International Journal of Computers, Communications & Control
Vol. II (2007), No. 2, pp. 159-173

An Enhancement of the Random Sequence 3-Level Obfuscated Algorithm for Protecting Agents Against Malicious Hosts

Kamalrulnizam Abu Bakar, Bernard S. Doherty

Abstract: With the advent of agent technology, the need for security of applications becomes a concern that is beyond classical data security capability and considerations. Without proper security protection especially against a malicious host attack, the widespread use of agent technology can be severely impeded. The Random Sequence 3-level obfuscated algorithm has been proposed by the authors to improve agent security: in this paper, an enhancement to the protection level of this algorithm is proposed. The effectiveness of the obfuscation algorithm is enhanced by addition of noise, which surrounds the true value carried by the agent with false values. A malicious host can thus at best guess the true value carried by the agent.

Keywords: Agent security, Malicious host, Spying attack, Noise code.

1 Introduction

Security is considered as an important factor for an agent used in real world applications. The problems in agent security arise when an agent is used in open and unsecured environments [8, 3]. One kind of attack by an execution host (malicious host) is spying on the agent's code, data and state [6, 11]. Spying attack by the malicious host may invade the agent's privacy, especially an agent's critical data, for example a user's maximum budget carried by the agent. Knowledge of an agent's critical data gives a malicious host an advantage in any competition over other hosts because the malicious host knows what is expected by the agent. For example, a customised agent is sent out (in an open and unsecured environment) to find a suitable flight with the fare price under or equal to 500 pounds. Malicious host attack based on a spying attack is to raise the offered price until it meets the maximum price that has been set by the agent's owner, even though the normal price is much lower. Spying attack by the malicious host on an agent's data or state is difficult to detect, because the attack doesn't leave any detectable trace [6, 11, 5]. The executing host has to read the agent's code, must have access to the agent's data, and must be able to manipulate the agent's variable data in order to execute the agent [6, 5, 8]. Therefore, the executing host can see and access all of the agent's code including data and state, and is thus able to hide any traces of an attack, which makes any attempt to address spying attack difficult.

To reduce the likelihood of this attack succeeding, the authors have proposed the Random Sequence 3-level (RS3) obfuscated algorithm [1], which obfuscates the actual value of an agent's critical data to prevent a malicious host spying attack. The RS3 obfuscated algorithm consists of multiple polynomial functions which obfuscating the actual value of the agent's critical data to produce an obfuscated value that is meaningless to the attacker. Only selected polynomial functions are used in the conversion process and in each selected function, multiple random inputs are implied. The main objective of using the RS3 obfuscated algorithm is to enable the comparing of confidential values within an unsecured remote host environment without exposing the actual confidential value to an unauthorised party, using comparison of obfuscated values rather than actual values. Unfortunately, implementing the RS3 obfuscated algorithm alone may expose the algorithm to an attack, which could execute multiple copies of the algorithm many times in order to analyse it, and propose actual values that will obfuscate to give the comparison outcome sought by the malicious host.

This paper proposes an enhancement of the protection level of the RS3 obfuscated algorithm by adding noise code in the agent application that executes in the remote host environment, to hide the actual obfuscated value among a number of false values. The enhancement makes it more difficult for the attacker to analyse the actual obfuscated value in order to discover the actual value of an agent's critical data: the attacker can deobfuscate the set of values carried by the agent, but can at best guess which is the correct value.

This paper is organized as follows: Section 2 present an overview of the Random Sequence 3-level obfuscated algorithm. Section 3 describes the random sequence 3-level obfuscated algorithm with noise code, the implementation of noise code to enhance the obfuscated algorithm protection level and the deobfuscation time of the RS3 obfuscated algorithm. Section 4 presents the experimental results on the overhead of implementing noise code with the RS3 obfuscated algorithm. Section 5 analyse the strength of the RS3 obfuscated algorithm with noise code. Section 6 presents a discussion and the conclusion is presented in section 7.

2 An Overview of The Random Sequence 3-level Obfuscated Algorithm

The Random Sequence 3-level obfuscated algorithm [1] is an algorithm that is designed to protect the confidentiality of an agent against malicious host spying attack. This algorithm uses three polynomial functions for obfuscating the actual value of the agent's critical data to an obfuscated value that is meaningless to the malicious host, in order to prevent the malicious host from spying on the agent critical data (Figure 1 show the obfuscation process of the RS3 obfuscation algorithm). The obfuscated method used in the Random Sequence 3-level obfuscated algorithm enables the execution host to execute the process of comparing its offer with the agent budget with both values in an obfuscated format without the execution host having any knowledge of the actual value of the agent's critical data. This comparing process can be done without needing deobfuscation of the data, unlike cryptographic methods, which require decryption of the data, thus revealing its value, before a comparison can be made.

Although the RS3 obfuscated algorithm is able to obfuscate an agent's critical data to make it more difficult for the malicious host to spy, the malicious host can execute multiple copies of the obfuscated algorithm in parallel in order to analyse the algorithm quickly, making this obfuscated algorithm vulnerable. This attack can be addressed by limiting the processing time available to the host before the agent is discarded [6]. However, the problem in determining an effective protection interval that can prevent the malicious host having enough time to execute multiple copies of the obfuscated algorithm also makes it difficult for this obfuscated algorithm to be implemented in real applications, where sufficient time must be allowed for legitimate processing. In order to overcome the problem of a malicious host executing multiple copies of the RS3 obfuscated algorithm and of determining an effective protection interval to protect the algorithm, noise code [9, 10] is introduced for an agent that executes in the remote host environment. The RS3 obfuscated algorithm with noise code is described in the next section.

3 The Random Sequence 3-level Obfuscated Algorithm with Noise Code

The objective for implementing the noise code in the agent application is to hide the actual obfuscated value among a numbers of fake obfuscated values so the malicious host can at best guess at the true value of the agent's critical data [9, 10]. The difference between ordinary RS3 algorithm and RS3 with noise code algorithm is in the number of obfuscated values generated and added by the master agent into the slave agent application before the slave agent is dispatched to the remote host execution environment to execute its tasks.

