Jun KAWANO[1], Hiroshi KAI[1], Yoshinobu HIGAMI[1], Shinya KOBAYASHI[1]

Graduate School of Science and Engineering, Ehime University (1)

Dummy code insertion and its efforts on concealment for secure processing

Abstract. *Technology of concealing purpose of program is needed for profitable uses of an external grid. We propose dummy code insertion technique for concealment technology. We have implemented one kind of technique of dummy code insertion on trail. Moreover we evaluate strength of concealment against malicious inspection. We explain the detail of evaluation of dummy code insertion technique in this paper.*

Streszczenie. *Technologia zaciemniania przeznaczenia programu jest niezbędna podczas odpłatnego korzystania z zewnętrznych sieci gridowych. W artykule zaproponowano jeden z typów technik wstawiania atrap kodu. Dodatkowo oceniono odporność techniki zaciemniania przed złośliwą penetracją. Szczegółowo wyjaśniono także sposób oceny zastosowanej techniki wstawiania atrap kodu. (**Wstawianie atrap kodu oraz wpływ tego typu zaciemniania na bezpieczeństwo przetwarzania**)*

Keywords: external grid, concealment, secure processing, dummy code
Slowa Kluczowe: zewnętrzne sieci gridowe, zaciemnianie, bezpieczne przetwarzanie, atrapa kodu

Introduction

The program is distributed to computers connected in a network and executed in the computer in the grid computing. The more computers are in the grid, the higher throughput than the grid computing has.

There are two types of grid computing. One is the external grid, and another is the internal grid. The external grid is composed of computers on the Internet, while the internal grid is composed of computers in an intranet. The external grid has the higher throughput than the internal grid, because the external grid is composed of more computers than the internal grid. However, we cannot identify who is an owner of each computer in the external grid, so we cannot trust the owner. In the internal grid, we can identify who they are.

The owner of the computer in the external grid might get the intention of the distributed program by investing the program. This cheat is called as “analyze”. Furthermore, the owner of the computer in the external grid might reply a fake result to us. This cheat is called as “alter”.

We have already proposed a technology called secure processing technology, which is a set of methods against analyzing and altering. Dummy code insertion is one method against analyzing. However, it is not clear how effectively the dummy code insertion works for concealment of the program intention and how to insert dummy code. In this paper, we propose a new algorithm to insert dummy codes.

On the other side, we must stand on the defense side and offense side, inserting dummy codes and finding out the dummy code in the program. In this paper, we also proposed an algorithm to find out dummy codes in the program. Moreover, we evaluate how difficult to find out dummy codes from a program.

Data dependence [1]

Data dependence is required for parallel processing or distributed processing. Data dependence relations are used to determine which codes we execute at the same time.
For the instance, in the codes

S_1: A = B + C
S_2: D = A + 2

statements S_1 and S_2 cannot be executed at the same time since S_2 uses the value of A that is computed by S_1. This is called “true dependence” since the data value flows from S_1 to S_2.
In the program segment
S_1: A = B + C
S_2: B = D / 2

S_1 uses the value of B before S_2 assigns a new value to B. Since S_1 is to use the “old” value of B, it must be executed before S_2. So this is called “antidependence”.

The third kind of dependence is shown in the program segment below:

S_1: A = B + C
S_2: D = A + 2
S_3: A = E + F

Here S_3 assigns a new value to A after S_1 has already given a value to A. If S_1 is executed after S_3, then A will contain the wrong value after this program segment. Thus, S_1 must precede S_3. This is called “output dependence”.

In data dependence, we can delete antidependence and output dependence using a method called “renaming”.

Secure processing [2][3][4]

We have already proposed the secure processing technology against analyzing and altering. In this paper, we will explain about methods against analyzing, because we discuss on analyzing in this paper.

There are three types of methods against analyzing. The first is “program divide”, the second is “program reconstruction”, and the last is “dummy code insertion”.

At first, we will explain about program divide. The program is divided to some codes. Each code is distributed to computers in the external grid. So it is hard to analyze, because the owners get the less information about program than in the case of inspecting an original program and the data dependence between codes cannot be recognized.

However, if malicious owners conspire with other owners, they can collect some codes, and then they can get more information about program. Therefore program divide isn’t sufficient to conceal the information of program.

