

SOFTWARE WATERMARKING VIA ASSEMBLY CODE TRANSFORMATIONS

A Thesis

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Masters of Science

by

Smita Thaker

May 2004

© 2004

Smita Thaker

ALL RIGHTS RESERVED

APPROVED FOR THE DEPARTMENT OF COMPUTER
SCIENCE

Dr. mark Stamp

Dr. Chris Pollett

Dr. Dave Blockus

APPROVED FOR THE UNIVERSITY

ABSTRACT

SOFTWARE WATERMARKING VIA ASSEMBLY CODE TRANSFORMATIONS

By Smita Thaker

By some accounts, piracy costs the software industry in excess of \$10 billion annually [10]. One possible defense against piracy is to add a watermark to software. While this will not prevent all piracy, a robust watermark would make it possible to determine the source of pirated software, and hence would likely discourage a significant amount of piracy.

To date, it has been difficult to design digital watermarking schemes that can function well in a hostile environment. Most watermarking techniques are easily removed or distorted beyond recognition. For example, “stirmark” will effectively remove a watermark from a digital image; see [12] for a discussion.

This project, presents a watermarking scheme based on modifications to software at the level of assembly code. These modifications result in multiple versions of a given piece of software, with each version having a different code sequence, but with all versions being functionally equivalent.

Our watermarking approach was inspired by metamorphic computer viruses. Metamorphic viruses are transformed into distinct code each time they replicate. This makes detection far more difficult, since there is no constant signature for virus scanning software to detect. In the context of watermarking, metamorphism allows the watermark to be uniquely embedded in each instance of the software. While it is still possible to

remove (or obscure) a watermark in a given instance of the software, the variability of the code makes it extremely difficult to automatically remove watermarks from a significant number of instances of the code. Consequently, watermark removal will remain a manual and labor-intensive process.

Metamorphic code transformations have also been discussed as a method for increasing software diversity. In an analogy to the genetic diversity of biological systems, it is argued that increased software diversity can limit the effects of malicious attacks on computing systems [12]. A side benefit of our approach to software watermarking is that it results in diverse (or metamorphic) software, and hence may increase the resistance of such software to many types of attacks.

TABLE OF CONTENTS

1	Introduction.....	7
2	Requirements	9
3	Potential Applications:	11
4	Assembly language programming - Background information	13
4.1	Inside the CPU.....	13
4.1.1	General Purpose Registers	14
4.1.2	Segment Registers	15
4.1.3	Special Purpose Registers	16
4.2	8086 Instruction Set:.....	16
5	Design.....	18
5.1	Part A: Inserting the watermark into the code.....	18
5.2	Part B: Read Watermark.....	21
6	Implementation	23
6.1	Setting up the environment / Tools used.....	23
6.2	Implementation Details.....	23
6.2.1	Inserting the watermark in the code.....	25
6.2.2	Reading the watermark from the executable:.....	35
7	Code Transformations.....	39
7.1	Inserting Jump statements - JMPs.....	39
7.2	Adding redundant labels:	40
7.3	Using arithmetic and logical instructions for transformation.....	41
7.4	Inserting NOPs	43
7.5	Using PUSH and POP instructions.....	43
8	Challenges.....	45
9	Deployment.....	46
10	Summary.....	49
11	Directory Structure.....	50
12	Conclusion.....	52
13	References:.....	53
14	Appendices.....	54
14.1	runproj.cmd.....	54
14.2	properties.txt.....	55
14.3	keyTranx.txt.....	56
14.4	keyPattern.txt.....	57
14.5	CreateTranx0.java.....	58
14.6	ReadWM.java.....	59
14.7	WriteWM.java.....	63

1 Introduction

Software security is constantly compromised by attacks from hackers. Typically, hackers spend time and resources to tamper with the code to discover vulnerabilities then exploit these to carry out an attack. Such attacks, when conducted in a distributed fashion across a network, severely disrupt the functioning of corporations, online service providers, government agencies, etc.

One possible defense against hackers is to have multiple copies of software which are functionally equivalent but where each copy is distinct (Software Diversity). In this case, an attack is likely to be effective against only one instance (or a small fraction of instances) of the software. Some tools and techniques have been devised for this purpose. Moreover, these tools modify software at the source code level. We believe that transformations at the assembly code level are more robust and secure.

Another use of code transformations is for software watermarking. In this research project, we intend to focus primarily on this particular application of such transformations. I propose to make the code transformations at the assembly level. This creates unique, but functionally equivalent instances of a given piece of software. The aim of this project is to effectively insert a watermark in assembly code software and retrieve the watermark from the executable. This watermark carries customer-specific information. We will be working on assembly language 80x86 as it is widely used by several of the current processors.

The project has two principal goals.

1. To produce multiple copies of a given piece of software, making an attacker's job more difficult and thereby reducing the potential damage in the event of an attack.
2. To provide anti-piracy protection by embedding a customer's information in each instance of the software. If pirated versions of the software are found in the market, we can retrieve the watermark from its code and determine which legitimate customer originally received this copy of the software. As a side benefit, the reverse-engineering problem is made more difficult and this in turn makes the software more secure. We must make it difficult for an attacker to identify the actual location of the watermark in the software. And even if an attacker does succeed in identifying the watermark, tampering with it should be a challenge, since he has to work at a very low level. In short, the task of modifying the watermark and having the resulting code to function correctly is a challenging task.

Definition

Watermarking is a technique of embedding some special mark in an object to use it as identification. This project addresses software watermarking, i.e., embedding some special pieces of code in software so as to uniquely identify software. These special pieces of code (transformations) serve as a watermark and can carry any special information we wish to embed.

2 Requirements

The project has the following goals.

1. **Watermark** – To hide certain customer-specific information within each copy of software. To be able to successfully read the watermark back to find out at any point of time, who the actual owner of the software copy is.

If we always embed, say, the customer-id in the form of a watermark into the software we are selling, we will be able to track down the owner of any software instance. The purpose of this feature is to discourage software pirates and reduce the industry losses.

2. **Software Diversity** - To produce multiple diverse copies of a given piece of software making it harder for an attacker to attack a particular software weakness and exploit it.

Hackers generally study software looking for a bug or weakness and thereafter use this weakness to carry out malicious attacks. When searching for potential victims, attacker generally tend to look for software with a known weakness. Now if we can introduce software diversity, then although all our software instances will be functionally the same, they will differ in their lines of code. So, the hackers' pattern matching criteria will work only for one instance (or a very few limited instances). His line of attacking will fail against the majority of software instances providing a higher level of protection and a major reduction in damages.

3. **Robustness** – To make the watermark robust enough to survive an optimizing compiler.

An optimizing compiler tends to remove pieces of code it determines are unnecessary. Since the watermarked lines of code have been added into the actual software code, there is a possibility that the compiler will filter out the watermark on compilation. Our job is to make sure that the transformations we select to carry our watermark can survive this optimization.

4. **Reliability** - To ensure that the project application returns the correct watermark each time.

In spite of the loss of watermarking information by the optimizer, we should always be able to obtain correct and accurate results from the project. If the results obtained are incorrect we may end up incriminating a totally innocent customer to whom the customer-id returned belongs to. On the other hand, if the project returns partial customer-id results (say some bits of the customer-id are missing) we will be unable to decide if the optimizer is to blame or it's some smart reverse engineering job. Even slight loss of reliability can jeopardize the utility of the software. Hence, the project application needs to be virtually 100% reliable.

5. **Obscurity** – To introduce obscurity in the resulting files so as to make the job of reverse-engineering more challenging.

Reverse engineering is a process of breaking down a software to understand how it works so as to copy it, tamper with it or maybe enhance it. In an application with software diversity, the reverse engineering process involves

comparing multiple instances of the software and trying to determine where and how the watermark has been inserted. The introduction of real and dummy watermark transformations into the code should be done so that even after comparing several different instances of a software, a hacker cannot extract any information regarding the watermark location or the watermark application techniques.

6. **Security and Tamper Proofing** – Tampering with the watermark should be a challenging job. Also, in the event of code tampering, the application should be able to detect it.

Software tampering is a by product of successful reverse-engineering. With respect to this project, the most obvious task a hacker might carry out on successful reverse-engineering is to tamper with our inserted watermark. If he can figure out the embedded lines of code in the software, he can determine the location of some of these watermark transformation. His next interest would be to tamper with this watermarking by changing it or removing it, although it will be a manual and labor-intensive process. Our aim is to make the application as secure and tamper-resistant as possible. Moreover, if inspite of all this protection someone still manages to tamper with the watermark, we should have some mechanism to be able to detect that the watermark has been tampered with.

3 Potential Applications:

Some of the potential applications of this application are listed below:

1. Invisible Watermark:

In the words of Isenberg [6], "Digital watermarking, allows copyright owners to incorporate into their work identifying information invisible to the human eye. When combined with new tracking services offered by some of the same companies that provide the watermarking technology, copyright owners can, in theory, find all illegal copies of their photos, music, documents, applications, etc. on the Internet and take appropriate legal action". It thus, protects webmasters against the dangers of copyright infringement. Insertion of an invisible watermark in our application, gives us some ability to track the source of a stolen document [11].

2. Software Diversity:

Is a technique by which we can produce multiple copies of software which are functionally equivalent but where each copy is unique. Software Diversity does not directly provide protection against reverse-engineering. But it creates a system where an attacker may be able to break one particular instance of the software without necessarily breaking the entire system [11].

3. Protection against virus:

A virus, is a computer program that targets a known vulnerability to create havoc on the computer. It scans files, looking for some vulnerable pattern or it marks the entry-location in the victim file [9]. In our case, after undergoing transformations at the assembly level, the application-code becomes highly polymorphic. Now, the virus may be able to search for a pattern or mark a point-of-infection-entry in one particular instance of the software but cannot carry out the same procedure for all the other instances. The virus

writer will need to customize his virus, for different instances of the software for it to work. Thus, this provides a partial protection against viruses [9].

4 Assembly language programming - Background information

As already mentioned, we will be working on assembly language 80x86, which is widely used by several current processors. Here we provide some background information on this programming language, which will help us understand the transformations we have picked and the code into which we will insert our watermarking transformations.

4.1 Inside the CPU

Intel started the 80x86 family in 1981 with 8086 processors, followed by Pentium in 1994. Each of the processors are backward compatible but each one is faster than its predecessor and has more features [5]. They support assembly language 8086.

Following is a picture of the internals of a CPU (Central processing unit) also called a microprocessor. General purpose registers, segment registers, special purpose registers and ALU (Arithmetic and Logical Unit) are among some of its key components.

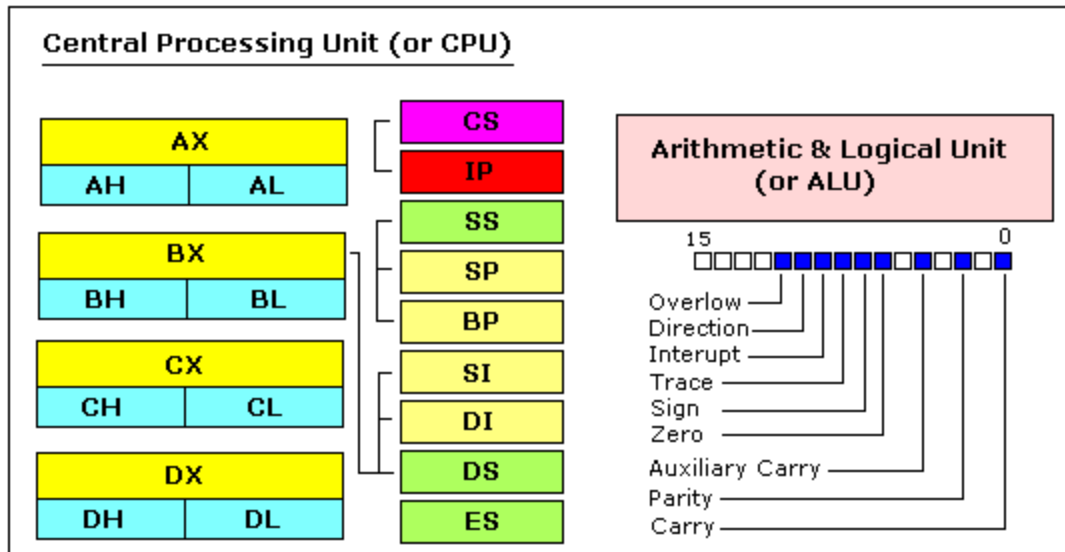


Figure 1: CPU Internals
 Source: EMU tutorials [4]

Registers are used to carry out operations, and temporarily store the results of such operations. However, registers are limited in number and small in size, and must be used judiciously. It's much faster to access these registers than accessing the memory locations, as registers are located within the CPU.

