SandMark User's Guide

Christian Collberg

July 3, 2002

Contents

Chapter 1

Introducing SandMark

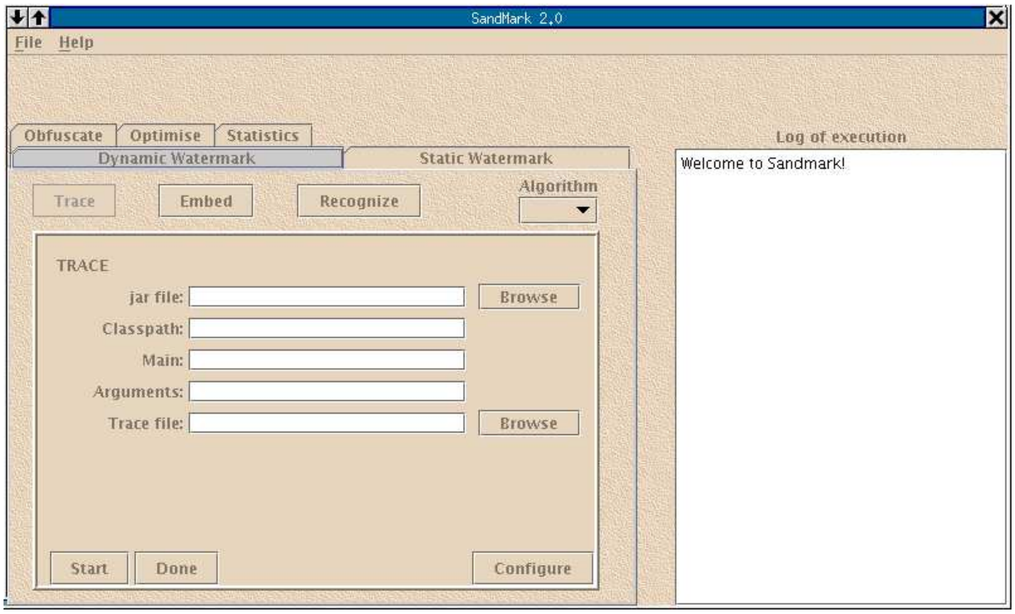
1.1 Introduction

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 `smark` in the `smbin` 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:

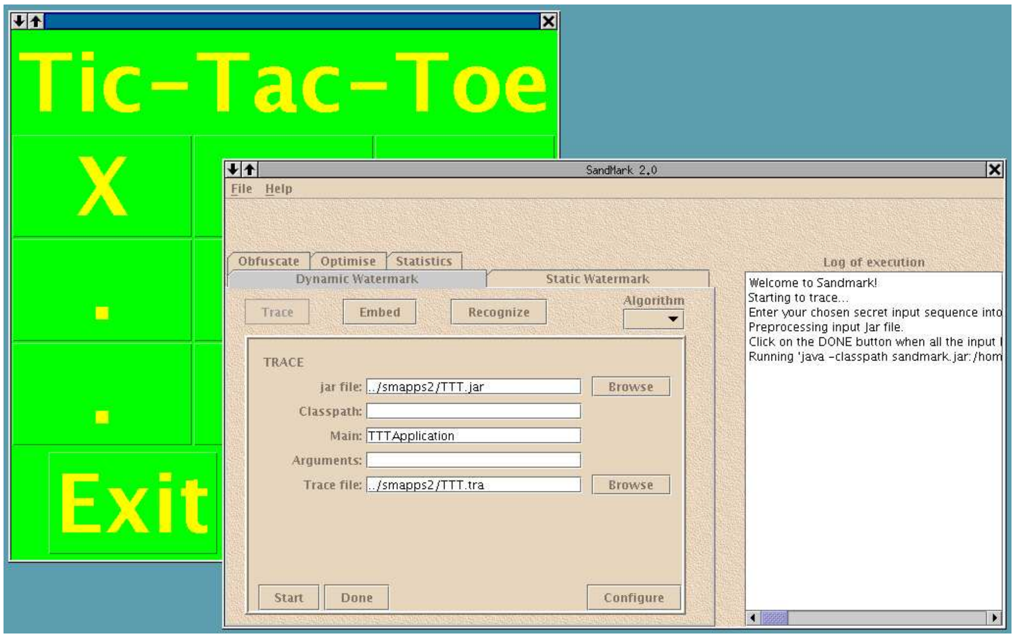
1. In the *trace* pane enter

   Jar-file to watermark: `TTT.jar`

   Main class name: `TTTApplication`

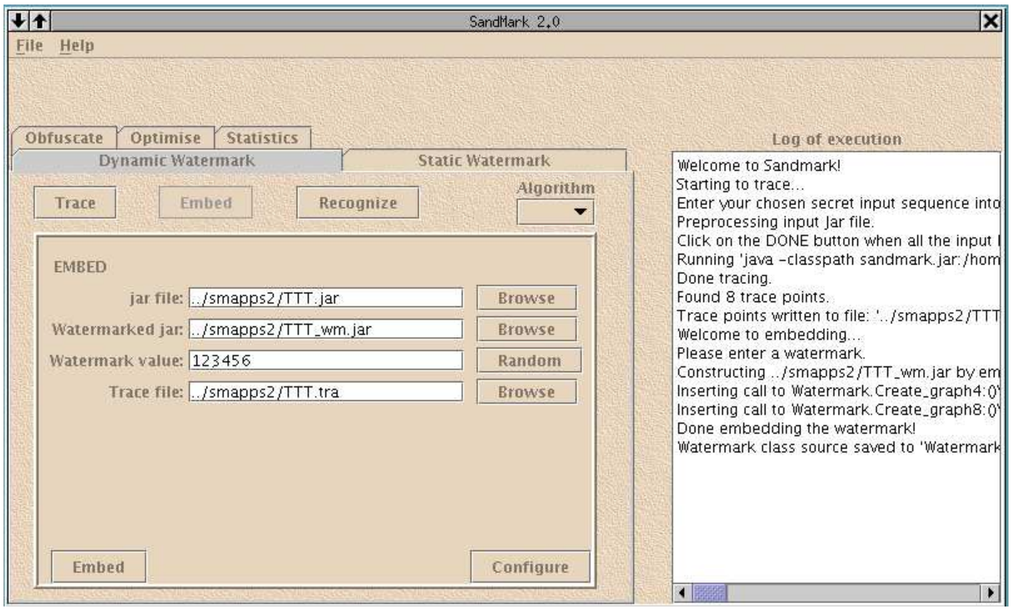
2. Hit $\boxed{\text{START}}$.

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 $\boxed{\text{DONE}}$.

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

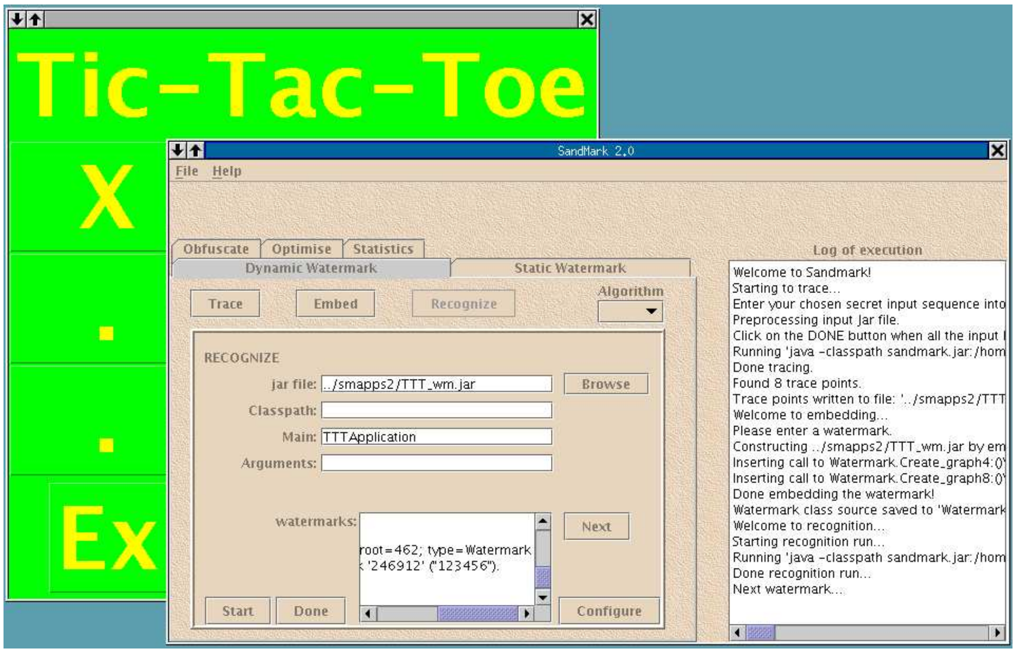


6. Hit $\boxed{\text{EMBED}}$.

7. Go to the *recognize* pane and hit the $\boxed{\text{START}}$ button.

8. Click on the same X's and O's as you did in step 3), in the same order.

9. Hit $\boxed{\text{DONE}}$:



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 $\boxed{\text{DONE}}$ 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 smark2 smextern smapps2 smbloat2 smbin
```

or, if you have got 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 smark2 smextern smapps2 smbloat smbin
```

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

You should now have four directories:

smark2: The SandMark sources.

smbin: Scripts.

