International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-2 Number-3 Issue-5 September-2012

Tag Based Client Side Detection of Content Sniffing Attacks with File Encryption and File Splitter Technique

Syed Imran Ahmed Qadri[#1], Prof. Kiran Pandey[#2]

All Saints' College of Technology, Bhopal[#]
iqadri@gmail.com[#1], kiran.mishra.bhu@gmail.com[#2]

Abstract

In this paper we provide a security framework for server and client side. In this we 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. We 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.

Keywords

Content sniffing, Encryption, Decryption, Message Digest, Tag Bit

I. Introduction

In today's scenario we rely on web-based programs or web applications to perform many essential activities. They usually reside on a server-side and are accessed from its client-side. There are some approaches which is either applied on client side as well as the server side but overall the approaches are not well enough to protect with the vulnerabilities. As a result users are fear and sometimes he/she may be suffering from those vulnerabilities.

The above scenario might result in stealing of session information and generation of anomalous runtime behaviors. The situation further worsens when many web-based programs are deliberately designed and deployed to mimic trusted websites that have explicit authentication mechanisms and employ active session information to perform for stealing personal information for example phishing websites [1] instead of providing legitimate functionalities. Thus, the mitigation of web-based security vulnerability exploitations is extremely important to reduce some of the consequences.

For this reason we study a number of common program security problems and vulnerabilities [2][3]. Our study focuses that the number of web-based attacks has increased in recent years [4][5], existing research has addressed a subset of security vulnerabilities in web applications for example SQL Injection. After observation from several research by different authors, we analyze there are several numbers of vulnerabilities are still in the communication process when we want to access data from the web. We believe that if we prevent the attack from the server side and it will be notified to the client then we can prevent the attack.

The remaining of this paper is organized as follows. In Section 2 we discuss about problem domain. The Evolution and recent scenario in section 3.In section 4 we discuss about proposed approach. In section 5 we discuss about result analysis. The conclusions and future directions are given in Section 6. Finally references are given.

2. Problem Domain

There are several attack detection approaches that are deployed at program runtime [6][7][8][9][10]. We identify several limitations for these approaches. First, most of the attack detection approaches rely on the modification of both server and client-side environments and the exchange of sensitive information between the two sides. Second, existing approaches do not adequately address some attack types like injecting legitimate JavaScript code and

content sniffing. Third, most approaches assume that web based programs are trusted and legitimate. But in the real scene this assumption does not hold in many cases such as suspected phishing websites that are deliberately designed to steal personal credential information. Taking consideration on the above point, there are some considerations for the research orientation:

- Is it is possible to detect an automatic system which automatically enable the tag bit and the associated clients if the client is the part of the network.
- Are we detecting attacks at the client-side without any a priori information from remote site?
- As a part of security we observe the need a proper security in the form of encryption and decryption.
- Are we reducing the overhead of transferred data by applying some file splitting technique?

We come with the solution in the subsequent section.

3. Evolution and Recent Scenario

In 2009, Adam Barth et al. [11] focused on Cross-site scripting defenses often on HTML documents, neglecting attacks involving the browser's content sniffing algorithm, which can treat non-HTML content as HTML. Web applications, such as the one that manages this content, must defend themselves against these attacks or risk authors uploading malicious papers that automatically submit stellar self-reviews. In this research, they formulate content-sniffing XSS attacks and defenses. They study content sniffing XSS attacks systematically by constructing high fidelity models of the content-sniffing algorithms used by four major browsers. They compare these models with Web site content filtering policies to construct attacks. To defend against these attacks, we propose and implement a principled content-sniffing algorithm that provides security while maintaining compatibility. Their principles have been adopted, in part, by Internet Explorer 8 and, in full, by Google Chrome and the HTML 5 working group.

In 2010, Zubair M. Fadlullah et al. [12] 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 our unique approach DTRAB since it focuses on both Detection and TRAceBack in the MS level. The effectiveness of the proposed detection and traceback methods are verified through extensive simulations and Internet datasets.