In order to discover the true value of the agent's critical data, the malicious host must first guess the actual obfuscated value among a number of fake obfuscated values. Any wrong decision in choosing the obfuscated value will result in using a wrong true value of the agent's critical data. For instance, if the agent is equipped only with the actual obfuscated value, X without adding any noise code, the probability that the malicious host could discover the actual obfuscated value by searching a range of values is one,

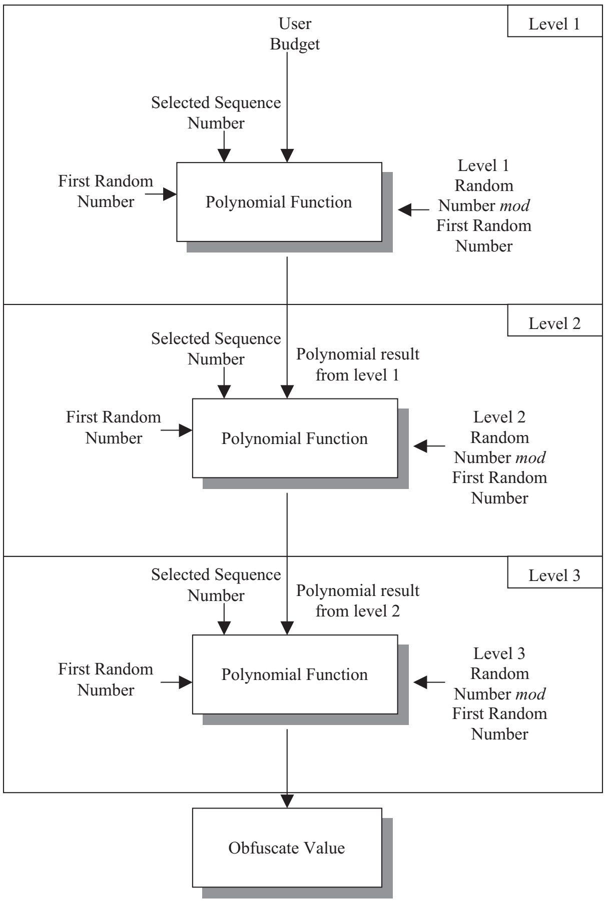


Figure 1: The RS3 Obfuscation Algorithm Obfuscation Process

i.e. $P(X) = 1$. However, if noise codes E_i (fake obfuscated values) are added to the agent, where $i = 1,2,\dots,100-1$, the probability of discovering the actual obfuscated value is $\frac{1}{100}$. The probability of guessing the actual obfuscated value becomes smaller as more noise codes are added. Figure 2 illustrates the effect of introducing noise codes into the agent application. In addition, the time needed to guess the actual obfuscated value will delay the malicious host in analysing the obfuscated algorithm. Therefore, the use of an effective protection interval in enhancing the obfuscated algorithm protection will be less important.

3.1 Implementing RS3 Obfuscated Algorithm with Noise Code

The operation of the RS3 obfuscated algorithm with noise code is almost the same as the operation of the RS3 obfuscated algorithm without noise code (refer to [1]). The only difference between these two obfuscated algorithms is in the number of obfuscated values generated and added by the master agent into the slave agent application before the slave agent is dispatched to the remote host execution environment to execute its tasks.

In the operation of RS3 obfuscated algorithm without noise code, the master agent only has to obfuscate the value of user's budget and add the obfuscated value into the slave agent application before dispatching the slave agent to execute its tasks in the remote host execution environment. However, in the operation of RS3 obfuscated algorithm with noise code, the master agent has to generate more than one obfuscated value (the extra obfuscated values serve as noise codes) and add these obfuscated values

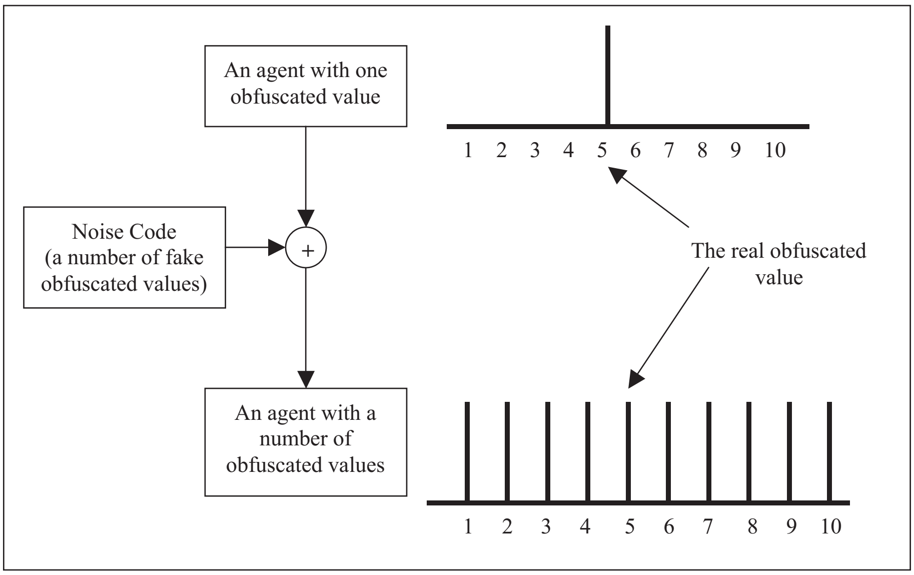
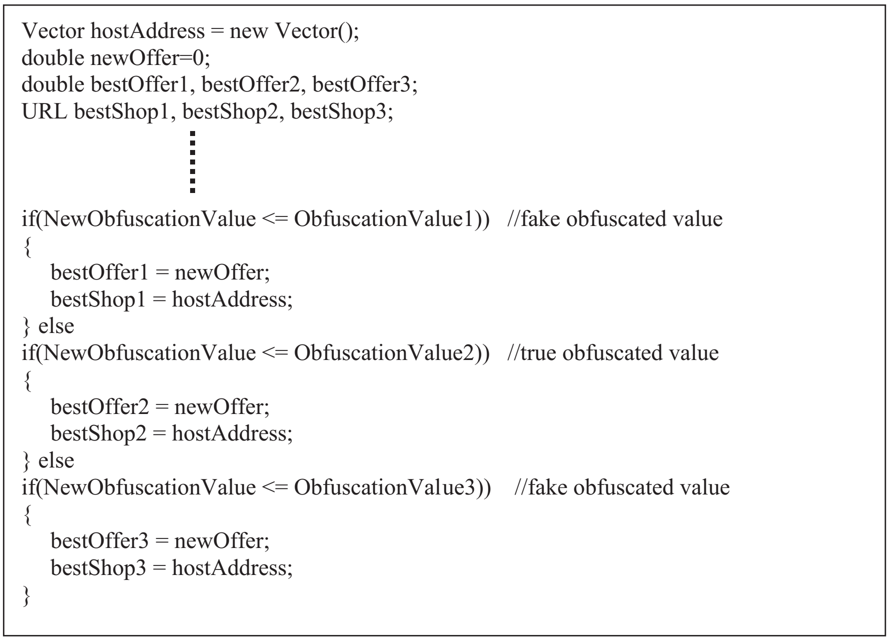

