# Storage Security and Predictable Folder Structures in Cloud Computing

Arpita Jain
Department of Computer Science and Engineering
Marudhar Engineering College, Bikaner
*arpitajain7310@gmail.com*

Sunita Chaudhary
Department of Computer Science and IT
Jagannath University, Jaipur
*er.sunita03@gmail.com*

***Abstract*—**The open nature of the html content and URLs used to access other resources used to render the page leaves the folder structure and location of those files vulnerable to robots, external hackers and malicious insider attacks, typically referred to as XSS attack. A malicious user can study the html structure and find out the pattern or folder structure of stored files and with the help of robots or crawlers it can try to access reset of the files residing there on server irrespective of whether he was or was not authorized to get them and could use those files file ï vary from simple ones based on is only the resources are stolen from the web page content or the directories are crawled and all the resources from those locations are accessed, listed or used. XSS attack is easy to be launched with little efforts while its damage is severe in case of cloud.

***Keywords*—***Cloud computing; SAAS (Software as service); insider attack; storage security.*

*****

## I. INTRODUCTION

Web applications have evolved from a set of static web pages' files ma Ide from static HTML to fully dynamic applications where end user could come and post there custom contents on portals.[1, 10] The combined usage of client-side and server-side made it possible to provide users with highly sophisticated user functionality and interface through web browsers. Current web security model relies on storing the binary resources at a location in scripts file server and rendering them in web pages through urls[13, 17]. However, attackers always try to fetch the data stored on server by crawling or by robots even when he is not authorized to access those files[10]. XSS is one of the most popular web attack techniques. It is ranked third in the top ten listing of the most popular web applications vulnerabilities by OWASP group[16]. Contents Security Policy (CSP) promises to mitigate such attacks to a great extent. CSP is a mechanism that helps web applications to prevent a wide class of injection attacks including XSS vulnerability[17]. To prevent the resources to be accessible publicly without user access control through url we must have some algorithm to generate dynamic urls which keeps on changing with every request and by not storing the files in file system so that even by crawling the source code one is not able to access the user's' binary data.

Similarly if the adversary is some insider who have access to the server and from the url he have an idea of possible location of storing file then without any brute force attack or hit and tries policy or boat he can simply navigate to the folder location and steal the resource[1, 5, 6, 8] according to the recent trends there are lots of measures being taken to prevent the server administrators also to access the end user data like as not allowing to store the server passwords in plain text,[1,3,4] files storing the password is not readable by any user accept the root user of the server[7], SSH login are required, log in through username and passwords is not promoted instead login through PPK or PEM keys is encouraged where private key is with end user but and public keys are stored on servers[7,8] but still it is not impossible to steal the data, to prevent this we are storing data on 2 servers 1st code server which stores the logic of SAAS applications and 2nd the file server which stores the encrypted physical resources uploaded by end users.

## II. LITERATURE SURVEY

### A. Automatic Generation of Content Security Policy to Mitigate Cross Site Scripting

Content Security Policy (CSP) is effective customer side security layer that aides in alleviating and identifying wide scopes of web assaults including cross-website scripting (XSS).

In any case, using CSP by site managers is an error prone process what's more, may require critical changes in web application code. In this paper, we propose a way to deal with help site directs to overcome these constraints to use the full advantages of CSP component which prompts more safe destinations from XSS. The calculation is actualized as a module. It doesn't meddle with the web application unique code. The module can be "introduced" on any other web application with least endeavors. The calculation can be actualized as a feature of Web Server layer, not as a component of the business rationale layer. It can be stretched out to help creating CSP for substance that are adjusted by

JavaScript in the wake of stacking. Current approach investigates the static substance of URLs.