In 2011, Misganaw Tadesse Gebre et al. [13] 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 shows that the proposed automata-based scheme is highly efficient and more accurate than existing signature-based approach.

In 2011, Anton Barua et al. [14] 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.

4. Proposed Approach

In this paper we have proposed a secure server client environment for detecting content sniffing attack.

This approach provides the security in the server side and alert the client which reduces the non secure violation with data use. In this approach client want to establish a secure connection from the server for gathering data from the server. Client simply requests the data and the admin provides the available resources from the server database. Admin first encrypt the data by Private key cryptography, which

uses the same key to encrypt and decrypt the message. This type is also known as symmetric key cryptography. In java we can use Base64 encoding and decoding as defined by RFC 2045 which provide a symmetric key encryption. With symmetric encryption, both parties use the same key for encryption and decryption purposes.

Each user must possess the same key to send encrypted messages to each other.The sender uses the key to encrypt their message, and then transmits it to the receiver. The receiver, who is in procession of the same key, uses it to decrypt the message.

The security of this encryption model relies on the end users to protect the secret key properly. If an unauthorized user were able to intercept the key, they would be able to read any encrypted messages sent by other users. It's extremely important that the users protect both the keys themselves, as well as any communications in which they transmit the key to another person.

Symmetric is conceptually simple. It's the "secret decoder ring" model. The same "secret decoder ring" is used to encrypt and decrypt messages.

Conceptually you might think of it as similar to physical lock, perhaps a door lock. The same key is used to lock and unlock the door [Figure 1]. Java supports encryption based on base 64.

Content-Transfer-Encoding from RFC 2045 Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies by Freed and Borenstein. The class can be parameterized in the following manner with various constructors:

URL-safe mode: Default off.

Line length: Default 76. Line length that aren't multiples of 4 will still essentially end up being multiples of 4 in the encoded data.

Line separator: Default is CRLF ("\r\n")

Since this class operates directly on byte streams, and not character streams, it is hard-coded to only encode/decode character encodings which are compatible with the lower 127 ASCII chart (ISO-8859-1, Windows-1252, UTF-8, etc).

Creates a Base64 codec used for decoding (all modes) and encoding in URL-unsafe mode.

When encoding the line length is 0 (no chunking), and the encoding table is STANDARD_ENCODE_TABLE. When decoding all variants are supported. Base64

Public Base64 (boolean urlSafe) [java Supported Encryption]

Creates a Base64 codec used for decoding (all modes) and encoding in the given URL-safe mode.

When encoding the line length is 76, the line separator is CRLF, and the encoding table is STANDARD_ENCODE_TABLE. When decoding all variants are supported.

Parameters:

urlSafe - if true, URL-safe encoding is used. In most cases this should be set to false.

Then we split the file according to the length which reduces the complexity span and send to the user. We also provide a tag bit checking based which alerts the client if any content based alteration is done. We also provide the memory buffer which detects the content alteration; this is done by any message digest algorithm.

MD5 algorithm was developed by Professor Ronald L. Rivest in 1991. According to RFC 1321, "MD5 message-digest algorithm takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input …The MD5 algorithm is intended for digital signature applications, where a large file must be "compressed" in a secure manner before being encrypted with a private (secret) key .[Figure 2]

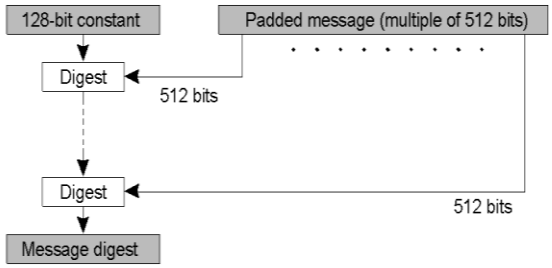

Figure 2: Message Digest

The flowchart for this algorithm is shown in Figure 3.