4.1.1 General Purpose Registers

The 8086 CPU has 8 general-purpose registers [4]:

- **AX, BX, CX, DX** - these are 16-bit data registers each divided into a higher and a lower 8-bit each registers (e.g. AH/AL where H denotes the higher byte and L denotes the lower byte)
- **SP** - is a 16-bit stack pointer register.
- **BP** - is a 16-bit backup stack pointer.

Both SP and BP are stack registers. Stack is a special memory location onto which we can save values and retrieve them back. Since 6 basic registers used for most operations are not sufficient, we use the stack for additional flexibility.

There are 3 index registers (SI, DI and IP) used primarily for string instruction.

- **SI** (Source index)- is a 16-bit source index register.
- **DI** (Destination index)- is a 16-bit destination index register.

4.1.2 Segment Registers

To access 1 MB of memory, 20-bits of information is required, while the registers are all 16-bit. So, the concept of segment offset pairs was introduced. When segment and offset registers are used together, it gives a 20-bit address [4].

$$\text{Offset} = \text{Segment} * 16$$

Following are the segment registers.

- **DS** - data segment register, points to the segment, which contains the variable definitions.
- **CS** - code segment register, points to the segment, which contains the current program being executed.
- **SS** - stack segment register, points to the segment, which contains the stack.
- **ES** - extra segment register

The segment registers DS, CS, SS and ES are used to point to various segments in memory. These registers pair up with certain general-purpose registers to access specific memory locations [4].

4.1.3 Special Purpose Registers

IP - is the instruction pointer register. It stores the address of the instruction to be executed.

Flags Register - keep track of the CPU state and changes on its own when mathematical operations are carried out. Based on the values set in flag registers, the program control is transferred to different locations in the program, at certain decision points.

4.2 8086 Instruction Set:

Some of the basic 8086 instructions we will be using are as follows:

Data Transfer Instruction	
MOV	Move byte or word to register or memory
IN, OUT	Input byte or word from port, output word to port
LEA	load effective address
PUSH, POP	Push word onto stack, pop word off stack
Logical Instructions	
NOT	Logical NOT of byte or word (one's complement)
AND	Logical AND of byte or word
OR	Logical OR of byte or word
XOR	Logical exclusive-OR of byte or word
Arithmetic Instructions	
ADD, SUB	Add, subtract byte or word
INC, DEC	Increment, decrement byte or word
NEG	Negate byte or word (two's complement)
MUL, DIV	Multiply, divide byte or word (unsigned)

Transfer Instructions	
JMP	Unconditional jump
JE/JNE	Jump if equal/Jump if not equal
Loop control	
LOOP	Loop unconditional, count in CX, short jump to target address
Subroutine and Interrupt Instructions	
CALL, RET	Call, return from procedure
Inactive states:	
NOP	no operation

Table 1 : 8086 Instruction Set
Source : 8086 Instruction Set [1]

- Some instructions are made up of 3 elements:
 1. Name of the instruction
 2. Operand 1 followed by a comma
 3. Operand 2.

Eg. Mov eax, 09
- Several other instruction are made up of only 2 elements:
 1. Name of the instruction
 2. Operand

e.g. push eax.

The operand can take 3 different values. It can be

- value - like a number '45'
- register value – like eax

- memory location – refers to the contents at say memory location (address) “2000”

As you will see later on in the implementation, each assembler has its own way of writing assembly instructions. E.g. During watermark write, we will see instructions like `movl $23, %eax` (Case (i)), which means move the value 23 to the register `eax`.

However when an assembly code is generated by a disassembler such as IDA Pro, the same thing reads as `mov eax, 17h` (Case (ii)).

Note the change in the instruction-name, the position of the register and number, and also the number (value) itself. In case (i) it is 23 in decimal whereas in case (ii) it is written in hexadecimal and so the value becomes 17. The letter h following 17 denotes hexadecimal.

5 Design

5.1 Part A: Inserting the watermark into the code

- **Step1:** Compile the source code to generate an assembly.

We begin with the source code of the software and compile it to obtain an assembly file, because our watermarking scheme is designed for the assembly level insertions.

- **Step2:** Apply some scheme to the list of transformations and the customer-id to create a customized transformation-set. Embed this customized transformation set at certain locations in the software assembly file. These act as the watermark and carry customer related information.

We generate a transformation file which has the list of all possible transformations. We take the customer-id apply it to this list of transformations to come up with a customized list of watermarking transformation. These customized watermarking transformation are created specifically for that particular customer-id only. We embed this watermark into our assembly file from step 1. The schemes used for this purpose are discussed in detail in the next section, Implementation.

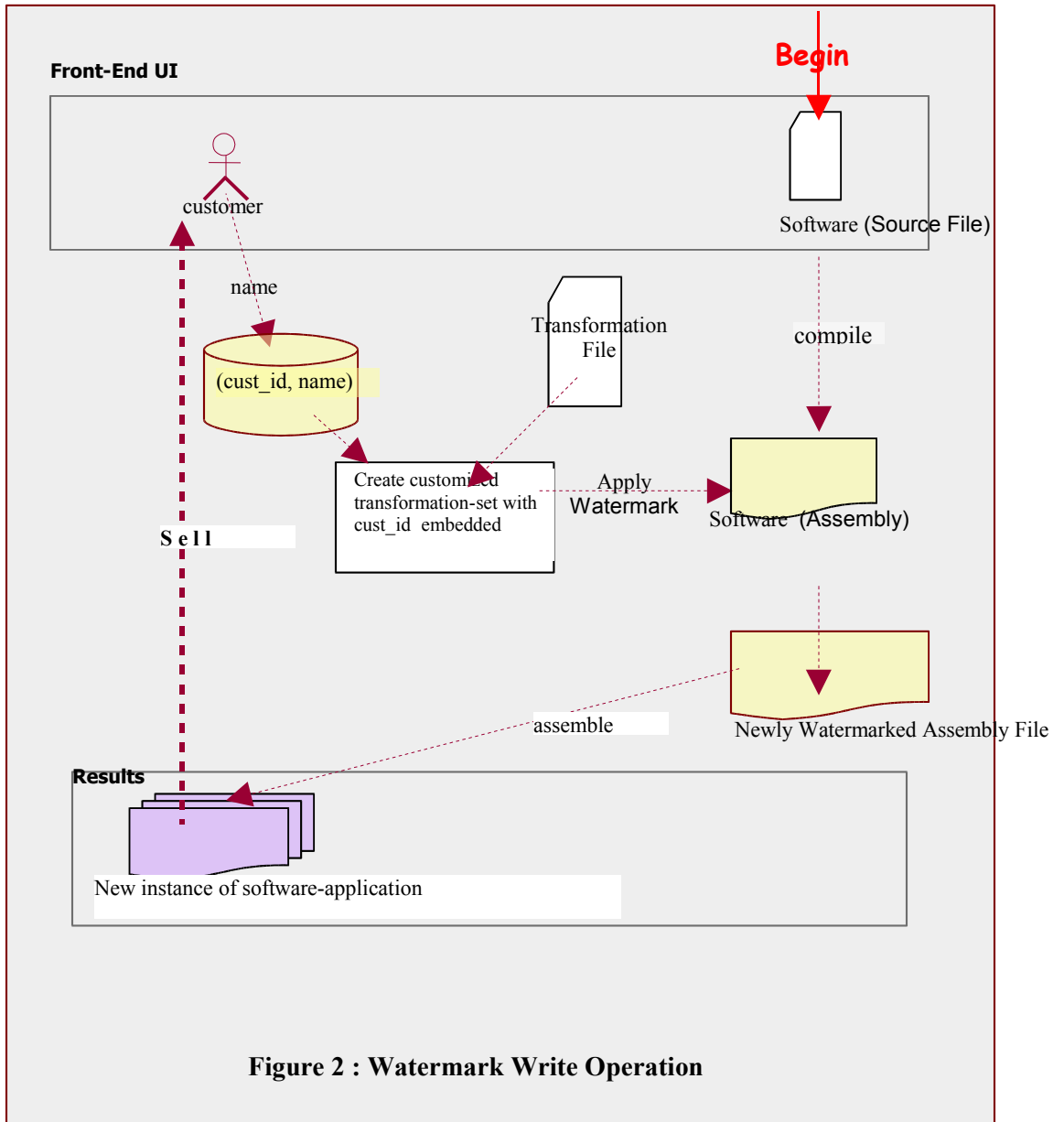
- **Step3:** Generate an executable from the assembly code.

Now, that we have embedded the watermark, we generate an executable.

- **Step4:** Save information regarding the customer (id, name) into the database.

We will not be focusing on this aspect in this project.

- **Step5:** Sell this software executable to the customer whose ID is embedded.



5.2 Part B: Read Watermark

- **Step1:** Disassemble the software executable to generate an assembly.

Now to retrieve the watermark we need to get back to the assembly level again. We have to disassemble the executable using disassembler IDA Pro and save the results into assembly format.

- **Step2:** Scan it for special patterns and attempt to extract all possible embedded data from it.

The assembly syntax used by our assembler is different than that used by our disassembler. Hence, the watermarking transformations we embedded during the write procedure get translated to slightly different lines of transformation-code. These are the special patterns we search for in the assembly file and try to extract all such transformation patterns we can.

- **Step3:** Translate the patterns found to the appropriate customer-id based on the embedding schemes used.

When we read the watermark patterns we need to reverse the procedure used to embed the mark to deduce the customer-id from the extracted data.

- **Step4:** Extract the owner's data from the database corresponding to this particular customer-id.

We will not be focusing on this step in this project.

In this project, we will be concentrating on getting the watermark to work correctly and to display the correct customer-id as our result on the screen. We are not

concerned about being able to obtain the corresponding customer-name by doing read/write with the database, as it is unimportant to the purpose of our project. It has only been included in the diagrams for the sake of completeness.

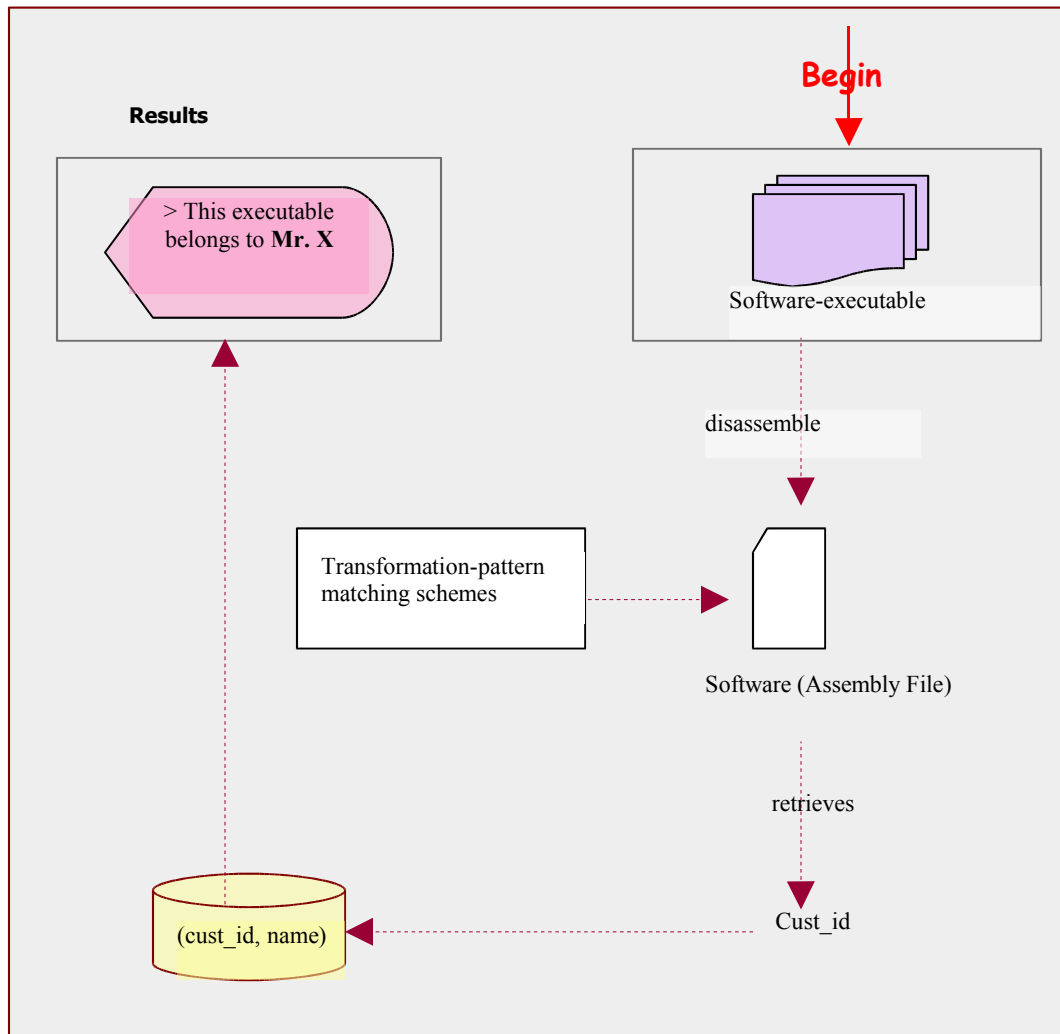


Figure 3 : Watermark Read Operation

6 Implementation

6.1 Setting up the environment / Tools used

To execute this application we need the following 4 components:

- ❑ Java compiler – JVM (java virtual machine), available from the download section of sun-java website.
- ❑ C compiler – we are using gcc, which is freely available on the net.
- ❑ Disassembler – IDA Pro, Version 4.6.0.785
- ❑ Software – we are using a project written in C, as our source application in order to demonstrate the watermark embedding/retrieval.
- All source codes and binaries necessary for the setting up the working environment for the project were downloaded and installed.
- The classpath variable was modified accordingly.

