12-15-2006

Some Novice methods for Software Protection with Obfuscation

SaiKrishna Aravalli
University of New Orleans

Some Novice methods for Software Protection with Obfuscation

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
In partial fulfillment of the
Requirements for the degree of

Masters of Science
In
The Department of Computer Science

By

SaiKrishna Aravalli

B.Tech, Jawaharlal Nehru Technological University, India, 2003

December, 2006

Acknowledgements

I would like to express my gratitude to the Almighty, to my professors, my parents and to the number of people who became involved with this thesis, one way or another.

I am eminently grateful to my supervisor, Dr. Bin Fu whose help, suggestions and encouragement helped me in all the time of research for and writing of this thesis. In addition, I would like to thank Dr. Shengru Tu and Dr. Golden Richard III for being on my thesis defense committee. It was a great honor having them both.

I am very much thankful to my family and friends for the continuous support and help they provided to me.

Table of Contents

List of Figures

Some Novice methods for Software Protection with Obfuscation

Abstract:

Previously software is distributed to the users by using devices like CD'S and floppies and in the form of bytes. Due to the high usage of internet and in order to perform the tasks rapidly without wasting time on depending physical devices, software is supplied through internet in the form of source code itself. Since source code is available to the end users there is a possibility of changing the source code by malicious users in order to gain their personnel benefits which automatically leads to malfunctioning of the software.

The method proposed in this thesis is based on the concept of using hardware to protect the software. We will obfuscate the relation between variables and statements in the software programs so that the attacker can not find the direct relation between them. The method combines software security with code obfuscation techniques, uses the concepts of cryptography like hashing functions and random number generators.

Chapter 1

Introduction

Software security has become mandatory, from the time software is made available through online .Many software vendors are selling their software's by the means of online from remote servers to remote clients, in which there is a probability for software theft from malicious users .Protection against piracy is nothing but protection against malicious host attackers .Due to software piracy there is a great loss to the nations revenue and integrity. If once the pirate gets the software he can modify it by using many conceivable tools thus breaking protection. There are some tools which allow system calls monitoring, file activity process and modification of important system functions.

If there is a massive source code, then the attacker cannot obfuscate whole source code, in order to obfuscate he will look into the program randomly and tries to identify the important parts of the source code such as important functions, statements and expressions and then he will try to change those parts which leads to malfunctioning of the code.

Most of the techniques proposed to prevent source code piracy are based on code transformations, obfuscation using automated computer machines, and altering the code functionality by modifying arrays, pointers and loops in the source code. The reverse-engineering community is so advanced that they have many tools which can attack above mentioned techniques thus leading to source code piracy.

In order to avoid piracy, we need to develop such techniques which make the job of reverse-engineer difficult and time consuming and the source code should be designed by implementing some time constraints so that there exist some dynamic changes in the

source code. Though the concept of software obfuscation provides security to the source code, the level of security provided can be broken if the attacker tries to deobfuscate the source code many times.

1.1 Most common types of attacks are:

Software Piracy [1]: It is the most common attack on intellectual property .It is nothing but unauthorized copying of software by breaking the "password" which is used as a protection. Previous methods of software protection were based on typical password methods. The supplier used to hide the password and handed over to the user secretly. But the attackers used to break this method by using trail and error methods and using some software's which can remember the password.

Reverse engineering [1,6,15] :The reverse engineer generally studies the terminology behind the software and tries to understand the software product so that he can identify the "secret key" and then he manufactures his own software by utilizing this key. Generally reverse-engineering takes place by reversing the program machine code back to the source code in which it was designed previously. By doing this the attacker can get complete knowledge about the source code and its functionality and then he will customize the source code based on his requirements. Reverse engineering is done by using some software tools which can disassemble the source program. A tool called Hexadecimal dumper, whose functionality is to display or print the binary numbers of the program in hexadecimal format which is very flexible to read when compared to binary format. By gaining the knowledge of bit patterns and instruction lengths the reverse engineer can understand the functionality of the certain parts of the program.

A disassembler is a tool which reads binary code and then displays executable instructions in text form. The drawback of disassembler is that it cannot differentiate between executable instruction and the data used by the source code, for which a debugger is used.

By using the above tools the, the hacker tries to understand the source code and tries to change its functionality which leads to malfunctioning of the source code.

Reverse-engineering tries to reverse a piece of source code.

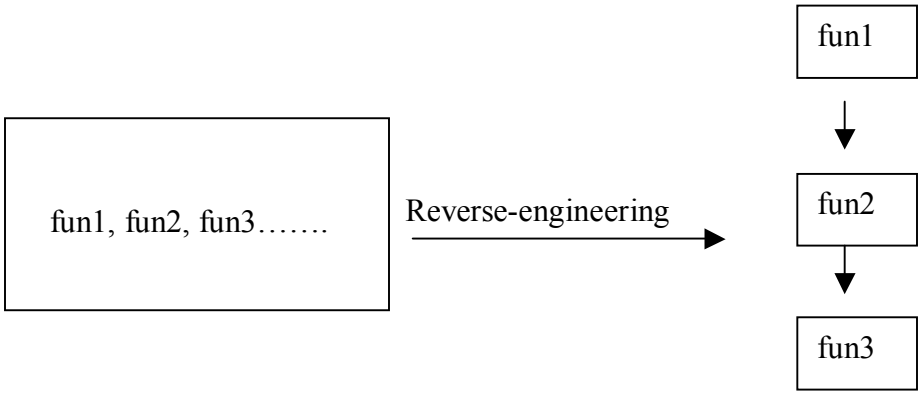


Fig 1.1 Reverse-engineering

1.2 Tampering [6,9]**:** It is nothing but changing the actual code so that it losses its functionality. Generally tampering is combined with reverse engineering and software piracy hence it is really a dangerous attack.