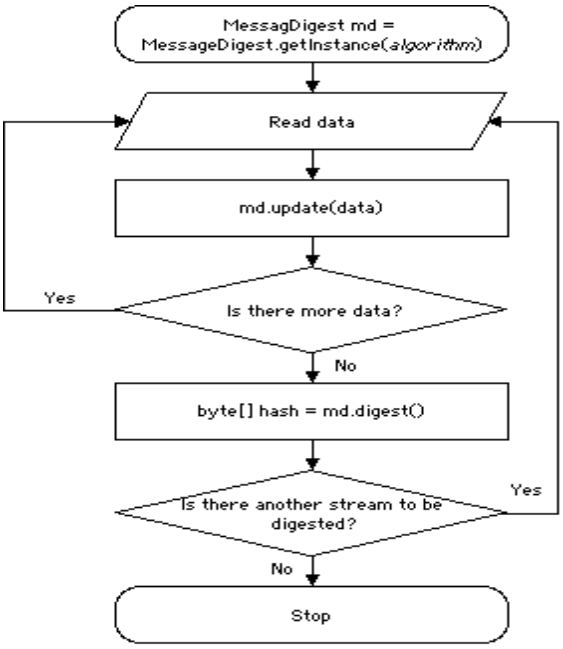

Figure 3: Flowchart for Message Digest

In a content sniffing attack, rendering of downloaded non-HTML files result in the generation of HTML pages or the execution of JavaScript code at victim's browser . The files are uploaded by attackers that contain malicious payloads. These files seem benign when we consider their content types or Multipurpose Internet Mail Extension (MIME) information. For example, a GIF file having a MIME image/gif might contain JavaScript code (<script>...</script>). An attack occurs when a victim's browser renders a non-HTML file as an HTML file. A successful attack might result in severe consequences such as stealing of session information and passing information to third party websites.

Browsers employ content sniffing algorithms to detect file content types and render them accordingly by scanning the initial bytes of a downloaded file to identify the MIME type. For example, Internet Explorer 7 examines the first 256 bytes of a file for specific signatures that represent specific file types. Internet Explorer 7 treats a file as image/gif, if the file begins with GIF87 or GIF89. Firefox performs the same, if the file begins with GIF8. Browsers also differ in ways they search for HTML tags for matching with HTML signatures and enforcing the rules when response contents are sniffed as HTMLs. For example, Google Chrome does not sniff a file as an HTML when the Content-Type header is known, text/plain, or application/octet-stream. However, Internet Explorer 7 sniffs a file as an HTML, if the first 256 bytes contain any of the predefined signatures such as <html> and <script>. These inconsistencies among widely used browsers motivate attackers performing content sniffing attacks.

Table 1: Examples of File and MIME Types

File Type	MIME Type
HTML	text/html
Textual data	text/plain
JavaScript	application/JavaScript
Arbitrary binary data	application/octet-stream
Portable Document	Format application/pdf
GIF image	image/gif
JPEG image	image/jpeg

In a content sniffing attack, an attacker exploits the difference between a website's file upload filter (assuming that a website is legitimate) and a browser's content sniffing algorithm. An attacker uploads a seemingly benign file to a website that accepts the uploaded file and does not check the contents. Later, a victim views the file by downloading it in his/her browser. A typical response by a server with respect to a file request from a browser contains two parts: response header and response body. A response body contains the actual resource that has been requested. A response header defines various characteristics of the response body.

It is possible to provide more malicious payloads that can access a web program's session or cookie information and transfer to third party websites. This simple example illustrates some idea about different ways of performing content sniffing attacks. However, setting wrong Content-Type information might not always result in content sniffing attacks. A sample code for uploading and testing are given below.

Algorithm:

Step 1: Append padding bits

The input message is "padded" (extended) so that its length (in bits) equals to 448 mod 512. Padding is always performed, even if the length of the message is already 448 mod 512.

Step 2: Append length

A 64-bit representation of the length of the message is appended to the result of step1. If the length of the message is greater than 2^64, only the low-order 64 bits will be used.

