SandMark User's Guide

Christian Collberg

January 28, 2003

Contents

Chapter 1

Introducing SandMark

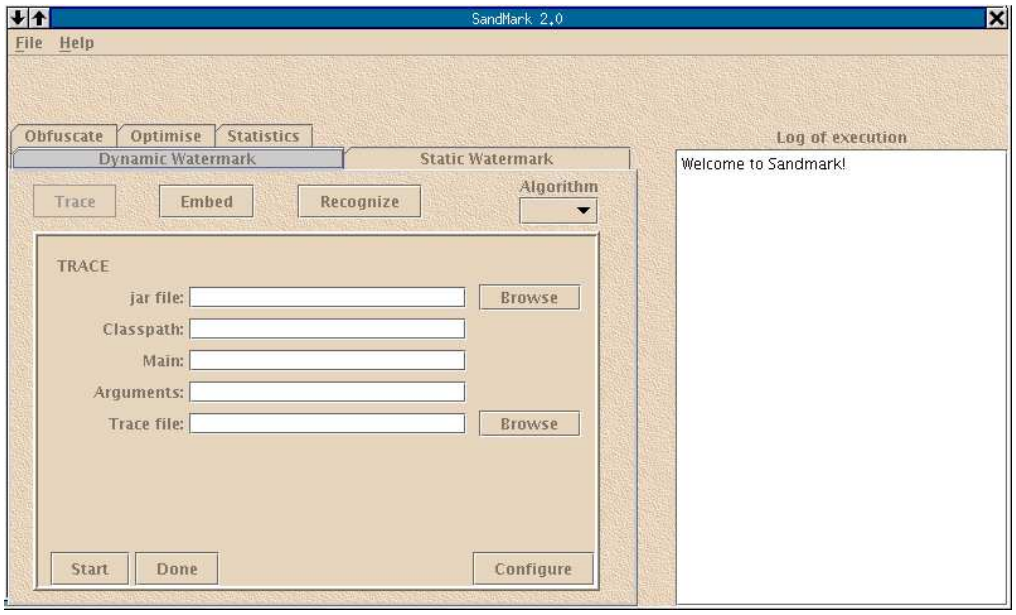
1.1 Introduction

SandMark is designed to be the Swiss-Army-Chainsaw of software protection research. In other words, we hope to build an infrastructure that makes it easy to implement algorithms for

1. code obfuscation,
2. softeware watermarking, and
3. tamper-proofing.

In fact, we hope to implement *every* software protection algorithm know to man, so that we can compare and evaluate them.

You normally interact with sandmark through its graphical interface. Start it by typing `make run` in the `smark3` directory:



Trying out the Watermarker

To get started try watermarking the `TTT` (tic-tac-toe) application. It can be found in the `smapps2` directory. Start SandMark, then do the following:

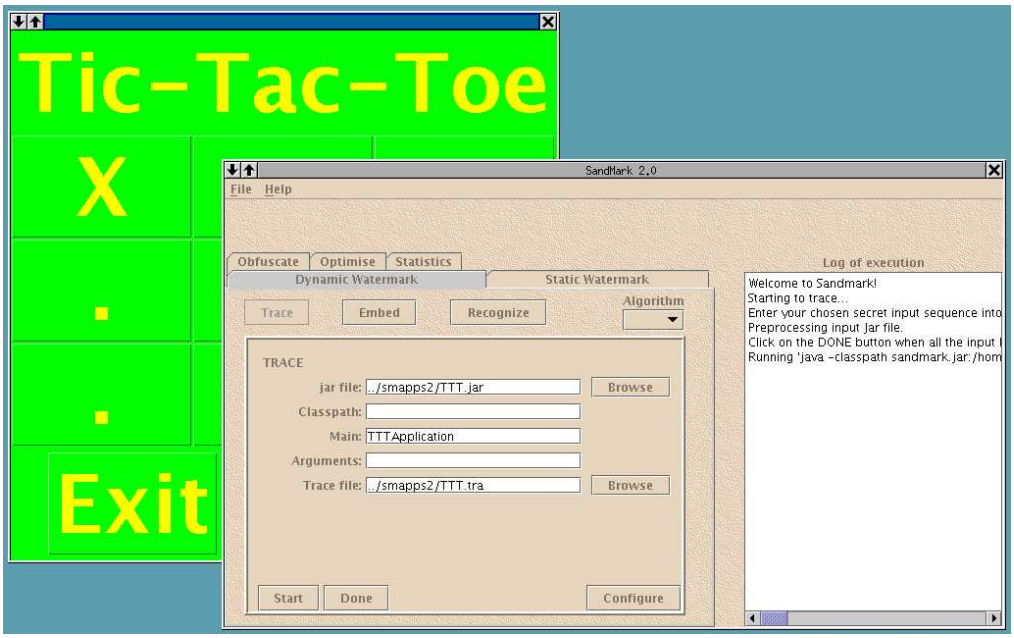
1. In the *trace* pane enter

   Jar-file to watermark: `TTT.jar`

   Main class name: `TTTApplication`

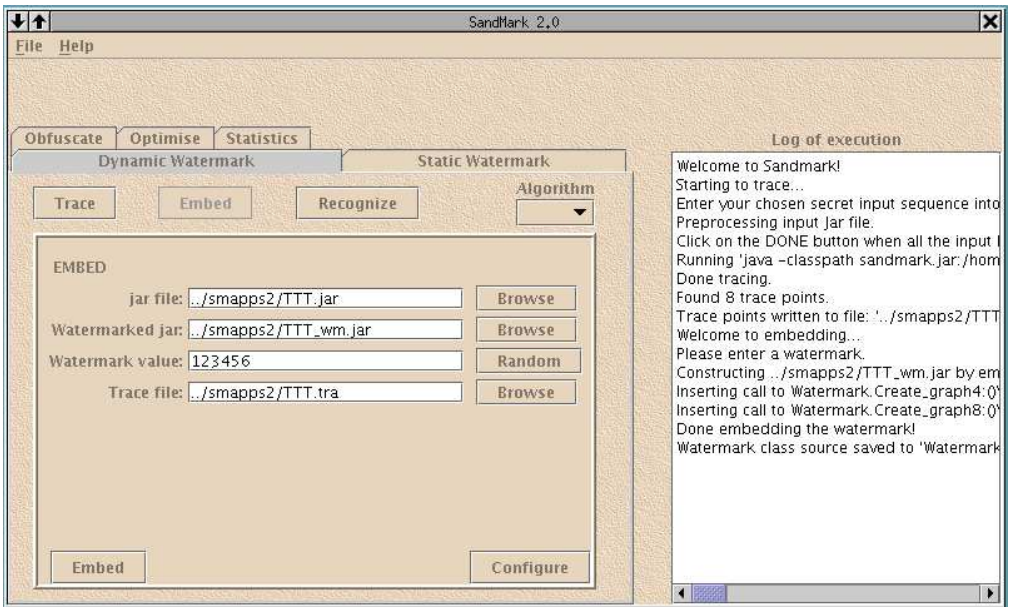
2. Hit START.

3. Click on a few X's and O's. Remember the order in which you do the clicks! It should look something like this:

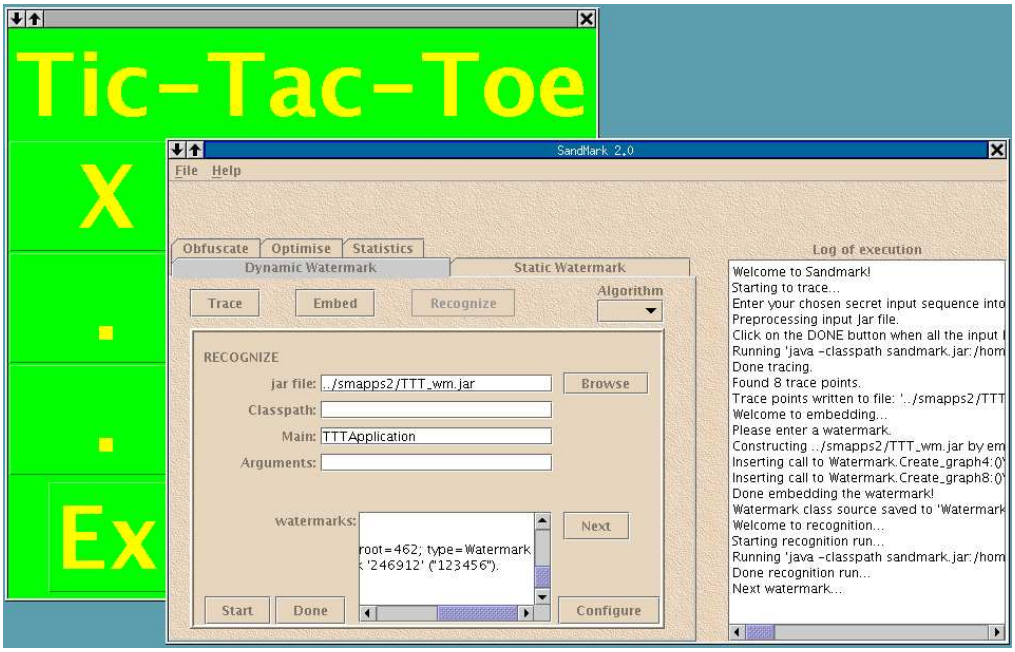


4. Hit DONE.

5. Go to the *embed* pane and enter the watermark value `123456`, like this:



6. Hit EMBED.
7. Go to the *recognize* pane and hit the START button.
8. Click on the same X's and O's as you did in step 3), in the same order.
9. Hit DONE:



You should see the watermark `123456` extracted from the watermarked `TTT` application.

