Obscuring Mobile Agents by Source Code Obfuscation

Veena Garg
Department of Computer Engineering
YMCA University of science & Technology, Faridabad, India

Atul Srivastava
Department of Computer Science
Echelon Institute of Technology, Faridabad, India

Atul Mishra
Department of Computer Engineering
YMCA University of science & Technology, Faridabad, India

ABSTRACT
Security and performance are most essential and prime challenges for networking phenomenon. Computation on the remote host is generally done through links. Thus security is needed when the code is on the way to the destination host. A program travelling over the link is extremely venerable to be forged for malfunctioning. On the other hand Software's are commonly distributed with all information in the code itself, for example java byte codes. Byte codes are easy to reverse engineer. Any rival company may get the algorithms and techniques used in the product. Therefore a protection is needed to keep information secret. In both the cases obfuscation seem to be promising solution to the problem. Obfuscation makes code less understandable without changing its functionality. In this paper we have proposed an obfuscator that converts source code of a mobile agent into unintelligible code. Whole paper mainly focuses on obfuscating mobile agents whereas the technique can be used for any software obfuscation.

Keywords
Mobile agents, network management, network security, obfuscation, code confidentiality.

1. INTRODUCTION
Application of networks in daily life has grown enormously in last few years. Therefore network management has become biggest challenge to achieve. Traditionally network management is done using client server approach to poll agents on network elements. But this approach has several limitations like poor bandwidth utilization; extensive processing capability at the manager node that imposes delays in the process. Mobile agents offer comparatively better approach to perform same task with less cost in terms of time and bandwidth capacity. But the facility does not come without cost, as mobile agents suffer from security point of view. Mobile agents are java programs that have capability to move from node to node in a network with code and state.
On the other hand, mobile agent technology has some limitations, primarily in the area of security. This unbarred mobility of mobile agents makes them venerable to be caught by malicious invader who can modify the code or state and resend it with her functionalities. These limitations have raised many concerns about the practical utilization of mobile agents.

It becomes essential to protect mobile agents from such attacks. The mobile agent and mobile agent platform i.e. host both have same security requirements as network confidentiality, integrity, anonymity, availability. Threats to the security generally fall into the classes [3]: masquerade, denial of service, eavesdropping, alteration, unauthorized access, copy and replay and so on. The components of a mobile agent system categorize the security threats by acting as source and target of an attack [3]: Threats from external entities to hosts/agents, Threats from hosts to mobile agent, Threats from mobile agent to other mobile agents, Threats from mobile agent to hosts.

This paper mainly focuses on the solution to protect mobile agents from malicious hosts. Malicious host can capture the mobile agent and can read sensitive information or modify the code to alter its functionality or implanting a virus, worm or Trojan horse within the code according to her intension.

Of late, many researchers have shown their sincere efforts to make mobile agents more and more protected such as reference states [1], state appraisal mechanism[2], itinerary recording with replication [3], cryptographic traces [4], computing with encrypted functions [5], environmental key generation [6] to name a few.

But these approaches fail provide code confidentiality as they either address preventive measures or detective mechanisms. Obfuscation seems to be strong enough to opt it for code confidentiality. Obfuscation is a technique that makes the object code or data illegible without changing its functionality or meaning. It is widely used by the application developers to protect their algorithms to be copied by the rival developers.

Rest of the paper is organized as, section two throws light on some of the existing techniques for obfuscation. Section three explains the obfuscation techniques proposed in this paper. Section four gives complete design of the obfuscator. Section five shows experimental results and section six finally concludes the paper.

2. RELATED WORK
Mobile agents have additional capability of mobility over the links without losing the computation data along with the computing capabilities. This makes mobile agents superior than many other approaches for remote computation. One drawback that obstructs mobile agents to be used universally is assurance that mobile agents travels in the network and returns back to the manager intact. Numerous efforts have been made to make mobile agents resilient against manipulation in the functionality while travelling over the links.

Encryption looks to be working in this scenario but that incurs additional overhead at both the ends of communication. Other

techniques like police office model [7, 9] and to configure additional hardware [3, 9] are also attempts in the same direction but these are expensive and also restrict autonomy of mobile agent. Obfuscation comes out to better option that generates a program harder to understand without compromising with the functionality. Control flow obfuscation techniques [8] make transformations in the control flow of the program that is difficult to understand. Control aggregation obfuscations group the program statements in different way.

These include inline methods, outline methods, interleave methods and clone methods. Control ordering obfuscations alter the execution order of the statements. These include reverse loop counter and control flow flattening. Control computation obfuscations hide the real control flow in the program. These include insert dead code, remove common library calls and programming idioms, reducible to non-reducible code, parallelize code and loop transformation.

