Practical View of Mobile Agent Obfuscation in Digital Cash System

Mohamed AboRizka

Chairman of Ecommerce department
Arab academy for science and technology

Mohamed Kouta

Chairman of MIS department
Arab academy for science and technology

Hany Mohamed
Teaching Assistant, CS department
Shorouk academy

Abstract

Mobile agents play a great role in powerful digital cash systems. They can carry a coin that travels from one platform to another. Agents can be designed to learn and adapt to user needs and habits. Also, they can be able to negotiate with bank agents for deposit, withdraw and so on. According to their great role that they can play, so that they fire a lot of security issues. In this paper we will focus on hiding agent structure based on obfuscation techniques. We will discuss general idea of digital cash systems and their schemas. Also we will provide overview of old and new obfuscation techniques Finally we address implementation issues related to obfuscation algorithms and propose MAObfuscation (Mobile Agent) Tool for protecting mobile agent in digital cash system.

Keyword: Digital Cash, Mobile agent, aglet, BCEL, Security services.

1. Introduction

Digital Cash is a payment system which enables offline transaction without revealing the payers identity [1]. The drawbacks for paper cash exist; such that it can be stolen or lost and no one can compensate it; besides drawbacks of credit card that make people lose their privacy in their transactions. All these drawbacks make the necessity for more efficient electronic payment system is an essential fact. Digital Cash system offers a solution to these problems such that it enables securing user's transactions, protecting people privacy and working offline without the need to a third trusted party like a bank to complete its processing. Three types of transactions occurred in this system: Withdrawal, Payment and deposit Figure1 shows general structure of cash system with its transactions.

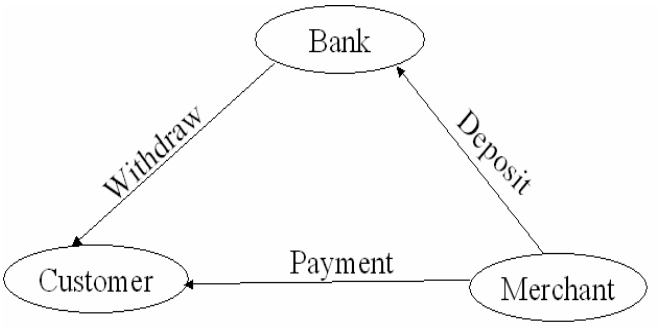


Figure 1 General structure of digital cash system

The Main characteristics of an ideal digital cash system can be summarized as security, anonymous, portability, two-way, transferability, offline and wide acceptability.

- Security: protecting the mobile agent during the transmission from one platform to another.
- Anonymous: hiding the mobile agent ownership to ensure privacy.
- Portability: no dependency on exact type of physical location.
- Two-way: peer to peer payments.
- Transferability: ability to spend coin just received without involving bank.
- Offline: working without involving a third trusted party.
- Wide acceptability: well known and accepted in a large zone.

Most of digital cash systems should handle critical problems if it is have a chance to succeed. These problems include double spending, anonymity, processing Time, tracing fraud, detecting criminal activity, and ensuring payment or delivery of goods. A lot of digital cash schemas are proposed to address these challenges. Some of these schema are Simple Anonymous Cash (Chaum-Fiat-Naor), Traceable anonymous Cash(Ferguson schema), Brands scheme, Digital coin scheme based on multi-agent system (MAS) which take our interest in this paper.

Referred to Secure Multi-Agent Based Digital Coin Scheme[2]; Each agent is designed to achieve its goals or services with guaranteed quality and performance, also agent can adapt dynamically and defend against unforeseen attacks on their integrity. The Following figure shows schema interactions between agents in digital cash system.