6.2 Implementation Details

As already described in the Design phase, there are two parts to this project.

1. Inserting the watermark in the code
2. Reading the watermark from the code.

We have automated the steps for running this project by using a command file “runproj.cmd”.

```
echo off
echo ..... Let's begin .....
echo 1. Creating an assembly TMain.s
gcc -S Traffic/TMain.c
pause
echo .....
echo 2. Writing watermark to assembly generating new file
wmTransform.s
javac WriteWM.java
java WriteWM
echo ...
pause
echo .....
echo 3. Generating executable (wmExe.exe) from the newly
watermarked file (wmTransform.s).
gcc -o wmExe wmTransform.s init.o rsrc.o
pause
echo .....
echo 4. Disassembling executable (wmExe.exe) using disassembler
IDA Pro; generates assembly file (wmExe.asm)
idag -B wmExe.exe
pause
echo .....
echo 5. Reading the watermark from the disassembled file ".asm"
javac ReadWM.java
java ReadWM | more
pause
echo .....
echo END
```

Figure 4 : runproj.cmd

This file contains all required commands to run the project from the beginning to the end. It demonstrates both a read and write of a watermark. If you need to change any file-names, you need to make changes to this file as well as the properties.txt file.

To run this file, you need to open the command prompt and go to the directory where this file is stored. Then at the prompt you just type the name of the file “runproj” and press enter and the complete project will run. More details are available in Section 9.

6.2.1 Inserting the watermark in the code

In this section we discuss the watermark insertion steps by taking as a simple example the “Hello world” code (hello.c).

- **Step 1:** Compile the source code to generate an assembly.

We begin with the source file, hello.c and generate an assembly file for it hello.s, using the option `-S` in gcc.

```
>gcc -S hello.c
```

This generates an assembly file hello.s and this is how it looks:

```

        .file "hello.c"
        .def  __main;    .scl 2;    .type 32;    .endif
        .text
LC0:
        .ascii "Hello, world\12\0"
        .align 2
        .globl _main
        .def  _main;    .scl 2;    .type 32;    .endif
_main:
        pushl %ebp
        movl  %esp, %ebp
        subl  $8, %esp
        andl  $-16, %esp
        movl  $0, %eax
        movl  %eax, -4(%ebp)
        movl  -4(%ebp), %eax
        call  __alloca
        call  __main
        movl  $LC0, (%esp)
        call  _printf
        movl  $0, %eax
        leave
        ret

        .def  _printf;    .scl 2;    .type 32;    .endif

```

Figure 5 : hello.s

Next we will insert a few transformations to the assembly file hello.s. We call the modified file wmHello.s. Here, we are trying to embed a watermark with customer-id '27' into the code. More information about transformations can be found in section 7.

- **Step 2:** Insert the watermark into assembly file hello.s.

We have employed two different schemes here to embed the customer-id into the software assembly code hello.s.

Scheme 1: For this particular scheme we will be dealing with the following files:

Input: properties.txt

keyTranx.txt

dummyTranx.txt

hello.s

Results: wmHello.s

- **properties.txt** – This file stores the names of different input and output files as well as important variable values. This file acts like a header file, which we use to declare important variables and files. However, the advantage of using properties.txt file over a header file is that we can change this file at runtime, which is not true in the case of the header file. Also, we do not wish to hardcore these file-names and variables in the code, as it becomes very inflexible.

We are using a concept of java called properties, whereby we write a variable name followed by an “=” sign, followed by the value of the variable. Now when we want to declare our Input-file as “hello.s”, we just declare it here. And when we want to obtain the name of the file in our program-code, we make a reference to this variable “p_wtWMInputFile” and read the value “c:\\wmProj\\hello.s” from here.

This is the file properties.txt

```
p_wtWMInputFile = c:\\wmProj\\hello.s
p_wtWMInputKey = c:\\wmProj\\keyTranx.txt
p_wtWMInputDummy = c:\\wmProj\\dummyTranx.txt
p_wtWMOutputFile = c:\\wmProj\\wmHello.s

p_rdWMInputFile = c:\\wmProj\\wmExe.asm
p_rdKeyPattern = c:\\wmProj\\ keyPattern.txt

p_custID = 27
p_dummy = Geos72ak8LAKD8lancu20sdkc
```

Figure 6 : properties.txt

We read the customer-id to be inserted from the properties.txt file. Its value is “27”.

Now, we refer to the keyTranx.txt file to obtain the list of all possible watermark transformations. For each number from 0-9 we have an associated transformation listed. These transformations have been selected such that they do not change the register values and yet introduce some software diversity with each instance. More information about transformation can be found in Section 7.

```
0 = push %eax
    movl $xx, %eax
    pop %eax
1 = ----
2 = xor $0, %ecx
3 = -----
4 = -----
5 = -----
6 = -----
7 = addl $13,%ebx
    subl $13,%ebx
8 = -----
9 = -----
```

Figure 7 : keyTranx.txt

Assumptions

Since we are inserting multiple copies of the transformation to make our watermark robust, we cannot use customer-ids like 3445. i.e. the immediate neighbors of any digit, has to be different than the digit itself. i.e. Two 4s cannot show up side-by-side. If we do not follow this rule, then we cannot determine if in the result “3445”, the number 4 is repeated or it occurs twice in the customer-id.

Customer-id “27” means that we will be inserting transformations 2 and 7 in this particular order. Moreover, we will insert multiple copies of these transformations to make it more robust. So, on inserting these transformations into our hello.s file, we will obtain a new file wmHello.s as shown below.

```

file "hello.c"
  .def __main; .scl 2; .type 32; .
  .text
  LC0:
    .ascii "Hello, world\12\0"
    .align 2
  .globl _main
  .def _main; .scl 2; .type 32; .
  .endef
  _main:
  pushl %ebp
  movl %esp, %ebp
  xor $0, %ecx ; Tranx #2 . key = 2.
  subl $8, %esp
  xor $0, %ecx ; repeat, Tranx #2
  andl $-16, %esp
  movl $0, %eax
  addl $13,%ebx
  subl $13,%ebx ; Tranx #7 key = 27
  movl %eax, -4(%ebp)
  addl $13,%ebx
  subl $13,%ebx ; repeat, Tranx #7
  movl -4(%ebp), %eax
  addl $13,%ebx
  subl $13,%ebx ; repeat, Tranx #7
  xor $0, %ebp
  call __alloca
  call __main
  movl $LC0, (%esp)
  push %ebp
  movl $23, %ebp
  pop %ebp
  call _printf
  movl $0, %eax
  leave
  ret
  .def _printf; .scl 2; .type 32; .endef

```

But, a hacker can compare the code of software and compare them, so we throw in several dummy transformations, which we have stored in a separate file, dummyTranx.txt. This file looks similar to keyTranx.txt except that this time we have transformations associated with numbers from 0-9 and with alphabets from a-z and A-Z.

Now, to read customer-id we read the value of variable p_custID from the properties.txt file (i.e. “27” and then inserted transformations 2 and 7 from keyTranx.txt file into wmHello.s Similarly to insert the dummy-transformations, we read the value of p_dummy from properties.txt file, i.e., Geos72ak8LAKD8lanc ... and accordingly insert these transformations – G, e, o, s, 7, 2, a, k, 8, ..., at random throughout the code. This helps us hide the actual watermark and severely obfuscates the code. So our wmHello.s will also carry several transformations from this list shown below.

```

0 = ---
1 = -----

9 = |-----
a = ---
b = -----

z = |-----
A = ---
B = -----

Z = |-----

```

Figure 9 : dummyTranx.txt

After addition of the dummy-transformation, the highly obfuscated code of wmHello.s will look something like this

```

file "hello.c"
    .def __main;    .scl 2;    .type 32;    .endef
    .text
LC0:
    .ascii "Hello, world\12\0"
    .align 2
    .globl _main
    .def _main;    .scl 2;    .type 32;    .endef
_main:
pushl %ebp
xxxxxxx      ; dummy-Tranx #G
movl %esp, %ebp
xor    $0, %ecx      ; Tranx #2 . key = 2.
XXXXXXXXXX   ; dummy-Tranx #G, repeat
subl $8, %esp
xxxxxxx      ; dummy-Tranx #e
xor    $0, %ecx      ; repeats, Tranx #2
xxxxxxx      ; dummy-Tranx #o
andl $-16, %esp
movl $0, %eax
XXXXXXXXXX   ; dummy-Tranx #s
addl $13,%ebx
subl $13,%ebx      ; Tranx #7 key = 27
movl %eax, -4(%ebp)
xxxxxxx      ; dummy-Tranx #2
addl $13,%ebx
subl $13,%ebx      ; repeats, Tranx #7
xxxxxxx      ; dummy-Tranx #7
movl -4(%ebp), %eax
xxxxxxx      ; dummy-Tranx #7, repeat
addl $13,%ebx
subl $13,%ebx      ; repeat, Tranx #7
x;And so on for the rest of the
dummy-tranxs' ak8LAKD8lancu20sdc
xor $0, %ebp
call __alloca
call __main
movl $LC0, (%esp)
push %ebp
movl $23, %ebp
pop %ebp
call _printf
.....

```

Figure 10 : wmHello.s

Scheme 2: We are going to call this our backup scheme. We will use the transformation# 0 for this purpose.

Transformation #0 is shown below

```
pushl %eax
movl $00, %eax
popl %eax
```

In this scheme we take the customer id, scramble it mathematically and then insert the results into transformation #0. When looking for the watermark, we read the inserted customer-id from this transformation #0 pattern, descramble it and obtain the results.

Example: Suppose the customer-id is 27 and mathematical scrambling the id involves adding 2 to its value i.e $27 + 2 = 29$. Now we insert 29 into our transformation #0 to obtain

```
push %eax
movl $29, %eax
pop %eax
```

We insert the 0th transformation before we insert the actual watermark of scheme1, indicated above. So basically, Scheme 2 insertions precede Scheme1 insertions.

So after all these insertions, Scheme1 insertions, Scheme2 – watermark & dummy insertions we move onto the next step.

The modified wmHello.s will look as follows:


```

file "hello.c"
.def __main; .scl 2; .type 32; .endif
.text
LC0:
.ascii "Hello, world\12\0"
.align 2
.globl _main
.def _main; .scl 2; .type 32; .endif
_main:
pushl %ebp
push %eax ; Transformation #0
movl $29, %eax ; Carries the key for scheme2
pop %eax
xxxxxxx ; dummy-Tranx #G
movl %esp, %ebp
xor $0, %ecx ; Tranx #2 . key = 2.
XXXXXXXXXXXX ; dummy-Tranx #G, repeat
subl $8, %esp
xxxxxxx ; dummy-Tranx #e
xor $0, %ecx ; repeats, Tranx #2
xxxxxxx ; dummy-Tranx #o
andl $-16, %esp
movl $0, %eax
xxxxxxx ; dummy-Tranx #s
addl $13,%ebx
subl $13,%ebx ; Tranx #7 key = 27
movl %eax, -4(%ebp)
xxxxxxx ; dummy-Tranx #2
addl $13,%ebx
subl $13,%ebx ; repeats, Tranx #7
xxxxxxx ; dummy-Tranx #7
movl -4(%ebp), %eax
xxxxxxx ; dummy-Tranx #7, repeat
addl $13,%ebx
subl $13,%ebx ; repeat, Tranx #7
x;And so on for the rest of the
dummy-tranxs' ak8LAKD8lanCu20sdkc
xor $0, %ebp
call __alloca
call __main
movl $LC0, (%esp)
push %ebp
movl $23, %ebp
pop %ebp
call _printf
movl $0, %eax
leave
ret
.def _printf; .scl 2; .type 32; .endif

```

Figure 11 : wmHello.s

The xx's in figure11 denotes dummy transformation instructions, which are all listed in file dummyTranx.txt.

