Algorithm Design

The algorithm functionality for data security and validation for preventing external object injection or unauthorized access is achieved using key sharing method when sending packets from one end to another. During any file upload two keys are generated; private and public keys based on Secure Hash Algorithm (SHA1) where the current time stamp, user mac identification number, IP address and other extra slats are added to strengthen the data. The Pseudocode for public key generation in PHP is depicted below and illustrated by a flowchart in Figure 3.

Define file_type_variable
if detect file type:
Get current time stamp
Get original filename
Get file extension
Temporary file name = current time stamp + original filename + file extension
*Temporary file name = strlen(temporary file name) * 8*

```
        Temporary file name .= chr(128)
        while ((strlen(Temporary file name) + 8) % 64 !== 0)
                Temporary file name .= chr(0)
        endwhile
        for each (string split(printf('%064b', Temporary file name), 8) as new variable)
                Temporary file name .= chr(bindec(new variable))
        endforeach
        Public key = Temporary file name
else
        Goto file_type_variable
endif
```

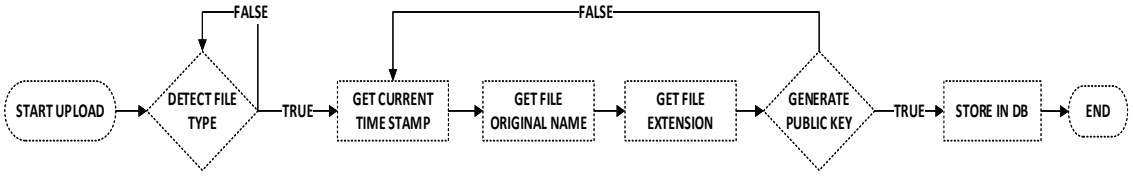

Figure 3. Public Key Generation Process

As depicted in Figure 4, the customized development mechanism for the private key differs from the public key in their encryption type. For the private key, it takes the time stamp of uploading a file and make a strtoupper() function to generate its uniqueness then uses file type, file original name, and file extension to generate a temporary file name. This temporary file name is then encrypted by MD5() algorithm first before the next initiatives which are to use SHA1() for encryption.

```
Define file_type_variable
if detect file type:
        Get current time stamp
        current time stamp = strtoupper(current time stamp)
        Get original filename
        Get file extension
        Temporary file name = current time stamp + original filename + file extension
        for each 512-chunk of temporary message
                break chunk into sixteen 32-bit words
                for i from 0 to 63
            var int F, g
            if 0 ≤ i ≤ 15 then
               F := (B and C) or ((not B) and D)
               g := i
            else if 16 ≤ i ≤ 31
               F := (D and B) or ((not D) and C)
               g := (5×i + 1) mod 16
            else if 32 ≤ i ≤ 47
               F := B xor C xor D
               g := (3×i + 5) mod 16
            else if 48 ≤ i ≤ 63
               F := C xor (B or (not D))
               g := (7×i) mod 16
            F := F + A + K[i] + M[g]
            A := D; D := C; C := B
            B := B + leftrotate(F, s[i])
          end for
                Temporary file name . = Temporary file name * A * B * C
        end foreach
```

*Temporary file name = strlen(temporary file name) * 8*
Temporary file name .= chr(128)
while ((strlen(Temporary file name) + 8) % 64 !== 0)
Temporary file name .= chr(0)
end while
for each (string split(printf('%064b', Temporary file name), 8) as new variable)
Temporary file name .= chr(bindec(new variable))
end for each
Public key = Temporary file name
Endif