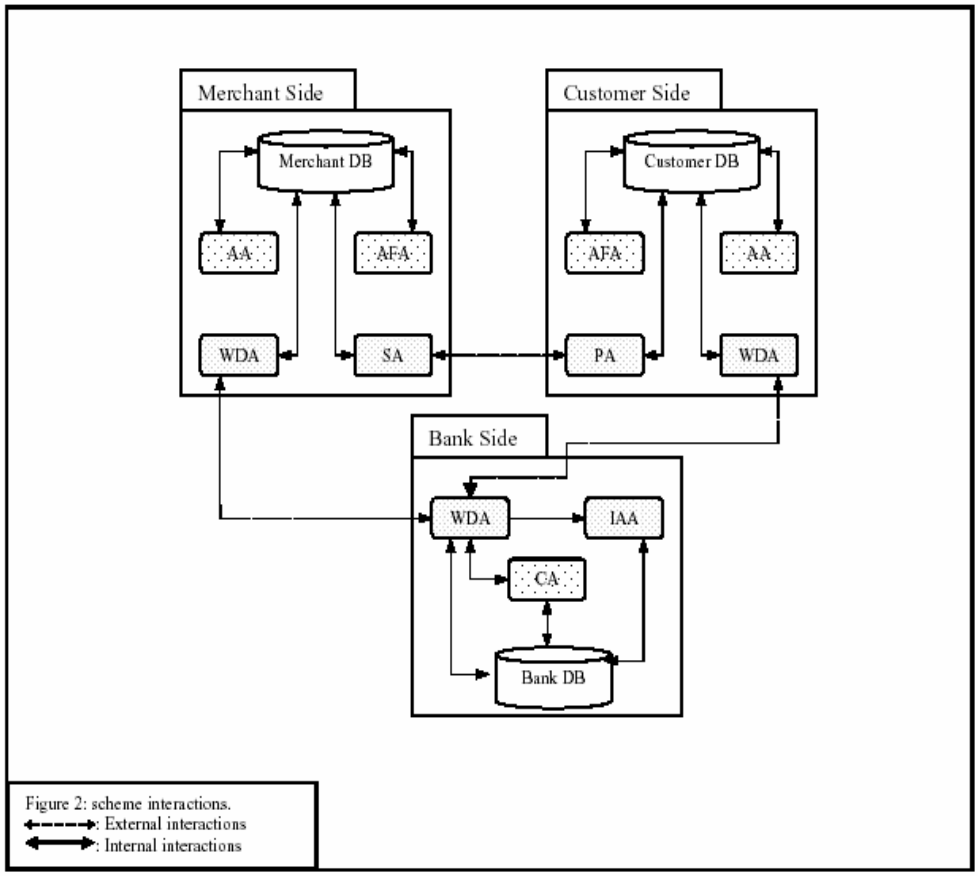


Figure 2: schema interaction between agents in digital cash system

The preceding proposed schema contains several types of agents which are distributed between the main parts of system which are customer, merchant and the bank. Each Agent in the schema plays an important role based on the special characteristics of agent (autonomous, reasoning, adaptable, proactive and so on).

Mobile agents such as payment, sales and coin agents have different security threats; Detection and analysis of agent structure become one of the most threats that can be made by the malicious host. This is our problem here in this paper. So protection the code from being analyzed and understood is our objective. We will build our work based on obfuscation technology. Obfuscation is a technique in which the mobile code producer enforces the security policy by applying a behavior-preserving transformation to the code before it sends it to run on different platforms that are trusted to various degrees. Consequently, the host should not be able

to modify the mobile code's behavior or expose sensitive information that is hidden inside the code such as a secret key, coin value or bank signature. Typically, the transformation procedure that is used to generate the obfuscated code aims to make the obfuscated code very hard to understand or analyze by host. This paper will provide overview of obfuscation techniques and the general structure of our system that is applied to mobile agent. Organization of paper is as follow; Section2 give us a survey of obfuscation techniques. We will describe implementation issues of our system in section3. Section4 shows us the testing case we take as input and the experimental results.

2 Obfuscation

2.1 current Protection Techniques

The most common protection techniques for software code against the mentioned attacks are protection by Server Side Execution, Hardware based Solution, Protection by encryption, protection through signed native code , Software Aging, Watermarking and code obfuscation[8]. Although encryption algorithms like to be useful approach that encrypts code before distribution to users. This approach has several drawbacks such as the difficulties that face architecture of runtime environment which is responsible for running encrypted application. The better is to transform application into one that is functionally equivalent to the original but is so difficult to be understood or decompiling by reverse engineering. No one can say that this technique can protect against attacks completely but we try using this methodology to reach maximum effort and time for reverse engineering. Example of obfuscator is a J-obfuscator that performs a lot of transformation until reaching a maximum cost for de-obfuscator. Another tool called J-Hide that support about 30 obfuscation algorithms.

Definition**:** Transform a program **P** into another program **P'** that is harder to reverse engineering with the same observable behavior. If P fails to terminate or terminates with an error, then **P'** fails to terminate or terminate with an error. Otherwise, P' must terminate and produce the same output as P.