An agent with one obfuscated value
Noise Code (a number of fake obfuscated values)
+
An agent with a number of obfuscated values
1 2 3 4 5 6 7 8 9 10
The real obfuscated value
1 2 3 4 5 6 7 8 9 10

Figure 2: The Effect of Adding Noise Codes Into The Agent Application

into the slave agent application before dispatching the slave agent to execute in the remote host execution environment (see figure 3).



```
Vector hostAddress = new Vector();
double newOffer=0;
double bestOffer1, bestOffer2, bestOffer3;
URL bestShop1, bestShop2, bestShop3;
                        ⋮

if(NewObfuscationValue <= ObfuscationValue1))   //fake obfuscated value
{
    bestOffer1 = newOffer;
    bestShop1 = hostAddress;
} else
if(NewObfuscationValue <= ObfuscationValue2))   //true obfuscated value
{
    bestOffer2 = newOffer;
    bestShop2 = hostAddress;
} else
if(NewObfuscationValue <= ObfuscationValue3))    //fake obfuscated value
{
    bestOffer3 = newOffer;
    bestShop3 = hostAddress;
}
```

Figure 3: A Slave Agent Program added with Noise Code (data block)

To illustrate, the noise code is a fake obfuscated value that is generated by the master agent from a fake user budget value. This fake user budget value is created by the master agent by adding or subtracting a random number to or from the actual user budget value. For example, say the actual user budget value is č500. To create a fake user budget value, the master agent needs to generate a random number, e.g. 176. If the master agent chooses to add the random number to the actual user budget value, the fake user budget value becomes 676. This value is then converted to the obfuscated value to represent the fake obfuscated value. On the other hand, if the master agent chooses to subtract the random number with the actual user budget value, the fake user budget value becomes 324. The same conversion process will be applied to convert the fake user budget value to a fake obfuscated value. To generate more fake obfuscated values, the master agent has to generate more random numbers.

Once the obfuscation process in the home host is completed, the master agent dispatches the slave agent together with all the obfuscated values generated (including fake obfuscated values) to the remote host to execute its given tasks. In the remote host execution environment, the slave agent starts its execution process by converting any offer that was gathered from the remote host into an obfuscated value to be used in the comparing process. For example, if the slave agent has 4 obfuscated values (one actual and three fake values), the slave agent has to execute 4 comparing processes. If any of the obfuscated user budget value matches the obfuscated offer value, the corresponding obfuscated user budget value will be excluded from the obfuscation stage (see figure 4). The next obfuscation stage starts by obfuscating a new remote host offer. In the authors' work, a maximum of three obfuscation stages are used to searches for a flight offer (the obfuscated user budget value is compared with the first class obfuscated fare value in the obfuscation stage 1, business class obfuscated fare value in the obfuscation stage 2 and economy class obfuscated fare value in the obfuscation stage 3) or less than three times if all the obfuscated user budget values have been excluded.

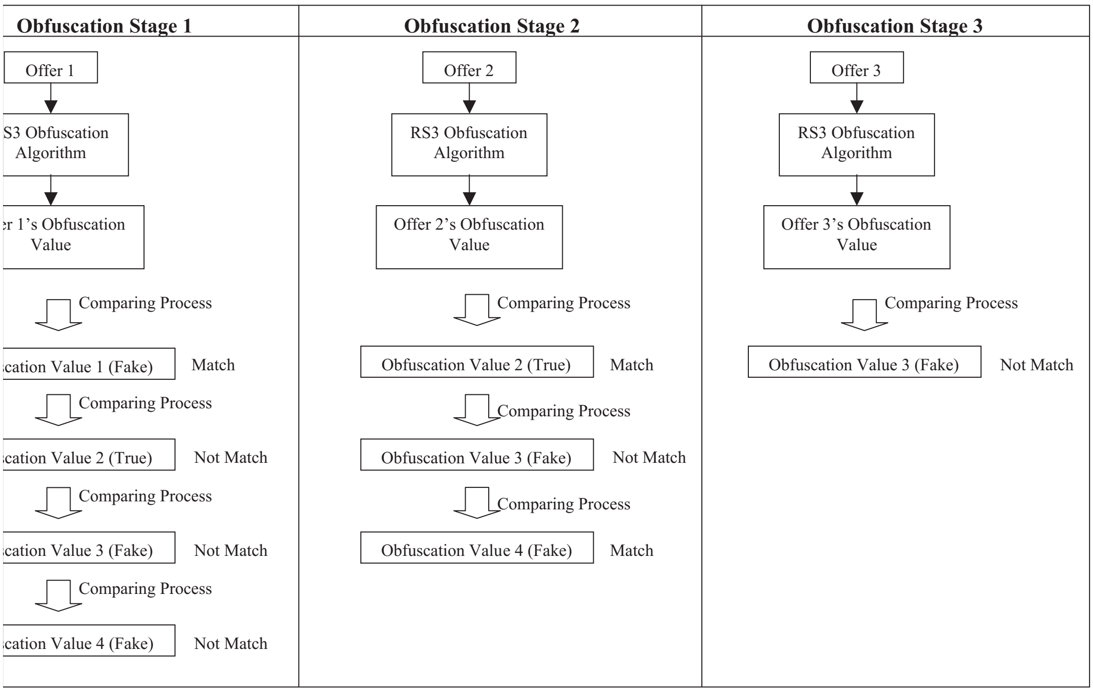


Figure 4: The illustration of the Obfuscation Stages and the Comparing Process

After completing the execution process in the remote host, the slave agent returns to its home host together with the remote host offer. The home host then extracts only the offer that fulfils the requirement of the true obfuscated value for further actions.