Step 3: Initialize MD buffer

A four-word buffer (A, B, C, D) is used to compute the message digest. Each of A, B, C, D is a 32-bit register. These registers are initialized to the following values in hexadecimal, low-order bytes first):
word A: 01 23 45 67
word B: 89 ab cd ef
word C: fe dc ba 98
word D: 76 54 32 10

Step4:

It processes the message in 16-word blocks Four functions will be defined such that each function takes an input of three 32-bit words and produces a 32-bit word output.

F (X, Y, Z) = XY or not (X) Z
G (X, Y, Z) = XZ or Y not (Z)
H (X, Y, Z) = X xor Y xor Z
I (X, Y, Z) = Y xor (X or not (Z))

For maintain the information we create two types of databases one from the server side and one from the client side. In server side we maintain two copies of the same table one for Before Send and other for after send. E/D is the encryption decryption key. If the content is altered automatically tag bit is 1 which implies that there is a change in the file. It is automatically alerted to the client, so those clients rerequest the data from the server. Server also maintain the time of sending and receiving of files.

Table 2: Server Side Database (Before Send)

Name	**Tag Count**	**Methods**	**JavaScript**	**PHP**	**LOC**	**E/D Key**	**Tag Bit**
Ab.html	2	5	12	10	150		0
Pq.html	6	7	2	5	200		0
Rs.html	3	8	4	7	180		0

Table 3: Server Side Database (After Send)

Name	Tag Count	Methods	Java Script	PHP	LOC	Tag Bit
Ab.html	2	5	12	10	150	0
Pq.html	6	7	2	5	200	1
Rs.html	3	8	4	7	180	0

Table 4: Client Database (Before Send)

Name	Time	E/D Key	Tag Bit
Ab.html	2		0
Pq.html	1		1
Rs.html	3		0

5. Result Analysis

The result produce by the above algorithm is shown in Figure 5 to Figure 7. When a client sends a request to the server. Server first assign a key to the client for the particular web file and the tag bit is set to be 1.This phenomena is shown in Figure 5.Then server decompose and encrypt it for the purpose of sending data. In this stage if any content sniffer change or delete the data, it is automatically replicated to the server and the tag bit is changed to 0 instead of 1 which shows that the values are changed by the outsiders. Then server replicates the tag bit to client also so that client must aware of that data changes and beware of the use of data. Our server alerts times shows this mechanism with time calculation when server knows the information about the change data. The time period which our mechanism shows is in millisecond which shows that it is better than the previous mechanism.

Figure 6 shows the data which was send to the client. Some of the data was attacked and some was not which will be identify by the tag bit. The time of attack is shown in table 8.

Table 5: Data before Send from the Server

beforesend						
fname	tagcount	js	php	loc	tag	key
file1.html	101	33	0	365	1	iC0Ye9
file2.html	146	48	0	530	1	iI8Ca9
file4.html	451	150	0	1651	1	uK7Pa2
file3.html	385	128	0	1409	1	vF4Ix8
file4.html	451	150	0	1651	1	bO6Ai4
file5.html	493	164	0	1805	1	rK5Ej7

Table 6: Data after Send from the Server

AFTERSEND								
FNAME	TAGCOUNT	JS	PHP	LOC	TAG	KEY	SENDINGTIME	RECTIME
file1.html	101	33	0	365	1	iC0Ye9	10:39:40:7	10:39:40:54
file2.html	146	48	0	530	1	zP6Xl4	10:44:2:586	10:44:2:649
file3.html	385	128	0	1409	1	xI5Ux5	10:46:12:67	10:46:12:116
file4.html	451	150	0	1651	1	bO6Ai4	10:54:55:506	10:54:55:569
file5.html	493	164	0	1805	1	rK5Ej7	11:2:17:174	11:2:18:237

Table 7: Data after Attack

afterattack			
fname	size	attacktime	servertime
file1.html	2822	10:40:1:332	10:40:1:480
file2.html	4104	10:44:22:461	10:44:22:650
file3.html	10945	10:46:28:431	10:46:28:610
file4.html	12826	11:1:23:570	11:1:23:713
file5.html	14023	11:2:45:231	11:2:45:370

6. Conclusion and Future Direction

Web-based attacks due to program security vulnerabilities are huge concerns for users. While performing seemingly benign functionalities at the browser-level, users might become victims without their knowledge. These might lead to unwanted malicious effects such as the execution of JavaScript code that accesses and transfers credential information to unwanted websites and the filling of forms that result in stealing login credentials. In this paper, we address the mitigation of some of these exploitations by developing automatic attack detection approaches at both server and client-sides.