B. Insider Threat Identification by Process Analysis

The insider risk is a standout amongst the most hurtful assault in PC security. Customary methodologies commonly instrument frameworks with actualizes interruption identification components to identify people who mishandle their benefits (the dependable or approved "insider"). These assaults require that the clients approach assets or information keeping in mind the end goal to degenerate or reveal them. In this paper, we examined the utilization of process demonstrating and ensuing examinations to the insider issue. With process demonstrating, we initially contemplated how a SAAS application functions, it's work process and its fundamental engineering in formal terms. We at that point contemplated diverse specialists who convey specific undertakings, investigated distinctive examinations to see how the procedure can be bargained, and its countermeasures that can be fused into the procedure model to enhance its protection from insider assault.

C. Securing Against Insider Attacks The combination of the Web an open system de into the correspondence structure of most associations has fundamentally changed the way IT security is actualized in for all intents and purposes each business condition. What's more, as of not long ago, the essential reason for most security systems to manage this has been the "channel and château" show: a solid border is set up that partitions the earth into a confided in inside and untrusted outside, with security concentrated on setting up the edge, upholding access control procedures, and securing information as it streams from outside to edge. This way to deal with security is not really new. Ancestral resistance is an attempted and put stock in solution for a threatening world and has at its base a survival quality that has served us well for many years, ideal back to the time when ancient man initially began to walk upright and bunch together into bunches for barrier against an exceptionally antagonistic world.

D. Insider threats: Detecting and controlling malicious insiders

Malignant insiders are posturing exceptional security difficulties to associations because of their insight, abilities, and approved access to data frameworks. Information robbery and IT attack are two of the most repeating subjects among violations conferred by noxious insiders. This paper means to research the scale and extent of malevolent insider hazards and investigate the effect of such dangers on business operations. Associations need to actualize a multi layered guarded ways to deal with battle insider dangers; shielding touchy business data from pernicious insiders require right off the bat, a compelling security arrangement that imparts outcomes of taking or releasing private data in an unapproved way. Besides, logging and observing representative action is basic in recognizing and controlling framework vulnerabilities to vindictive insiders. Thirdly, directing intermittent and steady insider defenselessness appraisals is basic to recognizing any holes in security controls and keeping insiders from abusing them. What's more, finally, yet unquestionably not slightest, taking additional alert with advantaged clients is imperative to proactively shielding data framework from insider dangers.

E. A System Architecture for the Detection of Insider Attacks in Big Data Systems

In enormous information frameworks, the foundation is with the end goal that a lot of information are facilitated far from the clients. In such a framework data security is considered as a noteworthy test. From a client point of view, one of the enormous dangers in embracing huge information frameworks is in believing the supplier who plans and claims the foundation from getting to client information. However there does not exist much in the writing on location of insider assaults. In this work, we propose another framework design in which insider assaults can be identified by using the replication of information on different hubs in the framework. The proposed framework utilizes a two-advance assault identification calculation and a protected correspondence convention to investigate forms executing in the framework. The initial step includes the development of control direction successions for each procedure in the framework. The second step includes the coordinating of these direction successions among the reproduction hubs. Starting analyses on true hadoop and start tests demonstrate that the proposed framework needs to consider just 20% of the code to dissect a program and acquires 3.28% time overhead. The proposed security framework can be executed and worked for any enormous information framework because of its extraneous work process.

## III. DESIGN AND ARCHITECTURE

### A. Algorithm Overview

Let’s assume there is an application example.com coded in PHP using an MVC framework, say laravel. The admin of the website is willing to protect the user uploaded files to be directly accessible through urls. The application displays the user specific resources from specific sources. For example

1. http://example.com/storage/something.png
2. http://example.com/storage/somethingelse. pdf
3. http://example.com/storage/somemore.jpe g
4. http://example.com/storage/somemorefiles.doc
5. http://example.com/storage/somefile.xlsx As we

can notice that by looking at the url patterns one can guess

 ISSN: 2321-8169
109 - 116

the actual file location of stored files on server i.e. the web root must have a folder name storage in it and that folder would contain all the files, this is not good for hackers outside the system to get list of all the other files stored in this location through brute force attack and download them through urls and for malicious insider also it is easy to locate the files and access/manipulate them.
Suppose if it is user based system like dropbox, google drive, facebook then it is more harmful because anyone can access files of all the user without their permission.