Some efforts are also made in the form of data obfuscations [8] in which transformations alter the data and data structures of the program. These obfuscations can be categorized as: Data storage and encoding obfuscations data storage affects the way data stored in memory and data encoding affects the way data is interpreted. These obfuscations include substitute a variable with code, promote scalar to objects, changing scope of the variable like local to global and changing the variable by an expression.

Data aggregation obfuscations alter the grouping of data. These include splitting the array into multiple arrays, merging the multiple arrays into one, fold the array and flatten the array. Similarly split the class into multiple classes, insert new bogus classes and merge variables. Data ordering obfuscations change the order of data. These include reorder arrays, reorder the variables and methods.

Some preventive transformations [8] aim not to obscure the code but to make it more difficult to break for the de-obfuscator. Target transformations try to make automatic de-obfuscation techniques more difficult. Inherent transformations try to explore known weaknesses in the de-obfuscators.

Undoubtedly obfuscator adds up the cost of computation and other parameters [10] that evaluate the quality of an obfuscation method. Potency defines to what degree the transformed code is more obscure than original code. Resilience defines how well the transformed code can resist with the automated de-obfuscation attacks. Stealth defines how well the obfuscated code mixes with the rest of the program. Cost is the execution time and space overhead in the obfuscated code as compared to the original code.

3. PROPOSED TECHNIQUES

Java programs are easy to reverse engineering as they are well defined. A malicious user may reverse engineer the mobile agent code and can understand and/or change the functionality. Obfuscation is the technique to confuse the attacker while understanding the code of the mobile agents. Obfuscation can be applied on source program before compilation and then letting obfuscated then compiled file travel over the network or on the other hand obfuscating compiled file i.e. byte codes. In this paper first kind of obfuscation is addressed. The technique proposed in this paper comprises of four steps to obfuscate a program. First step renames the identifiers to conceal the purpose associated with them, second addresses data structures like scalar variables and arrays and globalization of local variables, in third step all matrices are represented using sparse matrix representation. In fourth step flow of control is handled by interleaving the methods. Detailed obfuscator design is described in following subsections:

3.1 Hiding Variable Names

It is often seen that programmers name the variables according to their purpose e.g. ipAddress, hostname etc. Such variable names directly imply meaning that can ease the attacker to understand the motive of using the variables. Same venerability lies with functions as well. In the first step variable names and method names are identified in the program and a kind of crossover between them is performed. Crossover is a technique used in neural networks [11]. Slightly manipulated crossover is applied on the variable names and function names that lead a strong illusion for the attacker. Variable names may be purpose specific like ipAddress, hostname, etc. similarly function names may be like getIpAdd (), getHostname () etc. A crossover is performed between these names and these identifiers are exchanged amongst themselves such as variable ipAddress is replaced by getHostname, getIpAdd () is replaced by hostname () and so on. This kind of exchange leads different name with different purpose associated with it. Attacker will try to put her intelligence in that to get the idea of code segment according to the name assigned to it. Some dummy variables are also defined with the same strings identified in this step that have no significance with the functionality of the program. Names similar to the actual variables such as ipAdd, hostip, hostAdd etc can be defined in the program to make it more complex to understand the purpose of the variable.

3.2 Dimensionality Manipulation

Generally it is easier to understand scalar variables than a multidimensional array. In the second step the obfuscator converts all scalar variables into matrix like structure in slightly different manner. Any program has two types of variables viz. global and local. Initially all the variables are made global variables whether they are local or global (Fig 1).

A two dimensional array is defined with size decided by number of variables in the application. A separate global variable matrix is defined for each type of variables. Let there be n variables of integer type then an integer type matrix of size n*n is defined to store these variables at random places. Other locations are marked unnecessary and are fed with random values. Location information of actual variables is preserved by the obfuscator and is used throughout the program.

```
main ()                          main ()
{                                {
        int i;                           int i, j;
        .....                            .....
        func ()                          func ()
        {                                {
                int j;                                  ....
                .....                    }
        }                        }
}
```

Fig 1: Globalizing all variables

This scheme incurs surplus use of space. This drawback is overcome in the next section where two dimensional arrays are considered to be sparse matrices and a better representation scheme makes optimum use of space.