Because of all these attacks there is a great financial loss for software vendors and some times to the integrity of nation .Hence many software security programs are actively going on in order to avoid these software piracy.

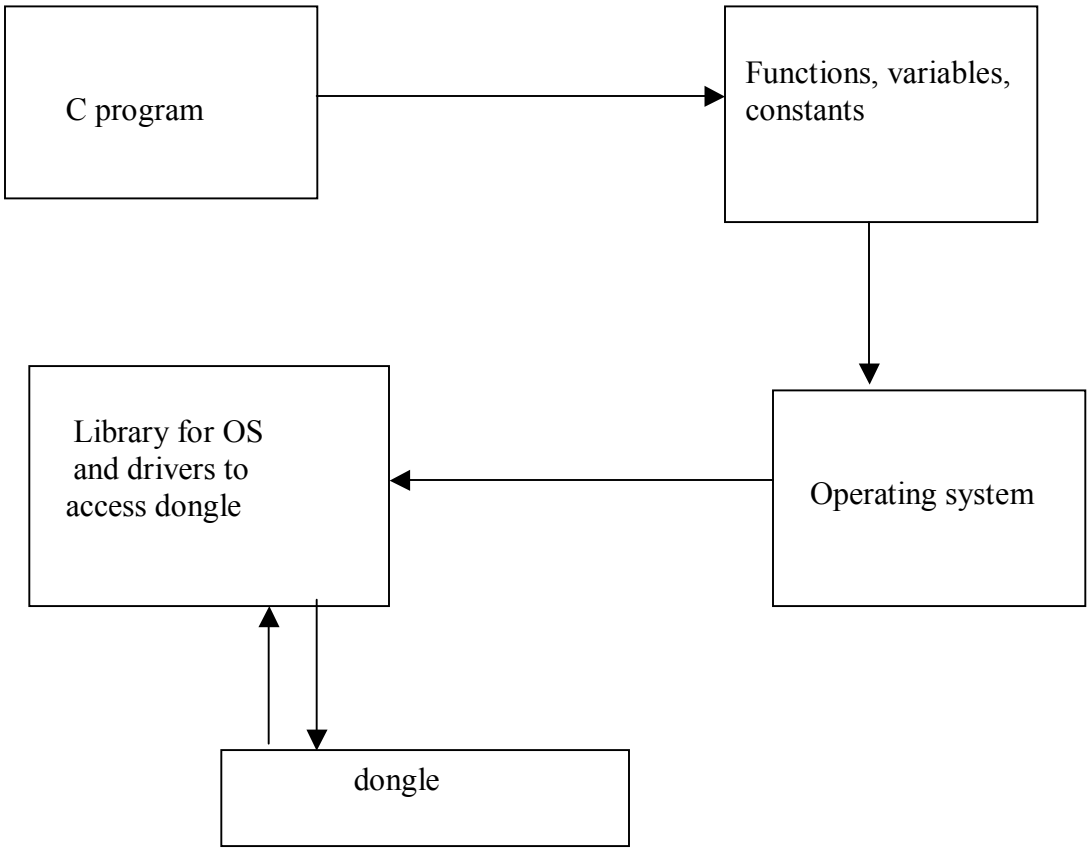
1.3 In order to protect software from piracy, many techniques have been proposed using both hardware and software fields.

Hardware Methods**:** Using smart cards, secure processors and hardware dongles.

A dongle is a hardware device which is attached to the port of the computer. The dongle uses passwords and secret keys which are embedded inside for controlling and accessing software applications. The software associated with dongle runs only when the dongle is attached to the terminal.

Fig 1.2 : Dongle

Fig 1.3 : architecture of dongle



Software Methods:

1) Software obfuscation
2) Watermarking and
3) Tamper-proof are some of the techniques.

Explanation in brief:

1.3.1 Software obfuscation [1,7,10], is nothing but the process of confusing and obscuring the software from malicious users so that the security of the software can be preserved without losing its functionality. There are many methods which provide software obfuscation such as layout obfuscation, data obfuscation and control flow obfuscation [1].

If prog1, prog2 are the original programs, after applying obfuscation methods they are transformed to prog11, prog22.Though the obfuscated program contains different variables and functions, the functionality will be same as original program.