To avoid all this we need to prevent following things

1. Resources are accessible through same url throughout their lifespan
2. Resources are readable or accessible from there file locations, neither to hacker through urls nor to malicious insider
3. Location of physical files is predictable through their urls.
4. Hacker/malicious insider is able to predict/read all the other files except the one which he is authorized to.

We can prevent this by separating code base's server and file server and keep them isolated so that each server do not have any information about each other except there URL each time where the code server wants the file it sends a request to file server and display the file. And the url used to get the file would expire in 10 second and so thus the request sent from code server to file server.

*B. Algorithm Design*

We suggest that the algorithm is carried on using an MVC framework build up in PHP namely Laravel. Our script will first authenticate user and if authenticated then only will allow the end-user to access his/her resources. Once user gets login then only he/she would be allowed to view the files he /she has uploaded or upload new files.
User uploads the file on code server through html form provided to him in front end. On receiving request from client on code server it encrypts the file content using Key(i) and AES256 with mode CBC and pass it on to file server
File server received the encrypted file and re encrypts it with key(ii) store it in its own storage system in path = Storage/iv of file name/mac of filename.value of filename and return concatenated encrypted id, iv and hash and code server save this for future reference.

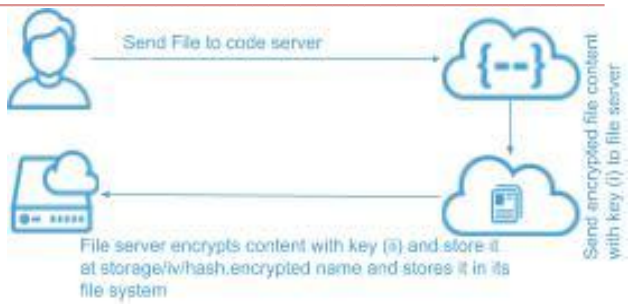

Figure 1: Save file uploaded by end user.

Figure 2: Cloud Server Stores the Reference returned by file server in response after storing the file securely .

We are storing file on an isolated server so that even if end+user tries to crawl the code server through boat he won't be able to get the file location of physical file.
We are doing double encryption so that even if one server is hacked the hacker won't be able to access the actual records even if he is an insider because none of the server knows the actual encoding/or encryption key as it is combination of 2 encrypted strings. for example even if someone tries to hack the code server (even if he is an insider) he won't be able to access the physical files because they are not there, if someone tries to study the code and figure out the encryption for stored files then also he will get partial process as second half of encryption procedure is on files server.
Similarly if someone tries to hack file server then also even after knowing the physical file location he won't be able to get the actual file s it is encrypted and if he tries to decrypt the file and get actual content then also he won't be successful in getting file content because one mode decryption is yet to be figured whose details are on code server.

To Retrieve the the stored file we need to perform following sequence of operation

URL generation: When the page loads to get the user data the Code server generates a unique URL which is valid for 10 seconds by encrypting the time and code server's id at which we have saved and time and return the encrypted string along with iv and hash in url.

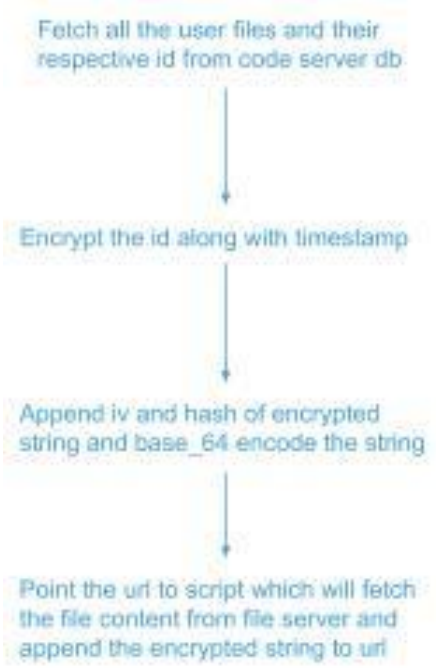


Figure 3: Generating unique urls to access the resource

To access the files we first need to generate the urls which are globally accessible through http request. But in our case they must be valid for only a predefined time. And after that they expire. For that what we do is we extract the id from code server combine it with date time and encrypt it, and because same date and time won't repeat ever thus every time we will have a different string to encrypt. Once we encrypt the string we append it to the url which points to our script which would fetch the file from file server and return the file content. Along with this encrypted string to be passed as GET parameter.