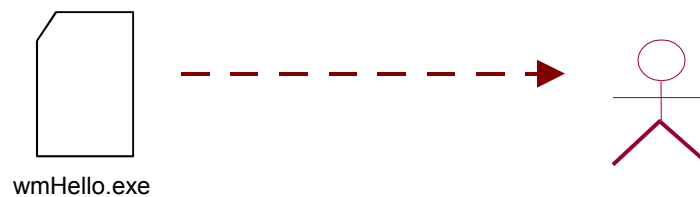
- Step 3: Generate an executable from the assembly code.

Now generate an executable from this newly transformed wmHello.s

```
>gcc -o wmHello wmHello.s
```

Similarly, for all N instances of the software we sell to N different customers, we repeat the above steps to create executables wmHello1.exe, wmHello2.exe, wmHello3.exe, and so on... All these newly created executables are functionally same, they differ however in the set of transformations used, their locations and the customer-id embedded in the watermark.

- Step 4: Sell this software executable (hello.exe) to the specified customer.



6.2.2 Reading the watermark from the executable:

To read the watermark, following are the files we will deal with:

Input: wmHello.exe

wmHello.asm

keyPattern.txt , this file is for our reference only

Now to extract/read a watermark back from a given executable, this is how we proceed.

Step 1: Disassemble the software executable to generate an assembly.

We are using disassembler IDA Pro Version 4.6.0.785 to disassemble our software executable wmHello.exe. IDA Pro provides an option to generate an assembly file with .asm extension for an opened executable. The command to do this has already been listed in the runproj.cmd file. Note that we cannot use any other disassembler other than IDA Pro as the assembly syntax used by different disassemblers is different. Our read program can detect the assembly patterns generated by IDA Pro, but may not work with the assembly patterns of some other disassembler.

On disassembly of wmHello.exe we obtain an assembly file wmHello.asm.

Step 2: Scan assembly file wmHello.asm for the special patterns and attempt to extract all possible embedded data from it.

Based on the embedding schemes used, translate the patterns found to the appropriate customer-id. The transformations translate to a different format after the write/read procedures. Through trial and error we inserted the transformations one at a time to see what they translate to. Finally we came up with a list of 10 different

patterns corresponding to the 10 transformations 0-9. We stored this pattern results in file keyPattern.txt for our reference.

```
0. push    eax
   mov     eax, xxh
   pop     eax
1. ----
2. xor     ecx, 0
3. ----
4. ----
5. ----
6.
7. add     ebx, 0Dh
   sub     ebx, 0Dh
8. -----
9. -----
```

Figure 12 : keyPattern.txt

Schemel:

Now, we have fed all the 10 patterns into our readWM.java file. This file compares the assembly code of wmHello.asm with each of the pattern and retrieves the key “27”.

```

push  ebp
push  eax
mov   eax, 1Dh ; Tranx pattern #0, key = 0
pop   eax
xxxxxxx ; dummy transformations
mov   ebp, esp
xor   ecx, 0 ; Tranx Pattern #2, key = 02
xxxxxxx ; dummy transformations
sub   esp, 8
xor   ecx, 0 ; Tranx Pattern#2, key = 022
and   esp, 0FFFFFF0h
mov   eax, 0
xxxxxxx ; some dummy tranx
add   ebx, 0Dh ; Tranx Pattern#2, key = 0227
sub   ebx, 0Dh
xxxxxxx ; some dummy tranx
mov   [ebp+var_4], eax
add   ebx, 0Dh ; Tranx Pattern#2, key = 02277
sub   ebx, 0Dh
mov   eax, [ebp+var_4]
add   ebx, 0Dh ; Tranx Pattern#2, key = 022777
sub   ebx, 0Dh
---xxxxxx-x—x—x-x-x—x-----

```

Figure 13 : wmHello.asm

The xx's in figure13 indicate other dummy-transformations; we are not interested in.

When our java program readWM.java scans through the assembly wmHello.asm, it carries out a pattern match. Corresponding to each pattern is a digit from 0-9. So, the program extracts data 022777. Thus, we can deduce our key as: "027".

Now, we need to compare our results with that of backup scheme 2

Scheme 2: While reading the watermark back from the executable, we first search for the pattern of transformation #0 which will now look like

```
push  eax
mov   eax, 1Dh    ; Tranx pattern #0, key = 0
pop   eax
```

The key we extracted in scheme 1 is “027”. So, we are able to find pattern 0 within the assembly of software executable. On finding this pattern we extract the information from it, which is the numbers 1Dh. 1D in hexadecimal is equivalent to 29 in decimal. To descramble 29 we subtract 2 from it giving us the customer-id 27. Thus, scheme 2 also provides us with the same results, assuring us that the customer-id has not been tampered with. Note that disassembler IDA Pro supports Hex number system at the assembly level.

The remaining transformations were just obfuscation or camouflage and we are not concerned of how a disassembler reads them and disassembles them.

To summarize, given a source file (hello.c), we create an assembly file .s from it (hello.s). We add some transformations to hello.s (key-watermarking transformations plus dummy transformations), and then link this new transformed file (wmHello.s) to produce an executable. (wmHello.exe)

So now, to read the watermark back, we ignore all the dummy transformations we had inserted except the key-watermark ones, and search for their corresponding patterns throughout the code. Once these patterns are found, we extract the watermarked information from it.

Transformations are added to the code to either, carry the watermark or act as an obfuscation to camouflage the actual watermark locations in the code. Some of these transformations are discussed in the next section.

7 Code Transformations

Code Transformations are special sequences of code, which are inserted into the program to make the program more complex and difficult to understand. This discourages an attacker from altering the program. The code sequence selected as transformations do not cause any change to the data stored in different registers. Thus, insertion of these code sequences into the program does not change them functionally. Some of the potential transformations are listed below. These can be used by a piece of code to attain both diversity and uniqueness.

7.1 Inserting Jump statements - JMPs

Jump statements can be very tricky to use in the assembly code, so they need to be used carefully. However, in our case, we intend to cause jump to go to the very next statement and so program flow is not affected. Thus, by introduction of a few dummy 'jmp' labels, we can make the code more obscure.

Example

<i>Original Code</i>	<i>Transformed Code</i>
#make_BIN# MOV AX, 5 MOV BX, 10 ADD AX, BX ;AX=5+10=15 (000F)h SUB AX, 1 ;AX=15-1=14 (000E)h HLT	#make_BIN# MOV AX, 5 MOV BX, 10 ADD AX, BX ;AX=5+10=15 (000F)h jmp proc_sub proc_sub: SUB AX, 1 ;AX=15-1=14 (000E)h HLT

Source : EMU 8086 Tutorial [4]

7.3 Using arithmetic and logical instructions for transformation

Certain logical and arithmetic instructions can be used in a certain fashion such that the net effect on the registers values is 0. As an example, we can XOR a register value with 0, and its value remains unchanged. Similarly, we can increment the register values by one and then decrement it by one, to leave the values unchanged. We can also OR a register value with 0, or NEG (negate) the value twice or use ADD/SUB as the transformations and the registers remain unchanged. However, we learnt during testing that an arithmetic or logical instruction cannot be inserted just before a conditional jump statement as it affects the flag-register values and causes the executable to run incorrectly.

Arithmetic and Logical Instructions

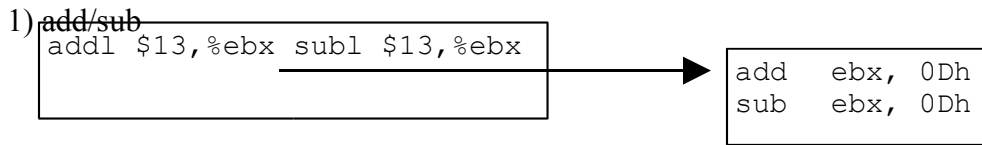
ADD, SUB, OR, XOR, INC, DEC, NEG

- XOR AL, 0
- OR BL, 0
- ADD AL, 8
SUB AL, 8
- INC SI
DEC SI
- NEG AX
NEG AX
- NOT BX
NOT BX

Example:

```
MOV AL, 05          ; AL ← 05
XOR AL, 0           ; AL ⊕ 0 = AL, Thus, AL = 05
```

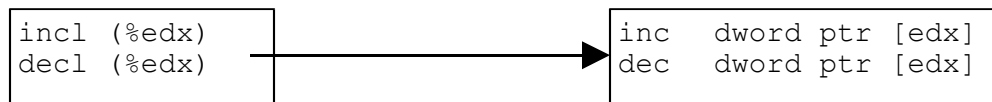
Let us see how some of the other transformations in this category translate.



This is the watermarking transformation we insert into the software. This is the transformation we read back from the executable.

Note that \$13 indicates value in decimal whereas 0Dh indicates its hex value.

- 2) ~~inc/dec~~ – increments register edx- value by one and then decrements by 1.



This is the watermarking transformation we insert into the software. This is the transformation we read back from the executable.

This transformation, when inserted into the assembly caused the executable to run incorrectly and hence was dropped from the transformation list.



This is the watermarking transformation we insert into the software. This is the transformation we read back from the executable.



This is the watermarking transformation we insert into the software. This is the transformation we read back from the executable.

During several test-runs we noticed that this transformation at times could not survive the compiler optimization and got lost. So we finally decided not to use this

transformation to carry the watermark, as it could result in the loss of embedded data.

However, we will be using it as one of our several dummy transformations.

7.4 Inserting NOPs

NOP stands for "no operation". NOP instructs the CPU to "skip this instruction."

Therefore, a sequence of nops could be inserted in a program if for example one would, at some later time, want to add to the program without changing its size [8].

Example:

```
MOV  AL, [SI]
MOV  AH, 0Eh
ADD  SI, 1

NOP
NOP
NOP
JMP  next_char
```

7.5 Using PUSH and POP instructions

PUSH instructions can be used to store the contents of the register onto the stack.

We can use that a particular register for some transformations and restore the register contents using POP instruction. This PUSH/POP pair comprises a new set of transformations.

Example:

```
PUSH AX
MOV  AX, 5678h
POP  AX
```

PUSH/POP can also be used to swap the values of two registers, twice leaving the register contents unchanged.

Example:

The following code swaps the values of registers CX and DX twice, leaving the register values unchanged[4].

```
MOV    CX, 0000h    ; store 0000h in CX.
MOV    DX, FFFFh    ; store FFFFh in DX

; the stack top always contains the value last pushed in.
; pop instruction always pop the value on stack top.
; swapping once

PUSH   CX           ; copy value of CX(i.e. 000h) in stack.
PUSH   DX           ; copy value of DX(i.e. FFFFh)in stack.

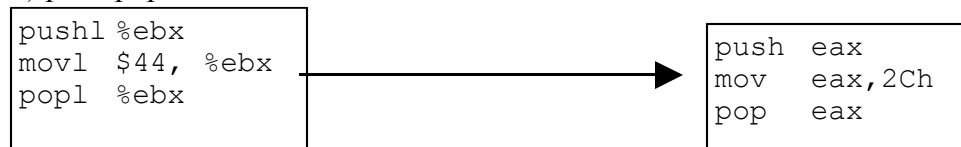
POP    CX           ; set CX to FFFFh, as FFFFh is stored at
the stacktop.
POP    DX           ; set DX to 0000h, as now stack top has
0000h.

; swapping twice
PUSH   CX           ; copy value of CX(i.e.FFFFh) to stack.
PUSH   DX           ; copy value of DX(i.e.0000h) to stack.

POP    CX           ; set CX to 0000h.
POP    DX           ; set DX to FFFFh
```

Let us see how a push/pop transformation translates.

1) push/pop



This is the watermarking transformation we insert into the software. This is the transformation we read back from the executable

Note that \$44 indicates value in decimal whereas 2Ch indicates its hex value.

8 Challenges

Reading the watermark is the most difficult aspect of this project, i.e. how to read these transformations from the exe, so as to determine the watermark and thus the customer -id. Every disassembler produces a different version of the disassembled code. Moreover slight change in the assembly code can result in major changes when its associated exe is disassembled. So it is difficult to read back all of the same transformations. Several of these are lost during assembly/disassembly. The goal was to find a few simple transformations, which can survive assembly/disassembly, and to pick appropriate locations for the same. These transformations carry the watermark. After testing several transformations, we found a set, which could survive assembly/disassembly, and used them to carry our watermark. For detailed list of these transformations, refer to section 15.2

Also, we tried inserting one transformation at a time into our program to determine what a particular transformation will translate to, after the assembly and disassembly operation. We used these results during our read pattern search.

We cannot disassemble known executable files, embed transformation into it and reassemble them back. This approach fails because, there is no tool, which can disassemble an executable and reassemble it back. Hence, instead of an executable, we decided to work with some C source code. We embed transformations into its assembly, we assemble it, then we disassemble the executable.

9 Deployment

Step 1: Make sure, all required components to run the project are installed. (JVM, IDAPro, gcc)

Step 2: Make sure, you have a working copy of any project C files (software).