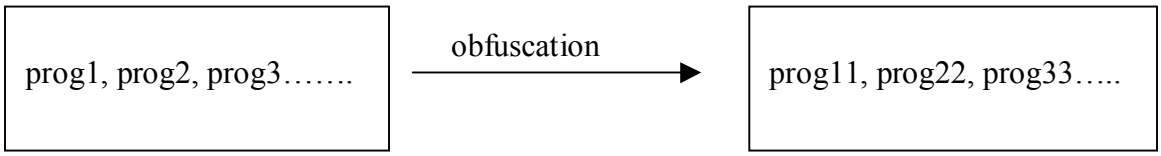


fig 1.4 Obfuscation

The obfuscated program should follow

1) High obscurity: The process of deobfuscation should be very high when compared to original program

2) High resilient: The transformed obfuscated program should have high resistant towards deobfuscator devices.
3) Cost: Care should be taken while applying obfuscation techniques so that it should be minimum so that obscurity and resilient are maximum.
4) Stealth: Generally resilient programs are less prone to deobfuscation tools though they can be tracked by human beings. If the transformed program looks completely different from the original program , then the reverse engineer easily understands that the transformed program is an obfuscated program.

   Therefore stealth is nothing but transforming the original program into obfuscated program with maximum resemblance to original program.

1.3.2 Software watermarking [8] provides security to the source code by mentioning copyright notice which implies that the particular software belongs to that user. The aim is to avoid software theft by mentioning ownership. When watermarking is embedded in a program care should be taken so that the program does not lose its performance. There exists a tradeoff between watermarking methods and resilience, cost and stealth properties.

Software watermarking algorithms [16] are classified into static or dynamic .

Static watermarking depends on the properties of application that are available at compile time so that it can embed watermarking properties.it is classified into two types

1. Data watermark
2. Code watermark

Dynamic watermarking depends on the information gathered during the execution of the application. Though we cannot prevent piracy from this process, we can control illegal copy of the software. It is classified into three types

1. Easter eggs
2. Execution trace watermark
3. Dynamic data structure watermark

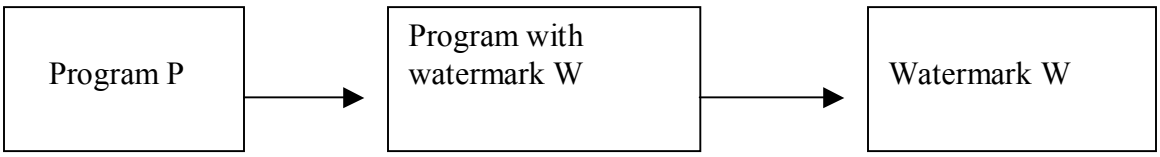


Fig 1.5 Watermark flow

Software watermark should follow these properties:

Resilience: Resilience should be high because it should resist de-watermarking techniques

Stealth: It should never lose its property; original program should not lose its functionality when it is transformed to watermarked program.

1.3.3 Tamper-proof [1] is a technique which prevents unauthorized modifications to the source code .If a program is watermarked and the source code that designed the watermark has been changed, then the goal of tamper-proof method is to prevent that program from executing.

If a malicious code has been attached to the source code, or if the source code which is useful for business applications has been modified, then by applying tamper proof code we can avoid some of tampering attacks

The main aim of tamper proof techniques are to detect whether the original source code has been modified and to prevent execution of tampered code.

Generally tampering of source code can be detected by:

a) Testing the executable program whether it is similar to original program
b) By checking the out put of the executable program so that it matches with the original program.

Custom operating system [3]: This is not a good method but still gives security to the programs to some extent. Here, a customizable operating system is created where in the security, check summing and obfuscation are secretly embedded.

Software aging: This is based on periodic program updating. These provide bug fixes and new features to users along with keeping program synchronization with other software's on which they depend. By providing frequency update the functionalities of older versions are lessen and this leads to constant work for pirates to track the software.

Cryptographic technology [3,5,12]: These technology gives good protection by hiding the software code from the pirate eyes. But these technologies have to overcome some difficulties like unless a good trusted in-fracture is used, giving cryptographic keys can leads to problem, bad performance.

Chapter 2

Objectives

Main objectives of obfuscation are:

1) Trying to preserve semantic meaning of original source code.
2) Must be time consuming for the reverse engineering which is called “obscurity”
3) It should be resistant to automatic tools nothing but resilient.

Software obfuscation is classified into

1) Lexical transformations: The goal of this transformation method is to change the variable names and replacing them with obscure method.
2) Control transformations: Deals with changing the flow of the program by altering loops and conditional statements while preserving its functionality
3) Data transformations : Deals with changing the data structures of the program

In order to provide code obfuscation in C there was a tool called C-Shroud [4,11] which was used to obfuscate source code .The code for which obfuscation is to be implemented is passed to the C-Shroud tool, which applies some standard algorithms and results in obfuscated code which is difficult for the malicious users to deobfuscate. If the attacker obtains the algorithms which are used in the tool then he can deobfuscate the source code. In order to eliminate this problem, the algorithms should be customized and designed according to the user application.

2.1 Difference between our method and other commercial methods:

Our approach is for source codes which are in C programming language and it is based on the concepts of control flow obfuscation techniques .The method is based on cryptographic concepts like hashing functions and random number generators. In our method, we are trying to hide the relation between variables in the source code so that the attacker can not understand where the particular variable is used in the expressions.

Here we have obfuscated the correspondence between variables by passing variables between another source code and the main source code.

