Fine-Grained Binary Code Authorship Identification

Xiaozhu Meng
Computer Sciences Department
University of Wisconsin-Madison
Madison, WI 53706, USA
xmeng@cs.wisc.edu

ABSTRACT

Binary code authorship identification is the task of determining the authors of a piece of binary code from a set of known authors. Modern software often contains code from multiple authors. However, existing techniques assume that each program binary is written by a single author. We present a new finer-grained technique to the tougher problem of determining the author of each basic block. Our evaluation shows that our new technique can discriminate the author of a basic block with 52% accuracy among 282 authors, as opposed to 0.4% accuracy by random guess, and it provides a practical solution for identifying multiple authors in software.

CCS Concepts

•**Applied computing → Investigation techniques;**

Keywords

Software forensics; Basic block level; Code features

1. INTRODUCTION

Authorship identification is the task of determining the authors of a computer program from a set of known authors. This task has significant application to forensics of malicious software (malware), detecting software plagiarism, and identifying untrusted software components in the software supply chain. Compared to source code authorship identification [5, 6, 8, 12, 17, 30], binary code authorship identification [2, 7, 28] can be applied under broader scenarios, such as when the source code is not available, which is often the case of handling malware, proprietary software, and legacy code, or when only a code byte stream is discovered in network packets or memory images.

Current software is often the result of team efforts. Open source and proprietary software often contains code from multiple authors. Even malware development has become similar to normal software development, evolving from an individual hacking to cooperation between multiple authors [10, 20, 26]. Malware writers share functional components and adapt them [18, 22, 23, 29], by forming co-located teams [19] or through the Internet [1, 4, 16]. This trend requires authorship identification techniques to recognize multiple authors within a binary and understand their contributions.

However, existing techniques have assumed that each binary is written by a single author [2, 7, 28]. When applied to multiple author binaries, they can identify at most one of the multiple authors or report a merged group identity, making it difficult to detect the presence of multiple authors and detect plagiarized or untrusted code when it consists of only a small fraction of the whole binary.

We present a new fine-grained technique to identify multiple authors in a program binary. Our idea is to identify a reasonable unit of code that can be attributed a single author. We then divide a program binary into smaller pieces and determine the author of each unit of code. The most obvious candidates for the unit of code are the function and basic block. When making this granularity decision, we consider two factors: how well a unit of code can be attributed to a single author and whether it contains sufficient information for attribution. For the first factor, we conducted an empirical study on several open source projects, including Apache HTTP Server [3], Dyninst [24], GCC [13] and other projects from Github. Our study shows that 88% of basic blocks have a major author who contributes more than 90% of the basic block, while only 67% of functions have such a major author, supporting using the basic block as the unit of code. For the second factor, we designed a set of new basic block level code features, covering code properties such as control flow and data flow. In addition, we designed a new type of code features, *context features*, to summarize the context of the function or the loop to which the basic block belongs. We evaluated our new technique on a data set derived from the open source projects used in our empirical study. The results show that our new technique can identify the author of a basic block with 52% accuracy among 282 authors.

2. RELATED WORK

Existing techniques for binary code authorship identification [2, 7, 28] share a common workflow and have four major steps: (1) designing a large number of binary code features to capture programming styles, (2) extracting the defined features and converting a program binary to a feature vector by using binary code analysis tools such as Dyninst [24] and IDA Pro [14], (3) determining a small set of features that are indicative of authorship by using feature selection techniques such as ranking features based on mutual information be-

FSE'16, November 13–18, 2016, Seattle, WA, USA
ACM. 978-1-4503-4218-6/16/11...$15.00
http://dx.doi.org/10.1145/2950290.2983962

tween features and authors [28], and (4) applying supervised machine learning techniques such as Support Vector Machine (SVM) [9] and Random Forests [15], to learn the correlations between features and authorship. We discuss the features used in existing techniques and their evaluation results.

Existing binary code features are extracted at the function and block level and then accumulated to the program level. Block level features include byte N-gram [7, 28], instruction idioms [2, 28], and library call targets [28]. We will include these features when working at the basic block level. Function level features include Graphlets [28, 7], which represent subgraphs of the control flow graph (CFG) of a function, and register flow graphs [2], which perform backward slicing on the registers in all `cmp` and `test` instructions and convert each slice to a hash value. We cannot directly reuse these function level features, so need new basic block level features to capture code properties such as control flow and data flow.

Previous projects have evaluated their techniques on single author programs. Rosenblum et al. [28] used features that described instructions, control flow, and library call targets, reporting 51% accuracy for classifying 191 authors on -O0 binaries. Caliskan et al. [7] added data flow features, improving this accuracy to 63% for classifying 191 authors on -O0 binaries, and 61% for classifying 100 authors on -O2 binaries.

