Detecting Software Theft with API Call Sequence Sets

David Schuler Valentin Dallmeier
Saarland University
Department of Computer Science
Saarbrücken, Germany
{schuler,dallmeier}@cs.uni-sb.de

Abstract

Software birthmarking uses a set of unique characteristics every program has upon creation to justify ownership claims of thefted software. This paper presents a novel birthmarking technique based on the interaction of a program with the standard API. We have used this technique to succesfully distinguish 4 different implementations of PNG image processing.

1 Introduction

In the last few years we have witnessed a huge increase in the amount of open source projects. Sourceforge.net currently hosts over 115.000 projects, a large portion of which is licensed under the Gnu Public License (GPL). This type of license allows users to pass on the program without charge, but requires new projects using it to also be licensed under GPL.

Unfortunately for the spirit of open source, not all individuals or companies obey these requirements. A popular example of the resent past is the XCP copy protection software present on many audio discs. It contains and uses several GPL projects (LAME, DRMS and others), but is not distributed under the terms of the GPL. There are many more examples of companies that illegally use GPL projects to save development resources. This is a severe disadvantage for companies that either don't use such projects or also put their products under the GPL.

In order to address the problem of software theft, researchers have proposed the use of birthmarks for the prove of ownership. Birthmarking relies on a set of characterstics a program originally possessed. Two programs that have similar or identical birthmarks are very likely to be the same.

This paper proposes a new dynamic birthmark. It is based on the observation that the way a program uses standard libraries (e.g. the JAVA API) is a unique characteristic of every program. The remainder of this paper gives details about our birthmark, presents a small experiment to evaluate it's effectiveness, and discusses the results.

2 Birthmark Extraction

Since the last release of the java development kit, the standard API consists of more than 4600 classes. As virtually every JAVA program uses classes of the API during it's execution, API interaction is a good candidate for a birthmark. This section explains the way we capture a program's API interaction and the determination of the birthmark.

Figure 1 shows a small example where a user implementation of a depth first search (`DFSEarch`) uses a `Stack` provided by the API. To characterize the usage of the API, we use the sequence of calls that originate in a user class and invoke a method that is part of the API. We call this the sequence of API method calls. For our example, the sequence of API method calls is `push(Node)`, `pop()` and `top()`.

For realistic program runs, a trace of the sequence of API method calls quickly reaches several gigabytes in size and becomes difficult to handle. Rather than using the whole trace, we extract sets of call sequences from it. A *call sequence set* is obtained by sliding a window with a fixed size over the trace and remembering all window contents observed. This is described in detail in (Dallmeier et al., 2005).

Using call sequence sets, we can compare two programs P and Q in the following way. We trace executions of both P and Q and calculate the sequence sets S_P and S_Q. If P and Q are the some program or at least closely related, we expect a large number of sequences to occur both in S_P and S_Q. The birthmark value B_{S_P,S_Q} measures the fraction of common call sequences in the union of S_P and S_Q:

$$B_{S_P,S_Q} = \frac{\#(S_P \cap S_Q)}{\#(S_P \cup S_Q)} \quad (1)$$

A birthmark value close to 1 indicates that P and Q have very similar API interaction and thus we suspect P and Q to be the same program.

3 Evaluation of Credibility

We require a birthmark to be *credible*: Two independently written programs P and Q that may accomplish the same task must have different birthmarks. In other words: If P and Q have the same birthmark,

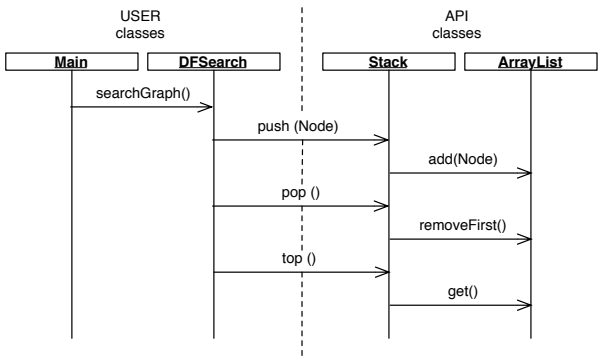

Figure 1: Example Interaction with JAVA API.

	JAI	JIMI	JIU	SIXLEGS
JAI	**0.94**	0.01	0.02	0.02
JIMI	0.01	**1.00**	0.13	0.00
JIU	0.02	0.13	**1.00**	0.04
SIXLEGS	0.02	0.00	0.04	**1.00**

Table 1: Credibility results.

then $P = Q$ should be true in almost all cases.

In order to evaluate the credibility of our birthmark technique, we have conducted an experiment with 4 libraries for reading PNG images. For each library we collected the API call sequence sets from reading a test suite of 100 PNG images. Table 1 shows the birthmark values for comparing each run of a library against all others. When comparing a library with itself (e.g. JIMI with JIMI), we compare the call sequence sets extracted from two runs using the same test suite as input.

Our results show that using API call sequences is a credible birthmark: All birthmark values for executions of different libraries are close to zero, indicating a strong difference in API interaction for all libraries although they implement the same task. The results for comparing two runs of the same library are located on the main diagonal. Except for JAI, all values are 1.00, indicating that the call sequence sets for both runs were the same. The slight difference in the call sequence sets for JAI are due to some GUI threads running in the background.

4 Related Work

In previous work (Dallmeier et al. (2005)) we have used call sequences to compare passing and failing runs of a program. Call sequences are collected on a per-class basis, and compared across runs to find the class that is most likely to contain the defect.

Tamada et al. (2003) describe the first practical application of birthmarks to identify the theft of programs. They propose a set of static birthmarks for classes, such as the constant values used to initialize the fields of a class. A preliminary evaluation reports that the proposed birthmarks identify a class within a program with high precision, but can easily be confused by several obfuscation techniques.

Tamada et al. (2004) introduce a definition of dynamic birthmarks and propose two such birthmarks based on the trace of system calls for windows programs. They claim that these birthmarks are reasonably robust against obfuscator attacks, but give no experimental evidence.

5 Conclusions and Future Work

We have presented a new dynamic birthmark technique based on the extraction of API call sequence sets from program runs. In a preliminary experiment, our technique successfully distinguished 4 different libraries for PNG image processing. These results are promising and justify further evaluation of our technique. Among other things, our future work will concentrate on the following topics:

Birthmark Refinement A possible improvement of the birthmark may be to ignore calls to commonly used API classes like `String` to avoid pollution of the call sequence sets.

Evaluation of Resilience An effective birthmark must be resilient to semantic preserving transformations like obfuscation. We plan to evaluate the resilience of our technique against several available obfuscation techniques.

Acknowledgements Christian Lindig provided valuable insights during birthmark design and evaluation.

References

Valentin Dallmeier, Christian Lindig, and Andreas Zeller. Lightweight defect localization for Java. In Andrew Black, editor, *European Conference on Object-Oriented Programming (ECOOP)*, pages 528–550, 2005.

Haruaki Tamada, Masahide Nakamura, Akito Monden, and Ken ichi Matsumoto. Detecting the theft of programs using birthmarks. Information Science Technical Report NAIST-IS-TR2003014, Graduate School of Information Science, November 2003.

Haruaki Tamada, Keiji Okamoto, Masahide Nakamura, Akito Monden, and Ken ichi Matsumoto. Dynamic software birthmarks to detect the theft of windows applications. In *Proc. International Symposium on Future Software Technology 2004 (ISFST 2004)*, 2004. Xi'an, China.