Generally, obfuscated variables and other obfuscation operations will be present on same source code unlike considering other source code reference to main source code. Lot of obfuscators are based on Java and .NET languages and there are many commercial products available in this area. Some of them are code obfuscation tools by 9Rays.Net Inc, by Informentum.com, Semantic Designs,Inc. and many other tools are available based on the requirements.

Generally obfuscation can be done by selecting a crucial part of the source code wherein the variables and other functions are obfuscated. Based on the confidentiality of the source code, there exist levels of obfuscation.

If we want to obfuscate some common parts of the source code which is not much important we can apply low level of obfuscation, and if we want to obfuscate some crucial parts of the source code, higher level of obfuscation is employed.

While employing obfuscation techniques, we can obfuscate some crucial variables, crucial functions or some variables and constants can be introduced into the source code and they are never used, arrays can be obfuscated.

Many obfuscation methods are based on changing variable names and function names. This type of obfuscation is called low level obfuscation, because an attacker can easily identify the correct variable and he can substitute it how large the source code is.

In order to eliminate these kinds of issues, we have tried to hide the variables by obfuscating the relation between the areas where the variable is used such as expressions, functions and some arithmetic operations.

2.2 Related work:

A lot of research was done previously in the area of software protection. A technique proposed by wang et al. is based on the concepts of aliases. According to his technique, the precision of static analysis in a program reduces if the program contains aliases[16]. Interprocedural analysis is required in order to understand the functionality of the source code whether we obfuscate it or not. Generally interprocedural analysis follows intraprocedural analysis and any obfuscation technique which can break interprocedural analysis can also breaks intraprocedural analysis. Therefore obfuscation techniques based on interprocedural analysis are effective.

Previous methods of variable obfuscation involved changing the names of the variables in the source code. The variables are named in such in way that the attacker should get confused in order to identify a particular variable.

Chapter 3

Obfuscation Method 1

3.1 Establishing the communication

Here server acts as a hardware component [2] for protecting the software which is on client side. On client side, the user specifies the input, which will be passed as a parameter to a hashing function 'h' and a function 'f'.

The hashing value h(x) returned by the hashing function is passed as a parameter to another hashing function 'g' which is present on both client and server side.

The hashing value h(x) obtained from the client hashing function 'h' is passed as a input to the hashing function 'g' on server side, and the hashing value g(h(x)) obtained from the hashing function of server is sent to the client. The hashing value g(h(x)) from server is sent to the client . The hashing value h(x) is combined with the hashing value g(h(x)) on client side. Here we are actually trying to obfuscate the relation between a variable and the statement where it is used.

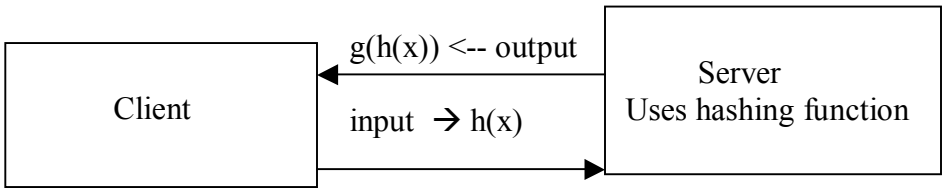


Fig 3.1 General client and server

3.2 Reason for choosing hashing functions between communications [17].

A hash function H is a transformation that takes an input m and returns a fixed-size string, which is called the hash value h (that is, h = H(m)). Hash functions with just this

property have a variety of general computational uses, but when employed in cryptography, the hash functions are usually chosen to have some additional properties.

The basic requirements for a cryptographic hash function are as follows.

- The input can be of any length.
- The output has a fixed length.
- H(x) is relatively easy to compute for any given x.
- H(x) is one-way.
- H(x) is collision-free.

A hash function H is said to be one-way if it is hard to invert, where ``hard to invert'' means that given a hash value h, it is computationally infeasible to find some input x such that H(x) = h. If, given a message x, it is computationally infeasible to find a message y not equal to x such that H(x) = H(y), then H is said to be a weakly collision-free hash function. A strongly collision-free hash function H is one for which it is computationally infeasible to find any two messages x and y such that H(x) = H(y).

Hence by using the hashing value as an input, the input used for communication between the client and server will always change rather than to be fixed, since the system clock changes its timings.

Unlike the “Password” which is fixed and can be easily cracked by the attacker, by using the hashing value we can send different inputs between the communication channels thus creating confusion for the user and thus security can be obtained.

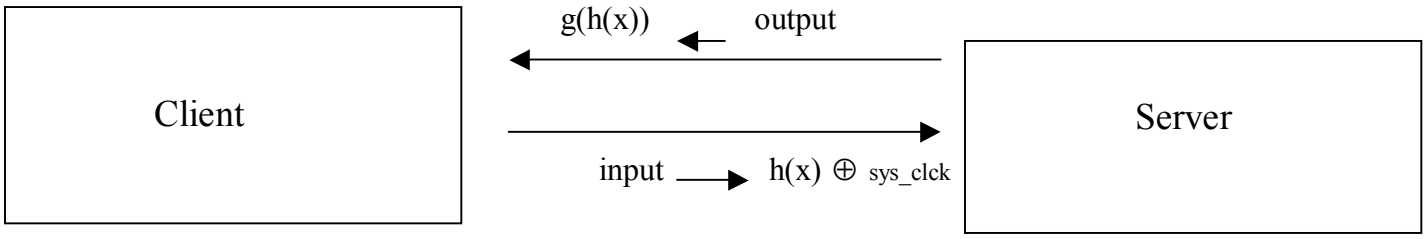


Fig 3.2 Client and Server showing Hashing function

Algorithm:

double x,y;

Assume variable to be x, a function 'f' and a hashing function 'h'. The input is applied to both the functions on client side and the resulting hashing value h(x) is sent to the hashing function 'g' which is on server.