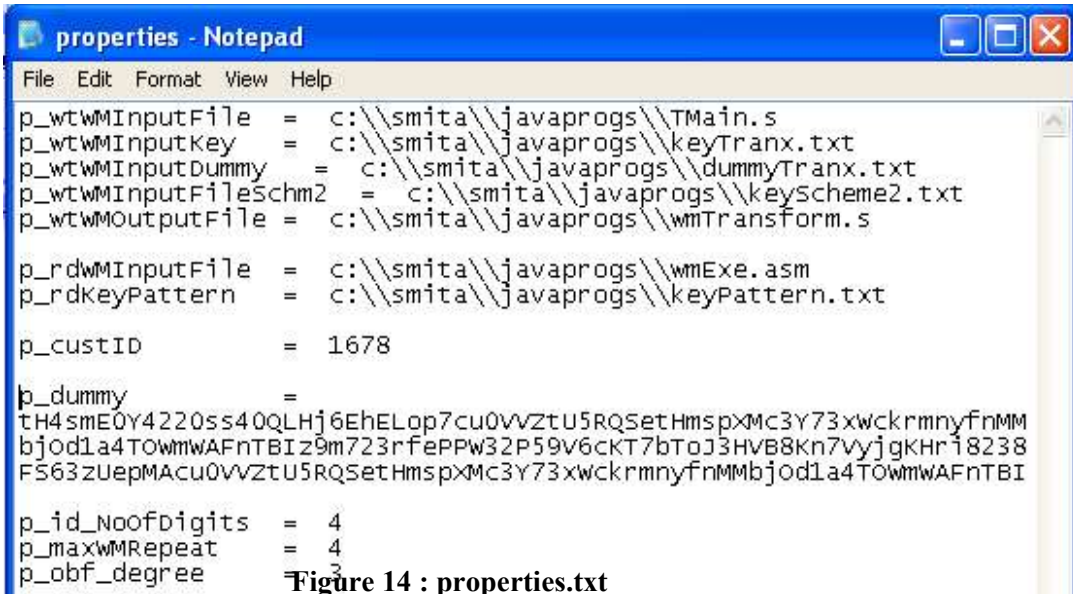
Step 3: Start command prompt and go to the directory where project is stored.

Step 4: At the prompt run the command file runproj.

```
> runproj
```

Step 5: Follow the steps indicated on the screen. To change the customer-id at any point before the write begins, open properties.txt file and make appropriate changes. To make changes to the file-names(i.e. to watermark your own C software) you will also have to make changes to the runproj.cmd file in addition to the properties.txt file.

This is how a property file looks which you can edit during runtime to change the customer-id.



```
properties - Notepad
File Edit Format View Help
p_wtwMInputFile = c:\\smita\\javaprogs\\TMain.s
p_wtwMInputKey = c:\\smita\\javaprogs\\keyTranx.txt
p_wtwMInputDummy = c:\\smita\\javaprogs\\dummyTranx.txt
p_wtwMInputFileSchm2 = c:\\smita\\javaprogs\\keyScheme2.txt
p_wtwMOutputFile = c:\\smita\\javaprogs\\wmTransform.s

p_rdwMInputFile = c:\\smita\\javaprogs\\wmExe.asm
p_rdkeyPattern = c:\\smita\\javaprogs\\keyPattern.txt

p_custID = 1678

p_dummy =
tH4smEQY4220ss40QLHj6EhELop7cu0vvZtU5RQsetHmspxMc3Y73xwckrmnyfnMM
bjod1a4TOWmWAFnTBIz9m723rfePPW32P59V6cKT7bToJ3HVB8Kn7vyjgKhr18238
F563zUepMACu0vvZtU5RQsetHmspxMc3Y73xwckrmnyfnMMbjod1a4TOWmWAFnTBI

p_id_NoofDigits = 4
p_maxwMRepeat = 4
p_obf_degree = 3
```

Figure 14 : properties.txt

You also need to reset the no-of-digits in the variable `p_id_NoOfDigits` as the size of customer-id changes. The variable `p_obf_degree` is used to set the degree of obfuscation. The higher the value of this variable, the more obfuscating the code is. This picture shows how the project actually runs and the steps it goes through.

```

C:\ Command Prompt - runproj
C:\smita\javaprogs>runproj
C:\smita\javaprogs>echo off
..... Let's begin .....
1. Creating an assembly TrafSim.s
Press any key to continue . . .
2. Creating custom-tranx 0
.....
2. Writing watermark to assembly generating new file wmTrafSim.s
FILE HAS 3498 lines.
Line#30 : Scheme2- Inserting Transformation#0
Line#40 : Scheme2- Inserting Transformation#0
Line#164 : KeyTransformation #1
Line#228 : DummyTransformation #4
Line#289 : KeyTransformation #1
Line#298 : DummyTransformation #m
Line#360 : KeyTransformation #1
Line#398 : DummyTransformation #4
Line#425 : DummyTransformation #2
Line#510 : DummyTransformation #0
Line#511 : DummyTransformation #s
Line#557 : KeyTransformation #6
Line#573 : DummyTransformation #Q
Line#626 : KeyTransformation #6
Line#640 : KeyTransformation #6
Line#645 : KeyTransformation #7
Line#648 : KeyTransformation #7
Line#746 : KeyTransformation #7
Line#759 : DummyTransformation #E
Line#783 : DummyTransformation #o
Line#810 : KeyTransformation #7
Line#838 : KeyTransformation #8
Line#849 : DummyTransformation #u
Line#923 : DummyTransformation #0
Line#985 : DummyTransformation #U
Line#989 : KeyTransformation #8
Line#1051 : DummyTransformation #t
Line#1071 : KeyTransformation #8
Line#1199 : DummyTransformation #5
Line#1256 : KeyTransformation #8
...
Press any key to continue . . .
3. Generating executable (wmExe.exe) from the newly watermarked file (wmTrafSim.s).
Press any key to continue . . .
4. Disassembling executable (wmExe.exe) using disassembler IDA Pro; generates assembly file (wmExe.asm)
Press any key to continue . . .
.....
4. Disassembling executable (wmExe.exe) using disassembler IDA Pro; generates assembly file (wmExe.asm)
Press any key to continue . . .
5. Reading the watermark from the disassembled file ".asm"
Scheme 1 Results, Customer-id = 001116
Scheme 1 Results, Customer-id = 0011166
Scheme 1 Results, Customer-id = 00111666
Scheme 1 Results, Customer-id = 001116667
Scheme 1 Results, Customer-id = 0011166677
Scheme 1 Results, Customer-id = 00111666777
Scheme 1 Results, Customer-id = 001116667777
Scheme 1 Results, Customer-id = 0011166677778
Scheme 1 Results, Customer-id = 00111666777788
Scheme 1 Results, Customer-id = 001116667777888
Scheme 1 Results, Customer-id = 0011166677778888
Scheme 1 Results, Customer-id = 0011166677778888
Following are the results we receive from our backup Scheme2.
Lets see if they compare to our results from scheme1
SCHEME2 RESULTS
Instance#0: Customer-id = 1678 (decimal); hexvalue = 104E
Instance#1: Customer-id = 1678 (decimal); hexvalue = 104E
Press any key to continue . . .

```

Figure 15 : Project execution on running runproj.cmd file

10 Summary

We successfully carried out the project goals of inserting a watermark into the assembly code and reading the watermark from the executable. This is how we achieved our results.

1. Watermark - We embedded the customer-id into the software assembly via transformation-insertion schemes.
2. Software Diversity – By embedding a different watermark and a different set of dummy-transformations into appropriate random locations throughout the software code, we are able to produce several copies of software which are functionally equivalent but where each copy is unique. An attacker who manages to break one particular instance of the software cannot use the same techniques to break the entire system [11]. It thus is able to provide partial protection against reverse-engineering.
3. Robustness – By trying out several different transformations we managed to find out the ones with a high survival rate after being processed by the optimizing compiler. We also figured out the locations where they tend to get lost. So using this knowledge and by inserting the watermark multiple times throughout the code we are able to make the watermark robust.
4. Reliability – The transformations selected for watermarking are highly robust, and are inserted multiple times. Thus, we are able to obtain accurate results everytime.
5. Obscurity – Large number of dummy transformations have been inserted alongwith the watermark throughout the code to make every instance of the software

significantly different from the other and to avoid any sort of pattern formations in the code.

6. Security and Tamper Proofing - In general, it is impossible to disassemble an executable, modify its assembly-code and re-assemble the code and obtain an executable, which can work. So, if someone tampers with the watermark at the assembly level he will have to put in significant amount of time and labor to generate a working executable out of it. Moreover, we also have a backup scheme to retrieve the watermark. So each time we read the watermark we obtain results from both scheme1 and scheme2 to see if any of the cust-id bits are lost or tampered with. Any discrepancy in results will immediately indicate tampering. Since both schemes work very differently, it's nearly impossible to figure out how each one of them works and tamper the code in such a way that results from both schemes match.

11 Directory Structure

- runproj.cmd – the command file which comprises of all instructions to read/write WM. More details about runproj.cmd can be found in the appendix section.
- properties.txt – a text file which stores the value of all crucial project-parameters like file names, customer-ids, etc. This allows us to change the parameters at runtime.
- keytranx.txt - this file stores the list of all possible transformations we embed as our watermark

- keyPattern.txt - this file stores the patterns into which the transformations get changed to after passing through the read/write procedures. This file is for our reference only. These patterns have already been included in the read procedure, for doing a pattern search.
- dummyTranx - this file stores the list of all dummy transformations, which we embed alongwith with our watermark-transformation to introduce obscurity and thereby making the problem of reverse engineering much more challenging.
- ReadWM.java - carries the code to read the watermark from the indicated executable-file.
- WriteWM.java – carries the code to write the watermark based on the given customer-id

We have used the hello program as a sample to explain our implementation. It is very short in size, which makes it easy to understand. But for an actual demonstration we will be using a bigger C project. It's a traffic simulation project whose source files are listed below. It gives us more flexibility and variation when playing around with the customer-id, its size and other parameters.

- Source C files – this list depends on what software you pick to be watermarked.

We have used a traffic intersection simulation project as our source.

- Tmain.c - the main file
- Init.o – object file of init.h. File init.h contains code to initialize the system before the start of the simulation.
- Rsrc.o – object file

- **Generated Files**

Tmain.s – the assembly file generated from the source-C file.

wmTransform.s – Tmain.s gets transformed to wmTransform.s after applying the watermark

wmExe.exe – the executable generated for this particular software-project.

wmExe.asm - the assembly file generated when disassembler IDA Pro disassembles wmExe.exe. We read the watermark from this file.

12 Conclusion

The technique of software watermarking discussed in this paper appears to be a viable method for embedding a robust and invisible watermark into software. The technique has several potential benefits. First, the removal of the watermark is difficult, if not impossible, to automate. In general, it is impossible to disassemble an executable, modify its assembly-code and re-assemble the code to obtain an executable that works. Second, the watermark can be made highly robust by inserting the mark multiple times into the code. Third, the technique has a positive security side effect, since it results in diverse software.

Software watermarking field is still in the developing phase and a lot needs to be done in this area. At times, compiler optimizations remove inserted watermarks. Research needs to be done on how optimization works and what can be done to make the watermark more robust. Methods to make the watermark more stealthy need to be

discovered. The project also needs to be software independent i.e. we need to find how applications written in languages other than C can use these watermarking schemes. Different assemblers and disassemblers use different assembly syntax. So this project code may not work if we use an assembler other than gcc or a disassembler other than IDA Pro. We need to find some common grounds such that we can use this project with most commonly used assemblers and disassemblers. The steps involved in reading the watermark are complex and tedious. The most difficult part is to determine how the watermarking transformations get changed after they are written, the code is assembled and the executable is disassembled. Coming up with the correspond patterns involves trial and error mechanism and is time-consuming. Moreover these patterns need to be hard coded into the read mechanism. Hence, its not easy to change the transformation sequence selected for watermarking.

13 References:

- [1] 8086 instruction set summary, Retrieved on September 24, 2003 from <http://www.eie.polyu.edu.hk/~enyhchan/8086inst.pdf>
- [2] Brey, B., Assembly Language Programming: *8086-8088, 80286, 80386, 80486*, Macmillan Publishing Company, 1997.
- [3] Collberg, C., Thomborson C.,and Low, D., A taxonomy of obfuscating transformations. Technical Report 148, Department of Computer Science, University of Auckland, New Zealand, July 1997. Retrieved on September 3, 2003 from <http://www.cs.auckland.ac.nz/~collberg/Research/Publications/CollbergThomborsonLo97a/index.html>
- [4] EMU8086: Tutorials, Retrieved on October 20, 2003 from <http://www.emu8086.com/Help/tutorials.html>
- [5] Gavin (1995), Gavin's Guide to 80x86 Assembly, Retrieved on March 10, 2004 from <http://burks.brighton.ac.uk/burks/language/asm/asmtut/asm2.htm>.