2.2 Obfuscation evaluation Criteria

- Potency**:** It measures how much obscurity adds to the obfuscated code to be understood more than the original code.
- Resilience: it is evaluated by two measure

1. Programmer effort: amount of time required to construct automatic de-obfuscator.
2. De-obfuscator effort: execution time and space required by automatic de-obfuscator.

- Execution cost: how much time and space are added to obfuscated code.
- Quality: we can determine quality of transformed code according to previous measures:

$$Tqual(P) = (T_{pot}(p) , T_{res}(P), T_{cost}(P))$$

Where P is the input program.

2.3 Obfuscation Techniques

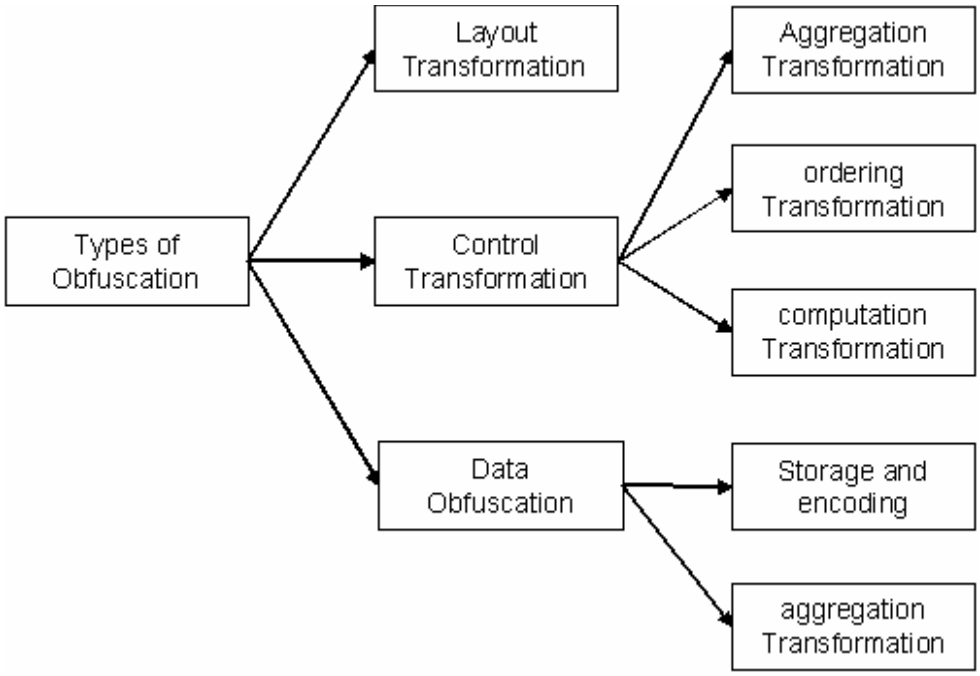


Figure 3: main obfuscation transformation functions

- Layout Transformation (obscuring the logic inherent in a program)
  1. Remove class information like comments, tabulation, carriage return and debugging information in java class files.
  2. Scramble identifiers: identifier beside method names declare its role in program such that reverse engineer could understand whole program easily.
- Control Transformation (make the control flow of a program difficult to understand).it consists of the following functions:

1. Aggregation transformation: breaks up operation that belong logically together or merge computation that Doesn't; ex: replace procedure calls with its code statements.
   a- Inline Method: replacing procedure calls with its block of code and remove the procedure.
   b. Outline Statements: put block of statements in a procedure then call it.
   c. Interleave Methods: it is so powerful to make a section of code has more than one purpose.
   d. Clone Methods: Applying different versions of a method to make examining of method structure and behavior more complex.
   e. Unroll Loops: replicates the body of loop more than one times.

2- Ordering Transformation: alter order of statements. Examples of these transformation:
   a- Reorder statement: changing the sequence of execution of instructions that each instruction doesn't depend on each other.
   b. Reorder Loop: changing the sequence of loops but changing should give the same results.
   c. Reorder expressions or blocks expression: breaking code statements with jump instructions make analysis of program by human more difficult.