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

smapps2: Some simple applications you can use to try out SandMark

smbloat2: Some simple test cases for Bloat. Read these to get a feel for how Bloat should be used.

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

```
> cd smark2
> 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 smark2/Makedefs.std smark2/Makedefs
# Make the 'obvious' changes to Makedefs.
# In particular, you should set these variables:
#   JDK =
#   HOME =

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

> chmod a+rx smbin/smark
> make

# Build applications to watermark
> cd smapps2
> make

# Start SandMark
> smbin/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 also build the manual:

```
> cd smark2/doc
> make
> make again
> make again
```

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

```
> make jdoc
> make jtex
```

This generates a PostScript file `classes.ps` and a directory `jdoc` of html files.

1.3 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 SandMark Packages

SandMark is a large application. Currently it is made up of some 29000 lines of java code distributed over 200 classes. The code is organized in a fairly deep package hierarchy.

sandmark

Classes at the top level:

sandmark.Console: The main entry point to SandMark. It contains code that starts up the GUI and acts as an interface between the GUI and the rest of the application.

sandmark.CLI: The command line interpreter.

sandmark.Scripting: The script interpreter.

sandmark.gui

`sandmark.gui` contains the code that build up the graphical interface.

sandmark.gui.AboutDialog:

sandmark.gui.ButtonRenderer:

sandmark.gui.CodeDialog:

sandmark.gui.EmbedConfigDialog:

sandmark.gui.ExtensionFileFilter:

sandmark.gui.GUIListener:

sandmark.gui.HelpFrame:

sandmark.gui.IntegerInput:

sandmark.gui.LayoutConstraints:

sandmark.gui.LightRenderer:

sandmark.gui.MultiHeaderRenderer:

sandmark.gui.ObTableModel:

sandmark.gui.ObfuscateConfigDialog:

sandmark.gui.RelativeLayout:

sandmark.gui.SandMarkFrame:

sandmark.gui.SandMarkGUIConstants:

sandmark.gui.SkinPanel:

sandmark.gui.StatDialog:

sandmark.gui.StatTableModel:

sandmark.gui.TablePanel:

sandmark.gui.SMarkGUIConstants:

sandmark.gui.SandMarkOSConstants:

sandmark.html

`Html` help files are kept here.

sandmark.obfuscate

Each obfuscation lives in its own subdirectory in `sandmark.obfuscate`. `sandmark.obfuscate.setfieldspublic`, for example, is an obfuscator that sets all fields to `public`.

sandmark.obfuscate.AllClassesObfuscator:

sandmark.obfuscate.AppObfuscator: The abstract base-class from which any obfuscator that works on an entire application should inherit. It inherits from `sandmark.obfuscate.GeneralObfuscator`.

sandmark.obfuscate.ClassObfuscator: The abstract base-class from which any obfuscator that works on a single class should inherit. It inherits from `sandmark.obfuscate.GeneralObfuscator`.

sandmark.obfuscate.GeneralObfuscator: The top-level abstract class for an obfuscator.

sandmark.obfuscate.MethodObfuscator: The abstract base-class from which any obfuscator that works on a single method should inherit. It inherits from `sandmark.obfuscate.GeneralObfuscator`.

sandmark.obfuscate.NameOverloadObfuscator: An advanced name obfuscator based on Paul Tyma's patent.

sandmark.obfuscate.Obfuscate: This class handles the incoming obfuscation requests from the GUI. It is called by `sandmark.Console`.

sandmark.obfuscate.Obfuscator: The top-level obfuscator that decides on overall strategy – which algorithms should be applied to which pieces of code.

sandmark.obfuscate.setfieldspublic

An obfuscator that sets all fields and methods to public.

sandmark.obfuscate.setfieldspublic.SetFieldsPublic:

sandmark.obfuscate.splitclass

An obfuscator that splits a class in two pieces. Non-functional.

sandmark.obfuscate.splitclass.DummyClass:

sandmark.obfuscate.splitclass.Obf:

sandmark.obfuscate.splitclass.SplitClass:

sandmark.obfuscate.util

`sandmark.obfuscate.util` contains classes of interest to all obfuscators.

sandmark.optimise

Routines to optimize Java programs. Currently based solely on the BLOAT package.

sandmark.optimise.Optimise:

sandmark.optimise.Main:

sandmark.optimise.Optimiser:

sandmark.util

`sandmark.util` contains classes of interest to all SandMark tools.

sandmark.util.Compile:

sandmark.util.BCEL: Utility routines to work with the `BCEL` (formerly `JavaClass`) package.

sandmark.util.ByteCodeLocation: Represents a location within a Java application. Essentially a tuple

$$\langle \texttt{sandmark.util.MethodID}, \texttt{lineNumber}, \texttt{byteCodeIndex} \rangle.$$

sandmark.util.CallGraphNode: The node in a call graph.

sandmark.util.CircularBuffer: A bounded buffer of arbitrary objects.

sandmark.util.ClassFileCollection: Routines for loading the classes of the program to be watermarked, obfuscated, etc. A class can be parsed into a `BCEL` or a `BLOAT` object.

sandmark.util.ClassHierarchy: A package for building the complete class hierarchy from a jarfile. There are many routines for querying the hierarchy, such as which methods override a particular method, which classes extend a particular class, etc.

sandmark.util.DependencyGraph: Used by `sandmark.obfuscate.splitclass.SplitClass`.

sandmark.util.Editor: Utility class for use with `BLOAT`.

sandmark.util.EditedClass: A class for maintaining a cache of classes and methods edited with `BCEL`.

sandmark.util.FileClassLoader: Used to load classes on the fly.

sandmark.util.FindClassFiles: Used to load classes on the fly.

sandmark.util.GraphViewer: Graphical display of `sandmark.util.graph.Graph`.

sandmark.util.InstructionTree:

sandmark.util.LabeledGrid:

sandmark.util.Log: Routines for printing out messages (both to a file and to the screen) to the user.

sandmark.util.MethodID: Represents a method within a Java application. Essentially a tuple

$$\langle \texttt{name}, \texttt{signature}, \texttt{className} \rangle.$$

sandmark.util.MethodNode:

sandmark.util.Misc:

sandmark.util.Options: Routines for cracking command line parameters.

sandmark.util.PriorityQueue:

sandmark.util.SparseVector: A specialized version on `java.util.Vector`.

sandmark.util.StackFrame: Used to build call graphs.

sandmark.util.StatisticsRecord:

sandmark.util.Stats:

sandmark.util.StringInt: A routine to package up an int inside a string.

sandmark.util.TempDir: Routines to manage temporary directories.

sandmark.util.Time: Routines for timing Java programs.

sandmark.util.exec

`sandmark.util.exec` contains classes that allow SandMark to run another Java application under debugging. This uses Java's JDI package. JDI allows you to start up a Java program, set breakpoints, trace method calls, examine variables, etc. We use this in the implementation of the trace and recognition steps of the dynamic watermarking algorithms.

sandmark.util.exec.Breakpoint: A class for creating breakpoints.

sandmark.util.exec.DumpAll:

sandmark.util.exec.EventHandler:

sandmark.util.exec.EventThread:

sandmark.util.exec.Heap: A class for iterating through all the objects on the heap.

sandmark.util.exec.HeapData: Objects returned by `sandmark.util.exec.Heap`.

sandmark.util.exec.MethodCallData:

sandmark.util.exec.Output: A class that handles input and output from the application being executed.

sandmark.util.exec.Overseer: Extend this class to run a program under JDI.

sandmark.util.exec.TracingException:

sandmark.util.graph

Many watermarking algorithms make use of graphs. This package contains a multigraph package `sandmark.util.graph.Graph` as well as operations on such graphs, such as all-pairs-shortest-path, depth-first-search, etc.

sandmark.util.graph.GraphOp: Additional algorithms on graphs.

sandmark.util.graph.Dfs: Classify the edges of a graph as `tree`, `cross`, and `back`.

sandmark.util.graph.Edge: Edge objects of a `sandmark.util.graph.Graph`.