- [6] Isenberg, D., Digital Watermarks: New Tools for Copyright Owners and Webmasters, Retrieved on February 8, 2004 from <http://www.webreference.com/content/watermarks/>.
- [7] Low, D., Protecting Java Code Via Code Obfuscation, 1998, Retrieved on Nov 10, 2003 from <http://www.cs.arizona.edu/~collberg/Research/Students/DouglasLow/obfuscation.html>.
- [8] Mayer, J., Assembly Language Programming: 8086/8088, 8087, John Wiley & Sons, 1988.
- [9] Mishra, P. A Taxonomy of Uniqueness Transformations, 2003.
- [10] Report: Software Piracy Dips. Retrieved on January 24, 2004 from <http://www.cnn.com/2003/TECH/biztech/06/03/software.piracy.reut/>.
- [11] Stamp, M., Digital Rights Management: The Technology Behind the Hype, Retrieved on October 15, 2003 from <http://home.earthlink.net/~mstamp1/papers/DRMpaper.pdf>.
- [12] Stamp, M., Risks of Monoculture, to appear in *Communications of the ACM*.
- [13] Szor, P., Ferrie, P., Hunting for Metamorphic, Virus Bulletin Conference, September 2001, Retrieved on August 15, 2003 from <http://www.peterszor.com/metamorp.pdf>.
- [14] Zhang, Y., Shao, Q., Kwon, D., Krishnaswamy, S., Ma, D., Palsberg, J. Software Watermarking, Retrieved on October 30, 2003 from <http://www.cs.purdue.edu/homes/madi/wm/#Intro>

14 Appendices

Following is the code of the basic files used :

14.1 runproj.cmd

This file contains all the commands to automatically run the project.

```

echo off

echo ..... Let's begin .....
echo 1. Creating an assembly TrafSim.s
gcc -S Traffic/TrafSim.c
pause
echo .....
echo 2. Creating custom-tranx 0
javac CreateTranx0.java
java CreateTranx0
echo .....
echo 2. Writing watermark to assembly generating new file wmTrafSim.s
javac WriteWM.java

```

```

java WriteWM
echo ...
pause
echo .....
echo 3. Generating executable (wmExe.exe) from the newly watermarked file
(wmTrafSim.s).
gcc -o wmExe wmTrafSim.s init.o rsrc.o
pause
echo .....
echo 4. Disassembling executable (wmExe.exe) using disassembler IDA Pro;
generates assembly file (wmExe.asm)
idag -B wmExe.exe
pause
echo .....
echo 5. Reading the watermark from the disassembled file ".asm"
javac ReadWM.java
java ReadWM | more
pause
echo .....
echo END

```

Here is an explanation for all the commands used.

- `gcc -S Traffic/TMain.c` – generates an assembly file `Tmain.s` for
 - `Tmain.c`
- `javac CreateTranx0.java` – compiles java program `CreateTranx0.java` generating a
java byte code file `CreateTranx0.class`
 - `java CreateTranx0` - runs the program
- `javac WriteWM.java` - compiles java program `WriteWM.java`. This program
writes the new watermarked data to file `wmTransform.s`
 - `java WriteWM` - runs the program
- `gcc -o wmExe wmTransform.s init.o rsrc.o` – generates an executable for the
watermarked file
- `idag -B wmExe.exe` - IDA provides this command line function to disassemble an
executable and store the results into an assembly `.asm` file. `wmExe.exe` is
disassembled and the results are stored in assembly file `wmExe.asm`.
- `javac ReadWM.java` - compiles java program `ReadWM.java`. This program searches
assembly file `wmExe.asm` for the watermark and displays watermarked data.
 - `java ReadWM` - runs the program

14.2 properties.txt

```

p_wtWMInputFile = c:\\smita\\javaprogs\\TrafSim.s
p_wtWMInputKey = c:\\smita\\javaprogs\\keyTranx.txt
p_wtWMInputDummy = c:\\smita\\javaprogs\\dummyTranx.txt
p_wtWMInputFileSchm2 = c:\\smita\\javaprogs\\keyScheme2.txt
p_wtWMOutputFile = c:\\smita\\javaprogs\\wmTrafSim.s

```

```

p_rdWMInputFile = c:\\smita\\javaprogs\\wmExe.asm
p_rdKeyPattern  = c:\\smita\\javaprogs\\keyPattern.txt

p_custID       = 1678
p_dummy        = t4smEOY4220ss40QLHj6EhELop7cu0VVZtU5RQSetHms
pXMc3Y73xWckrmnyfnMMbjOdl44TOWmWAFnTBIz9m723rfePPW32P59V6cKT7bToJ3HVB8Kn7VyjgKHri82
38FS63zUepMAcu0VVZtU5RQSetHmspXMc3Y73xWckrmnyfnMMbjOdl44TOWmWAFnTBIz9m723rfePPW32P5
9V6cKT7bToJ3HVB8Kn7VyjgKHri8238FS63zUepMAcu0

p_id_NoOfDigits = 4
p_maxWMRepeat   = 4

```

Always set p_maxWMRepeat to a value 2 and higher, to make the watermark more robust

```
p_obf_degree = 3
```

The larger the value of this variable is the higher the level of obfuscation. So an obfuscation degree of 3 indicates, for every 1 key transformation the probability of a dummy transformation showing up is 3.

p_wtWMInputFile indicates the name of the source assembly input file into which we insert the watermark. p_wtWMInputKey indicates the file containing list of all key transformations. p_wtWMInputDummy indicates the file containing list of all possible dummy transformations. p_wtWMOutputFile indicates the file generated as a result of the watermark insertions into the source assembly file. p_obf_degree sets the level of obfuscation. The larger the value of p_obf_degree, the higher the level of obfuscation.

14.3 keyTranx.txt

This file contains all key transformations. One transformation for each of the digits 0-9. Depending on the customer-id the corresponding transformation is inserted into the software assembly code.

```

1 = \
pushl  %ebx \n\
movl   $44, %ebx \n\
popl   %ebx \n\

2 = \
jmp proc_tanu \n\
proc_tanu: \n\

3 = \
xor    $0, %ecx \n\

4 = \
xor    $0, %ebp \n\

5 = \
or     $0, %ebp \n\

6 = \
or     $0, %esp \n\

```



```

7 = \
addl $8,%edx \n\
subl $8,%edx \n\

8 = \
addl $13,%ebx \n\
subl $13,%ebx \n\

9 = \
addl $17,%edx \n\
subl $17,%edx \n\

0 = \
pushl %eax \n\
movl $16, %eax \n\
movl $78, %eax \n\
popl %eax \n\

```

14.4 keyPattern.txt

This file contains patterns corresponding to each of the 10 transformations indicated in Section 15.2. This file is for our understanding only.

```

0.
push  eax
mov   eax, 0Fh
mov   eax, 4Eh
pop   eax

1.
push  eax
mov   eax, 2Ch
pop   eax

2.
jmp   short $+2

3.
xor   ecx, 0

4.
xor   ebp, 0

5.
or    ebp, 0

6.
or    esp, 0

7.
add   edx, 8
sub   edx, 8

8.
add   ebx, 0Dh
sub   ebx, 0Dh

9.
add   edx, 11h
sub   edx, 11h

```

14.5 CreateTranx0.java

This program is used to create a customized transformation #0, depending on the value of customer-id. This transformation #0 then becomes a part of the list of key transformations in keyTranx.txt

```
/*
 * CreateTranx0.java
 *
 * Author : Smita Thaker, Masters Project, San Jose State University.
 * Date : 25 May, 2004.
 * Version : 1.0
 * Purpose : Code to generate the 0th transformation and complete the list of
 * key transformation.
 * The 0th key transformation carries the watermark as per our backup
 * scheme2. Before we begin watermark-write, we embedd the customer-id
 * in the 0th transformation and copy it to our keyTranx.txt file.
 * Once this list of transformation file is complete we can begin
 * the write procedure which is described in file write.java.
 * Functions : main(String []),
 *
 */

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.LineNumberReader;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.lang.Math;
import java.lang.reflect.Array;
import java.util.Properties;
import java.util.Vector;

public class CreateTranx0 {

    /*
     * Function : main
     * Creates transformation#0 and writes it to keyTranx.txt
     * Input
     * custid : Customer-id
     * tmp.txt : Temporary file containing key-transformations in the range
(1-9).
     * tmp0.txt : Temporary file containing incomplete transformation 0.
     * Output
     * keyTranx.txt : File containing all transformations 0-9.
     */
    public static void main (String args[]) throws Exception {

        String line = null, s="", t="";
        int digit = 0, odd = 0;

        Properties props = new Properties();
        File propFile = new File("c:\\smita\\javaprogs\\properties.txt");
        InputStream propStream = new FileInputStream(propFile);
        props.load(propStream);
    }
}
```

```

        File tmpFile = new File("c:\\smita\\javaprogs\\tmp.txt");          //File
containing KeyTranx's
        File tmp0File = new File("c:\\smita\\javaprogs\\tmp0.txt");      //File
containing tranx0
        File createTranx0File = new File(props.getProperty("p_wtWMInputKey")); //
Result File containing all tranx's

        LineNumberReader rdr = new LineNumberReader(new FileReader(tmpFile)); //
LnNoRdr for tmp - file
        LineNumberReader rdr0 = new LineNumberReader(new FileReader(tmp0File)); //
LnNoRdr for tmp0 file
        PrintWriter writer = new PrintWriter(new FileWriter(createTranx0File)); //
writer for output(result) all-tranx list file
        int dig = Integer.parseInt((props.getProperty("p_id_NoOfDigits")));
        String custid = (props.getProperty("p_custID"));

        while((line = rdr.readLine())!=null) // copies from tmp.txt file to
keyTranx.txt file
            writer.println(line);

        if(dig != custid.length()) System.out.println("\nALERT !!! \n\t Values of
variables cust-id and id_NoOfDigits are set incorrectly.\n\t This will either give
an error or an incorrect result.\n\t Kindly reset it and restart this application.
\n\n");
        if(Math.IEEEremainder(dig,2) == 0) digit = dig/2;
        else {digit = (dig/2)+1; odd = 1; }

        /* appends Transformation0 from tmp0.txt file to keyTranx.txt file */
        while((line = rdr0.readLine())!=null) {
            if(line.indexOf("movl") != -1) { // means "movl" is found in "line"
                for(int i=0; i<digit;i++) {
                    if(odd == 1) { //cust-id-size is odd
                        if(i==0) t = custid.substring(i,i+1);
                        else {
                            t = custid.substring((i*2)-1, (i*2)+1);
                            if(t.indexOf("0")== 0) t = t.replaceFirst("0","");
                        }
                    } else { //cust-id-size is even
                        t = custid.substring((i*2), (i*2)+2);
                        if(t.indexOf("0")== 0) t = t.replaceFirst("0","");
                    }
                    s = line.replaceFirst("00",t); // creates the "movl" code sequence for
transformation#0
                    writer.println(s);
                }
            } else
                writer.println(line);
        } // end of while loop.
        rdr.close();
        rdr0.close();
        writer.close();
        propStream.close();
    } /* End of main function */
} /* End of class CreateTranx0 */

```

14.6 ReadWM.java

This file contains code to read the watermark from software executable.

```

/*****
* ReadWM.java
*

```

```

* Author      : Smita Thaker, Masters Project, San Jose State University.
* Date       : 25 May, 2004.
* Version    : 1.0
* Purpose    : Code for reading the watermark (customer-id) from the
*             software-executable's assembly file (.asm)
* Functions  : main(String []),
*             read(int id_noOfDigits, int maxRep, LineNumberReader readwmfile,
*             Properties props)
*             hexToDec(String hex)
* Issues     : Reading the transformations correctly
*
***** */

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.LineNumberReader;
import java.lang.Math;
import java.lang.reflect.Array;
import java.util.Properties;
import java.util.Vector;
import java.util.regex.*;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.lang.Math;
import java.lang.reflect.Array;

public class ReadWM {
    public static void main (String args[]) throws Exception {

        Properties props          = new Properties();
        File propFile             = new File("c:\\smita\\javaprogs\\properties.txt");
        FileInputStream propStream = new FileInputStream(propFile);
        props.load(propStream);

        File rdWMInputFile = new File(props.getProperty("p_rdWMInputFile")); // Our .
asm assembly file is our input.
        File rdKeyPattern  = new File(props.getProperty("p_rdKeyPattern"));
        int id_noOfDigits  = Integer.parseInt((props.getProperty("p_id_NoOfDigits")));
        int maxRep         = Integer.parseInt((props.getProperty("p_maxWMRepeat")));
        LineNumberReader rdInLnNoRdr      = new LineNumberReader(new FileReader
(rdWMInputFile)); //rdInLnNoRdr *
        LineNumberReader rdKPatternLnNoRdr = new LineNumberReader(new FileReader
(rdKeyPattern)); //rdKPatternLnNoRdr *

        String custid = read(id_noOfDigits, maxRep, rdInLnNoRdr);

        propStream.close();
        rdInLnNoRdr.close();
        rdKPatternLnNoRdr.close();
    } /* End of function main */

/*****
*
* Function      : read(int id_noOfDigits, int maxRep, LineNumberReader readwmfile)
* Input
*   id_NoOfDigits: Size of customer-id (fixed)
*   maxRep       : Maximum times a key transformation can be repeated
*   readwmfile   : Line-Numb-Reader for software-executable's assembly file, the
*                 file we read to find the watermark
* Output       : Returns the watermark. Prints the results from both Schemel and
*               Scheme2.
***** */