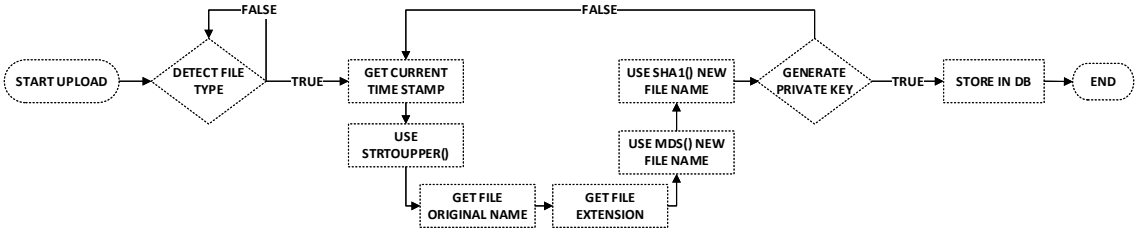

Figure 4. Private Key Generation Process

The proposed algorithm mitigates unauthorized access thereby preventing MITM attacks efficiently by handling header part of any uploading file. The header part is primarily design for restricting unauthorized access. During authorized access, the hidden header variable update sender out of the box that someone else authorized to access this file. On the other hand, this unauthorized access file will disappear from recipient side, so that there are no possibilities from recipient side to access any corrupted file. Figure 5 shows the flowchart of the full mechanism for preventing un-authorization of the data from MITM attack.

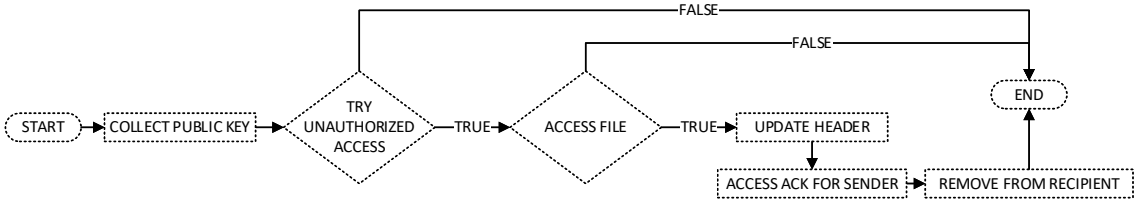

Figure 5. Factory Design Pattern

2.2. Algorithm Implementation

Taking into account cloud computing environment, the algorithm design was such that it conforms to the functional, structural and behavioral modelling of CC. Initially, 1600 raw files shown in Table 1 and Table 2 are randomly selected from document, text, pdf, jpg, png, gif, ppx, and pptx for training and evaluating the algorithm. All raw data are selected for different platforms and operating system. Mainly this test focused on Mac, Windows, and Linux (ubuntu) OS.

Table 1. Selected Raw Data Classification

File Type	MS word	MS Excel	MS PowerPoint	JPG	GIF	PNG	TXT	TOTAL
No of Files	350	200	200	200	400	150	100	1600
Mac OS	175	100	100	100	200	75	50	800
Windows OS	100	75	75	50	100	50	25	475
Linux OS	75	25	25	50	100	25	25	325

Table 2. Selected Raw Data Classification based on Data Size

File Size	< 1 MB	< 5 MB	< 10MB
Total	1045	545	10

3. RESULTS AND ANALYSIS

The evaluation benchmark is based on PHP unit test by directly inserting object in the uploading files. After randomized insertion, PHP unit test benchmark the uploaded file. In the first batch file, the randomized inserted object added for testing in PHP unit test whether the designed algorithm can detect the injected object or not. Figure 6 show the testing environments and the result of this benchmarking for the evaluation of inserted objects during file transfer form one state to another (Data in Transit).

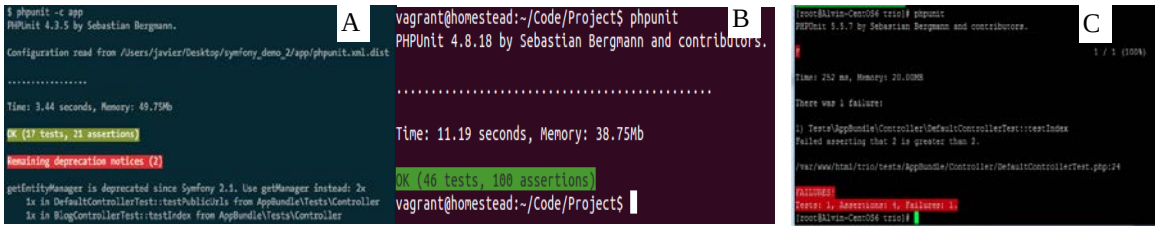

Figure 6. Testing Algorithm Code in (A) Mac (B) Linux and (C) Windows Environments