3. Computation transformation**:** insert new statements that have no effect. Examples of these transformation:
   a Insert Dead or irrelevant code: The inserted dead code disguise the control flow of a method by adding an opaque predicate to the code which ensures that dead code is never executed.
   b Reducible graph to non-Reducible graph: constructing loop by multiple headers.
   c Extend Loop Condition**:** making the termination condition more complex by using true or false predicates that doesn't affect number of iterations.
   d Table interpretation: convert a section of code into a different virtual machine code. This new code is then executed by a virtual machine interpreter included with the obfuscated applications.
   e Basic Block Fission: break the loop into several loops keeping the iteration space same.
   f Intersecting Loop**.**

g Replacing Goto Function**:** replaces the byte code goto instructions with conditional instructions.
h Opaque predicates: a variable V is opaque if it has some property q which is known a priori to obfuscator but it is difficult to de obfuscator to detect its value.
i Add redundant operand.
j Parallelize code: parallelization code makes it more complex and more confused to reverse engineering.

- Data Obfuscation (modifying data fields of a program)

1. Storage & Encoding
   a Split variables**.** For example: splitting a variable V into k variable
   b Promote scalars to objects**:** Such as in java; converting from one of its data type to one of its wrapper classes.
   c Convert static data to procedure**:** For example instead of using static string, we can generate a procedure that produces that string.
   d Discrete algorithms to pack words**:** Fundamentals of discrete logarithm can be used to protect variables from dynamic analysis of memory references. Instead of separate words being used to store different variables, multiple variables can be packed into the same word so that the adversary is presented with a storm of events if he sets a break point on all references to a word during dynamic analysis.
   e Define cipher technique to encode string literals**:** for a simple example; consider a string name="casier", Let the obfuscation parameter chosen at random be 5. It means that each letter in string will be pushed forward by 5 letters.
   f Change Encoding**:** using composite functions to encode variables.
   g Change Variable lifetime**:** Ex: transform local variables to global variables.
   h Hide method Arguments: use composite functions to hide method arguments.
   i Reorder array**:** Ex: Replace index j in an array by f(j).
   j Array indexed transformation using Composite function.
   k Let I=f(i)= 2*i+3 be a function representing the new value of I. Let J=g(I)=f(2*i+3) be a function representing the new position of the ith element.
   l Method Argument transformation using composite function**.**
   m Hiding Constants using Composite functions**.**

2 Aggregation
   a Merge Scalar Variable: Merge two 32 bits variables into 64-bits variable.
   b Modify Inheritance Relations

- Extending the inheritance tree**:** According to Chidamber matrix, the complexity of class increases with its depth (distance from root) in the inheritance hierarchy.
- False Re-factoring**:** Factoring is a technique for re-structing object oriented programs.
- Restructure (Split, fold, merges) Arrays.

3. Implementation issues

Java is a jet fuel for mobile agents, its advanced technologies like Object Serialization and RMI (Remote method invocation)[20] enable us to develop easily Mobile agent platform specific to digital cash system. Mobile Agent is obfuscated before dispatched from source platform to different destination platform using Obfuscation Techniques. It was observed that the most suitable way to create obfuscated code is by working in the byte code level that requires well understanding of java byte code specifications and java class format. Using BCEL (Byte code Engineering Library) saves from us a lot of Time. BCEL is intended to give us a convenient possibility to analyze, create, and manipulate (binary) Java class files (those ending with .class). Classes are represented by objects which contain all the symbolic information of the given class: methods, fields and byte code instructions. Such objects can be read from an existing file, transformed by a program (e.g. a class loader at run-time) and dumped to a file again. An even more interesting application is the creation of classes from scratch at run-time. The BCEL may be also useful if we want to learn about the Java Virtual Machine (JVM) and the format of Java class files. BCEL contains also a byte code verifier named Just-Ice, which usually gives us much better information about what's wrong with our code than the standard JVM message. The latest feature is very useful in our system that help us to verify agent functions after applying obfuscation transformations

In the current version of MAObfuscation system we apply about 25 techniques with new novel obfuscation methods in [13]. The system is implemented and evaluated in an execution environment that has several machine with the same capability: Pentium 4 CPU 3.06GHz, 512 of Ram and windows XP 2 as the operation system. System developing is made up of the following steps:

- Develop mobile agent platform responsible for performing agent activities like creating, dispatching, dispose and so on.

- Develop a coin agent class which has the ability of diving itself into sub coins, able to transfer itself to other platform and so on.
- Develop obfuscation package which has implantation of several old and new obfuscation techniques.
- Apply obfuscation functions for agent before dispatch.
- Developing methods for verifying agent functions after executing agent transformation functions.

