Volume 4, Issue 3, March 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

DES Encryption and Attack Detection in Client-Server Communication

Sapna Choudhary, Bhupendra Singh Thakur

Computer Science &Engineering

SRGI Jabalpur, India

Abstract—Today era and business processing is totally dependent upon client and server communication. So it needs more concentration in terms of safe and secure communication. So our paper mainly concentrated on secure data communication. Then if any unauthorized access is happened then it also alerts the client so that better timely prevention can be done. In this paper we apply Data Encryption Standard for sending the data from client to the server. So that unauthorized access can be prevented. We also maintain the log details if any attack on the data will apply by the outsiders so that proper precaution and resending status will be maintained. Our results show the effectiveness of our approach.

Keywords-Content Sniffing Attack, MIME, DoS, Security Provision

1. Introduction

Content sniffing and Cross-site scripting (XSS) vulnerabilities are the major security threats today when we are in the server-client environment or using any web browser. There are several other attacks which are discuss[1] and [2].XSS vulnerabilities allow an attacker to inject malicious content into web pages from trusted web servers. The malicious code with the source script run on the same ground as other web pages and shows the fake presence of trusted authority. Since the malicious content runs with the same privilege as the trusted content from the web servers, the malicious content can steal the victim users' private data or take unauthorized actions on the users' behalf. Content sniffing attack is an attempt to deduce the content of a file format of the data or alteration in the byte stream. It is also called media type sniffing or MIME sniffing. Traditional approaches for detecting and preventing include static analysis [3], combination of static analysis and dynamic monitoring [4, 5], and browser-based defenses [6, 7]. If we think of the aggregation of static and runtime approaches provide more accuracy in detecting [8] at the cost of the deployment of customized frameworks. Browser-based approaches require end user interventions and rewriting of entire implementations [8]. So in the above direction, we survey several web based attacks which can possible through communication, we also discuss about the security concern which can be applied in future for better security in web communication[9][10].

The remaining of this paper is organized as follows. The related work in section 2. In section 3 we discuss about proposed work. In section 4 we discuss the result analysis. The conclusions are given in Section 5. Finally references are given.

2. Related Work

In 2010, Zubair M. Fadlullah et al. [11] to combat against attacks on encrypted protocols; they propose an anomaly-based detection system by using strategically distributed monitoring stubs (MSs). They have categorized various attacks against cryptographic protocols. The MSs, by sniffing the encrypted traffic, extract features for detecting these attacks and construct normal usage behavior profiles. Upon detecting suspicious activities due to the deviations from these normal profiles, the MSs notify the victim servers, which may then take necessary actions. In addition to detecting attacks, the MSs can also trace back the originating network of the attack. They call their unique approach DTRAB since it focuses on both Detection and TRAceBack in the MS level. The effectiveness of their proposed detection and traceback methods are verified through extensive simulations and Internet datasets.

In 2011, Misganaw Tadesse Gebre et al. [12] proposed a server-side ingress filter that aims to protect vulnerable browsers which may treat non-HTML files as HTML files. Their filter examines user uploaded files against a set of potentially dangerous HTML elements (a set of regular expressions). The results of their experiment show that the proposed automata-based scheme is highly efficient and more accurate than existing signature-based approach.

In 2011, Anton Barua et al. [13] developing a server side content sniffing attack detection mechanism based on content analysis using HTML and JavaScript parsers and simulation of browser behavior via mock download tests. They have implemented our approach in a tool that can be integrated in web applications written in various languages. In addition, they have developed a benchmark suite for the evaluation purpose that contains both benign and malicious files. They have evaluated our approach on three real world PHP programs suffering from content sniffing vulnerabilities. The evaluation results indicate that their approach can secure programs against content sniffing attacks by successfully preventing the uploading of malicious files. In 2012, Syed Imran Ahmed Qadri et al. [9] provide a security framework for server and client side. In this they provide some prevention methods which will apply for the server side and alert replication is also on client side. Content sniffing attacks occur if browsers render non-HTML files embedded with malicious HTML contents or JavaScript code as HTML files. This mitigation effects such as the stealing of sensitive information through the

execution of malicious JavaScript code. In this framework client access the data which is encrypted from the server side. From the server data is encrypted using private key cryptography and file is send after splitting so that we reduce the execution time. They also add a tag bit concept which is included for the means of checking the alteration; if alteration performed tag bit is changed. Tag bit is generated by a message digest algorithm. We have implemented our approach in a java based environment that can be integrated in web applications written in various languages.