Our future work on content sniffing attack detection includes identifying ways to reduce the overhead for large files. We plan to evaluate our approach for some other file types such as flash. We convert the MIME type of any file into HTML manually. We plan to find an automated way to perform the MIME type conversion. The future work also includes the automated identification of file upload procedures to integrate our filter.

References

[1] D. Geer, "Security Technologies Go Phishing," Computer Archive, Volume 38, Issue 6, June 2005, pp. 18-21.

[2]H. Shahriar and M. Zulkernine, "Mitigating Program Security Vulnerabilities: Challenges and Approaches," ACM Computing Surveys, Vol. 44, Issue 3, September 2012.

[3] H. Shahriar and M. Zulkernine, "Taxonomy and Classification of Automatic Monitoring of Program Security Vulnerability Exploitations," Journal of Systems and Software, Elsevier Science, Vol. 84, Issue 2, February 2011, p. 250-269.

[4] Z. Mao, N. Li, and I. Molloy, "Defeating Cross-Site Request Forgery Attacks with Browser - Enforced Authenticity Protection," Proc. of Financial Cryptography and Data Security, Barbados, Feb 2009, p. 238-255.

[5] Phishing Activity Trends Report, 2010, Accessed from www.antiphishing.org/reports/apwg_report_Q1_2010.pdf.

[6] Y. Zhang, J. Hong, and L. Cranor, "CANTINA: A Content-based Approach Detecting Phishing Websites," Proc. of the 16th International Conference on World Wide Web (WWW),Banff, Alberta, Canada, May 2007, pp. 639-648.

[7] M. Alalfi, J. Cordy, and T. Dean, "WAFA: Fine-grained Dynamic Analysis of Web Applications," Proc. of the 11th International Symposium on Web Systems Evolution (WSE), Edmonton, Canada, Sept 2009, pp. 41-50.

[8]M. Gundy and H. Chen, "Noncespaces: Using Randomization to Enforce Information Flow Tracking and Thwart Cross-site Scripting Attacks," Proc. of the 16th Annual Network and Distributed System Security Symposium (NDSS), San Diego, California, USA, February 2009.

[9] Y. Nadji, P. Saxena, and D. Song, "Document Structure Integrity: A Robust Basis for Crosssite Scripting Defense," Proc. of the 16th Annual Network and Distributed System Security Symposium (NDSS), San Diego, California, USA, February 2009.

[10] T. Jim, N. Swamy, and M. Hicks, "Defeating Script Injection Attacks with Browser- Enforced Embedded Policies," Proc. of the 16th International Conference on World Wide Web, Banff, Alberta, Canada, May 2007, pp. 601-610.

[11] Adam Barth, Juan Caballero and Dawn Song , "Secure Content Sniffing for Web Browsers, or How to Stop Papers from Reviewing Themselves", 2009 30th IEEE Symposium on Security and Privacy.

[12] 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.

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

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

Syed Imran A. Qadri:- He received his B.E. degree in Computer Science & Engineering from Rajiv Gandhi Prodyogiki Vishwavidyalaya, Bhopal, M.P. India in 2009. Presently he is pursuing his M. Tech. degree in Computer Science & engineering from Rajiv Gandhi Prodyogiki Vishwavidyalaya, Bhopal, M.P. India.

Dr. Kiran Pandey:- She Received her B.E. in Computer Science & Engineering From KNMIET Gaziabaad India in 2002 and PhD. in Artificial Intelligence from BHU Banaras, India, in 2011. She is currently working as Professor with the Department of Computer Science & Engineering at All Saints' College Of Technology, Bhopal, India, from June 2012 to till date, her professional research interests include Artificial Intelligence, Network Security and Data Mining.