sandmark.util.graph.Graph: A class for constructing multi-graphs.

sandmark.util.graph.Matrix: A class where graphs are represented as an adjacency matrix.

sandmark.util.graph.Node: Node objects of a `sandmark.util.graph.Graph`.

sandmark.util.graph.Path: A class for representing a path in a `sandmark.util.graph.Graph`.

sandmark.util.graph.codec

This package implements various algorithms for embedding a number into the topology of a graph.

sandmark.util.graph.codec.DecodeFailure:

sandmark.util.graph.codec.GraphCodec: Base-class for graph codecs.

sandmark.util.graph.codec.PPCT: Non-functional.

sandmark.util.graph.codec.PermutationGraph: A permutation graph encoding.

sandmark.util.graph.codec.RadixGraph: A radix graph encoding.

sandmark.util.javagen

`sandmark.util.javagen` contains classes for building up Java methods and classes. It is essentially a layer on top of `BCEL`. `sandmark.util.javagen` allows you to build up a Java AST and then turn that into bytecode.

sandmark.util.javagen.AssignField:

sandmark.util.javagen.AssignIndex:

sandmark.util.javagen.AssignStatic:

sandmark.util.javagen.Block:

sandmark.util.javagen.Cast:

sandmark.util.javagen.Class:

sandmark.util.javagen.Comment:

sandmark.util.javagen.CondNotNullExpr:

sandmark.util.javagen.Discard:

sandmark.util.javagen.EmptyStatement:

sandmark.util.javagen.Expression:

sandmark.util.javagen.Field:

sandmark.util.javagen.FieldRef:

sandmark.util.javagen.Formal:

sandmark.util.javagen.IfNotNull:

sandmark.util.javagen.Java:

sandmark.util.javagen.List:

sandmark.util.javagen.LiteralInt:

sandmark.util.javagen.LiteralString:

sandmark.util.javagen.LoadIndex:

sandmark.util.javagen.Local:

sandmark.util.javagen.LocalRef:

sandmark.util.javagen.Method:

sandmark.util.javagen.MyClass:

sandmark.util.javagen.New:

sandmark.util.javagen.NewArray:

sandmark.util.javagen.Null:

sandmark.util.javagen.Return:

sandmark.util.javagen.Statement:

sandmark.util.javagen.StaticCall:

sandmark.util.javagen.StaticFunCall:

sandmark.util.javagen.StaticRef:

sandmark.util.javagen.Test:

sandmark.util.javagen.Try:

sandmark.util.javagen.VirtualCall:

sandmark.util.javagen.VirtualFunCall:

sandmark.watermark

This package contain all the watermarking algorithms. Each algorithm resides in its own directory.

sandmark.watermark.DynamicEmbed: Handles the embedding phase of a dynamic watermarker.

sandmark.watermark.DynamicRecognize: Handles the recognition phase of a dynamic watermarker.

sandmark.watermark.DynamicTrace: Handles the tracing phase of a dynamic watermarker.

sandmark.watermark.DynamicWatermarker: The base-class for all dynamic watermarkers.

sandmark.watermark.GeneralWatermarker: The base-class of all watermarkers

sandmark.watermark.StaticWatermarker: The base-class for all static watermarkers.

sandmark.watermark.StaticEmbed: Handles the embedding phase of a static watermarker.

sandmark.watermark.StaticRecognize: Handles the recognition phase of a static watermarker.

sandmark.watermark.WatermarkingException:

sandmark.watermark.Watermarking:

sandmark.watermark.CT

This package implements the Collberg-Thomborson dynamic watermarking algorithm.

sandmark.watermark.CT.CT: The main class for the Collberg-Thomborson algorithm. Extends `sandmark.watermark.DynamicWatermarker`.

sandmark.watermark.CT.embed

This package implements the embedding of a graph into a Java program. This is part of the Collberg-Thomborson watermarking algorithm.

sandmark.watermark.CT.embed.AddParameters:

sandmark.watermark.CT.embed.CallGraphPath:

sandmark.watermark.CT.embed.DeleteMarkCalls:

sandmark.watermark.CT.embed.EmbedData:

sandmark.watermark.CT.embed.Embedder:

sandmark.watermark.CT.embed.InsertStorageCreators:

sandmark.watermark.CT.embed.PrepareTrace:

sandmark.watermark.CT.embed.ReplaceMarkCalls:

sandmark.watermark.CT.encode

This package implements the encoding of a graph into a Java program. This is part of the Collberg-Thomborson watermarking algorithm.