```

```

public static String read(int id_noOfDigits, int maxRep, LineNumberReader
readwmfile)throws IOException {
    String line=null,rdline = null, key="", tmpkey2 = "", key_bit = "", tmpkey = "",
key2res = "", tmpscheme2res = "";
    String[] vline = null, key2 = null, scheme2res = null;
    vline = new String[100];
    key2 = new String[((id_noOfDigits/2) * maxRep)+2];
    scheme2res = new String[((id_noOfDigits/2) * maxRep)+2];
    int flag = 0, digit = 0, odd = 0, a=0;

    /* Read the assembly .asm file line-by-line looking for the keyPattern */
    while((rdline=readwmfile.readLine())!=null) { // while-line#42 begins
        tmpkey2 = "";
        flag = 0;
        if(Pattern.matches("(\\s)*push(\\s)*eax(\\s)*", rdline)){ // if-line#45
            flag = 1;
            key_bit = "0";

            if(Math.IEEEremainder(id_noOfDigits,2) == 0) digit = id_noOfDigits/2;
            else { digit = (id_noOfDigits/2)+1; odd = 1; }

            for(int i=0; i<digit; i++) { // for-line54 begins
                rdline = readwmfile.readLine();
                rdline.trim();
                if (Pattern.matches("(\\s)*mov(\\s)*eax,(\\s)(\\s)*(\\s)*",rdline)) {
                    int len = rdline.length();
                    int x = rdline.indexOf(",");
                    int y = rdline.indexOf("h");
                    if(y != -1) tmpkey = rdline.substring(x+2,x+4); // h exists means
its a 2-digit no.
                    else if(i!=0) tmpkey2 = tmpkey2.concat("0");
                    tmpkey = rdline.substring(x+2, x+3);
                    } // h doesn't exist means its a 1-digit no.
                    tmpkey2 = tmpkey2.concat(tmpkey);
                    key2res = hexToDec(tmpkey);
                    if ((i <digit-1) &&(key2res.length() == 1)) tmpscheme2res =
tmpscheme2res.concat("0");
                    tmpscheme2res = tmpscheme2res.concat(key2res);
                } else {
                    flag = 0;
                    break ; // end of if else
                }
            } // end of for-line43 loop

            if(flag != 0) {
                rdline=readwmfile.readLine();
                if (Pattern.matches("(\\s)*pop(\\s)*eax(\\s)*", rdline)) {
                    key = key.concat(key_bit);
                    scheme2res[a] = tmpscheme2res;
                    key2[a] = tmpkey2;
                    a++;
                    tmpscheme2res = "";
                } else {
                    flag = 0;
                    break ; // end of if else
                }
            } // end of if-statement.
        }
        if(Pattern.matches("(\\s)*push(\\s)*ebx", rdline)){
            flag = 1;
            key_bit = "1";
            for(int j=1;j<3;j++) {
                rdline = readwmfile.readLine();
                if ((j == 1) && (Pattern.matches("(\\s)*mov(\\s)*ebx,(\\s)*2Ch(\\s)
*",rdline))) ;
                else if ((j == 2) && (Pattern.matches("(\\s)*pop(\\s)*ebx(\\s)*",
rdline))) ;
            }
        }
    }
}

```

```

        else {
            flag = 0;
            break ;                // end of if else
        }
    } // end of for-loop
    if (flag == 1)
        key = key.concat(key_bit);
        flag = 0;
} else if (Pattern.matches("(\\s)*jmp(\\s)*short(\\s)*\\$\\$\\+2", rdline)){
    flag = 1;
    key_bit = "2";
    key = key.concat(key_bit);
} else if (Pattern.matches("(\\s)*xor(\\s)*ecx,(\\s)*0", rdline)){
    flag = 1;
    key_bit = "3";
    key = key.concat(key_bit);
} else if (Pattern.matches("(\\s)*xor(\\s)*ebp,(\\s)*0", rdline)){
    flag = 1;
    key_bit = "4";
    key = key.concat(key_bit);
} else if (Pattern.matches("(\\s)*or(\\s)*ebp,(\\s)*0", rdline)){
    flag = 1;
    key_bit = "5";
    key = key.concat(key_bit);
} else if (Pattern.matches("(\\s)*or(\\s)*esp,(\\s)*0", rdline)){
    flag = 1;
    key_bit = "6";
    key = key.concat(key_bit);
} else if (Pattern.matches("(\\s)*add(\\s)*edx,(\\s)*8", rdline)){
    flag = 1;
    key_bit = "7";
    if(Pattern.matches("(\\s)*sub(\\s)*edx,(\\s)*8", readwmfile.readLine()))
        key = key.concat(key_bit);
} else if (Pattern.matches("(\\s)*add(\\s)*ebx,(\\s)*0Dh", rdline)){
    flag = 1;
    key_bit = "8";
    if(Pattern.matches("(\\s)*sub(\\s)*ebx,(\\s)*0Dh(\\s)*",
readwmfile.readLine()))
        key = key.concat(key_bit);
} else if (Pattern.matches("(\\s)*add(\\s)*edx,(\\s)*11h(\\s)*", rdline)){
    flag = 1;
    key_bit = "9";
    if (Pattern.matches("(\\s)*sub(\\s)*edx,(\\s)*11h(\\s)*",
readwmfile.readLine()))
        key = key.concat(key_bit);
} else {
    flag = 0;
}
if (flag == 1)    System.out.println ("Scheme 1 Results, Customer-id = "+key);

} // end of while loop
System.out.println("\nFollowing are the results we receive from our backup
Scheme2.\nLets see if they compare to our results from scheme1 ");
System.out.println("SCHEME2 RESULTS\n");
for(int i=0; i<a; i++)
    System.out.println("Instance#"+i+":          Customer-id = "+scheme2res[i]+"
(decimal); hexvalue = "+key2[i]);

/* ENDED - Reading of .asm assembly file */
System.out.println();
return key;
}
/* End of function read */
/*****
*
* Function      : String hexToDec(String hex)
*                Converts a hex input to decimal value
*/

```

```

* Input
*   hex      : Watermark extracted from the assembly file is in hexadecimal.
*             It has to be converted to its equivalent decimal value.
* Output
*   result   : The decimal value of input-hex
***** */

public static String hexToDec(String hex) throws IOException {

    int res, res1, res2;
    String tmp, tmp1, tmp2, result = "";
    Properties props = new Properties();
    File hexToDecFile = new File("c:\\smita\\javaprogs\\hexToDec.txt");
    FileInputStream propStream = new FileInputStream(hexToDecFile);
    props.load(propStream);

    int len = hex.length();

    if(len == 1) {
        tmp = props.getProperty(hex.substring(0,1));
        result = result.concat("0");
        result = result.concat((String.valueOf(tmp)));
    } else {
        tmp1 = props.getProperty(hex.substring(0,1));
        res1 = Integer.parseInt((String) (tmp1));
        res1 = res1 * 16;
        tmp2 = props.getProperty(hex.substring(1,2));
        res2 = Integer.parseInt((String) (tmp2));
        res = res1 + res2 ;
        result = result.concat((String.valueOf(res)));
    }
    return result;
} /* End of function hexToDec */
} /* End of class ReadWM */

```

14.7 WriteWM.java

This file contains code to write the watermark into the software assembly.

```

/*****
* WriteWM.java
*
* Author      : Smita Thaker, Masters Project, San Jose State University.
* Date       : 25 May, 2004.
* Version    : 1.0
* Purpose    : Code for inserting the watermark (customer-id) into the
*             software assembly file (.s)
* Functions  : main(String []),
*             write(custID, id_NoOfDigits, dumVal, repTranx, obf_degree,
*             randLocs, noOfRandLocs, wtInLnNoRdr1, wtInLnNoRdr2,
*             wtInLnNoRdr3, wtOutPrtWtr, prop_tranx, prop_dummy),
*             findrandLocs(noOfRandLocs, randLocs, wtInLnNoRdr1).
* Problems   : If the size of file into which we are trying to write the
*             transformations is small and the values of obf_degree,
*             id_NoOfDigits and repTranx is comparatively larger, we might
*             lose some of the transformations inserted, resulting finally
*             into incorrect results during read operation.
***** */

```

```

import java.io.BufferedReader;
import java.io.FileWriter;
import java.io.File;
import java.io.FileReader;
import java.io.PrintWriter;
import java.io.OutputStream;
import java.io.FileInputStream;
import java.io.LineNumberReader;
import java.lang.String;
import java.util.Properties;
import java.util.Arrays;
import java.util.Vector;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.lang.Math;
import java.lang.reflect.Array;

public class WriteWM {

/*****
 *
 * Function : main
 * Input
 *   properties.txt : Several important variables required for the write
 *                   procedure are defined in this file. It also contains
 *                   names of files used for write purpose.
 * Output
 *   Inserts the required transformations into the indicated software assembly
 *   and generates a new watermarked assembly file.
 * Calls function   : write, findrandLocs
 *****/
public static void main (String args[]) throws Exception {

    Properties props = new Properties();
    File propFile    = new File("c:\\smita\\javaprogs\\properties.txt");
    FileInputStream propStream = new FileInputStream(propFile);
    props.load(propStream);

    Properties prop_tranx = new Properties();
    File propTranxFile    = new File("c:\\smita\\javaprogs\\keyTranx.txt");
    FileInputStream propTranxStream = new FileInputStream(propTranxFile);
    prop_tranx.load(propTranxStream);

    Properties prop_dummy = new Properties();
    File propDummyFile    = new File("c:\\smita\\javaprogs\\dummyTranx.txt");
    FileInputStream propDummyStream = new FileInputStream(propDummyFile);
    prop_dummy.load(propDummyStream);

    File wtWMInputFile = new File(props.getProperty("p_wtWMInputFile")); //
File we are adding WM to
    File wtWMInputKey   = new File(props.getProperty("p_wtWMInputKey")); //
File containing WM-Key
    File wtWMInputDummy = new File(props.getProperty("p_wtWMInputDummy")); //
File containing dummy tranx
    File wtWMOutputFile = new File(props.getProperty("p_wtWMOutputFile")); //final
result, watermarked assembly file.
    LineNumberReader wtInLnNoRdr1 = new LineNumberReader(new FileReader
(wtWMInputFile)); //LnNoRdr for input assembly file *
    LineNumberReader wtInLnNoRdr2 = new LineNumberReader(new FileReader
(wtWMInputKey)); //LnNoRdr for key tranx list file *
    LineNumberReader wtInLnNoRdr3 = new LineNumberReader(new FileReader
(wtWMInputDummy)); //LnNoRdr for dummy tranx list file *
    PrintWriter wtOutPrtWtr      = new PrintWriter(new FileWriter
(wtWMOutputFile)); //Writer for output (result)assembly file

    String custID = String.valueOf(props.getProperty("p_custID"));