The program reconstruction makes it hard to analyze the program. We will explain about the program reconstruction. First, we collect some programs which we can execute. Second, we divide these programs to some codes. At last, we reconstruct program from these codes. These reconstructed programs are distributed to computers in the external grid. For the instance, we assume that there are following two programs.

Program A:
A_1: x = y + z
A_2: y = z + 3
A_3: z = x / y;
Program B:

B_1: a = b – c
B_2: b = a * c * 2

Program A is divided three codes, A_1, A_2 and A_3. Program B is divided two codes, B_1 and B_2. The next programs reconstruct form A_1, A_2, A_3, B_1 and B_2.

Program R_1: A_1, B_1
Program R_2: A_2, B_2, A_3

Program R_1 is reconstructed from A_1 and B_1. Program R_2 is reconstructed from A_2, B_2 and A_3. If malicious owners analyzed, it is hard for them to recognize the original program since there are codes of Program A and B in Program R_1 or R_2.

Now we will explain the third method "dummy code insertion". The dummy code doesn't exist in the original program, and the dummy code doesn't influence to the processes of the original program. If malicious owners analyzed, it is hard to recognize the original program since there are dummy codes in the program.

How to insert dummy codes?

We have already proposed the basic idea of dummy codes. But we don't have proposed the adaptable method of dummy code insertion yet. So, we will propose a method of dummy code insertion in the following. In this paper, we assume that the program is written in C language.

The careful points when we insert dummy codes

When we insert dummy codes, we should make the dummy code that resembles the original codes. For example, we assume that there are following the program.

S_1: A = B + 5
S_2: C = A * 2
S_3: D = C / 3

S_2 depends on S_1. S_3 depends on S2. The data dependence relation of these codes is true dependence. We assume to insert the code dam as dummy code.

S_1: A = B + 5
S_2: C = A * 2
S_3: D = C / 3
Dam: F = A + D – C

Dam depends on three codes, S1, S2, and S3. If malicious person analyzed this program, they might guess that Dam may be a dummy code since Dam depends on larger codes than the number of codes that original codes depend. Unfortunately, his guess is right.

How to divide a program?

Before we insert dummy codes, we divide a program.

If we can know how many iterations are executed by for-loop or while-loop a priori, we divide the codes in for-loop or while-loop. If we cannot, we think the codes in for-loop or while-loop as one code.

How to insert dummy codes?

We will explain how to insert dummy codes.

step1: We decide how many codes the dummy code will depend on.
step2: We choose a code that the dummy code will depend on.
step3: We repeat step2 until the number of codes chosen by step2 reaches the number of codes decided by step1.
step4: We insert the dummy code.

The sample of dummy code insertion is shown in Figure 1. A white circle is a code, and an arrow is dependence between codes and codes.

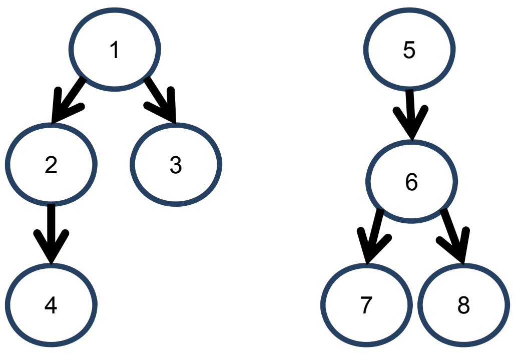

Fig. 1. Sample program for dummy code insertion

In Figure 1, code 2 and code 3 depend on code 1, and code 4 depends on code 2. Code 6 depends on code 5. Code 7 and code 8 depends on code 6.

First, in step1, we decide how many codes the dummy code will depend on. In this sample, we decide that the dummy code depend on become two codes.

In step2, we choose code that the dummy code depends on. In this sample, we choose code 5.

In step3, we repeat setp2 until the number of codes chosen by step2 reaches the number of codes decided by step1. Code 3 is selected as second. As we said, dummy code depends on two codes. It is not yet necessary to repeat.

Last, in step4, we can insert a dummy code that depends on code 3 and code5. The program that is inserted Dummy code 9 is shown in Figure 2. Gray colored circle means dummy code.