You can also try

```
> smark -f ../smapps2/TTT.script
```

This will run a script that traces, embeds, and recognizes a watermark in the TTT application. You still have to enter the X's and O's and hit the DONE buttons in the trace and recognize panes.

1.2 Installing SandMark

SandMark's source code is stored in a CVS repository at `cvs.cs.arizona.edu`. You can get the source anonymously (in which case you can't make any changes to it): To get the sources anonymously, do the following

```
> setenv CVS_RSH ssh
> cvs -d :pserver:anonymous@cvs.cs.arizona.edu:/cvs/wmark login
> cvs -d :pserver:anonymous@cvs.cs.arizona.edu:/cvs/wmark \
   checkout -P smark3 smextern3 smapps3 smbloat3 smbin3 smtest3
```

or, if you have write access to SandMark (i.e an account on `cvs.cs.arizona.edu`) you should instead do

```
> setenv CVS_RSH ssh
> cvs -d :ext:MyLogin@cvs.cs.arizona.edu:/cvs/cvs/wmark \
      checkout -P smark3 smextern3 smapps3 smbloat3 smbin3 smtest3
```

where `MyLogin` is your account name on `cvs.cs.arizona.edu`.

You should now have four directories:

smark3: The SandMark sources.

smbin3: Scripts.

smextern3: External Java code i.e. jar and zip files needed to run SandMark.

smapps3: Some simple applications you can use to try out SandMark

smbloat3: Some simple test cases for Bloat. Read these to get a feel for how Bloat should be used.

smtest3: A test suite for SandMark.

Once you have checked out SandMark you can get the latest version by running the `cvs update` command:

```
> cd smark3
> cvs update -dP
```

To add a new file you do `cvs add file`, to remove it `cvs rm file` and to commit your changes to the repository at `cvs.cs.arizona.edu` you say `cvs commit`.

Building SandMark

Note that you will need Java 1.4 to run SandMark properly. Get the latest version from `http://java.sun.com/j2se/1.4`.

Do the following to build SandMark:

```
> cp smark3/Makedefs.std smark3/Makedefs
# Make the 'obvious' changes to Makedefs.
# In particular, you should set these variables:
#   JDK =
#   HOME =

> cp smbin3/smark.std smbin3/smark
# Make the 'obvious' changes to smark.
# Again, you should set these variables:
#   JDK =
#   HOME =

> cp smapps3/Makedefs.std smapps3/Makedefs
# Make the same obvious changes as in smark3/Makedefs

> make -C smark3

# Build applications to watermark
> make -C smapps3

# Start SandMark
> ./smbin3/smark
```

SandMark gets compiled into a jar-file `sandmark.jar`. To execute it you also need some other packages (`bloat`, `BCEL`, etc.), which can be found in the `smextern` directory. The `smark` script takes care of setting Java's classpath correctly so that these get picked up.

You can verify that SandMark works using the test suite:

```
# Set some environment variables.
# The current directory should contain
# smtest3, smark3, smextern3, etc.
> export SMEXTERN=$PWD/smextern3/
> export SMJAR=$PWD/smark3/sandmark.jar

# JDK_ROOT should contain bin/, lib/, jre/
# and other stuff
> export JDK_ROOT=/path/to/jdk/

# run the test script
> ./smtest3/bin/runtests
```

See smtest3/README for details of how to add and remove tests.
You can also build the manual:

```
> make -C smark3/doc/ manuals.ps
> make -C smark3/doc/ manuals.ps
```

Finally, you can generate html from the JavaDoc comments in the source:

```
> make -C smark3 jdoc
```

This generates a directory `jdoc` of html files.

A minimal installation of SandMark contains 5 files: sandmark.jar, bloat-1.0.jar, BCEL.jar, grappa1_2.jar, and smark3. bloat-1.0.jar, BCEL.jar, and grappa1_2.jar should all be in a directory called smextern3, sandmark.jar should be in a directory called smark3, and these 2 directories should have the same parent directory. smark3 can have any path.

