Implementation of Obfuscation Technique on PHP Source Code

Maskur Maskur
Informatics Department
Universitas Muhammadiyah Malang
Malang, Indonesia
maskur@umm.ac.id

Zamah Sari
Informatics Department
Universitas Muhammadiyah Malang
Malang, Indonesia
zamahsari@umm.ac.id

Ahmad Sirojul Miftakh
Informatics Department
Universitas Muhammadiyah Malang
Malang, Indonesia
sirojulmiftakh@umm.ac.id

***Abstract*—Source code on web based applications can be altered easily. This occurred because the source code is not compiled into an executable file. Hence, it can be read and copied easily, or be changed without permission from the author. Obfuscation is a technique that commonly used to secure the source code in any websites based application. Obfuscation is a technique to randomize the source code that make the code harder to read but still runnable, but this make the running time increased and the application will run slower then it supposed to. This increased time caused by reverse obfuscation proses to bring back the source code into originally form before interpreted by web server. This studi intended to create an obfuscation technique that keeping the application run time performance as not obfuscated called Wanna Crypt. The methods to create this applications are (1) system design using UML, (2) implementation of the system, which is done by coding or writing scripts using PHP, HTML, JavaScript, CSS to build Wanna Crypt based website, (3) Blackbox and Whitebox testing to compare the execution time. From the tests, it can be concluded that web applications using Wanna Crypt provide a longer response time than web applications without using obfuscation.**

Keywords— php, source code, encryption, reverse engineer, obfuscation

I. INTRODUCTION

Open source technology used in any software development especially web based application can be build with a fairly low cost in its development. But keep in mind that any open source technology also opens up opportunities for vulnerabilities that can be misused by irresponsible parties due to the source code of software also included into the software deployment. Source code for web based software that is not compiled into executable files allows others to manipulate and even alter malicious code on the application to be built or developed. Aplication that are processed or executed without being transformed into executable files cause the source code easily known by people and modified or altered without permission[1] [2].

This may result that the software being vulnerable to manipulation, duplication, unauthorized permission, and even unknowingly altered code that can harm to the system when the application is in operation. To overcome the software security case, proposed an approach in the technique of obfuscation by distributing the ciphertext which is the result of the stages of randomization of data or strings in a particular form in accordance with the rules of php language code execution[3][4][5][6]. This research propose a solution through the development of a Wanna Crypt software using a text distribution approach to ciphertext generated through obfuscation process in accordance with the code execution code PHP so that the source code can still be processed by the web server. Also built a loader which is a new extension in the web server as a support for processing the source code that has been obfuscated using Wanna Crypt application.

Krisma Pradana in [7] was designed an encoder and decoder application intended to obfuscation against a web based software. But the software is focused on pure source code that is purely incompatible with scripts other than the php programming language. This causes problems when the source code has been mixed with other tags such as HTML tags, JavaScript or CSS which are the builder code of a web based software. While, in [8], Oki Setiawan, who also in his research applied the rivest cipher cryptography algorithm to secure the source code of php programming language with obfuscation technique resulted in execution time of 199.86%.

Those previous studies resulted in obfuscation technique that still incompatible to some elements in web based software, and much longer execution time compered to web without obfuscation technique. However, this study tries to solve the problem by designing system that provide obfuscation technique that more compatible with web based software tags and better execution time.

Wanna Crypt is designed and built on web based technology. Wanna Crypt version 1.0.0 is built using Codeigniter's PHP framework which refers to object-oriented programming [9] based approach and Model-View-Controller pattern design. Wanna Crypt is the result of Test Case Code Execution by observing the way of evaluation and execution of PHP code by web server. Wanna Crypt is also designed by building new extensions that will be integrated on the web server. Wanna Crypt designed with the hope of being accessible online on every mobile platform that will have low impact on execution time.

II. DESIGN AND IMPLEMENTATION

A. System Design

The design of the system is done to identify the framework or blueprint that can be used as a reference in making the system. System design is done by using flowchart or flowchart approach to arrange system framework. The flow diagram used to build the Wanna Crypt application uses the Unified Modeling Language (UML) scheme to construct the system functionality blueprint scheme to be built. UML scheme used is (1) Use Case Diagram, (2) Activity Diagram, (3) Component Diagram, and (4) Sequence Diagram.