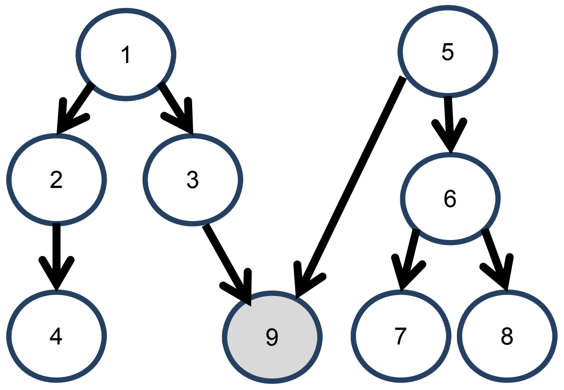

Fig. 2. The program that dummy code 9 is inserted

If we repeat this program, we can insert some dummy codes in the program. We can insert dummy codes to depend on another dummy codes. We show the sample that dummy depends on other dummy code.

In Figure 3, the number of codes decided by step1 is three. Moreover, from step2 to step3, code 7, code 8, and code 9 are chosen.

When the number of the codes decided by step1 is zero, we can insert dummy codes to depend no codes. Here, step2 and step3 are not executed. We show the sample that dummy code to depend no codes.

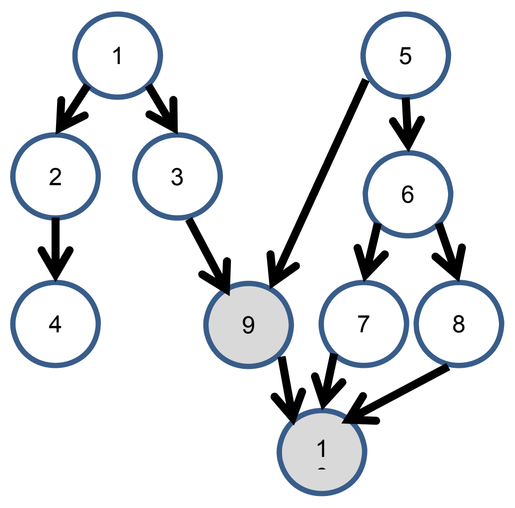

Fig. 3. The sample that dummy code 10 is inserted into Figure 2.

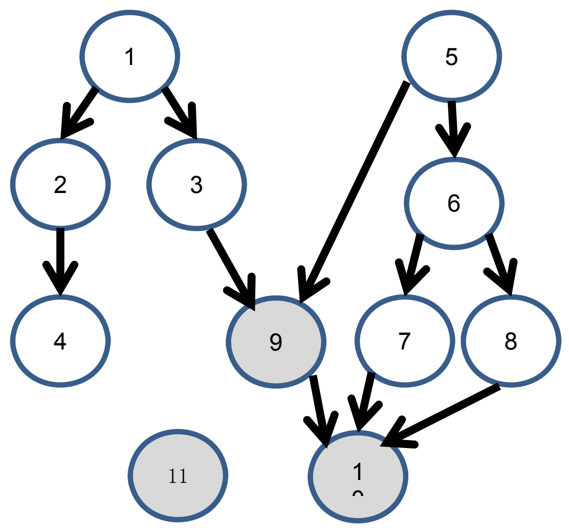

Fig. 4. The sample that dummy code 11 is inserted into Figure 3

In Figure 4, dummy code 11 depends on no codes.

How to find out the dummy codes

In order to propose strong concealment algorithm, we have to inspect as a malicious person. As you know, in research of cryptograph, malicious decryption progresses research. This story can be told in secure processing, too. So we will propose how to find out the dummy codes in the program.

The dummy code doesn't influence to the process of the original program. And the all codes in the original program are used to compute the result of the program. Therefore, we can draw the following two assertions.

- It is the dummy code that the code to depend on dummy codes.
- It is the dummy code that the code which dummy codes only depend on.

Malicious person can find out dummy codes in the program with these two assertions.

When malicious person find out dummy codes, at first, they suppose that a code is a dummy code. And they find out dummy codes by searching the supposed code. We explain how to find out dummy codes in the following.

step1: We suppose that anyone code is a dummy code.
step2: If there are codes that depend on the supposed code, we repeat from step1 for these codes.
step3: If there is a code whose all successors are dummy code, the code is also a dummy code.

