University of Arizona, Department of Computer Science

CSc 620 — Security Through Obscurity — Project 1

Christian Collberg
February 7, 2002

# 1 Introduction

The purpose of this first mini-project is for you to get familiar with the SandMark code-base and to implement your first, trivial, watermarking algorithm.

Make sure that you have downloaded the `smark2` and `smextern` directories from the `cvs` server:

```
cvs -d :ext:YourLogin@cvs.cs.arizona.edu:/cvs/cvs/wmark \
     checkout -dP smark2 smextern
```

Build the system according to the installation instructions in the manual.

# 2 Project and Project Team Selection

Do the following to get ready for this project:

1. Choose one of the mini-projects below.
2. Choose whom to work with (each team should consist of two students).
3. Pick a cool team name.
4. Come to my office to tell me who is on your team and which project you have chosen.
5. Make up some cool team T-shirts (optional).

**Project selection is on a "come-first-serve-first" basis! Don't wait to the last minute to choose your project — you may be left with the hardest/boringest one!**

## 2.1 Implementation

You should add your code to the `smark2 cvs` directory. Assume your chosen watermarking algorithm is named `WM1`. Do the following:

```
cd smark2/sandmark/watermark
cp -r static_template wm1
cvs add wm1
cd wm1
mv MyAlgorithm.template WM1.java
mv Test.template Test.java
cvs add WM1.java Test.java README doc
cd doc
cvs add help.html
```

Then edit `WM1.java`, `Test.java`, and `doc/help.html` to implement and document your algorithm. You may, of course, add any other files or sub-directories that you feel you need, as long as you only make changes to your own `wm1` directory.

To compile and run, you should be at the very top of the directory tree:

```
cd smark2
make
make run
```

When you have (at the very least) gotten rid of compilation errors, you can commit your code to the cvs repository:

```
cd smark2/sandmark/watermark/wm1
cvs commit
```

**Never commit code that does not compile cleanly! Whenever someone checks out a new version of SandMark they will get your code, too!**

## 2.2 Documentation

Finally, when your project is done, you should add documentation:

```
cd smark2/doc
cp watermarking.template WM1.tex
```

Edit the file `manual.tex` and add the lines below:

```
\part{\SM\ Algorithms}
\input{CollbergThomborson}
\input{ConstantString}
\hacker{My Name and My Partner's Name}  % <<<<<<< Add this line!
\input{WM1}                             % <<<<<<< Add this line!
```

Edit the `makefile` and add the line below:

```
SECTIONS = \
   ....\
   CollbergThomborson.tex\
   ConstantString.tex\
   WM1.tex \                            % <<<<<<< Add this line!
   ...
```

Edit the file `WM1.tex` to add your documentation. If you need to draw figures, use the `xfig` tool and store the `.fig`-file in the `FIGS` directory. You also need to make the "obvious" additions to the `makefile`.

Build the manual by saying:

```
make
make again
make again
```

If you don't know LaTeX, there is plenty of documentation on the web, or get a book from the bookstore or library.

# 3 Project Descriptions

Below I have outlined ten software watermarking projects for you to choose from. No project can be picked by more than one team. Many of the algorithm descriptions are deliberately vague; you may interpret them as you see fit. You can decide to implement something really clever, or something really dumb, I don't care. However, *your code must work*. In other words, you can implement the silliest software watermarking algorithm in the world (i.e., it is dead easy for an adversary to destroy the watermark) but the implementation cannot modify the user's program so that it no longer works.

## 3.1 Project 1: Rename Fields and Methods

Rename the fields and/or methods in the user's program such that one or more names embed the watermark. For example, to embed the watermark `1056` in the program

```
class C1 {
   int f1;
   int f2;
   void m1(){f2=f1+99; m2();}
   void m2(){}
   String toString(){}
}
```

we could modify it like this:

```
class C1 {
   int f1$10;
```

```
   int f2;
   void m1(){f2=f1$10+99; m2$56();}
   void m2$56(){}
   String toString(){}
}
```

This may seem easy, but there are some complications:

1. Not every method in a class can be renamed. For example, the `toString()` method in the above example cannot, since it overrides the `toString()` method in `java.lang.Object`.
2. If we rename a method, all references to that method, anywhere in the program, will have to be renamed.
3. If we rename a method `m()`, all methods that override this one will also have to be renamed.