978-1-5386-8402-3/18/$31.00 ©2018 IEEE

B. *Implementation*

Implementation of the system is done by coding or writing scripts using PHP, HTML, JavaScript, CSS to build Wanna Crypt based website. Implementation is done by using Object Oriented Programming (OOP) coding technique using Codeigniter framework which refers to Model-View-Controller (MVC) pattern design.

In addition, to build the loader is done by coding using C programming language and Visual Studio applications to compile the process into a library of Dinamic Link Library (DLL). Implementation of the system is done by referring to the results of the previous system design.

III. RESULT AND DISCUSSION

Wanna Crypt application testing will be done by applying black box testing techniques and also white box. Black box testing technique is done to ensure that the functionality built in Wanna Crypt application has been running properly and correctly. Furthermore, the white box tested to test whether the ciphertext generated from the obfuscation process can be read or re-executed by the web server and have the most efficient code structure. Other things observed by looking at the comparison of execution time between the source code in plaintext and ciphertext to see the performance comparison between the obfuscated and non-obfuscated web applications.

A. *Obfuscation Steps*

This stage is a series of process of transformation and repackaging of ciphertext generated by Wanna Crypt in randomization of source code in accordance with php programming language structure. The scheme of the obfuscation process designed in the Wanna Crypt application can be seen in the schema below.

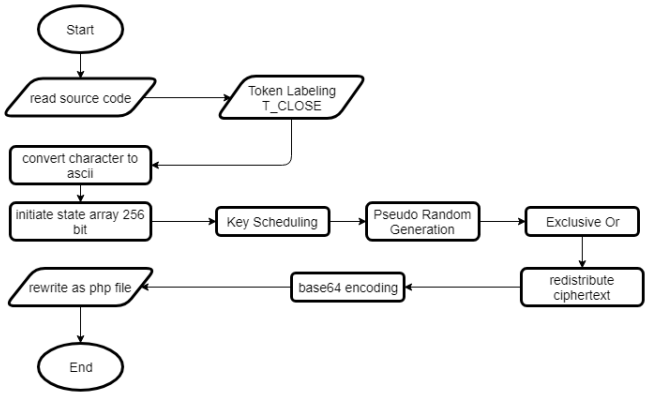

Fig. 1 Schema of obfuscation proses in Wanna Crypt Application

Figure 1 illustrates the path of the obfuscation stage designed in the Wanna Crypt application. The process of obfuscation is done by (1) reading the source code file, (2) tagging the php tag, (3) obtaining the ascii value of each character, (4) initializing the state array with the value 0 to 255, (5) scheduling, (6) running the pseudo random generation method, (7) doing exclusive or encryption (8) compiling the php code execution flow, (9) doing base64 encoding, and (10) rewriting the php source code source into php extension file.

B. *Obfuscated Source Code Proses Steps*

This stage is a series of code execution process to source code result of obfuscation by Wanna Crypt application. Here is the scheme of the process execution code results obfuscation using Wanna Crypt application. Obfuscaition code processing schemes are described in the following scheme:

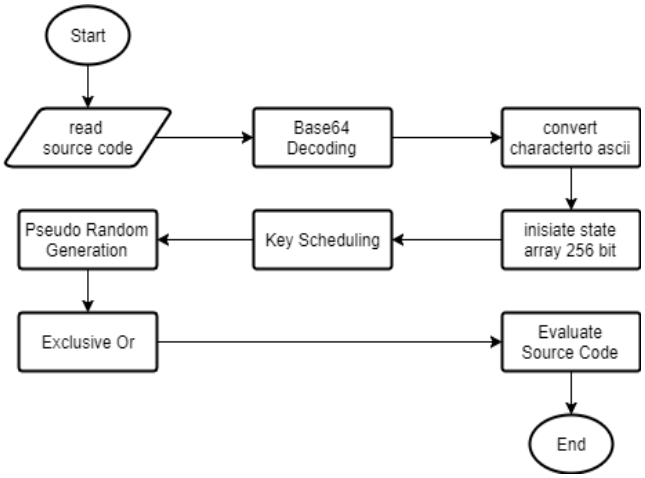

Fig. 2 Schema of Obfuscated Source code execution using Wanna Crypt Application

Figure 2 describes the obfuscation source code execution method using Wanna Crypt with the following paths: (1) reading the source code of the obfuscation result, (2) decoding using base64, (3) obtaining the ascii value of each character, (4) initializing state array (5) executes the key scheduling method, (6) runs the pseudo random generation method, (7) executes the exclusive method or, (8) executes the cipher text source code.