sandmark.watermark.CT.encode.Encoder:

sandmark.watermark.CT.encode.Graph2IR:

sandmark.watermark.CT.encode.Split:

sandmark.watermark.CT.encode.ir

This package implements the encoding of a graph into a simple intermediate code. This is part of the Collberg-Thomborson watermarking algorithm.

sandmark.watermark.CT.encode.ir.AddEdge:

sandmark.watermark.CT.encode.ir.Build:

sandmark.watermark.CT.encode.ir.Construct:

sandmark.watermark.CT.encode.ir.Create:

sandmark.watermark.CT.encode.ir.CreateNode:

sandmark.watermark.CT.encode.ir.CreateStorage:

sandmark.watermark.CT.encode.ir.Debug:

sandmark.watermark.CT.encode.ir.Destroy:

sandmark.watermark.CT.encode.ir.Destruct:

sandmark.watermark.CT.encode.ir.Field:

sandmark.watermark.CT.encode.ir.Fixup:

sandmark.watermark.CT.encode.ir.FollowLink:

sandmark.watermark.CT.encode.ir.Formal:

sandmark.watermark.CT.encode.ir.IR:

sandmark.watermark.CT.encode.ir.Init:

sandmark.watermark.CT.encode.ir.List:

sandmark.watermark.CT.encode.ir.LoadNode:

sandmark.watermark.CT.encode.ir.Method:

sandmark.watermark.CT.encode.ir.NodeStorage:

sandmark.watermark.CT.encode.ir.PrintGraph:

sandmark.watermark.CT.encode.ir.ProtectRegion:

sandmark.watermark.CT.encode.ir.SaveNode:

sandmark.watermark.CT.encode.ir.StaticCall:

sandmark.watermark.CT.encode.ir2ir

This package implements transformations of the intermediate code used in the encoding of graphs in the Collberg-Thomborson watermarking algorithm.

sandmark.watermark.CT.encode.ir2ir.AddFields:

sandmark.watermark.CT.encode.ir2ir.AddFormals:

sandmark.watermark.CT.encode.ir2ir.Builder:

sandmark.watermark.CT.encode.ir2ir.CleanUp:

sandmark.watermark.CT.encode.ir2ir.Debug:

sandmark.watermark.CT.encode.ir2ir.Destructors:

sandmark.watermark.CT.encode.ir2ir.InlineFixups:

sandmark.watermark.CT.encode.ir2ir.Protect:

sandmark.watermark.CT.encode.ir2ir.SaveNodes:

sandmark.watermark.CT.encode.ir2ir.Transformer:

sandmark.watermark.CT.encode.storage

This package implements operations on the storage of graph nodes in Collberg-Thomborson watermarking algorithm.

sandmark.watermark.CT.encode.storage.NodeStorage:

sandmark.watermark.CT.encode.storage.Array:

sandmark.watermark.CT.encode.storage.Hash:

sandmark.watermark.CT.encode.storage.GlobalStorage:

sandmark.watermark.CT.encode.storage.Pointer:

sandmark.watermark.CT.encode.storage.StorageClass:

sandmark.watermark.CT.encode.storage.Vector:

sandmark.watermark.CT.recognize

This package implements the recognition part of Collberg-Thomborson's software watermarking algorithm. It should be generalized to handle recognition in *any* dynamic software watermarking algorithm.

sandmark.watermark.CT.recognize.Heap2Graph:

sandmark.watermark.CT.recognize.RecognizeData:

sandmark.watermark.CT.recognize.Recognizer:

sandmark.watermark.CT.trace

This package currently implements the trace part of Collberg-Thomborson's software watermarking algorithm. It should be generalized to handle tracing in *any* dynamic software watermarking algorithm.

sandmark.watermark.CT.trace.Annotator:

sandmark.watermark.CT.trace.CallForest:

sandmark.watermark.CT.trace.Preprocessor:

sandmark.watermark.CT.trace.TracePoint:

sandmark.watermark.CT.trace.Tracer:

sandmark.watermark.constantstring

This package implements a trivial static watermarking algorithm.

sandmark.watermark.constantstring.ConstantString:

sandmark.watermark.constantstring.Test:

sandmark.watermark.constantstring.Program:

sandmark.statistics

This package counts all types of elements of a Java program.

sandmark.statistics.Statistics:

2.2 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.

```
java.util.Collection c =
   sandmark.util.classloading.ClassFinder.getClassesWithAncestor
      (sandmark.util.classloading.IClassFinder.ALGORITHM);
java.util.Iterator it = c.iterator();
while(it.hasNext()) {
   String className = (String)c.next();
   System.out.println(className);
}
```

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.