3.2 The Deobfuscation Time of the RS3 Obfuscated Algorithm with Noise Code

The experiment on the deobfuscation time of the RS3 obfuscated algorithm with noise code is conducted to examine the time taken by the execution host (assumed to be the malicious host) to deobfuscate a full set of obfuscated value. The experiment is conducted using one 700 MHz personal computer with 128 MB of main memory, which running Windows 98 operating system.

In this experiment, the deobfuscation time is taken starting from the start of the deobfuscation process for the first obfuscated value and ending when the last obfuscated value from a set of obfuscated value have been deobfuscated. The experiment is performed by the execution host by executing the RS3 obfuscated algorithm using different input value many times until the matching obfuscated value is produced and continue until a full set of the obfuscated value are deobfuscated (see Figure 5). Twenty sets of obfuscated values are examine with the number of values in each set ranging from five to one thousand obfuscated values. Each set of obfuscated values contained one correct value and the remainder false values.

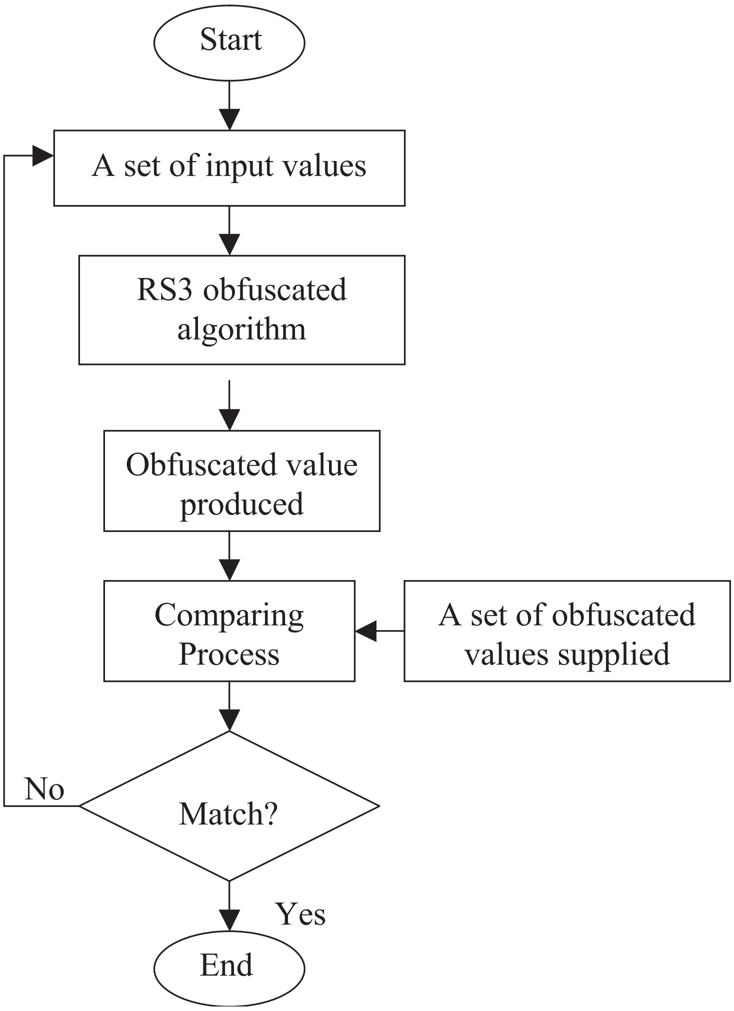


Figure 5: The Deobfuscation Process

The deobfucation time results (which were gathered in milliseconds and then converted into seconds) of the RS3 obfuscated algorithm with noise code are shown in Table 1 and 2, and illustrated in Figure 6 and 7 respectively.

Obfuscated Values	5	10	15	20	25	30	35	40	45	50
Deobfuscation Time	8.95	21.57	29.42	37.69	49.96	60.71	62.53	72.46	81.66	85.58

Table 1: Summary Statistics of the Random Sequence 3-level Obfuscated Algorithm Deobfuscation Time for Small Number of Obfuscated Values

Obfuscated Values	100	200	300	400	500	600	700	800
Deobfuscation Time	180.15	376.33	581.28	779.12	982.06	1142.27	1376.64	1568.75

Obfuscated Values	900	1000
Deobfuscation Time	1765.94	1887.83

Table 2: Summary Statistics of the Random Sequence 3-level Obfuscated Algorithm Deobfuscation Time for Large Number of Obfuscated Values