In case of one dimensional array a matrix folding technique [12] is used that folds the array from the middle and enhances dimensionality.

```
int D[10];                       int D1 [2.5];
for ( i = 0;  i <= 9; i++)       for (j = 0; j <= 1; j++)
D [i] = 2*D [i+1];               for (k = 0; k <=4; k++)
...                              if (k == 4)
...                              D1 [j, k] = 2*D1 [j+1, 0];
....                             else
                                 D1 [j, k] = 2*D1 [j, k+1];
```

Fig 2: Array folding

For arrays with dimensionality more than two can be flattened [12] to be two dimensional arrays. This is not done directly but first multidimensional array is flattened to linear array then array folding is applied.

```
int a [3][3][3];                 int a1 [27];
a [i][j][k] = ......;            a 1 [3(3 * i + j) +k] = ...;
```

Fig 3: Array flattening

This irritates the attacker while understanding the terminology of the program and makes a huge crowd of two dimensional arrays that is enough to vex the attacker.

3.3 Sparse Matrix Representation

The frustration is enhanced by converting the traditional representation of two or multidimensional arrays into sparse matrix representation [13]. Sparse matrix is a type of matrix in which most of the entries are zero. Thus traditional declaration of such matrices occupies a lot of space in storing zeroes that is not required. Sparse matrix representation provides better space utilization. Following figure shows a sparse matrix representation of a two dimensional array:

```
for (i = 0; i < n; i++)    for          k = 0;
(i = 0; i < n; i++) M[i][j] =           for (i = 0; i < n; i++)
......;                                 for (j = 0; j< m; j++)
                                        if (M[i ] [j] != 0)
                                        S[k] [0] = i;
                                        S[k] [1] = j;
                                        S[k] [2] = M[i] [j];
                                        k++:
```

Fig 4: Simple Matrix and Sparse matrix Representation

3.4 Function Infusion

Control flow reveals essential information of the program. Program method calls and definitions make it easier to understand the control flow of the program. By obscuring the methods and procedure calls the attacker can be confused to understand the actual flow of the code. In this step [14] an abstraction is introduced at method level of the program. Here all the method bodies and their parameters are integrated into one common method. Selection of the code to be executed can be made intelligently by adding a selection parameter in the argument list of the common method.

```
Class A                          Class A
{                                {
Method M1 (Type x)               Method M1 (Type x, Type
{                                y, Type d1, Type d2, Type
        S_M1.....;               d3)
}                                {
Method M1 (Type x)               D1 = d1++;
{                                D2 = d1;
        S_M1.....;               D3 = d2 * d1; d2 = d3*
}                                d1;   If(s (cond^n) || (D1 < =
}                                D2)  {
Method Call:                             S_M1.....;
A a;                             }
a.M1(x);                         If(s  (cond^n)  ||  (D1  >  D2
a.M2(y);                         &&                      D3)
                                 {

                                         S_M2.....;

                                 }
                                 }
                                 }
                                 Method          Call:
                                 A                   a;
                                 a.M1   (s_1   ,x,y,d1,d2,d3);
                                 a.M2(s_2 ,x,y,d1,d2,d3);
```

Fig 5: Infusing Methods

Moreover, for enhanced illusion some dummy variables are also introduced in the argument list and in the body of common method dead code, manipulating these dummy variables is inserted. The attacker would be hitting her head in understanding the actual flow of the code and the purpose of the dead code inserted. Following figure shows a demonstration of the technique discussed.

4. OBFUSCATOR DESIGN

Above described techniques can be implemented by parser that takes the program as input and obfuscates it and produces obfuscated program that has the same functionality with negligible overheads. This program is compiled and byte codes are sent.

To accomplish this, a huge and efficient parser is required which implements all four steps discussed above. We developed it in parts and tried to obfuscate the program in steps. We have shown sample codes and their obfuscated versions in the above section. The figure 6 shows complete design of the obfuscator. The AcceptInput () method accepts the un obfuscated program file as input and collects variable and method names. These variable and method identifiers are stored in an array named identifier. Second module generates a crossover point from which the rounded crossover between identifiers is applied. This practice exchanges the identifiers with other ones without changing their definition and purpose. The function LocalToGlobal () changes the scope of the local variables to global variables. In the next section three functions are applied simultaneously; first scalar variables are assigned to random location of a square matrix. Second, the functions FlattenArray () and FoldArray () changes other arrays into matrices. All these matrices are converted into sparse matrix representation in the function module ToSparseMatrix (). In the last step functions CollectArguements () and MarkBody () gathers argument list and body of each method in the program file F1. The function InfuseInOneFunction () makes a common big method and deploys all the arguments and bodies collected so far. After this complete obfuscated file is generated that functionally reflects the same behavior like the original program. I.