Request file from code server: When the browser try to receive the file from code server then server first decrypt the string with the help of details in url and cross check if this url was generated in last 10 seconds if yes then it fetch the code server id and with respect to code server id it figure out the string which file server returned in response when we saved the file.

Once we receive the request we cross check that the url was generated just a few seconds back if not then it won't process the request. This forces us to generate a new request every time a resource is to be accessed and add block access to content from code server from repeated request from previously stored urls i.e. replay attack. Say some one stored the http url and try to access the file later with same url then he/she won't be able to do that.

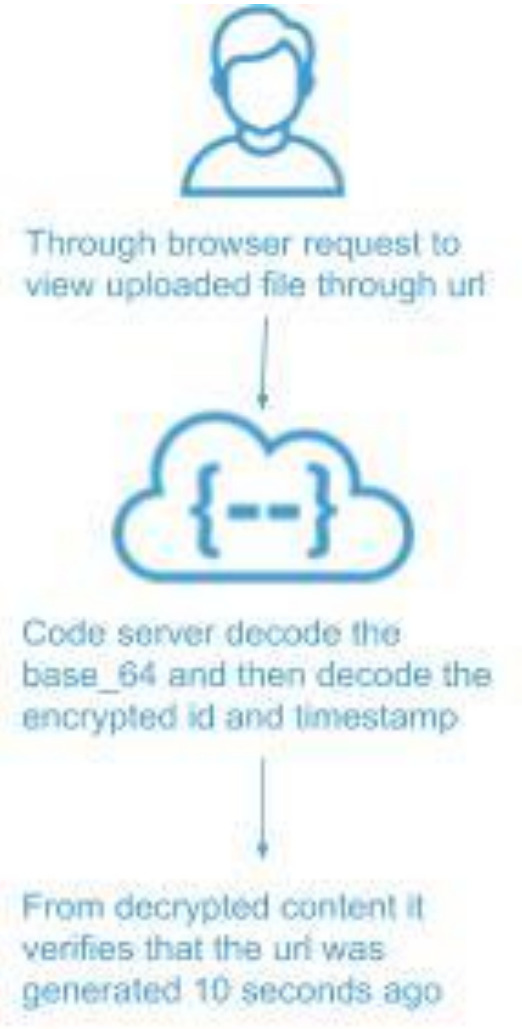


Figure 4: Code Server verifying the .url which is trying to access the resource.

Request file from file server: Once Code server validates the request it create the request to be sent to file server from code server to file server, for that it encrypts the response string from file server along with current timestamp using public key, and send the encrypted string to file server in POST

We are sending encrypted request from cloud server to file server so that if someone try to sniff requests from code server to file server then he or she could not understand what is being transferred here we are using openssl encryption we could have used ssl. But because of availability of resources and cost for developing the prototype of this approach openssl was more suitable.