C. *Rivest Chiper 4 Algoritm*

Keystream type algorithms that process plaintext changes into ciphertext bytes per byte with simpler and faster computations [10][11][12]. This algorithm can use a key length of 1 to 256 bytes which will be used to initialize the 256 byte substitution key table.

1) Encryption Method

a) Array Initialization: At this stage provided 2 pieces of arrays, each of which is a security key and substitution box. This array substitution box is initialized initially with values from 0 to 255 while the security key array is populated with the decimal value of each character on the security key until it is fully loaded in the array.

b) Key Scheduling Method: At this stage re-initialization process is done on array substitution box or sbox with value along 1 to 256 bit which is repeatedly filling security key that is inputted or used. The results of this stage will be reused in the encryption key generation process. Calculation of key scheduling method can be seen in mathematical calculation below:

```
i=0 dan j=0
j=(j+sbox[i]+key[i])  % 256
temp=sbox[i]
sbox[i]=sbox[j]
sbox[j]=temp                    (1)
```

Equation 1 contains an explanation of the substitution process of the encryption key generation array that will be used for the transformation of each character in the php source code.

c) Pseudo Random Generation Method: At this stage the encryption key is generated which will be processed using the exclusive method or with each character on the plaintext. The result of this step will be used as an exclusive or encryption key to any character contained in the plaintext or source code. Calculation of this method can be seen in the equation below:

```
a=0,j=0,dan i=0
a=(a+1)% 256
j=(j+sbox[a])% 256
temp=sbox[a]
sbox[a]=sbox[j]
sbox[j]=temp                                    (2)
```

Equation 2 contains an explanation of the calculations performed in the modification process of the state of the security lock generation. The substitution process is performed to randomize the contents of the sandbox data which is then used to generate the encryption key.

d) Exclusive Or Method: After the keystream generation of the specified security key is assigned, each character on the plaintext will be processed bytes per byte or character by character against each keystream that has been exclusively generated or so obtained by the ciphertext of each plaintext. Calculation of this method can be seen in the equation below:

```
k=sbox[((sbox[a]+sbox[j]))% 256]
cipher=plain[i]^ k                              (3)
```

Equation 3 contains an explanation of the computations performed in the exclusive or the method for the process of transforming the plaintext into ciphertext by using the xor logic to the bits of each character contained in the plaintext using the previously generated security key.

2) Decrypt Method

The method of decrypting or returning ciphertext to plaintext in the rivest cipher 4 algorithm uses the same path as in the encryption process. The security key used in the decryption process also exactly matches the security key used in the encryption process.

D. Web Application Test without Obfuscation

TABLE I. RESULT OF WEBSITE TEST PERFORMANCE WITHOUT OBFUSCATION

Response Time without Obfuscation		
User	***Avg. Time Spent (Ms)***	***Avg. Click Time(Ms)***
5	74.3	74.3
10	53.2	53.2
15	239.2	79.9
20	208.5	104.3
25	453.7	180.3
40	857.5	214.4
50	1340	268
80	3179.1	397.4
100	4811.8	481.1

The data in Table I is the data obtained from the test results of the application without using obfuscation. Testing is done by varying the number of users to observe the performance of the application.

E. Web Application Test with Obfuscation

Below is the data obtained from the test results of the application without using obfuscation. Testing is done by varying the number of users to observe the performance of the application. Notice Table 2.