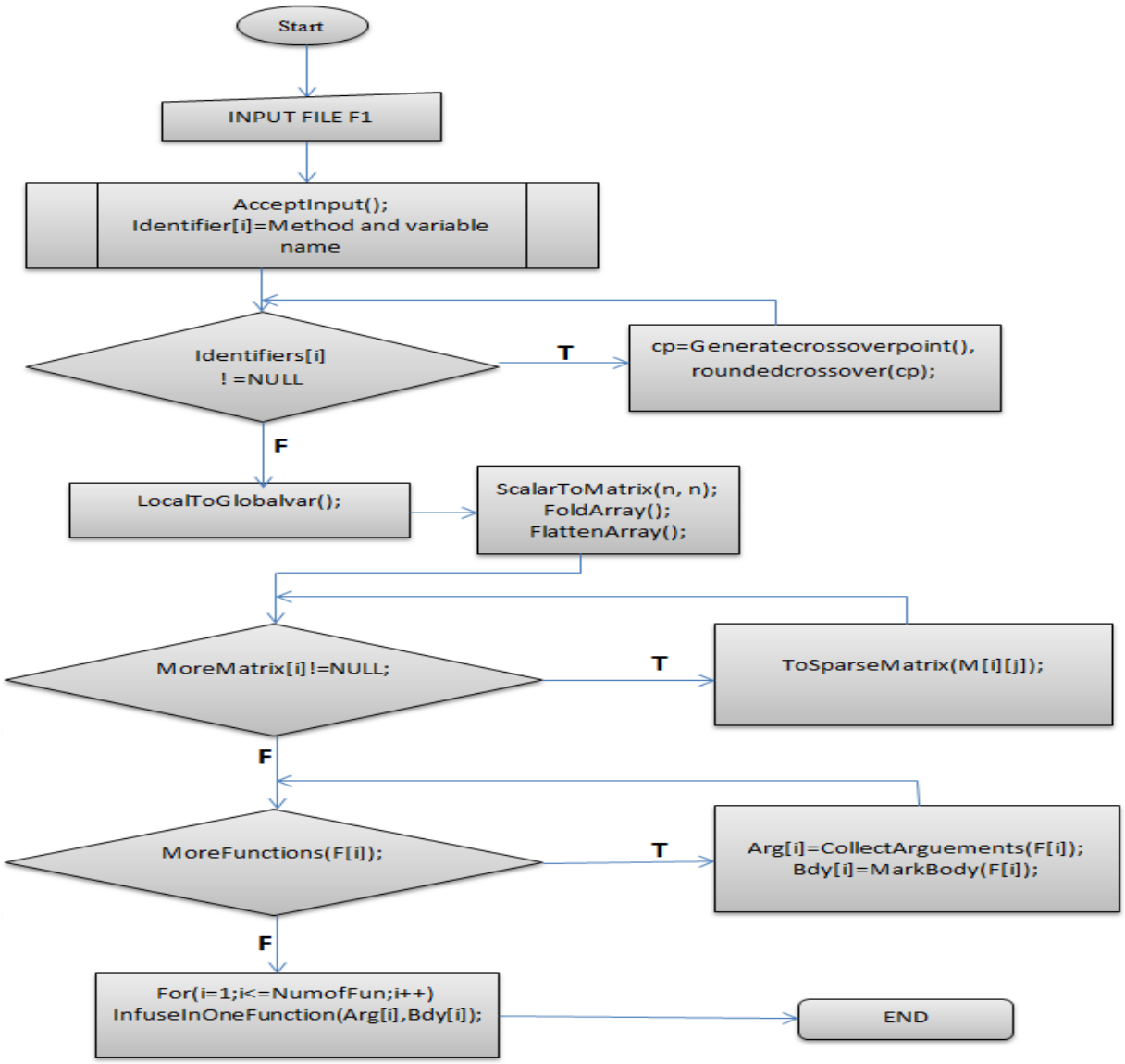

Fig 6: Complete Obfuscator Design

5. EXPERIMENTAL RESULTS

The way the agent code obfuscator was designed resulted in the obfuscated agent code being different every time. Also the output agent code appeared scrambled. However, the obfuscation process was evaluated based on the following metrics: Lines of Code (LOC), Complexity, and Reverse Engineering using Cavaj. The simplest way to measure the size of a program is to count the lines. This is the oldest and most widely used size metric.