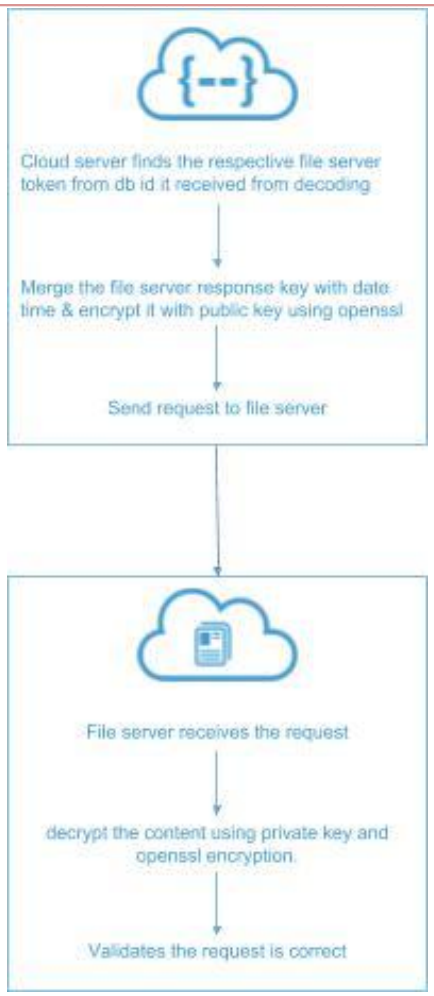


Figure 5: Code Server retrieving the resource from File server through API.

Return file from File server to Code server: On receiving request from code server file server verifies that code server generated the request in last 10 seconds if yes then it decodes the post data and retrieve the file id and with the help of it finds the encrypted name of file.

We decodes the file name and fetch the file path, access the file from the and decrypt the file content using key (ii) and return the decoded file content to code server along with mime type

We cross check the validation related to time for the same reason we discussed above in step 2 and once it is verified we return the decrypted content to code server without any additional layers because we needed to secure access and storage only

Returning file content to browser from code server: Code server decrypts the content received from file server using key (i) and return the content to browser with same file type which file server returned to code server.


Figure 6: Code Server returning the decrypted resource to end user.

## IV. IMPLEMENTATION

### A. Pseudo to upload file

This Section explains the pseudo code for uploading the file by a user in his SAAS account and uploading it on file server from all the way through code server.

### B. Pseudo for Code Server

This section explains the pseudo code fo code server which reads the user uploaded file , encrypts it and forwards it to file server.

1. Var fileContent = read(uploaded file content)
2. Encrypt AES256-CBC-ENC (fileContent,K(i))
3. Var fileServerReference = uploadFileOnFileServer()
4. Store fileServerReference in DB

### C. Pseudo for File Server

This section explains the pseudo code for file server which reads the file sent by code server, encrypts it and stores it on its own server.

Var fileContent = read(uploaded file content)

1. Var encFileContent = AES256-CBC-ENC (fileContent,K(i))
2. Var encName = AES256-CBC-ENC (fileName, key(ii))
3. Var encContentType = AES256-CBC-ENC (contentType,key(ii));
4. Store encFileContent at encName[String]/encName[Hash].encNa me[Iv]
5. Store encName, encContentType in DB
6. Var id= DB id
7. EncryptedId = AES256-CBC-ENC (id,key(ii))
8. return EncryptedId + iv + hash + length(encryptedId) + length(iv) + length(hash) + randomSalt(3Digit)

D. Pseudo to generate URLs

This section explains the pseudo code for generating unique URLs at the time on creating HTML page to render.

1. Var Id = fetch cloud server DB id
2. Var encId = AES256-CBC-ENC(id + timestamp,key(i))
3. Var URL = http://subdomain.com/URL/encId

4.3.3 Pseudo to retrieve the resources

This section explains the pseudo code for retrieving the original content from file server through code server.

4.3.3.1 Pseudo for Code Server

This section explains the how code server validated the request and if valid it sends request to file server to get the file content.

1. Var Decrypted String = AES256-CBC-DEC(URLsuffix)
2. Split timestamp and verify if not more than 10 seconds have passed since then.
3. Return 404 in HTTP response header if more than 10 seconds have passed
4. Var fileServerReference = Fetch Reference from DB
5. Var fileContent = requestContnetFromFileSever(OPENSSL-ENC(fileServerReference + timestamp))
6. Return AES256-CBC-DEC(fileContent, key(i))

4.3.3.2 Pseudo for File Server

This section explains the how file server validated the request and if valid it returns the decrypted content to code server.