Evaluation

We will insert dummy codes into tested program, and try to find out dummy code from the point of malicious person's view. The tested program is solving linear equation with Gaussian elimination algorithm, shown in [5]. This program can divide 222 codes. Dummy codes are inserted 10%, 20%, 30%, 40% and 50% of the program.

Result of simulation

A code on which no codes depend, is called "terminal code". Dummy code that implies no other dummy code is called "solitary code".

Table 1 shows the number of terminal codes in the original program.

Table 1. The number of terminal codes in the original program.

	Not solitary codes	Solitary codes	Totals
Original codes	1	73	74
Dummy codes	-	-	-
Totals	1	73	74

Table 1 shows that the original program has 74 terminal codes and 73 solitary code. So there are few solitary codes in the original terminal codes.

Table 2 shows the number of terminal codes in the program inserted 10% of dummy codes.

Table 2. The number of terminal codes in the program inserted 10% of dummy codes.

	Not solitary codes	Solitary codes	Totals
Original codes	1	62	63
Dummy codes	9	12	21
Totals	10	74	84

Table 2 shows that this program has 84 terminal codes and 10 not solitary codes. Moreover, there are 9 dummy codes in not solitary codes. So, we have inserted codes that are few codes in the original program, since it is few that the code which is terminal code but isn't solitiary code.

We have gotten the same result from the program of 20%, 30%, 40% and 50%.

Consideration

The reason that not solitary codes are inserted as dummy codes may be to choose the codes on which dummy codes depend randomly. We chose the codes on which dummy codes depend without thinking another code to depend on these codes. So we inserted dummy codes not to resemble original codes.

As the code on which dummy codes depend, we may have to choose the codes around point to insert dummy codes. We thought this idea from locality of reference.

Conclusion

From the result of simulation, we speculate that malicious person might find out dummy codes to be inserted by our method since the dummy codes don't resemble original codes.

It is a subject of future study to think how to insert dummy codes to resemble original codes. We will think the another method to find out dummy codes, too.

Acknowledgement

Thanks are due to Dr. Kouji Hirata for valuable discussion, who works at Tokyo University of science in Japan.

This work was supported by Japan Society for the Promotion of Science (JSPS) Grant-in-Aid for Scientific Research (22500060).

Wydanie publikacji zrealizowano przy udziale środków finansowych otrzymanych z budżetu Województwa Zachodniopomorskiego.

REFRENCES

[1] DAVID A. PADUA and MICHAEL J. WOLFE, "Advanced compiler optimizations for supercomputers", Communications of the ACM, vol. 29, No. 12, pp. 1184-1201, Dec 1986.

[2] Akihiko Funo, Kouji Hirata, Yoshinobu Higami, Shinya Kobayashi, "Optimistic Processing Protocol for Multiplexing in External PC Grids", in Proc. Advanced Computer Systems (ACS 2010), Pomerania, Poland, (CD-ROM), Oct. 2010.

[3] Satoshi Takasuka, Kouji Hirata, Yoshinobu Higami, Shinya Kobayashi, "Consideration of an Appropriate Program Segment Size on Method of Concealing Purposes of Processing", in Proc. Advanced Computer Systems (ACS 2008), Miedzyzdroje, Poland, (CD-ROM), Oct. 2008.

[4] Kensei Himeda, Kouji Hirata, Yoshinobu Higami, Shinya Kobayashi, "Consideration of Characteristics of Programs for Concealing Purpose of Processing in Distributed Computing Systems", in Proc. Advanced Computer Systems (ACS 2008), Miedzyzdroje, Poland, (CD-ROM), Oct. 2008.

[5] Teruya Minamoto, "The guide of numerical calculation by C language ~How to calculate, Algorithm, Program~", Japanese title "C-gengo ni yoru suchi keisan nyumon ~kaiho, arugorizumu, puroguram~ ", Saiensu-sha, Dec 2005(in Japanese).

***Authors:** Jun Kawano, Prof. Dr. Hiroshi Kai, Prof. Dr. Yoshinobu Higami, Prof. Dr. Shinya Kobayashi, Graduate School of Science and Engineering, Ehime University, 3 Bunkyou-cho, Matsuyama, Ehime, 790-8577 Japan, E-mail: kawano@koblab.cs.ehime-u.ac.jp*