A New Thread-Aware Birthmark for Plagiarism Detection of Multithreaded Programs

Zhenzhou Tian[1], Ting Liu[1]*, Qinghua Zheng[1], Feifei Tong[1], Ming Fan[1], Zijiang Yang[1, 2]
[1] MOEKLINNS, Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China
[2] Department of Computer Science, Western Michigan University, Kalamazoo, MI 49008, USA

ABSTRACT
Dynamic birthmarking used to be an effective approach to detecting software plagiarism. Yet the new trend towards multithreaded programming renders existing algorithms almost useless, due to the fact that thread scheduling nondeterminism severely perturbs birthmark generation and comparison. In this paper, we design a birthmark based on thread-related system calls. Such a birthmark is less susceptible to thread scheduling. The empirical study conducted on an open benchmark shows that the new birthmark is superior to existing birthmarks and is resilient against most state-of-the-art obfuscation techniques.

CCS Concepts
• Social and professional topics → Intellectual property; • Information systems → Near-duplicate and plagiarism detection.

Keywords
Software Plagiarism Detection; Multithreaded Program; Thread-aware Birthmark; Thread-related System Call

1. INTRODUCTION
Software plagiarism, ranging from open source code reusing to app repacking, severely affect both open source communities and software companies. It is widespread because software plagiarism is easy to implement but hard to detect. A study in 2012 [1] shows that about 5% to 13% of apps in six large third-party app markets are copied from the official Android market. The unavailability of source code and the existence of powerful automated code obfuscation techniques and tools [2]–[4] are a few reasons that make software plagiarism detection a daunting task. Nevertheless, significant progress has been made to address this challenge. One of the most effective approaches is software birthmarking, where a set of characteristics, called birthmarks, are extracted from a program to uniquely identify the program. As illustrated in previous works [5-8], with proper algorithms birthmarks can identify software theft even after complex code obfuscations.

Despite the tremendous progress in software birthmarking, the trend towards multithreaded programming greatly threatens its effectiveness, as the existing approaches remain optimized for sequential programs. Birthmarks extracted from multiple runs of the same multithreaded programs can be very different due to the inherent non-determinism of thread scheduling. As indicated in

*Corresponding Author

ICSE'16 Companion, May 14–22, 2016, Austin, TX, USA
ACM 978-1-4503-4205-6/16/05.
DOI: http://dx.doi.org/10.1145/2889160.2892653

```
pthread_t t[N]; sem_t a;
void *run(void *data){
  …
  sem_wait(&a);
  usleep(1000);
  ...
  sem_post(&a);
  if(((int)data)%2==0){
    usleep(0);}
  …}
```

```
main(int argc, char *argv[]){
  ...  //a set of operations
  sem_init(&sem_a, 0, 1);
  scanf("%d", &i); j = i;
  for (i; i < N; i++){
    pthread_create(&t[i], NULL, run, (void *)i);
    ...}
  for (i = j; i < N; i++)
    pthread_join(mThread[i], NULL);
  ...}
```

Figure 1. Simplified version of the sample program

Table 1. Similarity scores calculated between birthmarks of multiple runs of the same program. The column headings indicate the number of threads

SCSSB	1	2	4	6	8
Ex-Containment	1.000	0.898	0.670	0.569	0.469
Ex-Cosine	1.000	0.760	0.439	0.363	0.265
Ex-Dice	1.000	0.864	0.609	0.532	0.403
Ex-Jaccard	1.000	0.864	0.619	0.57	0.412

Table 1 the application of SCSSB [6] on the sample program in the work of Tian et al. [9] (Figure 1 depicts a simplified version of the program), the scores quickly deteriorate as the number of threads increase, the traditional birthmark SCSSB fails to declare plagiarism even for simply duplicated multithreaded programs.

In this paper, we introduce a thread-aware dynamic birthmark called TreSB (Thread-related System call Birthmark) that can effectively detect plagiarism of multithreaded programs. Unlike many approaches [10-11], TreSB operates on binary executables rather than source code. The latter is usually unavailable when birthmarking is used to obtain initial evidence of plagiarism. The extensive experiments conducted on a publicly available benchmark [9] consisting of 234 versions of 35 multithreaded programs show good resilience and credibility of TreSB. A comparison of TreSB against two recently proposed thread-aware birthmarks shows that TreSB outperforms both of them.

2. THE APPROACH
System calls recorded during program execution are a favorable basis for extracting high quality birthmarks. The hypothesis is that modifications of system calls usually leads to incorrect program behavior, and therefore a birthmark generated from system calls can be used to identify stolen programs even after they have been modified. Also previous empirical study shows that the system call based birthmarks [6] are resilient against various obfuscation techniques. Yet as illustrated in [9], these birthmarks become no longer efficient for multithread programs due to the nondeterminism of thread scheduling.