We take the dispatch method as a sample of our work; it consists of the following steps:

1- Take the selected coin agent that is had to be transferred to another platform.
2- Get object of java class file for selected coin agent; this object has all information of selected agent which is java class. This information like class name, methods, fields and so on.
3- Apply obfuscation transformation for resulted java class object from previous step.
4- Send the obfuscated object to another platform.
5- Send object of selected coin agent that represent the current state of agent like coin value.
6- Remove the coin agent from the local platform.

Here we will describe the steps of main obfuscation algorithm:

1- Take object of java class file of selected agent.
2- Get random r from 1 to 25(number of implemented obfuscation algorithms).
3- Select the obfuscation transformation based on r value.
4- Apply the selected transformation on java class object.
5- Verify agent functions after transformation; if verified go next step; Else go to step 2.
6- repeat the previous steps in n of times; take care the new value of r shouldn't be selected before.

We will list now pseudocode of some of selected transformations; most of these methods take object of ClassGen class(template class for building up a java class) as a parameter:

Here we will list the steps of renameclassMemebers method:

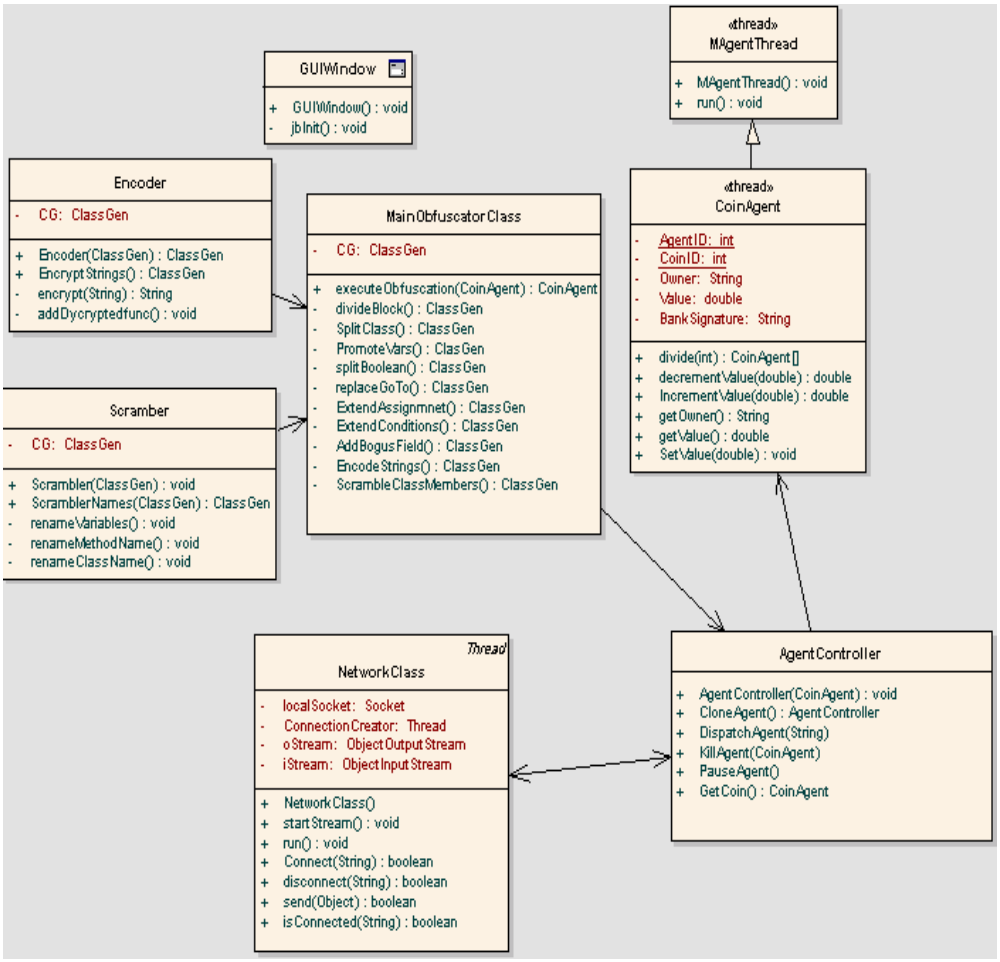
1- Get a list of all class fields from class gen object .
2- Take first field, search its constant name in constant pool.
3- Get index of constant from constant pool table.
4- Get object of ConstantUtf8 for selected constant.
5- Build a new name for selected field.
6- Replace old constant name with the new name
7- Repeat steps from 2 to 6 on each field in list resulted from step1.
8- Save and changes to input class.