On client side we can merge the function value f(x) and hashing value g(h(x)) as

x' = f(x) ⊕ g(h(x));

The hashing value sent by the server is

y' = g(h(x));

The hashing value received from the server is combined with the hashing values on the client side.

x'' = x' ⊕ y', where x' is the obfuscated value of f(x) ⊕ g(h(x)) and y' is identical to g(h(x))

From the above it implies that we have obfuscated the function f(x).

If we assume that we have a statement

x'' = x' + y'

= (f(x) + g(h(x))) ⊕ g(h(x))

= f(x)

Therefore in order to perform this obfuscation operation, we have considered network where server acts as a hardware and client acts as a software.

If the attacker wants to hack this method, he has to understand the source for x'' which can be understood only if he has access to server. Since client will be accessed by user, we have to obfuscate the client code so that the user should not understand it and the obfuscated code should not lose its original functionality by preserving the security of the code.

In order to maintain the performance of the software implemented using above method, we should control the frequency of communication between the client and the server. If the frequency is controlled then the waiting time between the client and the server for the communication to take place can be minimized. The code responsible for controlling the frequency of communication should be obfuscated.

If the malicious user wants to break the security of this code, he should need to get access to both client and server .On server side , just the input is combined with hashing function and its hashing value is sent back to client .Actually obfuscation takes place at client side. The malicious user has to understand the client code. By applying various obfuscation techniques such as

1) Merging of functions

2) Hiding loops

3) Inserting useless code

4) Changing control flow of the statements in the program

5) Changing variable names

6) Calling dead functions

we can make job of the attacker hard to understand the client code

We have a program which performs square of a number.

```
#include <stdio.h>
 int main()
{
int a ,x;
printf("enter the value of a \n");
scanf("%d",&a);
printf("square is %d",x = a*a);
return 0;
}
```

The obfuscation of the above program can be done as follows:

On client side:

double x,y;

The obfuscated program of square of a number can be obtained as follows.

Input is sent to a function 'f' and a hashing function 'h' and its corresponding function value f(x) and hashing value h(x) are obtained.

This hashing value in turn is passed as an input to another hashing function 'g'

thus producing hashing value g(h(x)).

From the above, function value and hashing value are merged together

thus x' = f(x) $\oplus$ g(h(x))

if we assume y' = g(h(x)) be the hashing value received from the server then

$x'' = x' \oplus y'$

$= (f(x) \oplus g(h(x))) \oplus g(h(x))$

$= f(x)$

On server side:

The input which is received from the client is passed as a parameter to the hashing function g.

The hashing value g(h(x)) is sent back to the client.

Chapter 4

Obfuscation method 2

4.1 Hiding the connections

double x;

Assume variable to be x, and a hashing function ‘h’. The input is applied to a hashing function on client side and the resulting hashing value h(x) is passed to another hashing function ‘g’ which is present on both client and server side.

Since we are passing hashing values in the network, there will be no problem with number of bytes sent.

Assume a simple function which performs addition of two numbers.

int add ();

in the function , we have z = x + y. We will try to obfuscate x and y so that the attacker will get confuse, and cannot understand that the function performs addition operation.

If we consider variable x, and its address as addr_x

addr_x = &x;

we pass this variable as a parameter to the hashing function ‘h’. If h(x) is the hashing value, that is the output of hashing function ‘h’, we pass this hashing value as a parameter to other hashing function ‘g’ which is present on both user side and server side.

Let us assume the hashing function be int hash_1(int x) , where x is the parameter.

4.2 Algorithm in brief for hiding the connections:

Here we are trying to hide a variable which is used in multiple places in the program, therefore the statements using that variable must be obscure in order to confuse the attacker. Basically we are trying to obfuscate the connection between the variable and the statements where it is used.

Continuing from the above:

Assume function be int hash_1(int x1).

If the hashing value of hash_1 hashing function is g(h(x)),based on the value of g(h(x)), we can obtain cli_value = &x - g(h(x));

Similarly the hashing value h(x) is sent to the hashing function 'g' on server side and the resultant hashing value g(h(x)) is obtained.

ser_value = g(h(x)); and s' = &ser_value;

final_value = *(cli_value + ser_value) based on the value of g(h(x)), where final_value is the obfuscated value of x.

If the obfuscated program receives correct message from the server,

*(cli_value+ ser_value) implies it is accessing actual value of 'x', else if it receives wrong message that implies the value accessed is from another address.

Here it is not easy for the attacker to figure out that *(cli_value+ ser_value) and x are the same.

Here we tried to hide the code that computes &x – g(h(x))

From the above we have obfuscated variable x in z = x + y as z = final_value + y;

in order to understand the final_value , the attacker has to know about cli_value and ser_value which can only be known if he/ she has the knowledge of the functions and which are present on client and server side.