The class `sandmark.util.ClassHierarchy` builds up the class hierarchy from a set of Java classes. It also contains methods which check whether a particular method can be renamed or not, and, if a method `m()` is renamed, which other methods will also have to be renamed. Also see `sandmark.obfuscate.nameoverloading.NameOverloading.java`.

Exactly how you embed and recover the watermark is up to you. An advanced algorithm would use the watermark key to seed a random number generator to pick out the locations of the code where the watermark is stored. A trivial algorithm would simply pick one method at random and somehow mark that this is the one that stores the watermark:

```
class C1 {
   void m2$1056(){String sm$ = "The watermark is in this method!!!";}
}
```

### 3.2 Project 2: Add Methods and Fields

This algorithm is the same as Project 1, in that we store the watermark in the names of the methods and fields of the program. However, in this case we add our own bogus methods and fields rather than modifying the user's ones. This is a bit easier, since we don't have to worry about renaming a method that cannot be renamed. On the other hand, whenever we add bogus code to the program we need to make sure that it doesn't stand out. Any method we add, for example, should contain "reasonable" code, and it should be called from at least one place in the program.

### 3.3 Project 3: Add Bogus Initializers

Add some bogus local variables to a method. The watermark is stored in the initial value these locals are given:

```
class C1 {
   void m(){
      ....
```

```
        int sm$1 = 10;
        int sm$2 = 56;
    }
}
```

Again, how you decide where to add the locals and how you find them again is entirely up to you. Relying on the name of a local variable is not such a good idea, since many Java compilers will strip this information out during compilation.

### 3.4 Project 4: Add Bogus Expressions

This is similar to Project 2, but here we store the watermark in the constant value computed by a bogus expression:

```
class C1 {
    void m(){
        ....
        int sm$1 = 400+600+56;
    }
}
```

To recognize the watermark you will need to look for the bytecode that computes the expression, and somehow "evaluate" it. It therefore makes sense to make the expression as simple as possible, maybe allowing integer addition as the only operation.

If you want to be even more clever you could add some random variables to the expression:

```
class C1 {
    void m(){
        ....
        int x = ...;
        int sm$1 = 400+x+600+56*x;
    }
}
```

When evaluating the expression you simply assume that `x=1` when you multiply with it and that `x=0` when you add to it, etc. Adding bogus variables to the expression would (maybe) prevent a bytecode optimizer from evaluating the expression and replace it with a constant.

### 3.5 Project 5: Add Bogus Switch Statements

Add one or more switch statements to the user's program, such that the case values stores the watermark:

```
class C1 {
    void m(){
```

```
      ...
      switch(e){
         case 10 : {...}
         case 56 : {...}
      }
      ...
   }
}
```

Again, the code in the switch arms must be "reasonable", as must the switch expression.

### 3.6 Project 6: Add Bogus Method Calls

Add a few bogus methods to the user's program. Store the watermark in the *order* in which these methods are called. For example, to hide the watermark 1056 we could add the following code:

```
class C1 {
   void sm$1(){int sm$x = 1; ...}
   void sm$5(){int sm$x = 5; ...}
   void sm$6(){int sm$x = 6; ...}
   void sm$0(){int sm$x = 0; ...}
   void m(){
       ...
      sm$1();
      sm$0();
      sm$5();
      sm$6();
       ...
   }
}
```

How you identify the methods is up to you. In the example above we've added some special code to the beginning of each bogus method that indicates what digit it encodes.