In 2012, Namrata Shukla et al. [14] present an efficient approach for fraud detection. In our approach they first maintain a log file for data which contain the content separated by space, position and also the frequency. Then they encrypt the data by substitution method and send to the receiver end. They also send the log file to the receiver end before proceed to the encryption which is also in the form of secret message. So the receiver can match the data according to the content, position and frequency, if there is any mismatch occurs, they can detect the fraud and does not accept the file.

In 2013, Animesh Dubey et al. [10] propose an efficient partition technique for web based files (jsp, html, php), text (word, text files) and PDF files. They are working in the direction of attack time detection. For this motivation they are considering mainly two factors first in the direction of minimizing the time, second in the direction of file support. For minimizing the time we use partitioning method. They also apply partitioning method on PDF files. There result comparison with the traditional technique shows the effectiveness of their approach.

3. **Proposed Work**

The proposed work shown in figure 1 clearly explains the concept. Figure 1 clearly shows that any unauthorized user cannot find space in the communication. For client and server communication first client will be registered in the admin as an authorized node. Our algorithm can work on word, pdf and html types of mitigated files. If it is registered successfully then it can demand the file from server. Server process the file in the following manner:

1) Key Generation: Key Generation is the first process as shown in figure 2. In this we use Java Random class for key generation, so that it can generate random key as many times as different client request the file. So the keys are different.
2) Partition Algorithm: Partitioning is needed so that the overhead of the file can be reduced[9][10].

1) Step 1: Initialization
2) int c=0; // counter is initializes to 0
3) int len=0; // Length of file is initializes to 0
4) Step 3: File f=new File(f1);
5) Step 4: long size=f.length()/1024;
6) Step 5: if(size<=100)
7) len=(int)f.length()/2;
8) Step 6: else if(size<=250)
9) len=(int)f.length()/3;
10) Step 7: else if(size<=500)
11) len=(int)f.length()/4;
12) Step 8: else
13) len=(int)f.length()/6;

As per the above algorithm if the size is less than or equal to100 KB it is partition into 2 parts, if it is less than or equal to 250 KB then it is partition into 3 parts, if it is less than or equal to 500 Kb then it is partition into 4 parts, otherwise it is partition into 6 parts. PDF is splitted page wise.

1) DES Algorithm

For encryption we apply DES algorithm.

1) Step 1: DES uses 16 rounds. Each round of DES is a Feistel cipher.
2) Step 2: The heart of DES is the DES function. The DES function applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output .
3) Step 3: Expansion P-box
   Since RI−1 is a 32-bit input and KI is a 48-bit key, we first need to expand RI−1 to 48 bits.
4) Step 4: Although the relationship between the input and output can be defined mathematically, DES uses Table to define this P-box.
5) Step 5: After the expansion permutation, DES uses the XOR operation on the expanded right section and the round key. Note that both the right section and the key are 48-bits in length. Also note that the round key is used only in this operation.
6) Step 6: S-Boxes
   The S-boxes do the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit.
7) Step 7: Apply s-box rule.
8) Step 8: We can make all 16 rounds the same by including one swapper to the 16th round and add an extra swapper after that (two swappers cancel the effect of each other). The round-key generator creates sixteen 48-bit keys out of a 56-bit cipher key.

We maintain the log files for the communication process which help in detecting the archives files. We can maintain the key for encryption and decryption process. We also maintain the time for communication sender and receiver. If any attack

will be happened then our alert system can alert the client and it will be faster as shown in the result section. So in this approach we can make the framework secure by applying Des algorithm as suggested in [15]. So the suggested framework can provide a seeds of better security with timeline alert in the client side.