1.3 SandMark in Windows

To run SandMark in Windows, a minimal installation is necessary, along with a java runtime environment version 1.4.0 or greater. Before running SandMark, you must set the environment variable CLASSPATH (case insensitive in Windows) to include the path of sandmark.jar BCEL.jar and bloat-1.0.jar.

Each path in the classpath is seperated by a semicolon. Once the environment variables are set, to run SandMark, open a command prompt window (Start Menu → Run, and type in `command` and press enter) and type `java sandmark.Console`.

1.4 Scripting SandMark

SandMark can be scripted. Either start SandMark from the command line with the `-f` option:

```
> smark -f file.script
```

or enter the script file in SandMark's `file` pull-down manu.

- You can set a property value using the command

  ```
  set PROPERTY VALUE
  ```

  The following property values are recognized:

  DWM_CT_AnnotatorClass:

  DWM_CT_Encode_NodeType:

  DWM_CT_Encode_ParentClass:

DWM_CT_Encode_ClassName:

DWM_CT_Encode_AvailableEdges:

DWM_CT_Encode_StoreWhat:

DWM_CT_Encode_StoreMethods:

DWM_CT_Encode_StoreLocation :

DWM_CT_Encode_ProtectionMethods:

DWM_CT_Encode_IndividualFixups:

DWM_CT_Encode_Encoding: Should be one of `perm` or `radix`. `"*"` picks a random encoding method.

DWM_CT_Encode_Components:

DWM_CT_Encode_Package:

DWM_MaxTracePoints:

DWM_CT_Encode_StoreLocation: One of `formal` or `global`.

DWM_CT_DumpIR:

DWM_ClassPath:

- To run tracing use the command

  ```
  trace input.jar trace.tra MAINCLASS ARGUMENTS
  ```

  The classpath is set through the command

  ```
  set ClassPath ...
  ```

  Tracepoints are written to the `trace.tra` file.

- Embed watermark in input.jar using the command

  ```
  embed input.jar output.jar watermark trace.tra
  ```

  Read the tracepoints from the file `trace.tra`.

- Obfuscate input.jar, creating output.jar:

  ```
  obfuscate input.jar output.jar
  ```

- Run recognition.

  ```
  recognize input.jar watermark_count MAINCLASS ARGUMENTS
  ```

  The classpath is set through the command

  ```
  set ClassPath ...
  ```

- If the first non-blank character on a line is # the rest of the line is ignored.

- Commands are case insensitive, arguments are case sensitive.

Chapter 2

The SandMark Code-base

2.1 Dynamic Class Loading

Dynamic class loading is used in several places in sandmark to allow for extensibility. The major problem in dynamic class loading in sandmark is finding classes to load. In particular, sandmark can be run from a directory or a jar file, and it may or may not know the location of that directory or jar file or even which of the two it is running from.

All of this detail is abstracted by sandmark.util.classloading.ClassFinder. ClassFinder allows a caller to specify one of a small number of classes (defined in sandmark.util.classloading.IClassFinder), and ClassFinder will return the names of all classes derived from that class. For example, the following code prints to standard output all classes derived from sandmark.Algorithm.

```
1  java.util.Collection c =
2     sandmark.util.classloading.ClassFinder.getClassesWithAncestor
3        (sandmark.util.classloading.IClassFinder.ALGORITHM);
4  java.util.Iterator it = c.iterator();
5  while(it.hasNext()) {
6     String className = (String)c.next();
7     System.out.println(className);
8  }
```

The class names returned by ClassFinder are those returned by an implementor of sandmark.util.classloading.IClassFinder. ClassFinder uses the first implementation in the following list whose default constructor does not throw an exception:

1. **FileClassFinder** This class reads class names and from which of the interfaces in sandmark.util.classloading.IClassFinder they derive from a file call **Algorithms.txt**. This file is opened by a call to **ClassLoader.getSystemClassLoader().getResourceAsStream("Algorithms.txt")**. If this call fails, FileClassFinder's default constructor throws an exception.

2. **JarClassFinder** This class searches through all entries in the jar file named by System.getProperty("SMARK_PATH"), loads all class files contained therein, and determines from which classes in IClassFinder each of these classes derives. If opening the jar file for reading throws an exception, JarClassFinder's default constructor throws an exception.

3. **DirClassFinder** This class is similar to JarClassFinder, but it searches through the file system hierarchy rooted at System.getProperty("SMARK_ROOT"). Its default constructor throws an exception if the contents of SMARK_ROOT cannot be read.