### 3.7 Project 7: Add Bogus Print Statements

Add one or more bogus print-statements to the user's program. The argument embeds (a part of) the watermark:

```
class C1 {
   void m(){
       ...
       System.out.println(10);
       System.out.println(56);
       ...
   }
}
```

Obviously, we have to make sure that the new statements are never executed. You can do that by wrapping them in an if-statement where you make sure that the body is never executed:

```
void m(){
    int e = 99;
    if (e < 0)
       System.out.println(1056);
}
```

in the `catch`-part of an exception-handler where the exception can never be thrown:

```
void m(){
    try {...}
    catch (VeryWeirdException e) {
       System.out.println(1056);
    }
}
```

in a loop that will never execute:

```
void m(){
    int e = 99;
    while (e > 100) {
       System.out.println(1056);
       e++;
    }
}
```

and so on. Ideally, the code you add should fit in with the code that is already in the method. Again, how you identify where the watermark is embedded is entirely up to you. In a simplistic implementation you could, for example, add a bogus `print`-statement to mark out the location of the code:

```
void m(){
    int e = 99;
    if (e < 0) {
       System.out.println("The watermark starts here!");
       System.out.println(1056);
    }
}
```

## 3.8 Project 8: Add Bogus Predicates

Add one or more bogus `if`-statements to the user's program. The boolean expression embeds the watermark:

```
class C1 {
   void m(){
```

```
        int sm$x = 99;
        ...
        if (sm$x == 10) {...}
        ...
        if (sm$x == 56) {...}
        ...
    }
}
```

Obviously, `while`-statements will work equally well, as long as the semantics of the program doesn't change.

### 3.9 Project 9: Add Bogus Locals

Add one or more bogus local variables to some method in the program. Encode the watermark in the types of these locals. For example, you could make up a table that maps common types to the digits of some base-$k$ number system:

| TYPE | DIGIT |
|---|---|
| `int` | 0 |
| `float` | 1 |
| `short` | 2 |
| `java.lang.Object` | 3 |
| `double` | 4 |
| `java.util.Hashtable` | 5 |
| `UserClass1` | 6 |
| `java.util.Properties` | 7 |
| `UserClass2` | 8 |
| `int[]` | 9 |

Embedding the watermark 1056 using this base-10 encoding would look like this:

```
class C1 {
    void m(){
        float sm$1;
        int sm$2;
        java.util.Hashtable sm$3;
        UserClass1 sm$4;
    }
}
```

Ideally, you should add bogus code to the method so that it appears as if these variables are actually used.

### 3.10 Project 10: Weird Jumps

In Java bytecode you can create control structures that you cannot create in Java source code, since in bytecode you have `goto`-statements which Java doesn't have. We can use this to embed a watermark by adding a set of weird jumps to the code. Here's a simple example how this can be done:

```
class C1 {
   void m(){
      ...
      if (false) {
         goto L1
         goto L0
         goto L5
         goto L6
         L0:
            goto EXIT
         L1:
            nop
            goto EXIT
         L5:
            nop
            push 1
            push 2
            add
            store sm$x
            goto EXIT
         L6:
            push 1
            push 2
            add
            push 3
            add
            store sm$x
            goto EXIT
         EXIT:
      }
      ...
   }
}
```

The idea here is that the number of instructions between a label (a location that we jump to in the bytecode) and the next jump represents a base-10 digit. The initial jumps (to `L0`, `L1`, `L5`, and `L6`) therefore encodes the watermark 1056.

The main issue with this idea is that we have to be careful to generate code that the verifier won't reject. The code in the example above, for example, would probably not pass the verifier since a large part of the code is not reachable.

### 3.11 Project 11: Your Own Algorithm

Invent your own static watermarking algorithm! Remember that it should be simple enough to implement in about a week. You may take inspiration from any of the algorithms above.

You should discuss your idea with me before getting started.

# 4 Administrivia

This assignment is due Sunday, February 17. It is worth 5% of your final grade. (The projects in this class are worth a total of 60% of your grade. There will be three projects: two mini-projects worth 5% each, and one final project worth 50% percent of the total grade.)

> **This is a team assignment! You may freely read the code already submitted by others! You may ask for assistance from other students! You may make use of any resources you find on the web or elsewhere!**
>
> **However, make sure to attribute the source of any help you get.**