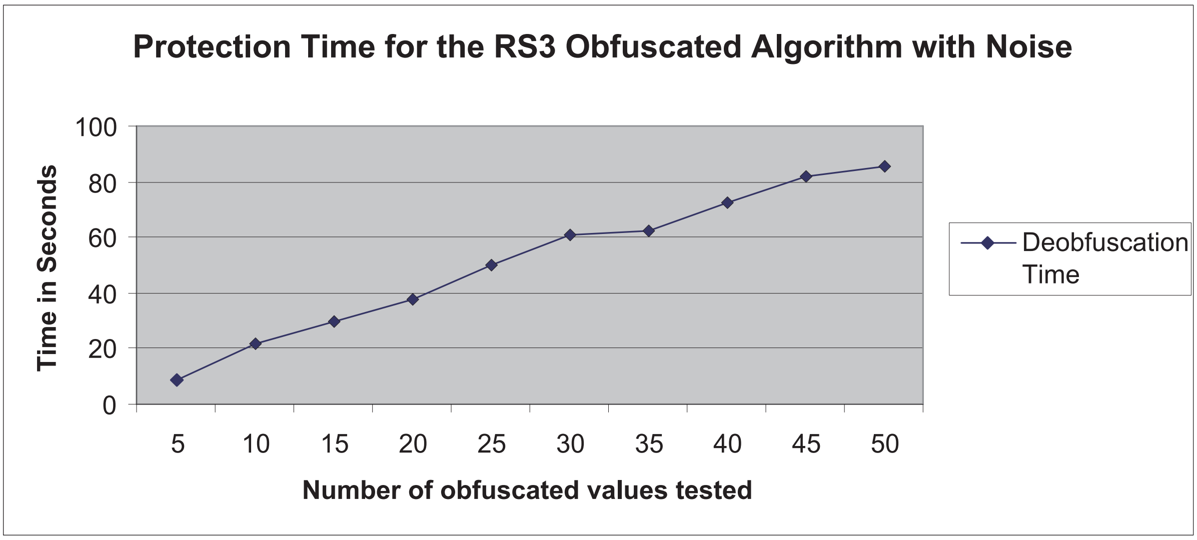


Figure 6: The Deobfuscation Time of the Random Sequence 3-level Obfuscated Algorithm for Small Number of Obfuscated Values

From the results given in Table 1 and 2, and illustrated in Figure 6 and 7 respectively, it can be seen that the deobfuscation time of the RS3 obfuscated algorithm with noise code increases linearly with the number of obfuscated values in the test set. Note that even after deobfuscation, the malicious host is left with a set of values of which only one is the correct value, and has no way of knowing which one is the correct value.

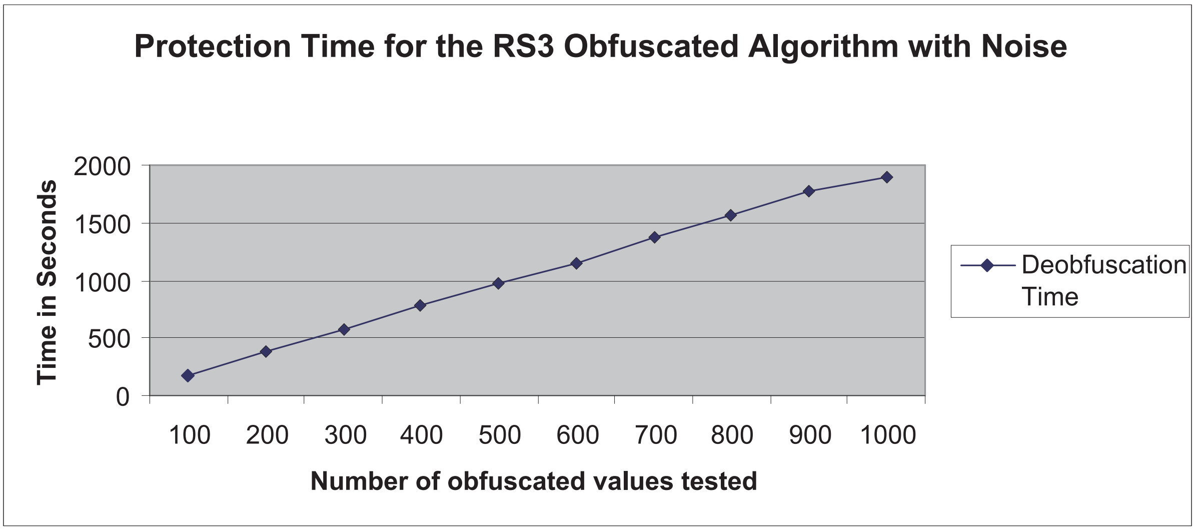


Figure 7: The Deobfuscation Time of the Random Sequence 3-level Obfuscated Algorithm for Large Number of Obfuscated Values

4 The Overhead of Implementing Noise Code with the RS3 Obfuscated Algorithm

The experiments to measure the overhead of implementing the RS3 obfuscated algorithm with noise code were conducted using six 400 MHz Sun Ultra Sparc 5 workstations with 128 MB of main memory. Each of the workstations is running the Solaris 8 operating system and is connected to the others using 100 Mbit/s UTP[1] cable. All of the workstations involved in this experiment were situated in the same room.

In this configuration, one workstation was chosen among the six workstations to be the home host for the agent, and only this host had permission to manage and dispatch the agent. The rest of the workstations were assumed to be remote hosts, having only the capability to receive the agent and to dispatch the agent back to its home host.

To examine the security overhead in implementing noise code with the Random Sequence 3-level obfuscated algorithm in an agent-based application, times are measured starting from sending of the agents to the remote hosts and ending with the home host receiving the agents back from the remote hosts. Four different experiments are conducted starting with one remote host, two remote hosts, three remote hosts and five remote hosts using three different kinds of agent: a plain agent[2], an agent with the Random Sequence 3-level obfuscated algorithm (RS3) and an agent with the RS3 obfuscated algorithm with noise code (RS3N). The times were measured using the “System.currentTimeMillis()” method in the Java language. This method produces a specific instant in time with millisecond precision [7].

The experiment is performed for 20 runs[3] and the time for each run gathered in milliseconds. The average result of the 20 runs is taken and converted into seconds. The result is then rounded and presented in two decimal places as shown and illustrated in Tables 4 to 7 and Figure 8 to 11 respectively.

The results of the security overhead of the Random Sequence 3-level obfuscated algorithm without noise (RS3) and the Random Sequence 3-level obfuscated algorithm with noise are compared to the plain agent as shown in Table 3. From these results, it can be seen that there is not much difference in

[1] Unshielded Twisted Pair Category 5e

[2] agents without security mechanisms

[3] The results were observed to be consistent after twenty runs

Number of Remote Hosts	Mean		
	Plain	RS3	RS3N
1	1.49	1.5	1.69
2	2.45	2.52	2.67
3	3.36	3.56	3.63
5	5.42	5.43	5.66

Table 3: Summary Statistics of The Random Sequence 3-Level Obfuscated Algorithm (1 Cycle, 1 Obfuscation Value Experiment(without noise code) and 1000 Obfuscation Value Experiment(with noise code))

the security overhead for the RS3 without noise and RS3 with noise for all numbers of remote hosts. The highest difference is for one remote host, where the security overhead for RS3 with noise is just 12.67 % higher than the security overhead of RS3 without noise. Therefore, it is concluded that RS3 with noise will add a small and acceptable overhead. More detail of the results now follows:

Firstly, consider plain and RS3, then plain and RS3N experimental results.