3. OUR APPROACH

Our new fine-grained technique follows a similar workflow described in the previous section with two key differences. First, we conduct an empirical study on several open source projects to quantify how well functions and basic blocks can be attributed to a single author. Our study supports using the basic block as the unit of code. Second, we design new basic block level code features, enabling authorship identification at the basic block level.

3.1 Determining the Unit of Code

In this study, we derive the authors and their contribution percentages for each function and basic block from the development history of open source projects. The more the major author contributes, the better the code can be attributed to a single author. Our approach to derive author contribution percentages is to first use *git-author* [21] to calculates a vector of author contribution percentage for each line of code, compile the source with debugging information using GCC 4.8.5 with -O2 optimization, map each machine instruction back to source lines, and accumulate machine instruction authorship to the basic block and function level.

Our study shows that 85% of the basic blocks are written by a single author and 88% of the basic blocks have a major author who contributes more than 90% of the basic block. On the other hand, only 56% of the functions are written by a single author and 67% of the functions have a major author who contributes more than 90% of the function. Therefore, the function as a unit of code brings too much imprecision, so we use the basic block as the unit for attribution.

3.2 New Code Features

Existing basic block level features miss several important properties of machine instructions, such as instruction prefixes, operand sizes, and operand addressing modes, and do not capture code properties such as control flow and data flow. We design new basic block level features to cover these missed code properties. We also design *context features* to summarize the context of the function or the loop to which the basic block belongs. Our new features are summarized in Table 1. Here, we focus on CFG edges and context features.

Table 1: An overview of new basic block level features

Code Property	New block level features
Instruction	Instruction prefixes, operand sizes and addressing modes, constant values
Control flow	CFG edge types, whether a block throws or catches exceptions
Data flow	# of live registers at block entry/exit, # of used/defined registers, stack height delta of the block, data dependencies of variables
Context	Loop nesting level, loop size, function CFG width/depth

To capture control flow at the basic block level, we design features that describe the incoming and outgoing CFG edges of a basic block in three dimensions: (1) the control flow transfer type (such as conditional taken, conditional not taken, direct jump, and fall through), (2) whether the edge is interprocedural or intraprocedural, and (3) whether the edge goes to unknown control flow target such as unresolved indirect jumps or indirect calls.

Context features capture the intuition that the context of a basic block such as the loop and the function to which the basic block belongs may affect how a programmer writes code. We design features such as loop nesting level and loop size to summarize the context of a basic block.

4. EXPERIMENT RESULTS

We evaluated our new technique based on a data set that contained 831,372 basic blocks, 170 binaries, and 282 authors. All binaries were compiled by GCC 4.8.5 with -O2 optimization. In practice, we can train separate models for different compilers or optimization levels and then apply compiler provenance techniques [25, 27] to determine which model to use. We used Dynisnt [24] to extract code features and Liblinear [11] for training and prediction. We performed the traditional leave-one-out cross validation, where each binary was in turn used for testing and all other binaries were used for training. Each round of the cross validation had three steps. First, each basic block in the training set was labeled with its major author. Second, we selected the top 50,000 features that had the most mutual information with the major authors. Third, we trained a linear SVM and predicted the author of each basic block in the testing set. We calculated accuracy for correctly attributed code bytes and correctly attributed basic blocks. Our new technique achieved 52% average accuracy for both metrics. Our result is comparable to the accuracy reported by the previous projects done at the program level, showing authorship identification is practical at the basic block level and our new technique provides a practical solution for identifying multiple authors.

5. CONCLUSION

We have presented our new fine-grained technique to determine the author of a basic block. Our new technique is based on an empirical study that supports using the basic block as the unit for attribution and a set of new basic block level code features. Our evaluation showed that our new technique can identify authors at the basic block level and help analysts perform authorship identification on multi-author software.

6. REFERENCES

[1] A. Abbasi, W. Li, V. Benjamin, S. Hu, and H. Chen. Descriptive analytics: Examining expert hackers in web forums. In *2014 IEEE Joint Intelligence and Security Informatics Conference (JISIC)*, Hague, Netherlands, Sep. 2014.

[2] S. Alrabaee, N. Saleem, S. Preda, L. Wang, and M. Debbabi. Oba2: An onion approach to binary code authorship attribution. *Digital Investigation*, 11, Supplement 1:S94 – S103, May 2014.

[3] Apache Software Foundation. Apache http server, http://httpd.apache.org.

[4] V. Benjamin and H. Chen. Securing cyberspace: Identifying key actors in hacker communities. In *2012 IEEE International Conference on Intelligence and Security Informatics (ISI)*, Arlington, VA, USA, June 2012.

[5] S. Burrows. *Source code authorship attribution*. PhD thesis, Melbourne, Victoria, Australia, RMIT University, 2010.