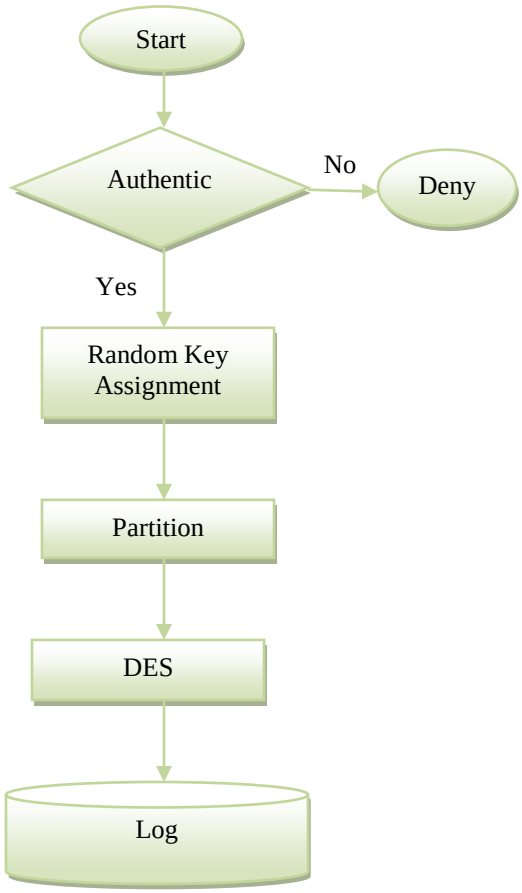

Figure 1: Flowchart

4. Result Analysis

First servers maintain the data before send to the client which is shown in table 1. In this table the key is also stored. Then after sending the data it stores the sending and receiving time also. After sending the data, client also maintains a log file with the user and the tag bit. If the tag bit is 0 it means the file is hacked as shown in table 3. Then client maintains the response time as shown in table 4. Then in table 5 clients maintains the alert time by which any client can receives the alert of attack. The response time is much better than the [13] and bit efficient than the [10]. Sometimes the results produces by our work is equivalent to the [10] but our work provides better security as it uses Des algorithm for encryption and decryption as shown in table5. The files taken are same as that in the previous work with the same size, so we can say that it is an efficient way of communication in the server and client environment.

5. Conclusion

Web-based attacks due to program security vulnerabilities are huge concerns for users. In this paper we proposed an efficient approach with DES encryption for better data receiving and sending mechanism. The future insights in this area are automatic file rendering. There are several file format which are not covered like .ps,.zip,.gif. There is also a scope in the direction of flash type files.

References

[1] Gaurav S. Kc, Angelos D. Keromytis, and Vassilis Prevelakis. Countering code-injection attacks with instruction-set randomization. In CCS '03: Proceedings of the 10th ACM conference on Computer and communications security, pages 272–280, New York, NY, USA, 2003.

[2] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic. Noxes: A Client-Side Solution for Mitigating Cross Site Scripting Attacks. In Proceedings of the ACM Symposium on Applied Computing (SAC), Dijon, France, April 2006.

[3] G. Wassermann and Z. Su, "Static Detection of Crosssite Scripting Vulnerabilities", Proceedings of the 30th ICSE,Leipzig, Germany, May 2008, pp. 171-180.

[4] Tramontana, "Identifying Cross Site Scripting Vulnerabilities in Web Applications", Proceedings of the Sixth International Workshop on Web Site Evolution (WSE 2004), Chicago, September 2004, pp. 71-80.

[5] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E.Kirda, C. Kruegel, and G. Vigna, "Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications", Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA, May 2008, pp. 387-401.

[6] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic, "Noxes: A Client-Side Solution for Mitigating Cross-Site Scripting Attacks", Proceedings of 21st ACM Symposium on Applied Computing, Dijon, France, April 2006, pp. 330-337.

[7] E. Ofuonye and J. Miller, "Resolving JavaScript Vulnerabilities in the Browser Runtime", Proceedings of the 19th International Symposium on Software Reliability Engineering, Washington DC, November 2008, pp. 57-66.

[8] Hossain Shahriar and Mohammad Zulkernine," MUTEC: Mutation-based Testing of Cross Site Scripting", IEEE 2009.

[9] Syed Imran Ahmed Qadri, Prof. Kiran Pandey, "Tag Based Client Side Detection of Content Sniffing Attacks with File Encryption and File Splitter Technique", International Journal of Advanced Computer Research (IJACR), Volume-2, Number-3, Issue-5, September-2012.

[10] Animesh Dubey, Ravindra Gupta, Gajendra Singh Chandel," An Efficient Partition Technique to reduce the Attack Detection Time with Web based Text and PDF files", International Journal of Advanced Computer Research (IJACR),Volume-3 Number-1 Issue-9 March-2013.

[11] Zubair M. Fadlullah, Tarik Taleb,Athanasios V. Vasilakos, Mohsen Guizani and Nei Kato, "DTRAB: Combating Against Attacks on Encrypted Protocols Through Traffic-Feature Analysis", IEEE/ACM Transactions On Networking, Vol. 18, No. 4, August 2010.

[12] Misganaw Tadesse Gebre, Kyung-Suk Lhee and ManPyo Hong, "A Robust Defense against Content Sniffing XSS Attacks", IEEE 2010.

[13] Anton Barua, Hossain Shahriar, and Mohammad Zulkernine, "Server Side Detection of Content Sniffing Attacks", 2011 22nd IEEE International Symposium on Software Reliability Engineering.