Based on similar principle, we also generate birthmarks from system calls. Yet rather than using all the calls, we extract TreSB just from the thread-related system calls, which are essential to the semantics and correct executions of a multithreaded program. A random or deliberate modification to the calls can result in very subtle errors and therefore they are the least possible code to be

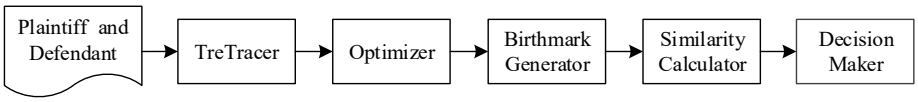

Figure 2. Overview of the TreSB based software plagiarism detection tool

changed. More importantly, these calls are the source to impose thread interleaving rather than being affected by the non-determinism, thus a birthmark extracted from these calls is less susceptible to thread scheduling. Specifically, we treat system calls that accomplish tasks including thread and process management (such as creation, join and termination, capability setting and getting), thread synchronization, signal manipulating, as well as thread and process priority setting, as thread-related.

Based on the thread-related system calls, we design the TreSB based plagiarism detection tool as depicted in Figure 2, where the plaintiff represents the program owned by its developer while the defendant represents the suspicious program that may have plagiarized the plaintiff. The tool consists of five modules, where:

- The first module, *TreTracer*, is implemented as a PIN [12] plugin to monitor program executions. It recognizes and records the thread-related system calls by instrumenting call sites dynamically. The output of this module is a thread-related system call sequence under a particular input vector. Each record in the sequence consists of a system call number and its corresponding system call name, and the return value during the execution.
- The raw sequences extracted by the *TreTracer* are not appropriate to be directly used for birthmark generation. The *Optimizer* performs a series of optimizations, including filtering noises such as failed calls [6, 9], and pre-selecting two most similar sequences from plaintiff and defendant to further reduce the randomness of thread interleaving.
- The *Birthmark Generator* obtains TreSBs from the sequences passed from the optimizer. Given a thread-related system call sequence $trs(p, I) = \langle e_1, e_2, \cdots, e_n \rangle$ recorded during the runtime of a multithreaded program p under input I, in which e_i is a thread-related system call instance. We adopt the k-gram algorithm [5, 13] to bound the sequences with a length k window, generating a set of fixed-length short subsequences called k-grams. Finally, as with the typical dynamic birthmarks [6-9], TreSB is a key-value pair set. The keys consist of all unique grams and values are the frequencies of the corresponding grams.
- The *Similarity Calculator* measures similarity between the birthmarks of plaintiff and defendant programs, and a decision is made in the *Decision Maker* based on the calculated similarity score and a threshold ϵ using Equation 1. Formally, let p and q be the plaintiff and defendant programs, respectively. Given a set of inputs $\{I_1, I_2, \cdots, I_n\}$, we obtain n pair of TreSBs $\{(p_{B_1}, q_{B_1}), \cdots, (p_{B_n}, q_{B_n})\}$. The similarity score between program p and q is calculated by, $Sim(p, q) = \sum_{i=1}^{n} sim_c\,(p_{B_i}, q_{B_i})/n$, where c ∈ $\{Ex\text{-}Cosine, Ex\text{-}Jaccard, Ex\text{-}Dice, Ex\text{-}Containment\}$ is the similarity metric defined in [9].

$$Sim(p,q) = \begin{cases} > 1-\epsilon & positive: q\ is\ a\ copy\ of\ p \\ < \epsilon & negative: q\ is\ not\ a\ copy\ of\ q \\ otherwise & inconcluisve \end{cases} \quad (1)$$

3. EXPERIMENTAL EVALUATION

We have conducted experiments for evaluating our method on an open benchmark [9] consisting of 234 versions of 35 multithreaded programs. Firstly, we evaluated its resilience and credibility [7],

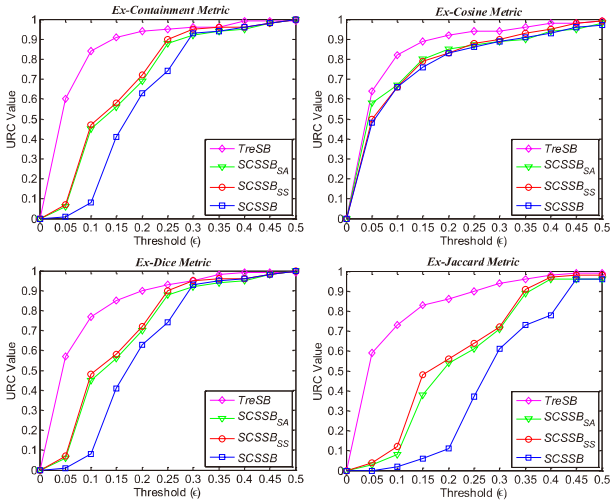

Figure 3. Performance comparison with respect to URC

two property a birthmark should possess. The empirical study shows that TreSB and its comparison algorithms are credible in differentiating independently developed programs, and resilient to most state-of-the-art semantics-preserving obfuscation techniques in the best tools such as SandMark [2] and DashO [3].