Number of Remote Hosts	Mean		Standard Error		Standard Deviation	
	Plain	RS3	Plain	RS3	Plain	RS3
1	1.49	1.5	0.001	0.001	0.003	0.004
2	2.45	2.52	0.003	0.005	0.011	0.022
3	3.36	3.56	0.003	0.004	0.011	0.02
5	5.42	5.43	0.007	0.006	0.031	0.025

Table 4: Summary Statistics of The Random Sequence 3-Level Obfuscated Algorithm Overhead (1 Cycle and 1 Obfuscation Value Experiment(without noise code))

Number of Remote Hosts	Mean		Standard Error		Standard Deviation	
	Plain	RS3N	Plain	RS3N	Plain	RS3N
1	1.49	1.54	0.001	0.001	0.003	0.003
2	2.45	2.52	0.002	0.007	0.011	0.031
3	3.36	3.51	0.003	0.005	0.011	0.022
5	5.42	5.44	0.007	0.007	0.031	0.031

Table 5: Summary Statistics of The Random Sequence 3-Level Obfuscated Algorithm (1 Cycle and 100 Obfuscation Value Experiment(with noise code))

From the results given in Tables 4 and 5 and illustrated in Figure 8 and 9 respectively, it can be seen that the mean of the security overhead for a plain agent is almost the same as the security overhead for RS3 and RS3N, where when comparing with RS3, the highest difference is just 5.95 % and 4.46 % when comparing with RS3N. However, from Table 6 and illustrated in Figure 10, it can be seen that as the number of noise codes is increased, the difference becomes larger, where the highest difference is 13.42 %, which is still considered acceptable.

From Table 7 and Figure 11, the security overhead for both the agents is almost the same as the

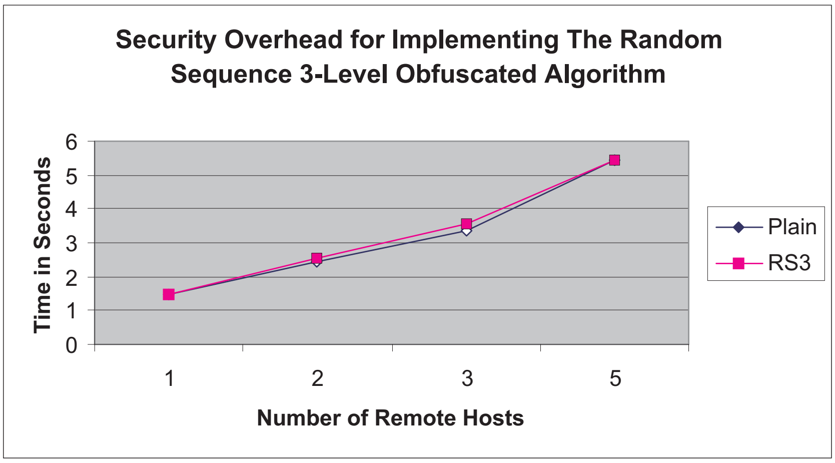


Figure 8: Security Overhead of The Random Sequence 3-Level Obfuscated Algorithm (1 Cycle and 1 Obfuscation Value Experiment(without noise code))

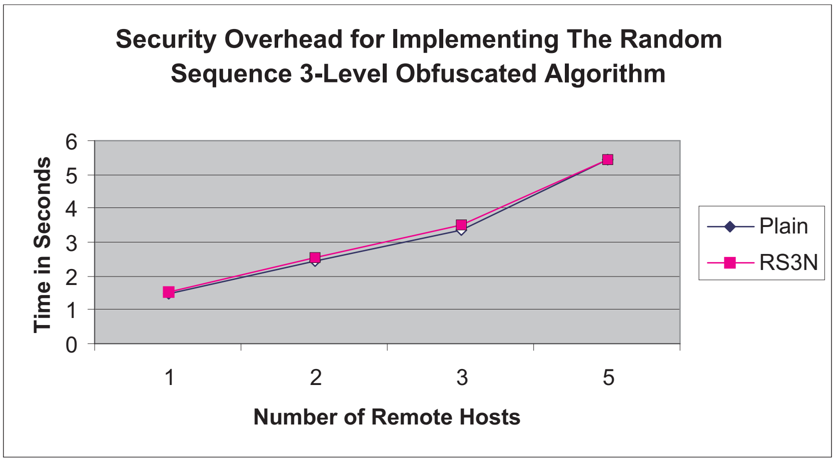


Figure 9: Security Overhead of The Random Sequence 3-Level Obfuscated Algorithm (1 Cycle and 100 Obfuscation Value Experiment(with noise code))

security overhead given in Tables 4 to 6, even though now the number of cycles[4] has been increased to 100.

These results show that implementation of the Random Sequence 3-level obfuscated algorithm with noise code does increase the overhead by up to 13.42 % compared to the plain agent, but the noise code adds little to the overhead.