Note: The previous steps work also with renaming methods

Steps of encrypting strings referenced in agent program are listed in the following table

1- Get a list of all class fields from class Gen object.
2- Get a list of class methods from Class Gen object.
3- Select first field in generated list of fields from step1.
4- Check type of field, if it is string go to step 4; else go to step.
5- Select first method in generated list from step2.
6- Search in all method instructions for instruction that try to access the selected field.
7- Check for each instruction if it is store or get instruction.
8- If it is store instruction; search for a constant that is used for changing field value and then encrypt it based on any encryption algorithm like Cesier algorithm.
9- Loop steps from 6 to 8 for each method exist in generated method list.
10- Loop steps from 4 to 9 for each field.
11- Save and changes to input class.

A sample view of our system model that is designed by enterprise architecture tool is illustrated in figure4.



. Figure 4: a general view of component diagram of our system

In the next section testing case shows us the effect of obfuscation on mobile agent.

4. Testing Case

We take the coin agent (responsible for creating coins for both customer and merchant) as an example for our work. We test our work by running two trusted platform that try to communicate together and a malicious host that try to access and reverse engineer coin agent code. Before sender platform apply obfuscation, a malicious host try to access agent structure and dump the transferred agent code. Byte code can be converted to java source by any de-compiler. a sample of java coin agent resulted from JDecompiler after decompile the byte code of transferred agent is displayed in Table 1 A .Table1 B shows us sample view of obfuscated coin agent.

```
class CoinAgent {
   private static final int AgentID = 34233;

   private static final int CoinID = 567498;

   private String Owner = "rwer34xcv34";

   private int value = 50;

   private String BankSignature = "HSBC Bank";

   public CoinAgent()
   {
   }

   public void divide(int dividedvalue) {
      for (int i = 0; i < (int)this.getValue() /
dividedvalue; i++) {
         CoinAgent newCoin = new CoinAgent();
         newCoin.setValue(dividedvalue);
      }
   }
   public void decrementValue(int decValue) {
      this.setValue(this.value - decValue);
   }

   public void incrementValue(int incValue) {
      this.setValue(this.value - incValue);
   }
   public String getOwner() {
      return this.Owner;
   }
   public long getValue() {
      return this.value;
   }
   public void setValue(int value) {
      this.value = value;
   }
```

A

```
class CoinAgent {
   private static final int YRJ = 34233;
   private static final int QBUI = 0x8a8ca;
   private String ZMXOU;
   private int SED;
   private String DOKIJ;
   private int HPDHR;
   private String WMJTZ;
   void LBXKQ() {
      ZMXOU = "54wdy54sfxs";
      SED = 50;
      DOKIJ = "lobC!DCTI";}
   public CoinAgent() {
      LBXKQ();          }
   public void JWPXJ(int dividedvalue) {
      int i;int E;int P;E = 0;
      i = 0 + ((P = 0) * P * 7) % 4;
      goto _L1
_L3:  CoinAgent newCoin = new CoinAgent();
      newCoin.RCU(dividedvalue);
      i++;
_L1:  i=(int)MGB() / dividedvalue;
      if (E * (E - 1) * (E - 2) != 0) goto _L3; else goto
_L2
_L2:  JVM INSTR icmplt 22;
      goto _L4
_L4:          }
   public void NHYRO(int decValue) {
      RCU(SED - decValue);       }
   public void WZEH(int incValue) {
     RCU(SED - incValue); }
   public String RMZMQ() {
      return JVK(ZMXOU);   }
   public long MGB() {
      return (long)SED;  }
   public void RCU(int SED) {
      this.SED = SED;    }
```

B

Table 1: sample of coin agent code before and after obfuscation

There exist a big difference between two versions of agent code; coin agent code appears in Table 1 B is so difficult to understand and reverse engineer.

In the previous example, a lot of transformations are applied:

- Rename class member: AgentID member variable is converted to another random name YRJ, also BankSignature member name is transferred to DOKIJ.
- Rename methods: setVAlue method name is transferred to RCU.
- Encrypt strings: The BankSignature string value which is "HSBCBank" is encrypted to "lobC!DCTI".
- Add bogus fields: WMJTZ string field is added to obfuscated code.
- And so on.