[14] Ms.Namrata Shukla, Ms. Shweta Pandey," Document Fraud Detection with the help of Data Mining and Secure Substitution Method with Frequency Analysis", International Journal of Advanced Computer Research (IJACR),Volume 2 Number 2 June 2012.

[15] Ashutosh Kumar Dubey,Animesh Kumar Dubey, Mayank Namdev, Shiv Shakti Shrivastava,"Cloud-User Security Based on RSA and MD5 Algorithm for Resource Attestation and Sharing in Java Environment", CONSEG 2012.

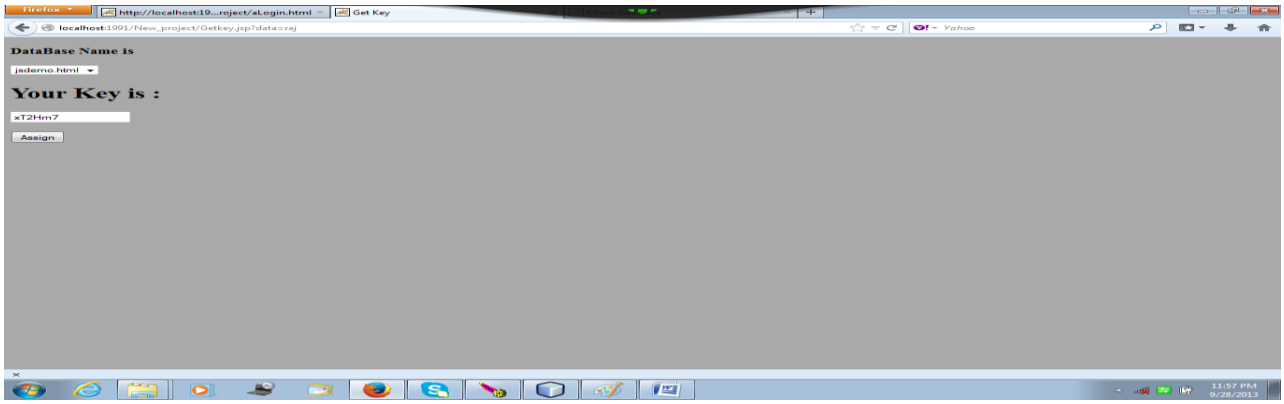

Figure 2: Key Generation

Table 1: Log File before Data Send to the Client

Log File after Data Send to the Client						
Fname	**tagcount**	**js**	**php**	**loc**	**tag**	**key**
fundemo2.html	0	0	0	0	1	bG3Md5
first.html	4	1	0	12	1	cK2Js1
demo.html	0	0	0	0	1	dF3Bi7
anim.html	0	0	0	0	1	hT7Bf5
regexp.html	4	1	0	25	1	iI7Wd6
54.pdf	0	0	0	0	1	iQ9Ef3
demo.html	0	0	0	0	1	kA2En7
ab1.html	0	0	0	0	1	kC3Yl6
ca.pdf	0	0	0	0	1	lU6Uv7

demo.html	0	0	0	0	1	nC6Qy8
jsdemo.html	0	0	0	0	1	rM8Ez8
54.pdf	0	0	0	0	1	sR3Gq5
regexp.html	4	1	0	25	1	sY5Ms2
ajax1.html	0	0	0	0	1	tA4Kr7
demo.html	0	0	0	0	1	uI3Pz8
54.pdf	0	0	0	0	1	wL0Nz7
demo.html	0	0	0	0	1	wM6Nk0
demo.html	0	0	0	0	1	xL2Iq1
jsdemo.html	0	0	0	0	1	xT2Hm7
first.html	0	0	0	0	1	yK0Mz6
demo.html	0	0	0	0	1	yY1Ux7
abc.html	0	0	0	0	1	yY8Ov2