4.3 Hiding of variable using mathematical concepts:

In this section, we are trying to obfuscate a variable by using mathematical calculations.

Here we are trying to obfuscate variable 'y' by introducing some dummy expressions along with the variable.

Say y = a*a – 2*a * b+ b*b + 2*a*b*y - (a-b) ^2 / 2*a*b;

the above expression is independent of the values of a and b and its value is based on algebraic operation.

The value of a*a – 2*a * b+ b*b + 2*a*b*y - (a-b) ^2 / 2*a*b is 'y' which can be found by solving the expression.

Here a*a = a^2

2*a* b = 2ab

b*b = b^2

(a-b)^2 = a*a – 2*a*b+b*b

By substituting the above values in the expression, we get the value of 'y' = 'y'.

Then the above equation becomes

f(x,y)= final_value + a*a – 2*a * b+ b*b + 2*a*b*y - (a-b) ^2 / 2*a*b;

in order to understand the above statement , the attacker need to know about final_value and the dummy expression which is introduced in place of y.

Thus we have obfuscated both x and y here.

Therefore in place of z = x + y;

We use the above statement

f(x,y) = final_value + a*a – 2*a * b+ b*b + 2*a*b*y - (a-b)^2 / 2*a*b;

Similarly if we have ‘n’ variables then we can obfuscate all the variables which makes the job of attacker very tough.

If the obfuscated program receives correct message from the server,

*(cli_value +ser_value) implies it is accessing actual value of ‘x’, else if it receives wrong message that implies the value accessed is from another address.

By applying the above methods, the probability of confusing the attacker is high since he cannot understand easily the value of

f(x,y) = *(cli_value + ser_value) + a*a – 2*a * b+ b*b + 2*a*b*y - (a-b)^2 / 2*a*b and

z = x + y are the same where z = f(x,y).

Client -Server implementation of the above algorithm:

In our thesis, a client is nothing but software a component which will be accessed by user. The user gives input information to the client and that input is sent to the server through ‘sent’ function .When the server receives the input, it passes it to a hashing function and the hashing value is obtained, which is sent back to client .The input is also passed to the hashing function which is present on client and its corresponding hashing value is obtained. We combine the hashing value with the address of the input operand and the new address is obtained.

The algorithm on server side follows this method

The input received from the client is passed to hashing function and its corresponding hashing value is sent back to client .Actual obfuscation takes place on client side. When once we get the hashing value from the server side it can be obfuscated by using obfuscation techniques.

Chapter 5

Obfuscation Method 3

5.1 Merging of Functions

Merging of functions plays a vita role on Software Protection. By introducing some new functions and by obfuscating existing functions we can generate good obfuscation methods.

Let us assume that we want to implement obfuscation method for merging of two functions. If we simply merge the two functions without applying obfuscation methods then the attacker can easily understands the functionality of the software and tries to pirate it.

From the previous methods, assume the function be f(x). Here we proposed a method that shows how to make obfuscation hard to extract f(x) from the software if it is merged with other functions.

Let us assume that we have two functions.

```
void f(double x)
{
s1;
s2'
}
void g(int y)
```

```
{
a1;
b1;
c1;
}
```

Here we introduce a new function h(double x, int y, int z) .

Based on the value of the parameter ‘z’ we execute the functions. We assume that if the value of ‘z’ is greater than ‘zero’ then we execute function ‘f()’ else we execute function ‘g()’.

```
void h(double x, int y, int z)
{
if (z>0)
{
s1;
s2;
}
else
{
a1;
b1;
c1;
```

```
}
```

Therefore we can replace f(x) by h(x,0,5) and g(y) by f(0,y,0).

Eliminating the boundary between the functions:

We can further obfuscate this method based on the values of parameters 'h' .If the resultant value of 'z' module is '1' then we should execute a1, else s1.

If we assume that if the value of 'x' is greater than '3' then execute 'b1' else 's2'..

```
void h(int x,double y,int z)
{
If (z%2 = = 1)
{
a1;
}
else {
s1;
}
if (x>3)
{
 b1;
 c1;
}
```

```
else
{
s2;
}
}
```

By implementing the above method, we tried to eliminate the boundary between the two functions f() and g().

By doing so the job of the attacker becomes hard in order to remove the code used for software protection.

Chapter 6

Generalization

In the above cases, we have applied obfuscation methods to a single variable.

By doing so the output of obfuscated program was similar to the original program, which implies the method performed its functionality correctly.

Since a program contains many variables, if we apply these obfuscation methods then the obfuscated program will be very confusing when compare to original program.

We can apply same obfuscation method to the same variable multiple times and care should be taken so that obfuscated program does not lose its functionality thus creating a more confusing program.

Assume 'F()' is original program 'O()' is obfuscation method then,

O(O(O(F))) is the obfuscated program which have been obfuscated multiple times by applying the same obfuscation techniques.