In addition, we compared the overall performance of TreSB against two latest thread-aware birthmarks SCSSB$_{SA}$ and SCSSB$_{SS}$ [9] that adapt SCSSB [6] for multithreaded programs, as well as the original SCSSB, with respect to the performance metric URC (Union of Resilience and Credibility) [7, 9]. URC is a metric that considers both resilience and credibility, whose value ranges from 0 to 1, with higher value indicating a better birthmark. As indicated by Equation 1, the detection result relies on the value of threshold ϵ. Figure 3 gives the URC values for each birthmark with respect to each similarity metric by varying the threshold value. As indicated by the pink lines, TreSB always performs better than that the other three birthmarks. SCSSB$_{SA}$ and SCSSB$_{SS}$ fail in that their assumption of events happened in each thread is stable is not always true. Therefore, although the method is able to alleviate the impact of thread scheduling, false negatives are not uncommon, especially when thread interplay is complex.

4. CONCLUSION AND FUTURE WORK

We introduced a new thread-aware birthmark called TreSB by abstracting program behavioral characteristics from thread-related system calls. It works efficiently for plagiarism detection of multithreaded programs, and outperforms the two only existing thread-aware birthmarks. Despite that, currently TreSB cannot handle the case where an obfuscation introduces new threads that frequently emit thread-related system calls to disguise plagiarism. Yet such obfuscations are difficult to achieve automatically. But to defend against such obfuscations, in the future we plan to optimize our approach by introducing trace filtering.

5. ACKNOWLEDGMENTS

The research was supported in part by National Science Foundation of China (91118005, 91218301, 61221063, 61428206, 61203174, 91418205, 61472318, 1500365), Ministry of Education Innovation Research Team (IRT13035), Key Projects in National Science and Technology Pillar Program (2013BAK09B01).

6. REFERENCES

[1] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting repackaged smartphone applications in third-party android marketplaces," in *Proceedings of the second ACM conference on Data and Application Security and Privacy*. ACM, 2012, pp. 317–326.

[2] C. Collberg, G. Myles, and A. Huntwork, "Sandmark-a tool for software protection research," *IEEE Secur Priv*, vol. 1, no. 4, pp. 40–49, 2003.

[3] "Dasho java obfuscator, http://www.cs.arizona.edu/collberg/teaching/620/2008/assignments/tools/dasho/index.html," 2008.

[4] K. A. Roundy and B. P. Miller, "Binary-code obfuscations in prevalent packer tools," *ACM Computing Surveys (CSUR)*, vol. 46, no. 1, p. 4, 2013.

[5] Z. Tian, Q. Zheng, T. Liu, and M. Fan, "DKISB: Dynamic key instruction sequence birthmark for software plagiarism detection," in *2013 IEEE International Conference on High Performance Computing and Communications. (HPCC'13)*, 2013, pp. 619–627.

[6] X. Wang, Y.-C. Jhi, S. Zhu, and P. Liu, "Detecting software theft via system call based birthmarks," in *Annual Computer Security Applications Conference*, 2009. (*ACSAC* '09), 2009, pp. 149–158.

[7] Z. Tian, Q. Zheng, T. Liu, M. Fan, E. Zhuang, and Z. Yang, "Software plagiarism detection with birthmarks based on dynamic key instruction sequences," *IEEE Trans. Software Engineering*, vol. 41, pp. 1217–1235, 2015.

[8] Z. Tian, Q. Zheng, M. Fan, E. Zhuang, H. Wang, and T. Liu, "DBPD: A dynamic birthmark-based software plagiarism detection tool," pp. 740–741, 2014.

[9] Z. Tian, Q. Zheng, T. Liu, M. Fan, X. Zhang, and Z. Yang, "Plagiarism detection for multithreaded software based on thread-aware software birthmarks," in *Proceedings of the 22nd International Conference on Program Comprehension*. ACM, 2014, pp. 304–313.

[10] C. Liu, C. Chen, J. Han, and P. S. Yu, "Gplag: Detection of software plagiarism by program dependence graph analysis," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '06. New York, NY, USA: ACM, 2006, pp. 872–881.

[11] G. Cosma and M. Joy, "An approach to source-code plagiarism detection and investigation using latent semantic analysis," *Computers, IEEE Transactions on*, vol. 61, no. 3, pp. 379–394, 2012.

[12] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '05. New York, NY, USA: ACM, 2005, pp. 190–200.

[13] D. Schuler, V. Dallmeier, and C. Lindig, "A dynamic birthmark for Java," in *Proceedings of the 22nd International Conference on Automated Software Engineering,* (*ASE'07*), 2007, pp. 274–283.