1. Var Decrypted String = OPENSSL-DEC(requestInput)
2. Split timestamp and verify if not more than 10 seconds have passed since then.
3. Return 404 in HTTP response header if more than 10 seconds have passed
4. Var encName =Fetch Name from DB
5. Var Name = AES256-CBC-DEC(encName)
6. Var contentType= AES256-CBC-DEC(Fetch contentType from DB)
7. Fetch File From encName[String]/encName[Hash].encNa me[Iv]
8. Var fileContent = requestContentFromFileSever(OPENSSL-ENC(fileServerReference + timestamp))
9. Return AES256-CBC-DEC(fileContent, key(i))

4.3.4 Common Functions

This section explains the common or generic process we used for encryption and decryption on both the servers.

4.3.3.1 AES256-CBC-ENC(string, key)

This section explains the encryption we used and how we exchanged the IV, hash and encrypted strings.

1. Standard AES Encryption on string with key and Random iv
2. Hash = SHA1(encrypted string +iv)
3. Create json of encrypted string, iv and hash
4. Base_64 encoding of json

4.3.3.2 AES256-CBC-DEC(string, key)

This section explains the decryption we used and how we retrieve the original string witht e help of shared IV, hash and encrypted strings.

1. Base_64 decoding of json
2. Fetch key,iv and hash from json
3. Verify hash
4. Decode the encrypted string with the help of key and iv

4.3.3.3 OPENSSL-ENC

Implementation of standard OpenSSL encryption in server side scripting in our case it is PHP with laravel framework.

4.3.3.4 OPENSSL-DEC

Implementation of standard OpenSSL decryption in server side scripting in our case it is PHP with laravel framework

4.3.3.5 uploadFileOnFileServer

Send a post request to file server with file content in post we can use CURL to implement this server side scripting language

4.3.3.3 uploadFileOnFileServer

Send a post request to file server with file content in post we can use CURL to implement this server side scripting language

## V. RESULT AND ANALYSIS

A. Impact of XSS on physical files uploaded We illustrates the impact of having predictable URLs for binary resources uploaded in a SAAS. We chose apache as server, php as server side scripting html,css and jquery as client side scripting. Aside from our methods, we also implemented a simple encryption by merging multiple strings in a predefined order along with some salt.

URLs are globally accessible and being indexed easily by search engines, If we need to secure our resources which are private and we do not want them to be accessible every time and by anyone we need to have dynamic URLs which keeps on changing from time to time.

An approach was suggested in (10.1109@ICSITech.2016.7852656.pdf) where we could change the URLs for all the http URLs of content in that

SAAS (namely wordpress) by adding some random numbers in the actual URLs
say if a URL was http://somedomain.com/my-article then the author suggested that with the help of a plugin admin can change the URL to something like http://somedomain.com/my-article/123 but in that case

1. The URLs are accessible until admin changes them again.
2. This approach is only concerned about the web pages and not about the physical resources user uploaded.
3. The folder structure or pattern of the web pages is still predictable

We resolved all these issues in our approach by

1. Generating unique URL for each request
2. Right now we have done this only for binary files uploaded but this could be used for any other webresource which is accessible through URL.
3. There is no specific pattern of URLs they all look similar with some encrypted strings.

B. Impact on real-time systems

Although this approach protects the files from being accessible publicly, but amount of processing time for encryption and decryption happening between end user-code server and code server-file server adds in processing time and CPU utilization of both the servers as a result it takes a little longer in accessing the file but this deviation in time to download is also dependent on network speed hardware of end user trying to access the resource geo location of both the server and in our case it is time to download is approximately double of time taken to directly download the file.

## VI. CONCLUSION AND FUTURE WORK

This chapter contains conclusion and future work of this dissertation.

A. Conclusion

We addressed the problem of global accessibility of resources from anywhere anytime. We considered a simple SAAS application which allows end-users to upload their resources and access them whenever they need it. So we introduced and approach where end-user uploads his binary data on SAAS application and this application encrypts the data and stores that on another server. We showed that with the conventional URLs it is easy for anyone to predict the folder structure and access all the files uploaded on server. We solved this issue by generating unique URLs for each http request these URLs could not be reused after a predefined time in our case it is 10 seconds. Our findings show that global access to files uploaded by end user or predictable location or accessibility leads to XSS attack leave those resources vulnerable to be accessed by unauthorized user which should not be the case otherwise. We used OPENSSL, and AES-256 with MODE CBC for encryption and decryption and used timestamps to prevent replay attack on URLs. Our schemes.