5 The analysis of RS3 obfuscated algorithm strength with noise code implementation

To analyse the strength of the RS3 obfuscated algorithm after the implementation of noise code, the authors have listed vulnerabilities and the way to address them as follows:

[4] a loopings that simulate an agent tasks

Number of	Mean		Standard Error		Standard Deviation	
Remote Hosts	Plain	RS3N	Plain	RS3N	Plain	RS3N
1	1.49	1.69	0.001	0.004	0.003	0.016
2	2.45	2.67	0.003	0.006	0.011	0.028
3	3.36	3.63	0.003	0.003	0.011	0.015
5	5.42	5.66	0.007	0.006	0.031	0.028

Table 6: Summary Statistics of The Random Sequence 3-Level Obfuscated Algorithm (1 Cycle and 1000 Obfuscation Value Experiment(with noise code))

Number of	Mean		Standard Error		Standard Deviation	
Remote Hosts	Plain	RS3N	Plain	RS3N	Plain	RS3N
1	1.48	1.65	0.0003	0.002	0.001	0.008
2	2.46	2.65	0.002	0.003	0.01	0.014
3	3.45	3.67	0.007	0.005	0.03	0.021
5	5.46	5.67	0.013	0.004	0.06	0.02

Table 7: Summary Statistics of The Random Sequence 3-Level Obfuscated Algorithm (100 Cycle and 1000 Obfuscation Value Experiment(with noise code))

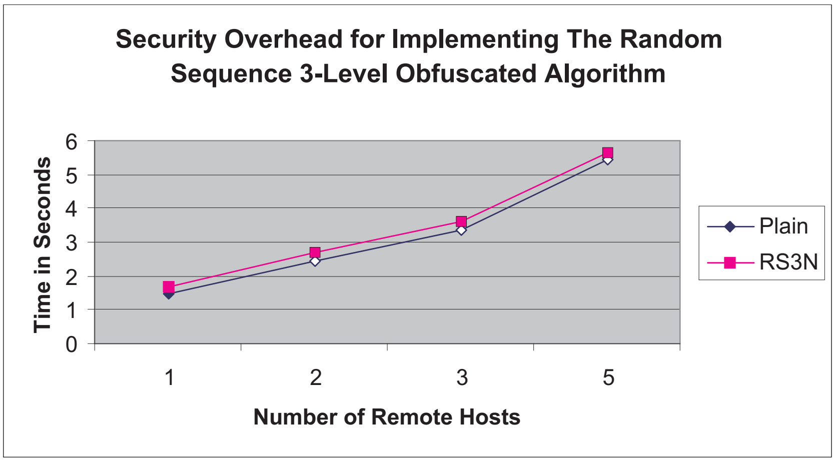


Figure 10: Security Overhead of The Random Sequence 3-Level Obfuscated Algorithm (1 Cycle and 1000 Obfuscation Value Experiment(with noise code))

5.1 The vulnerabilities of RS3 obfuscated algorithm with noise code

There are two main weaknesses of the RS3 obfuscated algorithm with noise code that have been identified.

- If the attacker (malicious host) takes a guess at the correct obfuscated value from many obfuscated values (including noise code) carried by the visiting agent and the attacker is given enough time to execute, the attacker can execute the RS3 obfuscated algorithm many times using different result values and watch the pattern of the RS3 obfuscated algorithm outcomes (which result value the

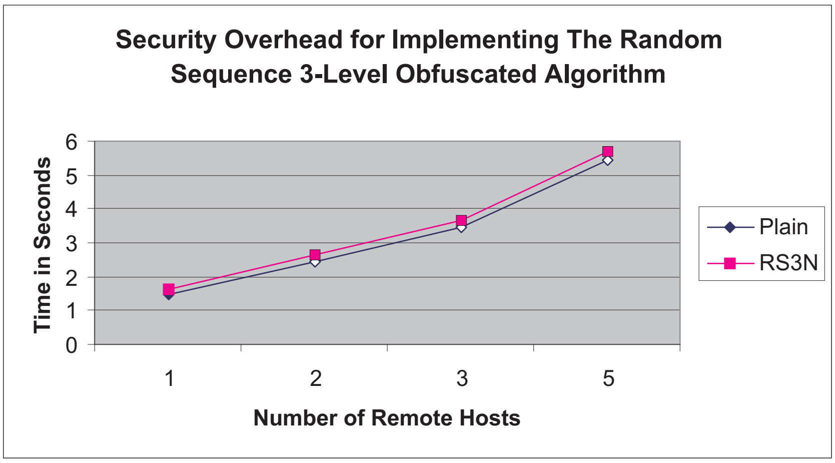


Figure 11: Security Overhead of The Random Sequence 3-Level Obfuscated Algorithm (100 Cycle and 1000 Obfuscation Value Experiment(with noise code))

agent accepts and which it rejects) to analyse the chosen obfuscated value in order to discover the actual value of agent's critical data.

- If the noise code generated from the actual value are very much out of range from the reasonable value, the malicious host could easily omit that values and take a guess from fewer values.

5.2 Addressing vulnerabilities of RS3 obfuscated algorithm with noise code

In order to overcome the weaknesses of the RS3 obfuscated algorithm with noise code, two main points are suggested:

- To prevent the attacker (malicious host) from being able to guess a correct obfuscated value among many obfuscated values in a short time, and running many tests on RS3 obfuscated algorithm by analysing which result value the agent accepts and which it rejects, the number of noise codes added in the agent application, could be made bigger. This is due to the fact that the probability to guess the correct obfuscated value becomes smaller as the number of noise code added become bigger, i.e. $P(X) \rightarrow 0$ as $N \rightarrow \infty$. The malicious host can at best guess the actual value from among the noise code.

- To prevent the malicious host able to omit any of the noise code values, the value of the noise codes must be within a reasonable range of the actual value and this can be done by limiting the range of the random numbers.

6 Discussion

In this paper, the problem of a malicious host spying on the actual value of an agent's critical data, such as the user maximum budget, has been discussed. The Random Sequence 3-level obfuscated algorithm, which is able to obfuscate the actual value of an agent's critical data in order to make it more difficult for the malicious host to spy on the actual value of the critical data, has been previously proposed by the authors. To address weaknesses in the RS3 obfuscated algorithm, an enhancement has been proposed in this paper.

The implementation of the RS3 obfuscated algorithm alone exposes the obfuscated algorithm to the attacker that could execute multiple copies of the obfuscated algorithm many times to analyse the algorithm. The agent owner could also face the problem of determining an effective protection interval for the obfuscated value that is carried by the agent in order to prevent the value from being analysed and discovered by the malicious host. These problems can be overcome by introducing noise codes carried by the agent application to force a malicious host to guess the actual obfuscated value, thus leaving a malicious host with at best a guess at the actual value of the user maximum budget. The noise code implementation also delays the malicious host in analysing the obfuscated algorithm, which has been shown in Section 3.2. Therefore, the use of an effective protection interval to enhance the level of obfuscated algorithm protection is less important.