TABLE II. RESULT OF WEBSITE TEST PERFORMANCE WITH OBFUSCATION

Response Time with Obfuscation		
User	***Avg. Time Spent (Ms)***	***Avg. Click Time(Ms)***
5	65.9	65.9
10	57.8	57.8
15	305.9	101.8
20	214.9	107.5
25	925.3	231.3
40	842.7	210.6
50	1375.8	275.3
80	3120.7	390.1
100	4913.1	491.4

F. Obfuscation Feature Testing

In this phase, an obfuscation of a PHP source code is mixed with HTML tags in order to see the success of the code execution process. Source code that has been obfuscation using Wanna Crypt placed back into the we server and then observed whether the obfuscated source code successfully processed and executed by the web server as the original source code or cause the source code becomes unreadable by the web server.

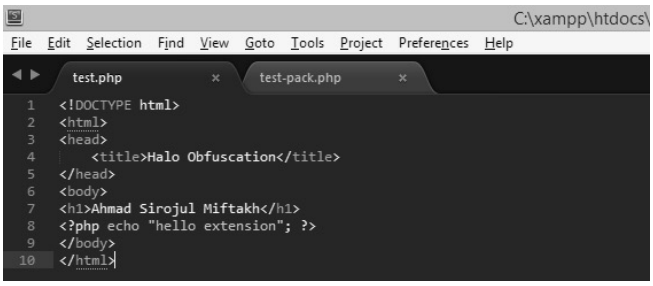

Fig. 3 The original software source code

Figure 3 shows the original source code of a software to be randomized using Wanna Crypt. Obfuscation testing is done by using PHP source code mixed with html tags. Obfuscation process is done by utilizing obfuscation method that has been designed in loader which built before and integrated into web server.

Fig. 4 Result of execution of original source code

Figure 4 contains the source code execution results on the web server. The original source code of a php-based language program that has been mixed with code or html tags is then randomized using obfuscation techniques then executed back into the web server. This test is performed to find out the results of obfuscation performed by Wanna Crypt successfully to be tested again on the web server that has been integrated with the loader.

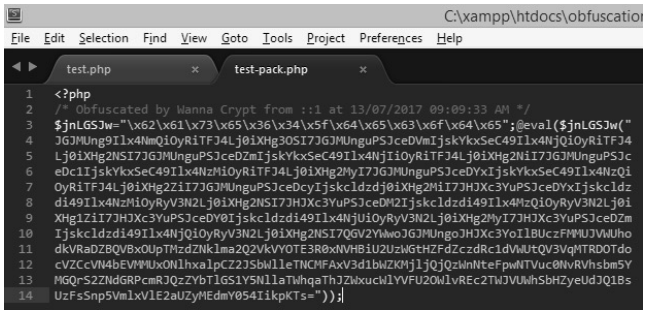

Fig. 5 Source code of obfuscation results using Wanna Crypt

Figure 5 shows the result of the obfuscation process using Wanna Crypt that has performed randomization or obfuscation on the PHP source code. Furthermore the source code generated will be executed again on web server that has been integrated with the loader which is also built in this research.

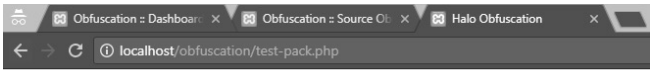

Fig. 6 The result of random source code execution

Figure 6 shows the results of the execution of the randomly generated source code producing exactly the same output as the original non-scrambled source code. This indicates that Wanna Crypt can scramble the source code properly without damaging or changing the program flow compiled in the original software.

Here is a comparison of the reliability of web application that use obfuscation against web application without using obfuscation. The observations in this test comparison are focused on the reliability of web applications that use obfuscation in response to user requests. Notice Table 3.