```



```

    int id_NoOfDigits = Integer.parseInt((String) (props.getProperty
("p_id_NoOfDigits")));;
    int obf_degree    = Integer.parseInt((String)props.getProperty
("p_obf_degree"));
    int repTranx     = Integer.parseInt((String) (props.getProperty
("p_maxWmRepeat")));
    String dumVal    = String.valueOf(props.getProperty("p_dummy"));
    int noOfRandLocs = (int)((obf_degree+1) * repTranx * id_NoOfDigits)*1.2);
    Vector randLocs  = new Vector(noOfRandLocs,4);

    int totLines = findrandLocs(noOfRandLocs, randLocs, wtInLnNoRdr1);

    wtInLnNoRdr1.close();
    wtWMInputFile = new File(props.getProperty("p_wtWMInputFile"));
    wtInLnNoRdr1  = new LineNumberReader(new FileReader(wtWMInputFile));

    write(custID, id_NoOfDigits, dumVal, repTranx, obf_degree, randLocs,
noOfRandLocs, wtInLnNoRdr1, wtInLnNoRdr2, wtInLnNoRdr3, wtOutPrtWtr, prop_tranx,
prop_dummy); // replace with maxNoOfLocs

    propStream.close();
    propTranxStream.close();
    propDummyStream.close();
    wtInLnNoRdr1.close();
    wtInLnNoRdr2.close();
    wtInLnNoRdr3.close();
    wtOutPrtWtr.close();

} /* End of function main */

/*****
*
* Function      : write(String id, int id_NoOfDigits, String dumVal, int repTranx,
*                int obf_degree, Vector randLocs, int noOfRandLocs,
*                LineNumberReader rdr, LineNumberReader keyrdr,
*                LineNumberReader dummyrdr, PrintWriter writer,
*                Properties prop_tranx, Properties prop_dummy)
* Input
* id            : customer-id
* id_NoOfDigits: Size of customer-id
* dumVal        : Value for dummy-transformations
* repTranx      : Maximum times a key transformation can be repeated
* obf_degree    : Degree of obfuscation, indicates the probable ratio of
*                key:dummy transformation insertions
* randLocs      : An array which stores random nos. between (0-totalLinesInFile)
*                Indicates the positions in file where insertion of
*                transformations takes place.
* noOfRandLocs : maximum possible size of array randLocs. Helps determine the
*                size of other arrays used in the function.
* rdr           : Line-Numb-Reader for software assembly file
* keyrdr        : Line-Numb-Reader for file containing list of key
transformations
* dummyrdr      : Line-Numb-Reader for file containing list of dummy
transformations
* writer        : Writer for output(result)assembly file
* prop_tranx    : properties file for key transformations
* prop_dummy    : properties file for dummy transformations
* Output       : Inserts the required transformations into the indicated software
*                and generates a new watermarked output (result)assembly file
* Issues       : we have picked random locations to insert the watermark, which
doesn't
*                always work. The generated executable is not able to function
*                correctly at times. After careful study of output files and their
*                assembly code, it was found that insertion of a mathematical
*                transformation right before a conditional jump statement affects
*                the value of flag-registers, causing the program to run incorrectly.
*****/

```

```

*           Thus, although, insertions are done randomly, this additional
condition
*           is checked before they are done. *
*
***** */

public static void write(String id, int id_NoOfDigits, String dumVal, int
repTranx, int obf_degree, Vector randLocs, int noOfRandLocs, LineNumberReader rdr,
LineNumberReader keyrdr, LineNumberReader dummyrdr, PrintWriter writer, Properties
prop_tranx, Properties prop_dummy) throws IOException
{
    String line = null, nextLine = "";
    int i, j, k=0, rep=0, redflag1 = 0, redflag2 = 0;

    int[] repeat, dumRepeat, wtChoice;
    repeat = new int[id_NoOfDigits]; //determines how many times we want to
repeat a dummy tranx insertion
    dumRepeat = new int[noOfRandLocs*2]; //determines how many times we want to
repeat a dummy tranx insertion
    wtChoice = new int[noOfRandLocs*2]; //determines if we want to insert a
dummy or key tranx
    String[] s;
    s = new String[id_NoOfDigits];

    /* COPY bits of the cust-id to array s */
    String cid = String.valueOf(id);
    for(i=0; i < id_NoOfDigits; i++)
        s[i] = cid.substring(i,i+1);

    /* copy bits of dum-Val to array d */
    String[] d;
    d = new String[dumVal.length()];
    for(i=0; i< dumVal.length(); i++)
        d[i]= dumVal.substring(i,i+1);

    /* Generates a random numb to use for key - repeatitions */
    for(j=0; j< id_NoOfDigits; j++) {
        int tmp = (int) Math.round(Math.random()*10000);
        if(Math.IEEEremainder(tmp,3) == 0) repeat[j] = repTranx;
        else repeat[j] = repTranx - 1 ;
    }

    /* Generates a random numb to use for dummy - repeatitions */
    for(j=0; j< noOfRandLocs*2; j++) {
        int tmp = (int) Math.round(Math.random()*10000);
        if(Math.IEEEremainder(tmp,4) == 0) dumRepeat[j] = 3;
        else if(Math.IEEEremainder(tmp,3) == 0) dumRepeat[j] = 2;
        else dumRepeat[j] = 1;
    }

    /* Generates a random number which decides if we insert the key or the dummy
(helps obfuscate the code) */
    for(j=0; j< noOfRandLocs*2; j++) {
        int tmp = (int) Math.round(Math.random()*10000);
        if(Math.IEEEremainder(tmp, obf_degree) == 0) wtChoice[j] = 1; // Insert
watermark - key
        else wtChoice[j] = 0; // Insert dummy
transformation
    }

    /* Copy lines from input file to result file */
    int m = ((Integer)randLocs.elementAt(0)).intValue();
    for(j=0;j<m;j++) {
        line = rdr.readLine();
        writer.println(line);
    }
    k=m;
}

```

```

/* Insert the 0th transformation to carry wm for scheme 2 */
line = prop_tranx.getProperty("0"); // String wrt contains key-transformation.
System.out.println("Line#" + m + " : Scheme2- Inserting Transformation#0");
writer.println(line);

/* Copy lines from input file to result file */
m = ((Integer)randLocs.elementAt(1)).intValue();
for(i=j;i<m;i++) {
    line = rdr.readLine();
    writer.println(line);
}
k=m;

/* Insert the 0th transformation to carry wm for scheme 2 */
line = prop_tranx.getProperty("0"); // String wrt contains key-transformation.
System.out.println("Line#" + m + " : Scheme2- Inserting Transformation#0");
writer.println(line);

/* Write to Transform file begins */
int p = 2, q=0;
for(int a = 0 ; a<id_NoOfDigits; a++) { // Is repeated (no-of-digits
in custid):: Times
    for(int b = 0 ; b<repeat[a];b++) { // Same tranx is inserted multiple
repeat[a] times
        m = ((Integer)randLocs.elementAt(p)).intValue();
        k++;
        if(nextLine.matches("")) ; else writer.println(nextLine);
        for(j=k; j < m; j++) { // Copy lines from inputfile to Transform
file.
            line = rdr.readLine();
            writer.println(line);
        }
        keyrdr.mark(10000);
        redflag1 = 0;
        redflag2 = 0;
        nextLine = rdr.readLine();
        if((nextLine.indexOf("ja")!=-1) || (nextLine.indexOf("jae")!=-1) ||
(nextLine.indexOf("jb")!=-1) ||
        (nextLine.indexOf("jbe")!=-1) || (nextLine.indexOf("je")!=-1) ||
(nextLine.indexOf("jg")!=-1) ||
        (nextLine.indexOf("jge")!=-1) || (nextLine.indexOf("jl")!=-1) ||
(nextLine.indexOf("jle")!=-1) ||
        (nextLine.indexOf("jc")!=-1) || (nextLine.indexOf("jo")!=-1) ||
(nextLine.indexOf("js")!=-1) ||
        (nextLine.indexOf("jnp")!=-1) || (nextLine.indexOf("jp")!=-1) ||
(nextLine.indexOf("jnbe")!=-1) || (nextLine.indexOf("jnb")!=-1) ||
(nextLine.indexOf("jnae")!=-1) ||
        (nextLine.indexOf("jna")!=-1) || (nextLine.indexOf("jz")!=-1) ||
(nextLine.indexOf("jnle")!=-1) ||
        (nextLine.indexOf("jnl")!=-1) || (nextLine.indexOf("jnge")!=-1) ||
(nextLine.indexOf("jng")!=-1) ||
        (nextLine.indexOf("jpo")!=-1) || (nextLine.indexOf("jpe")!=-1) ||
(nextLine.indexOf("jne")!=-1))
        {
            redflag1 = 1;
        }
        /* Write of dummy/key */
        /* Insert the KEY */
        if(wtChoice[q] == 1) { // Insert the KEY
            String wrt = prop_tranx.getProperty(s[a]); // String wrt contains key-
transformation.
            /* tranx 2 has definition of proc_tanu so has to be inserted with a
different name: proc_tanuI for each repetition.*/
            String wrt2 = "proc_tanu";
            if(s[a].matches("2")) {
                wrt2 = wrt2.concat(String.valueOf(b));
            }
        }
    }
}

```

```

        wrt = wrt.replaceAll("proc_tanu",wrt2);
    }
    /* If the nextline is a conditional jump statement */
    if(redflag1 == 1) {
        /* and we are trying to insert a mathematical transformation */
        if( (wrt.indexOf("xor")!= -1) || (wrt.indexOf("or")!= -1) ||
(wrt.indexOf("incl")!= -1) ||
            (wrt.indexOf("addl")!= -1) ) {
            redflag2 = 1;
        }
    } /* we will not insert any tranx because a conditional jump follows a
mathematical tranx */
    if ((redflag1 == 1) && (redflag2 == 1)) {
        b--; /* Since we will not be inserting the key-tranx we don't want
to increment b(repeat-value of key)*/
        p--;
    } else {
        writer.println(wrt);
        System.out.println("Line#"+m+" : KeyTransformation #"+s[a]);
    }
    } else {
        // Insert dummy
        String wrt = prop_dummy.getProperty(d[q]); // String wrt contains key-
transformation.
        if(redflag1 == 1) {
            /* and we are trying to insert a mathematical transformation */
            if( (wrt.indexOf("xor")!= -1) || (wrt.indexOf("or")!= -1) ||
(wrt.indexOf("incl")!= -1) ||
                (wrt.indexOf("addl")!= -1) ) {
                redflag2 = 1;
            }
        }
        if ((redflag1 == 1) && (redflag2 == 1)) {p--; b--; } // do nothing
        else {
            writer.println(wrt);
            System.out.println("Line#"+m+" : DummyTransformation #"+d[q]);
            rep++ ; // how many times we have to repeat this dummy
transformation.
            b--; // Since we just inserted a dummy-tranx, we don't want to
increment b(repeat-value of key)
            if(dumRepeat[q] == 3) {
                if(rep == 3) { rep = 0; q++; }
            } else if(dumRepeat[q] == 2) {
                if(rep == 2) { rep = 0; q++; }
            } else {
                rep = 0; q++;
            }
        }
    }
    q++;
    keyrdr.reset();
    k=j;
    p++;
}
}
writer.println(nextLine);
while((line = rdr.readLine())!=null) // prints the last few lines of file.
    writer.println(line);
} /* End of function write */

/*****
*
* Function : findrandLocs(int noOfRandLocs, Vector randLocs,
*                      LineNumberReader linerdr)
* Input :
*   randLocs      : An array which stores random nos. between (0-totalLinesInFile)
*                  Indicates the positions in file where insertion of
*                  transformations takes place.
*****/

```

```

*   noOfRandLocs : maximum possible size of array randLocs. Helps determine the
*                   size of other arrays used in the function.
*   linerdr      : Line-Numb-Reader for software assembly file
* Output :
*   totLines     : Returns the total number of lines in the software assembly
*                   file. Also, randomly selects line-numbs and stores it in vector
*                   randLocs. These determine where the actual insertions take
*                   place
* Issues        : Its difficult to generate truly random line-numbers evenly
*                   distributed throughout the file. The closer we get to achieving
*                   it, the higher is the probability of getting an
ArrayOutOfBounds
*                   exception in write function, as we try to access Vector
*                   randLocs whose values have not been set.
*
***** */

public static int findrandLocs(int noOfRandLocs, Vector randLocs,
LineNumberReader linerdr) throws IOException, ArrayIndexOutOfBoundsException
{
    String line = null;
    int totLines = 0, mulIncr = 0, prev = 0;
    int count = (int)(noOfRandLocs*0.99);
    int[] tmp;
    tmp = new int[count];

    /* FIND total number of lines in the InputFile */
    while((line = linerdr.readLine()) != null)    totLines ++;
    System.out.println(" FILE HAS "+totLines+" lines.");

    if(noOfRandLocs > totLines)
        System.out.println("\nALERT !!!! \nThe program may not work correctly unless
the degree-of-obfuscation/cust-id size is reduced. Press Cntl^C to exit.\n");
    else if (noOfRandLocs > totLines/2)
        System.out.println("\nCAUTION !!!! \nThe degree-of-obfuscation/cust-id size
is HIGH. For the program to run correctly, it is suggested you reduce these
variable values. Press Cntl^C to exit\n");
    else ;

    /* Generate random numbers and store in an array */
    for(int i =0;i<count; i++)
        tmp[i]= (int) (Math.round(Math.random()*totLines));

    Arrays.sort(tmp);

    /* Copy random no(line-nos) array to the vector */
    for(int i=0;i<count;i++) {
        if ((tmp[i]<totLines-10) && (tmp[i] > 10) && (prev!=tmp[i])) {
            randLocs.add(new Integer(tmp[i]));
            prev = tmp[i];
        }
    }
    return totLines;
}
/* End of function - findrandLocs */
} /* End of class WriteWM */

```