On the other hand, based on the experimental results on the overhead of implementing the RS3 obfuscated algorithm with noise code, the implementation of the RS3 obfuscated algorithm does increase the overhead by up to 13.42 % compared to the plain agent, but this is considered acceptable. This suggests the Random Sequence 3-level obfuscated algorithm can be implemented in "real world" agent applications to protect the agent application from the spying attack by the malicious host. Experiment shown that adding noise codes to the RS3 obfuscated algorithm gives little (12.67 %) increase in security overhead.

7 Conclusion

The Random Sequence 3-level obfuscated algorithm is an algorithm that improves the level of protection of an agent against malicious host spying attack. This obfuscated algorithm does not protect against all spying attacks by the malicious host, only an attack to the agent's critical data. However, the implementation of noise code in the agent application prevents the malicious host discovering the actual value of critical data carried by the agent; the malicious host can at best guess the actual value from among a number of noise values. The addition of noise code has strengthened the protection of the obfuscated algorithm and has reduced the likelihood of successful attack on the RS3 obfuscated algorithm, with very small increase in execution time.

References

[1] Abu Bakar, K. and Doherty, B. S.: A Random Sequence 3-level Obfuscated Algorithm for Protecting Mobile Agents Against Malicious Hosts. Proceedings of the 2003 International Conference on Informatics, Cybernetics and Systems. I-Shou University(2003) 525 – 530

[2] DiVincenzo, D. P., Leung, D. W. and Terhal, B. M.: Quantum Data Hiding. IEEE Transactions on Information Theory, Vol. 48, No. 3. IEEE(2002) 580–598

[3] Farmer, W.M. and Guttman, J.D. and Swarup, V.: Security for Mobile Agents: Issues and Requirements. Proceedings of the 19th National Information System Security Conference. Baltimore (1996) 591–597

[4] Harmsen, J. J. and Pearlman, W. A.: Steganalysis of Additive Noise Modelable Information Hiding. Proceedings of SPIE Electronic Imaging 5022. SPIE (2003) 21–24

[5] Hohl, F.: A Framework to Protect Mobile Agents by Using Reference States. In: Proceedings of the 20th international conference on distributed computing systems (ICDCS 2000). IEEE Computer Society (2000) 410-417

[6] Hohl, F.: Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts. In: G. Vigna (Ed.). Mobile Agent and Security. Lecture Notes in Computer Science, Vol. 1419. Springer-Verlag, Berlin(1998) 92–113

[7] Sun Microsystems, Inc. Java 2 Platform Std. Ed. V1.3.1 http://java.sun.com/j2se/ 1.3/docs/api/index.html (2004)

[8] Mandry, T., Pernul, G. and Rohm, A.: Mobile Agents in Electronic Markets: Opportunities, Risks, Agent Protection. International Journal of Electronic Commerce. M.E. Sharpe (2001) 47–60

[9] Ng, S. K. and Cheung, K. W.: Protecting Mobile Agents against Malicious Hosts by Intention Spreading. In H. Arabnia (ed.), Proc. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99). CSREA (1999) 725–729

[10] Ng, S. K. and Cheung, K. W.: Intention Spreading: An extensible theme to protect mobile agents from read attack hoisted by malicious hosts. In Jimming Liu, Ning Zhong(ed.), Intelligent Agent Technology: Systems, Methodologies and Tools. World Scientific (1999) 406–415

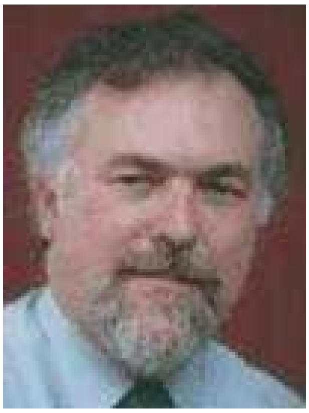
[11] Sander, T. and Tschudin, C.: Protecting Mobile Agent Against Malicious Hosts. In: G. Vigna (Ed.). Mobile Agent and Security. Lecture Notes in Computer Science, Vol. 1419. Springer-Verlag, Berlin(1998) 44–60

[12] Yeh, W. H. and Hwang, J. J.: Hiding Digital Information Using a Novel System Scheme. Computers and Security, Vol. 20, No. 6. Elsevier Science (2001) 533–538

Kamalrulnizam Abu Bakar
Faculty of Computer Science and Information System
Universiti Teknologi Malaysia
81310 UTM Skudai
Johor D. T.
Malaysia
E-mail: kamarul@fsksm.utm.my

Bernard S. Doherty
School of Engineering and Applied Science
Aston University
Aston Triangle, Birmingham B4 7ET
United Kingdom
E-mail: b.s.doherty@aston.ac.uk
Received: July 22, 2006

Kamalrulnizam Abu Bakar is a lecturer at Universiti Teknologi Malaysia, Malaysia. He received the diploma and degree of Computer Science in 1994 and 1996 respectively from Universiti Teknologi Malaysia, Malaysia. He then received Masters in Computer Communication and Networks degree from Leeds Metropolitan University, United Kingdom in 1998 and PhD in Network Security from Aston University, United Kingdom in 2004. His current research interests include computer and network security, distributed systems and parallel processing, grid computing, wireless and cellular network.



Bernard S. Doherty (born October 2nd, 1945) obtained the degrees of Bachelor of Engineering (Electrical), Bachelor of Arts and Master of Engineering Science from the University of Melbourne in 1967, 1971 and 1981 respectively. He has held positions with the State Electricicity Commission of Victoria, LM Ericsson Pty Ltd, Swinburne College of Technology (all in Melbourne) and, since 1980, at Aston University (Birmingham, UK), where is presently Lecturer in Computer Science. His main fields of teaching and research are Distributed and Networked applications and Information Security. In addition to supervising a number of Doctoral students, he has developed computer-based administration and teaching software, written a number of papers and presented papers at international conferences.