TABLE III. COMPARISON OF WEB APPLICATION WITH AND WITHOUT OBFUSCATION

Average Time Spent Analisys				
User	With Obfuscatoin (ms)	Without Obfuscation (ms)	Performance	Delta (ms)
5	65.9	74.3	Faster	-8.4
10	57.8	53.2	Slower	4.6
15	305.9	239.2	Slower	66.7
20	214.9	208.5	Slower	6.4
25	505.4	453.7	Slower	51.7
40	842.7	857.5	Faster	-14.8
50	1375.8	1340	Slower	35.8
80	3120.7	3179.1	Faster	-58.4
100	4913.1	4811.8	Slower	101.3

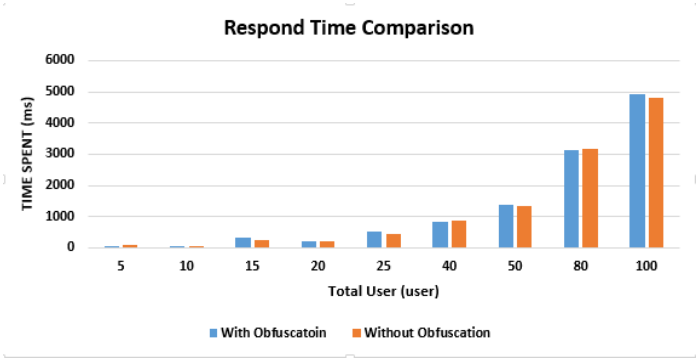

Fig. 7 Comparison Result of Web Application with and without Obfuscation

This longer execution time because of the existence of the obfuscation mechanism first in the source code that has been scrambled before finally processed again by web server.

IV. CONCLUSION

Based on the scenarios for testing in observing the performance of web applications using obfuscation and web applications without using obfuscation on Figure 7, it can be concluded that web applications using obfuscation provide longer response time than web applications without using obfuscation. This is due to the existence of deobfuscation mechanism first in the source code that has been scrambled before finally processed again by web server.

REFERENCES

[1] Kurniadi, Irwansyah F. Penerapan Algoritma Rc4 Untuk Enkripsi Keamanan Data (Studi Kasus : Dinas Pendidikan Dan Kebudayaan Kota Sekayu). J Ilm R B. 2015;8(12):1–13.

[2] Rahmatulloh A, Munir R. Pencegahan Ancaman Reverse Engineering Source Code Php Dengan Teknik Obfuscation Code Pada Extension Php. In: Konferensi Nasional Informatika. 2015.

[3] Ristić N, Jevremović A. An Open Source Solution For Protecting Php Source Code. In: Proceedings Of The 1st International Scientific Conference - Sinteza 2014 [Internet]. 2014. P. 616–9. Available From: Http://Portal.Sinteza.Singidunum.Ac.Rs/Paper/221

[4] Cimato S, Santis A De, Petrillo Uf. Overcoming The Obfuscation Of Java Programs By Identifier Renaming. 2005;78:60–72.

[5] Han S, Ryu M, Cha J, Choi Bu. Hotdol: Html Obfuscation With Text Distribution To Overlapping Layers. In: Proceedings - 2014 Ieee International Conference On Computer And Information Technology, Cit 2014. 2014. P. 399–404.

[6] Nugroho Ps, Ariwibowo E. Pengembangan Modul Enkripsi Dan Dekripsi Pada Php Dengan Modifikasi Metode Kriptografi Vigenere Cipher Dan Cipher Block Chaining (Studi Kasus Pada Geekybyte.Com). In: Jurnal Sarjana Teknik Informatika Universitas Ahmad Dahlan. 2014. P. 1004–12.

[7] Pradana K. Aplikasi Php Encoder Dan Decoder Menggunakan Algoritma Base64 Dan Kunci Keamanan. 2015;

[8] Setiawan O, Fiati R, Listyorini T. Algoritma Enkripsi Rc4 Sebagai Metode Obfuscation Source Code Php. In: Seminar Nasional Teknologi Industri Dan Informatika. 2014. P. 113–20.

[9] Suehring S, Valade J. Php Mysql Javascript Html5 All In One For Dummies. John Wiley & Sons, Inc. 2013. 724 P.

[10] Setiawan O, Fiati R, Listyorini T. Algoritma Enkripsi Rc4 Sebagai Metode Obfuscation Source Code Php. In: Seminar Nasional Teknologi Industri Dan Informatika. 2014. P. 113–20.

[11] Miles E, Sahai A, Weiss M. Protecting Obfuscation Against Arithmetic Attacks. Eprint [Internet]. 2014;878:221–38. Available From: Http://Dx.Doi.Org/10.1007/978-3-642-55220-5_13