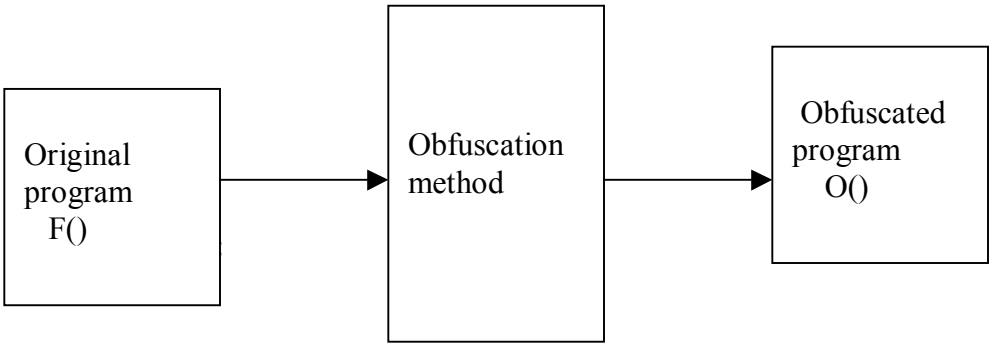


Fig 4.1 Generalization of obfuscation

Generally the performance of obfuscated program will be less than the original program because, in the process of obfuscating the program numerous logical operations are applied to the original program such as adding dead code, inserting extra variables changing loops which automatically reduces the performance of the obfuscated program. The complexity of obfuscated program will be higher than original program because of application of complex mathematical calculations. Even though the performance is less and complexity is more care should be taken so that these variations should be negligible.

6.1 Some various algorithms in brief:

Algorithm 1:

Hiding the counter:

Let us assume that we have a simple while loop such as

```
While (count<=100)
{
count = count + 2;
}
```

From the above, the user can easily understand that, if the count in the while loop is less than 100 , then count can be increased by 2.In order to confuse the user , we need to obfuscate the above while loop in such a way that the functionality of the loop should be same .

```
int count = 1;
while ( count + [(a+b) * (a+b) - 100 – (a-b) * (a-b) ]< = 4 * a * b)
{
```

Count = [count * a – b\ 4 + (a * a) - (b * b) \ (a + b)*2] 4 \ a – b;

}

In the above we are hiding both “count < = 100” and “count = count + 2”

Algorithm 2:

Obfuscating functions:

If the software consists of functions, then we can merge the functions or we can split the functions based on the requirements.

Example: let us assume there are two functions ‘h(x)’ and ‘g(x)’, then we can merge then into one using another function z(x).

$$z(x) = h(g(x))$$

if int add(int a,int b) and int sub(int c,int d) are two functions , then they can be merged into one as

```
  int arthematic(int a1,int b1)
{
int a,b,c1;
printf(“enter the values of a and b \n”);
scanf(“%d%d”,&a,&b);
if( a<= b)
printf(“sum is %d \n”, c1 = a + b);
else if(a >= b)
printf(“diff is %d \n”,c1 = a – b);
}
```

The above code represents merging of two functions. We can obfuscate the above code by using multiple ‘if else’ conditions.

6.2 Following is the example of obfuscating variables by changing their names and introducing dummy functions into the program.

We have considered Linear search of an array program, which compares each element of the array with the search key .Here we have inserted two functions into the source code whose output does not change the output of the original program.

Original program :

Linear Search of an array:

```
#include <stdio.h>
#define SIZE 100
int linearsearch (const int[],int,int);
int main()
{
        int a[SIZE],x,searchkey,element;
        for(x = 0; x<=SIZE - 1;x++)
                a[x] = 2*x;
        printf("enter integer search key : \n"); //takes input//
        scanf("%d", &searchkey);
        element = linearsearch(a, searchkey,SIZE);
```

```
        if(element != -1)
                printf("found value in element %d \n",element);
        else
                printf("value not found \n");
        return 0;
}
int linearsearch(const int array[],int key,int size)  //performs linear search//
{
        int n;
        for(n = 0;n<=size - 1;n++)
                if(array[n] == key)
                return n;
        return -1;
}
```

Obfuscated version of Linear Search of an array:

Here we have inserted two functions and have changed the variable names. Also removed the comments in the program so that the attacker gets confused.

Though this kind of method does not provide complete obfuscation to the source code, but it confuses the attacker. This is a general kind of obfuscation method which is applied to the programs.

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#define val 100
int seq_fin(const int[],int,int);
int doub_fin(int,int);
int pro_sol(int,int);

 main()
{
        int z11_11[val], xyz, key10,x,y,x1,y1,valu, paramet,addition, product;
        for(xyz = 0; xyz<=val - 1; xyz++)
        {
z11_11[xyz] = 2*xyz;
        }
        printf("request for  key \n");
        scanf("%d",&key10);
paramet = seq_fin(z11_11,key10,valu);
printf("key is %d",paramet);
addition = doub_fin(x,y);
printf("value is %d \n", addition);
product = pro_sol(x1,y1);
printf("value is %d \n",product);
```

```
}
int doub_fin(int x, int y)
{
        int z;
        printf("enter values of x and y");
        scanf("%d%d",&x,&y);
z = x + y;
        return z;
}

int seq_fin( const int array[100],int key10,int valu)
{
        int n;
        for(n=0;n<=valu - 1;n++)
                if (array[n] == key10)
                return n;
        return -1;
                        }
int pro_sol(int x1,int y1)
{
        int z;
        printf("enter the values of x and y");
        scanf("%d%d",&x1,&y1);
```

```
        z = x1 * y1;
        return z;
}
```

Algorithm 3:

Using multiple if else statements.

By introducing dummy conditions, we can generate many if else statements.