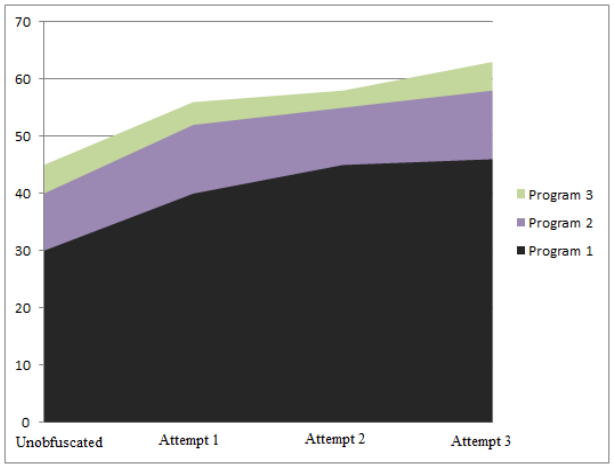

Fig 7: Variation in Lines of Codes

The larger the number of lines of code, the more it becomes difficult and time consuming to reverse-engineer a program. But adding more codes might also implies longer transmission times on network. Thus a compromise may have to be made to achieve agent code confidentiality. In each attempt of obfuscation the size of program slightly increases. Fig 7 shows the results of obfuscation of three programs in three attempts. Furthermore, complexity of program is another major issue. The obfuscator proposed in this paper changes flow of control by integrating all methods and aggregates their arguments and instruction bodies into one common method and differentiation between method call is made by an additional conditional statement. This additional conditional check adds some complexity overhead to the program but also strengthens security of program. But this exercise does not increase cyclomatric complexity of the program.

Many reverse engineer tools are available in the market for different language platforms like Cavaj is for Java. It is observed that by reverse engineered code of unobfuscated code is easily understandable. But in case of obfuscated code it is too difficult to understand flow of control and sparse matrix representation of variables as well as other data structures. Dead code inserted in the step of method infusion is not properly reverse engineered and produces errors while recompiled.

6. CONCLUSION

Code confidentiality is most important for the software's available openly in the market and for the agents travelling over the network. Many applications require agent to travel on the link and take decisions such as e-commerce or network management. Mobile Agent is one paradigm in which java code travels over the links with the capability to make changes at the receiving element. It has to be ensured that Mobile agents reach to the destination intact. Code obfuscation offers added security for a short period of time that is enough for the mobile agent to complete the task and move on to next node. In this paper we have opted for obfuscation of source code. There are many techniques available which obfuscate byte codes also. The combination of both, source code obfuscation and byte code obfuscation can be seen as future perspective for enhancement of this work.

7. REFERENCES

[1] Hohl, F. A Framework to Protect Mobile Agents by Using Reference States:Technical Report Nr. 2000/03, Universitt Stuttgart, 2000.

[2] Farmer, W., Guttman, J., and Swarup V.Security for Mobile agents Authentication and State Appraisal. Fourth European Symposium on Research in Computer Security, pages 118-130, 1996.

[3] W. Jansen and T. Karygiannis. NIST special publication 800-19 – mobile agent security, 2000, National Institute of Standards Technology.

[4] Sander, T., and Tschudin, C.F. Protecting Mobile Agents Against Malicious Hosts. Vigna G. (Ed.) Mobile Agents and Security. Springer-Verlag, 1997.

[5] Riordan, J., and Schneier, B. Environmental Key Generation Towards Clueless Agents. Vigna, G. (Ed.), Mobile Agents and Security, Springer-Verlag, 1998.

[6] Vigna, G. Cryptographic Traces for Mobile Agents. In Vigna, G. (Ed.): Mobile Agents and Security, pages 137-153, Springer-Verlag, 1998.

[7] Xudong Guan, Yiling Yang, Jinyuan You POM - A Mobile Agent Security Model against Malicious Hosts, Dept. of Computer Sci. & Eng., Shanghai Jiaotong University, 200030, China.

[8] C. Collberg, C. Thomborson, and D. Low "A taxonomy of obfuscating transformations", Technical Report 148,Department of Computer Science, University of Auckla, July 1997.

[9] Sandhya Armoogum and Asvin Caully "Obfuscation Techniques for Mobile Agent code confidentiality", March 2010.

[10] Arini Balakrishnan, Chloe Schulze "Code Obfuscation Literature Survey" , December 19th, 2005.

[11] David Beasley, David R.Bull, Ralph R.Martin "An Overview of Genetic Algorithms: Part 2, Research Topics", University Computing,1993.

[12] S.Praveen and P.SojanLal "Array based java source code obfuscation using classes with restructured arrays", 2008.

[13] Andrej Brodnik, Milena Kovaˇ, ˇSpela Malovrh "Optimal representation of sparse matrices, University of Ljubljana,1999.

[14] S. Rugaber, K. Stirewalt, and L. Wills, "The Interleaving Problem in Program Understanding," 2nd Working Conference on Reverse Engineering, Toronto, Ontario, Canada, pp. 166-175, July 14-16 1995.