B. Future work

In my future course of work, I will try to enhance the security of binary resources along with decreasing the download time, time complexity and space complexity. Right now in current system if we have a time limit of 10 seconds then a generated URL could be used for as many times as possible in the window of those 10 seconds I will try to update this as well so that if a URL is used once it could not be reused ever not even in that time frame. Moreover it is has great future scope to improve security by having dynamic encryption keys for each request such that even if a key is compromised this would not affect the other records lying in system as they would be using different keys for encryption.

## VII. REFERENCES

[1]. Samer Attallah Mhana, Jamilah Binti Din, Rodziah Binti Atan Department of Software Engineering and Information System “Automatic Generation of Content Security Policy to Mitigate Cross Site Scripting” 2016 2nd International Conference on Science in Information Technology (ICSITech)

[2]. Matt Bishop, Heather M. Conboy, Huong Phan, Borislava I. Simidchieva, George S. Avrunin, Lori A. Clarke, Leon J. Osterweil, Sean Peisert “Insider Threat Identification by Process Analysis” 2014 IEEE Security and Privacy Workshops

[3]. David M. Lynch, “Securing Against Insider Attacks”, The EDP Audit, Control, and Security Newsletter Volume 34, 2006 - Issue 1

[4]. Marwan Omar, “Insider threats: Detecting and controlling malicious insiders” 162-172. 10.4018/978-1-4666-8345-7.ch009.

[5]. Samer Attallah Mhana, Jamilah Binti Din, Rodziah Binti Atan “A System Architecture for the Detection of Insider Attacks in Big Data Systems” arXiv:1612.01587v1 [cs.CR] 5 Dec 2016

[6]. Miltiadis Kandias, Nikos Virvilis, Dimitris Gritzalis “The Insider Threat in Cloud Computing” Information Security & Critical Infrastructure Protection Research Laboratory Dept. of Informatics, Athens University of Economics and Business, Greece

[7]. Sunu Mathew1, Michalis Petropoulos, Hung Q. Ngo, and Shambhu Upadhyaya “A Data-Centric Approach to Insider Attack Detection in Database Systems” IEEE Journals & Magazine, ieeexplore.ieee.org/document/8093643/.

[8]. M. Bishop and C. Gates, "Defining the Insider Threat," Proceedings of the Cyber Security and Information Intelligence Research Workshop article 15 (May 2008).

[9]. Kandias M., Virvilis N., Gritzalis D. (2013) "The Insider Threat in Cloud Computing. In: Bologna S., Hämmerli B., Gritzalis D., Wolthusen S. (eds) Critical Information Infrastructure Security." CRITIS 2011. Lecture Notes in Computer Science, vol 6983. Springer, Berlin, Heidelberg

[10]. Mathew S., Petropoulos M., Ngo H.Q., Upadhyaya S. (2010) "A Data-Centric Approach to Insider Attack Detection in Database Systems." In: Jha S., Sommer R., Kreibich C. (eds) Recent Advances in Intrusion Detection. RAID 2010. Lecture Notes in Computer Science, vol 6307. Springer, Berlin, Heidelberg

[11]. Brad Ruppert. Protecting Against Insider Attacks. SANS Institute InfoSec Reading Room, 10 Aug. 2009.

[12]. Wali Ahmed Usmani , Diogo Marques , Ivan Beschastnikh , Konstantin Beznosov , Tiago Guerreiro , Luís Carriço "Characterizing Social Insider Attacks on Facebook" CHI 2017, May 6–11, 2017, Denver, CO, USA

[13]. B M Magklarasands, G & Furnell, Steven. (2018). The Insider Misuse Threat Survey: Investigating IT misuse from legitimate users

[14]. R Claycomb, William & Legg, Phil & Gollmann, Dieter. (2014). Guest Editorial: Emerging Trends in Research for Insider Threat Detection. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA). 5. 1-6.