```
if ( exp 1)
{
Statement 1;
elseif(exp 2)
{
Statement 2;
elseif(exp 3 || exp 4)
{
Statement 3;
elseif(exp 5 && exp 6)
{
Statement 4;
else
Statement 5;
}
}
```

```
}
}
```

By merging all these algorithms among each other and with our basic technique we can obtain good software obfuscation.

6.3 Algorithms implementation:

We have implemented the obfuscation techniques in C using Cygwin and the editor used is edit plus.

Since we are trying to obfuscate variables, based on the program we select some core variables which play a vital role in program execution.

We have applied different obfuscation techniques to the selected variables like changing variable name, obfuscating its location, and inserting the variable in between dummy expressions so that the ultimate result will be the selected variable.

Though the execution time of the obfuscated program will be slightly higher when compared to original program, the functionality did not change.

Chapter 7

Conclusion and Future work

In this method, we are using a hashing value which is obtained from server to the client, and we are obfuscating that hashing value in the client .Basically we are trying to obfuscate the connection between the variables and the locations where they are used.

Since the server can be a dongle or a small chip, therefore it is some what complex task for the attacker to break this concept even though he may have complete access to the source. Unlike other obfuscation methods which rely on confusing the source code by changing the variable names and inserting or deleting some dummy functions , they can be understand by static analysis , but our method makes the job of attacker complex because he may not able to understand the dongle or chip properties by static analysis .

Generally the size of obfuscated program will be larger than the size of original program, due to insertion of extra variables, constants, functions and some dummy codes.

Due to this, the execution speed of the obfuscated program will be slower when compare to original program. In order to mitigate above problems, obfuscation techniques should be applied to important parts of the source code only rather than applying to entire code.

In future, we can improve this concept by merging the functions among themselves and hiding the major variables and other address locations in different programs.

By improving this concept we can design our own obfuscation tool which contains a Graphical User Interface where the user inputs his source code and selects the level of obfuscation by selecting the appropriate buttons and thus obtains an obfuscated source code.

References

1) Christian S. Collberg and Clark Thomborson "Watermarking , Tamper-proofing , and Obfuscation tools for software protection" . IEEE Transactions on software engineering , vol.28, no.8 , August 2002.

2) Bin Fu , Golden Richard III , Yixin Chen "Some New Approaches For Preventing Software Tampering". ACM SE'06 March 10-12 , 2006.

3) Levent Ertual ,Suma Venkatesh "Novel Obfuscation Algorithm for Software Security".

4) Douglas Low "Protecting Java Code via Code Obfuscation".

5) Gimpel Software For C "http://www.gimpel.com/".

6) Randy Bentson "Issues (and solutions?) in Electronic voting systems".

7) D. Aucsmith. Tamper-resistant software: An implementation. In Proceedings of the First International Information Hiding Workshop, Lecture Notes in Computer Science 1174, pages 317--333, 1997.

8) Christian Collberg , Clark Thomborson , Douglas Low "A Taxonomy of Obfuscating Transformations" Technical Report # 148.

9) Christian Collberg, "SandMark : A Tool for the study of Software protection Algorithms" .

10) Paoco Falcarin , Mario Baldi, Daniele Mazzocchi "Software Tampering Detection using AOP and Mobile code".

11) Toshio Ogiso, Yusuke SAKABE , Masakazu SOSHI and ATSU'KO Miyaji "Software Obfuscation on a Theoretical basis and its implementation – IEICE TRANS.FUNDAMENTALS , VOL.E86-A, NO.1 January 2003.

12) Dr.John A. Hamilton, Wade Chatham, Brain Eoff, Eric Imsand, Adam Sachitano " Security Issues resulting from Interoperability".

13) Jon CROWCROFT "A Brief Introduction to Cryptographic Technology" .

14) Niklas Linde'n "Protecting Copyrighted Software: Code Obfuscation using Genetic Algorithms".

15) C.Wang, J.Hill, J.Knight and J.Davidson.Software tamper resistance:Obstructing static analysis of programs ,report CS-2000-12,department of CS ,University of Virginia,Dec. 2000.

16) Yong He "Tamper Proofing a software watermark by Encoding Constants". http://www.cs.auckland.ac.nz/~cthombor/Students/yhe/yhethesis.pdf.

17) RSA Laboratories Title: RSA Laboratories' Frequently Asked Questions About Today's Cryptography, Version 4.1 *Year:* 2000 Publisher: RSA Security Inc.

18) Reverse Engineering Community http://www.reverse-engineering.net/index.php?c=resgeneralrce

Books: Unix Network Programming by W.Richard Stevens

Vita

SaiKrishna Aravalli was born in the city of Hyderabad in India, on Aug 3, 1982. He got his Bachelors of Technology in Computer Science & Information Technology from Jawaharlal Nehru Technological University in May, 2003. He joined the Masters of Sciences in Computer Science at the University of New Orleans in 2004. During this time, he worked as a Graduate Assistant for Computer Science Department for a semester and for couple of semesters worked as a Research Assistant under Dr.Bin Fu.