This benchmarking shown in Figure 6(a) found two inserted object in uploading a file that has changed from the original file. Figure 6 (b) shows the screenshot of another batch of uploaded files, and the result confirmed that the uploaded data had not been tampered with. This evaluation was done on CentOS in Linux distribution. In each batch of this test, four objects were inserted during sending a file from one state to another. It is pertinent to note that, if a file is affected by the inserted object, the signature header change automatically by alerting the recipient that file has been modified during transferring from one state to another. In this case, the header detection in recipient section, send an acknowledgement to the sender that the last sending file has been modified/ corrupted by a third party, which requires being resent. This response is also generated by the proposed algorithm. Subsequently, 500-unit data test was conducted on different version of Windows OS and 200 from mac and Linux distribution in batches of randomized injected objects. The accuracy for detection rate is found at 87% from the overall data test.

4. CONCLUSION

This paper proposes an algorithm for data in transit security in cloud computing environments. The algorithm is designed to assign unique two-layer secure key detection system for preventing unauthorized access and detection of an injected object in Data in Motion. The proposed algorithm helps in mitigating Man In The Middle attack, which is one of the severe threat in transferring data from local machine to the cloud and vice versa. For the algorithm design, training and implementation, several randomized combinations from a set of 1600 distinct Data files were used. The accuracy of the algorithm was found to be 87%. It was discovered that the 13% undetected files were due to mass exceptions. The algorithm can also benefit from more training with more substantial data set to improve the detection rate and ease of deployment. Also, all files were tested on different types of operating systems such as MAC, Linux and Windows.

ACKNOWLEDGEMENT

This work was partially supported by Ministry of Higher Education Malaysia (Kementerian Pendidikan Tinggi) under Research Initiative Grant Scheme (RIGS) number RIGS 16357-0521.

REFERENCES

[1] D. Koizumi, *et al.*, "On the automatic detection algorithm of Cross Site Scripting (XSS) with the non-stationary Bernoulli distribution," in *2012 Mosharaka International Conference on Communications, Computers, and Applications (MIC-CCA)*, pp. 131–135, 2012.

[2] D. R. Tsai, *et al.*, "Optimum tuning of defense settings for common attacks on the web applications," in *43rd Annual 2009 International Carnahan Conference on Security Technology*, pp. 89–94, 2009.
[3] H. Shahriar and M. Zulkernine, "Information Theoretic Detection of SQL Injection Attacks," *Proceedings of 14th International Symposium on High Assurance System Engineering*, 2012.
[4] H. Shahriar, *et al.*, "Design and development of Anti-XSS proxy," in *Internet Technology and Secured Transactions (ICITST), 2013 8th International Conference for*, pp. 484–489, 2013.
[5] I. Lim, *et al.*, "Securing Cloud and Mobility: A Practitioner's Guide," CRC Press, New York, 2013.
[6] I. Balasundaram and E. Ramaraj, "An Authentication Scheme for Preventing SQL Injection Attack Using Hybrid Encryption (PSQLIAHBE)," vol/issue: 53(3), pp. 359-368, 2011.
[7] J. Shanmugam and M. Ponnavaikko, "A solution to block Cross-Site Scripting Vulnerabilities based on Service Oriented Architecture," in *6th IEEE/ACIS International Conference on Computer and Information Science*, pp. 861–866, 2007.
[8] K. X. Zhang, *et al.*, "TransSQL: A Translation and Validation-based Solution for SQLInjection Attacks," *Proceedings of First International Conference on Robot, Vision and Signal Processing, IEEE*, pp. 248-252, 2011.
[9] Kieyzun, *et al.*, "Automatic creation of SQL Injection and cross-site scripting attacks," *presented at the Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pp. 199–209, 2009.
[10] L. K. Shar and H. B. K. Tan, "Defending against Cross-Site Scripting Attacks," *Computer*, vol/issue: 45(3), pp. 55–62, 2012.