5. Conclusion and Future Work

Securing mobile agent is a critical challenge for digital cash based agent system designer due to its great role and its characteristics that require form us a lot of work. By using Obfuscation we prove that it is a useful way to protect mobile agents from interception and hacking. Although obfuscation mayn't look to be 100% secure algorithm but it can work as the first defense against Mobil Agent attacks. Our future work can be summarized into five steps

- Developing more advanced Obfuscation techniques.
- Applying all old and new obfuscation techniques in proposed digital cash system.
- Authenticate mobile agent using software watermarking.
- Assure for mobile agent integrity using tamper proof algorithms.
- Apply the three previous secure software layers to Mobile agents.

6. Acknowledgement

We would like to thank shaima toriah and sherif saad for their contribution in editing this paper and their efficient comments.

7. Reference

[1] Madana Jahanin Farsi, "Digital Cash" , Master Thesis, computer science, Department of mathematics and computing science, Goteborg university,1997.

[2] Mohammed Aborizka, Mohamed Kouta, Krishna Kavi, Sherif Saad, Mostafa Salama, “Secure Multi-Agent Based Digital Coin Scheme” 30th IEEE Annual International Computer Software and Applications Conference (COMPSAC), International Workshop on Engineering Semantic Agent Systems (ESAS 2006), Chicago, USA, September 17-21, 2OO6.

[3] Danny B.lang and Mitsuru Oshaima, "Mobile agent with java: The Aglet API", IBM Tokyo Research Laboratory,Yamato-shi, Japan,1999.

[4] Mousa Alfalayleh and Ljiljana Brankovic, "An Overview of Security Issues and Techniques in Mobile Agent", Eighth IFIP TC-6 TC-11Conference on Communications and Multimedia Security, University of NewCatle in September, 2004 .
[5] Nilson Minar, "Computational Media for Mobile Agents",1996.
[6] Jukka Ylitalo, "Secure Platforms for Mobile Agents", Department of Computer Science, Helsinki University of Technology , 2000-01-06.
[7] Wayne Jansen, Tom Karygiannis, " Mobile Agent Security " , NIST Special Publication by National Institute of Standards and Technology, Computer Security Division, Gaithersburg, 1999.
[8] Christian S. Collberg, Clark Thomborson, "Watermarking, Tamper-Proofing, and Obfuscation–Tools for Software Protection", IEEE Conference, 2002.
[9] Douglas Law , "Java Control Obfuscation", Master Thesis, Auckland university, 1998.
[10] Gael Hachez, "A Comparative Study of Software Protection Tools Suited for E-Commerce with Contributions to Software Watermarking and Smart Cards", 2003.
[11] Gregory Wroblewski, "General Method of code obfuscation", 2002.
[12] Francies Mautor ,"Reverse Engineering – Tools and techniques for obfuscating", Francis Mawutor Kugblenu", Blekinge Institue of Technology, MSc Computer Science,2006.
[13] Levent Ertaul Suma Venkatesh,"Novel Obfuscation Algorithms for Software Security", Department of Mathematics&Computer Scince , California Department,2005 .
[14] Christian Collberg Clark Thomborson, Douglas Low, "Breaking Abstractions and un-structuring Data Structure", Department of Computer Science, The University of Auckland, Auckland, New Zealand, 1998.
[15] Christian Collberg, Clark Thomborson, and Douglas Low,"ManuFacturing Cheap, Resilient, and stealthy opaque construct", Department of Computer Science, The university of Auckland, New Zealand,1998.
[16] Christian Collberg, Clark Thomborson, and Douglas Low, A taxonomy of obfuscation transformation, Department of Computer Science, The university of Auckland, New Zealand, 1997.
[17] Hou, H.Y.Chen and M.H.Tsai, "Three control flow obfuscation methods for Java software",IEEE Proceedings - Software -- April 2006 -- Volume 153, Issue 2, p. 80-86 .
[18] Levent Ertaul and Suma Venkatesh , "JHide- A tool Kit for code obfuscation", Cambridge university, 2004.
[19] Honggying Lai, "A comparative study of java obfuscator available in internet", Computer Science Department, university of Auckland, February 22,2001.
[20] Christian Nester, Michael Philippsen, Bernhard Haumacher, "A More Efficient RMI for Java", Java Grande, 1999.