[6] A. Caliskan-Islam, R. Harang, A. Liu, A. Narayanan, C. Voss, F. Yamaguchi, and R. Greenstadt. De-anonymizing programmers via code stylometry. In *24th USENIX Security Symposium (USENIX Security)*, Washington, D.C., Aug. 2015.

[7] A. Caliskan-Islam, F. Yamaguchi, E. Dauber, R. Harang, K. Rieck, R. Greenstadt, and A. Narayanan. When coding style survives compilation: De-anonymizing programmers from executable binaries. http://arxiv.org/pdf/1512.08546.pdf, Dec. 2015.

[8] E. Chatzicharalampous, G. Frantzeskou, and E. Stamatatos. Author identification in imbalanced sets of source code samples. In *2012 IEEE 24th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 790–797, Athens, Greece, Nov. 2012.

[9] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3), Sep. 1995.

[10] F. de la Cuadra. The geneology of malware. *Network Security*, 4:17–20, May 2007.

[11] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, June 2008.

[12] G. Frantzeskou, E. Stamatatos, S. Gritzalis, C. E. Chaski, and B. S. Howald. Identifying authorship by byte-level n-grams: The source code author profile (scap) method. *International Journal of Digital Evidence*, 6(1):1–18, 2007.

[13] GNU Project. Gcc: The gnu compiler collection, http://gcc.gnu.org.

[14] Hex-Rays. IDA, https://www.hex-rays.com/products/ida/.

[15] T. K. Ho. Random decision forests. In *3rd International Conference on Document Analysis and Recognition (ICDAR)*, Montreal, Canada, Aug. 1995.

[16] T. J. Holt, D. Strumsky, O. Smirnova, and M. Kilger. Examining the social networks of malware writers and hackers. *International Journal of Cyber Criminology*, 6(1):891–903, Jan. 2012.

[17] R. C. Lange and S. Mancoridis. Using code metric histograms and genetic algorithms to perform author identification for software forensics. In *9th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, London, England, July 2007.

[18] M. Lindorfer, A. Di Federico, F. Maggi, P. M. Comparetti, and S. Zanero. Lines of malicious code: Insights into the malicious software industry. In *28th Annual Computer Security Applications Conference (ACSAC)*, Orlando, Florida, USA, Dec. 2012.

[19] Mandiant. Mandiant 2013 Threat Report. https://www2.fireeye.com/WEB-2013-MNDT-RPT-M-Trends-2013_LP.html, 2013. Mandiant White Paper.

[20] M. Marquis-Boire, M. Marschalek, and C. Guarnieri. Big game hunting: The peculiarities in nation-state malware research. In *Black Hat*, Las Vegas, NV, USA, Aug. 2015.

[21] X. Meng, B. P. Miller, W. R. Williams, and A. R. Bernat. Mining software repositories for accurate authorship. In *2013 IEEE International Conference on Software Maintenance (ICSM)*, Eindhoven, Netherlands, Sep. 2013.

[22] N. Moran and J. Bennett. Supply chain analysis: From quartermaster to sunshop. https://www.fireeye.com/content/dam/fireeye-www/global/en/current-threats/pdfs/rpt-malware-supply-chain.pdf, Nov. 2013. FireEye Labs White Paper.

[23] G. O'Gorman and G. McDonald. The elderwood project. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the-elderwood-project.pdf, Sep. 2012. Symantec White Paper.

[24] Paradyn Project. Dyninst: Putting the Performance in High Performance Computing, http://www.dyninst.org.

[25] A. Rahimian, P. Shirani, S. Alrbaee, L. Wang, and M. Debbabi. Bincomp: A stratified approach to compiler provenance attribution. *Digital Investigation*, 14, Supplement 1, Aug. 2015.

[26] R. Roberts. Malware development life cycle. In *Virus Bulletin Conference (VB)*, Oct. 2008.

[27] N. Rosenblum, B. P. Miller, and X. Zhu. Recovering the toolchain provenance of binary code. In *2011 International Symposium on Software Testing and Analysis (ISSTA)*, Toronto, Ontario, Canada, July 2011.

[28] N. Rosenblum, X. Zhu, and B. P. Miller. Who wrote this code? identifying the authors of program binaries. In *16th European Conference on Research in Computer Security (ESORICS)*, Leuven, Belgium, Sep. 2011.

[29] B. Ruttenberg, C. Miles, L. Kellogg, V. Notani, M. Howard, C. LeDoux, A. Lakhotia, and A. Pfeffer. Identifying shared software components to support malware forensics. In *11th Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, Egham, London, UK, July 2014.

[30] M. F. Tennyson. On improving authorship attribution of source code. In *4th International Conference Digital Forensics and Cyber Crime (ICDF2C)*, Lafayette, IN, USA, Oct. 2012.