[15]. Shivam Jain, S. B. (2014). A survey on Software as a service (SaaS) using quality model in cloud computing. International Journal Of Engineering And Computer Science, 3(01), 3598-3602.

[16]. Ms. Pushpa B. Rajegore, Ms. Swapna G. kadam "Issues & Solution of SAAS Model in Cloud Computing" IOSR Journal of Computer Engineering (IOSR-JCE) e-ISSN: 2278-0661,p-ISSN: 2278-8727, PP 40-44

[17]. Santosh Kumar and R. H. Goudar, "Cloud Computing – Research Issues, Challenges, Architecture, Platforms and Applications: A Survey," International Journal of Future Computer and Communication vol. 1, no. 4 pp. 356-360, 2012.

[18]. Navneet Singh Patel, Prof. Rekha B.S. "Software as a Service (SaaS): Security issues and Solutions" ISSN (e): 2250 – 3005, Vol, 04, Issue, 6,| June – 2014, International Journal of Computational Engineering Research (IJCER)

[19]. Imran Ashrafa "An Overview of Service Models of Cloud Computing" Department of Information Technology, University of The Punjab, Gujranwala Campus, Pakistan, 15 Aug 2014, ISSN: 2321-3124

[20]. Rashmi, Dr.G.Sahoo, Dr.S.Mehfuz "Securing Software as a Service Model of Cloud Computing: Issues and Solutions" International Journal on Cloud Computing: Services and Architecture (IJCCSA) ,Vol.3, No.4, August 2013

[21]. S.Satyanarayana "CLOUD COMPUTING : SAAS" GESJ: Computer Science and Telecommunications 2012|No.4(36) ISSN 1512-1232

[22]. K.Kavitha "Study on Cloud Computing Model and its Benefits, Challenges" International Journal of Innovative Research in Computer and Communication Engineering (An ISO 3297: 2007 Certified Organization) Vol. 2, Issue 1, January 2014, ISSN(Online): 2320-9801

[23]. Akash Ahlawat "Implementing SAAS: Cloud Computing and Android Based Application Framework for .NET Programming" International Journal of Computer Science and Mobile Computing, Vol.4 Issue.4, April- 2015, pg.806-811, ISSN 2320–088X

[24]. T.Venkat Narayana Rao, V. Tejaswini ,K.Preethi "DEFENDING AGAINST WEB VULNERABILITIES AND CROSS-SITE SCRIPTING" Journal of Global Research in Computer Science, Volume 3, No. 5, May 2012, ISSN-2229-371X

[25]. Punam Thopate, Purva Bamm, Apeksha Kamble, Snehal Kunjir, Prof S.M.Chawre "Cross Site Scripting Attack Detection & Prevention System" International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 3 Issue 11, November 2014

[26]. Garcia-Alfaro, Joaquin & Navarro-Arribas, Guillermo. (2007). "Prevention of Cross-Site Scripting Attacks on Current Web Applications." http://hdl.handle.net/10363/606. 4804. . 10.1007/978-3-540-76843-2_45.

[27]. Gurvinder Kaur "Study of Cross-Site Scripting Attacks and Their Countermeasures" International Journal of Computer Applications Technology and Research Volume 3– Issue 10, 604 - 609, 2014, ISSN: 2319–8656

[28]. Manisha S. Mahindrakar "Prevention to Cross-site Scripting Attacks: A Survey" International Journal of Science and Research (IJSR) ISSN (Online): 2319-7064 Impact Factor: 3.358

[29]. Suman Saha "Consideration Points: Detecting Cross-Site Scripting" (IJCSIS) International Journal of Computer Science and Information Security, Vol. 4, No. 1 & 2, 2009

[30]. Mike Ter Louw, V.N. Venkatakrishnan "Robust Prevention of Cross-site Scripting Attacks for Existing Browsers" 2009 30th IEEE Symposium on Security and Privacy