Table 2: Log File after Data Send to the Client

Log File Before Data Send to the Client								
fname	**tagcount**	**js**	**php**	**loc**	**tag**	**key**	**sendingtime**	**rectime**
first.html	4	1	0	12	1	cK2Js1	4:13:47:656	4:13:47:703
54.pdf	0	0	0	0	1	wL0Nz7	4:43:15:421	4:43:15:437
demo.html	0	0	0	0	1	yY1Ux7	3:18:31:812	3:18:31:828
demo.html	0	0	0	0	1	yY1Ux7	3:25:4:906	3:25:4:921
54.pdf	0	0	0	0	1	wL0Nz7	3:46:4:187	3:46:4:218
demo.html	0	0	0	0	1	yY1Ux7	3:50:56:265	3:50:56:281
ab1.html	0	0	0	0	1	kC3Yl6	4:1:55:0	4:1:55:31
demo.html	0	0	0	0	1	yY1Ux7	4:18:9:656	4:18:9:671
demo.html	0	0	0	0	1	yY1Ux7	4:19:50:343	4:19:50:359
demo.html	0	0	0	0	1	yY1Ux7	4:20:39:859	4:20:39:875
demo.html	0	0	0	0	1	yY1Ux7	4:21:25:0	4:21:25:31
demo.html	0	0	0	0	1	yY1Ux7	4:22:52:515	4:22:52:515
abc.html	0	0	0	0	1	yY8Ov2	3:39:0:901	3:39:0:932
anim.html	0	0	0	0	1	hT7Bf5	3:50:29:636	3:50:29:636
first.html	4	1	0	12	1	cK2Js1	4:20:26:932	4:20:26:948
jsdemo.html	0	0	0	0	1	rM8Ez8	4:34:59:995	4:35:0:11
ajax1.html	0	0	0	0	1	tA4Kr7	4:41:23:714	4:41:23:729
jsdemo.html	0	0	0	0	1	rM8Ez8	11:59:50:304	11:59:50:429
ca.pdf	0	0	0	0	1	lU6Uv7	0:6:49:854	0:6:49:948
fundemo2.html	0	0	0	0	1	bG3Md5	0:14:4:95	0:14:4:195

Table 3: Client Table

Client Table								
fname	**tagcount**	**js**	**php**	**loc**	**tag**	**key**	**client**	**size**
fundemo2.html	0	0	0	0	0	bG3Md5	raj	352
first.html	4	1	0	12	1	cK2Js1	amit	184
first.html	4	1	0	12	1	cK2Js1	ram	
anim.html	0	0	0	0	1	hT7Bf5	amit	29789
ab1.html	0	0	0	0	1	kC3Yl6	ram	2348
ca.pdf	0	0	0	0	1	lU6Uv7	raj	20859
jsdemo.html	0	0	0	0	1	rM8Ez8	AMIT	316
jsdemo.html	0	0	0	0	1	rM8Ez8	raj	316
ajax1.html	0	0	0	0	0	tA4Kr7	amit	666

54.pdf	0	0	0	0	1	wL0Nz7	ram	190143
demo.html	0	0	0	0	1	yY1Ux7	ram	307
demo.html	0	0	0	0	1	yY1Ux7	ram	307
demo.html	0	0	0	0	1	yY1Ux7	ram	307
demo.html	0	0	0	0	1	yY1Ux7	ram	307
demo.html	0	0	0	0	1	yY1Ux7	ram	307
demo.html	0	0	0	0	1	yY1Ux7	ram	307
demo.html	0	0	0	0	1	yY1Ux7	ram	307
abc.html	0	0	0	0	1	yY8Ov2	amit	295

Table 4: Log File before Attack

Log File Before Attack			
fname	**size**	**respose**	**sendtime**
regexp.html	559	3:58:41:406	
first.html	184	4:13:40:937	4:20:26:932
54.pdf	190143	4:40:42:828	3:46:4:187
54.pdf	190143	4:42:12:171	3:46:4:187
demo.html	307	3:18:10:265	4:22:52:515
demo.html	307	3:24:22:609	4:22:52:515
54.pdf	190143	3:45:28:718	3:46:4:187
demo.html	307	3:49:6:890	4:22:52:515
ab1.html	2348	4:1:51:250	4:1:55:0
demo.html	307	4:18:6:687	4:22:52:515
demo.html	307	4:19:47:718	4:22:52:515
demo.html	307	4:20:37:500	4:22:52:515
demo.html	307	4:22:49:890	4:22:52:515
abc.html	295	3:38:52:42	3:39:0:901
anim.html	29789	3:50:18:495	3:50:29:636
first.html	184	4:20:21:651	4:20:26:932
jsdemo.html	316	4:34:56:651	11:59:50:304
ajax1.html	666	4:41:19:214	4:41:23:714
jsdemo.html	316	11:59:25:994	11:59:50:304
ca.pdf	20859	0:6:40:974	0:6:49:854
fundemo2.html	352	0:13:58:994	0:14:4:95

Table 5: Log File after Attack

Log File after Attack				
fname	**size**	**attacktime**	**servertime**	**Time (s)**
first.htm	304	4:24:23:776	4:24:23:792	16
jsdemo.htm	498	4:37:3:136	4:37:3:151	15
ajax1.htm	66	4:41:37:557	4:41:37:573	16
ajax1.html	101	4:42:39:698	4:42:39:714	16
fundemo2.html	37	0:15:54:594	0:15:54:731	137