that the function sequence is unchanged. However, we also evaluated the robustness of BCD using only call-graph and data-reference graphs. In our test, when function sequence graph is excluded, BCD F_1^w score reduces to from 0.86 to 0.78 for C++ applications. Although an adversary could use obfuscation to defeat BCD, those obfuscations might well have the side-effect of raising detection alarms. For example, if BCD encounters a binary that seems to have extremely chaotic locality properties, that in itself could be used as a malware detection strategy. One way to address obfuscation is to de-obfuscate before applying BCD, e.g., using solutions such as dynamic unpackers [9].

Dynamic features. For simplicity, we have focused on features extracted statically from the executable. However, features extracted from program executions could also be incorporated into the decomposition graph, improving its efficiency. Some example dynamic features that may provide useful modularity information are functions used in a certain order and functions that access the same data structures in heap-allocated memory.

6 CONCLUSION

This paper introduced the problem of binary code decomposition and addressed its challenges by proposing a novel approach, called BCD, for decomposing a program executable into components. BCD takes a binary executable as input, and extracts code locality, data references, and calling relationships to build a decomposition graph. It then applies a graph-theoretic approach to partition the decomposition graph into disjoint components. Our evaluation results show that BCD is able to achieve a high precision and recall for decomposing the tested programs into components having structurally related functions.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their valuable comments. This work was partially supported by AFOSR awards FA9550-14-1-0119 and FA9550-14-1-0173, ONR awards N00014-14-1-0030 and N00014-17-1-2995, and NSF awards 1054629 and 1513704. Partial support was also provided by the Regional Government of Madrid through the N-GREENS Software-CM project S2013/ICE-2731, the Spanish Government through the DEDETIS grant TIN2015-7013-R, and the European Union through the ElasTest project ICT-10-2016-731535.

REFERENCES

[1] Saed Alrabaee, Noman Saleem, Stere Preda, Lingyu Wang, and Mourad Debbabi. 2014. OBA2: An Onion Approach to Binary Code Authorship Attribution. In *Digital Investigation*, Vol. 11. S94–S103.
[2] Nicolas Anquetil and Timothy C. Lethbridge. 1999. Experiments with Clustering as a Software Remodularization Method. In *Proc. 6th Working Conf. Reverse Engineering (WCRE)*. 235–255.
[3] Tiffany Bao, Jonathan Burket, Maverick Woo, Rafael Turner, and David Brumley. 2014. BYTEWEIGHT: Learning to Recognize Functions in Binary Code. In *Proc. 23rd USENIX Security Sym.* 845–860.
[4] David Brumley, JongHyup Lee, Edward J. Schwartz, and Maverick Woo. 2013. Native x86 Decompilation Using Semantics-preserving Structural Analysis and Iterative Control-flow Structuring. In *Proc. 22nd USENIX Security Sym.*
[5] Juan Caballero, Noah M. Johnson, Stephen McCamant, and Dawn Song. 2010. Binary Code Extraction and Interface Identification for Security Applications. In *Proc. 17th Annual Network & Distributed System Security Sym. (NDSS)*.
[6] Juan Caballero, Pongsin Poosankam, Stephen McCamant, Domagoj Babic, and Dawn Song. 2010. Input Generation Via Decomposition and Re-Stitching: Finding Bugs in Malware. In *Proc. 17th ACM Conf. Computer and Communications Security (CCS)*. 413–425.
[7] Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. 2004. Dictionary Matching and Indexing with Errors and Don't Cares. In *Proc. 36th Annual ACM Sym. Theory of Computing (STOC)*. 91–100.
[8] Chris Eagle. 2008. *The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler*. No Starch Press, San Francisco, CA, USA.
[9] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. 2012. A Survey on Automated Dynamic Malware-analysis Techniques and Tools. *ACM Computing Surveys (CSUR)* 44, 2 (2012).
[10] Manuel Egele, Maverick Woo, Peter Chapman, and David Brumley. 2014. Blanket Execution: Dynamic Similarity Testing for Program Binaries and Components. In *Proc. 23rd USENIX Security Sym.*
[11] M. Van Emmerik and T. Waddington. 2004. Using a Decompiler for Real-world Source Recovery. In *Proc. 11th Working Conf. Reverse Engineering (WCRE)*. 27–36.
[12] Free Software Foundation. 1983. GNU Software Repository. www.gnu.org/software/software.html. (1983). Retrieved 3/30/2018.
[13] Debin Gao, Michael K. Reiter, and Dawn Song. 2008. Binhunt: Automatically finding semantic differences in binary programs. In *ICICS*.
[14] Emily R. Jacobson, Nathan Rosenblum, and Barton P. Miller. 2011. Labeling Library Functions in Stripped Binaries. In *Proc. 10th ACM SIGPLAN-SIGSOFT Work. Program Analysis for Software Tools and Engineering (PASTE)*. 1–8.
[15] Lingxiao Jiang and Zhendong Su. 2009. Automatic Mining of Functionally Equivalent Code Fragments Via Random Testing. In *Proc. 18th Int. Sym. Software Testing and Analysis (ISSTA)*. 81–92.
[16] Wei Ming Khoo, Alan Mycroft, and Ross Anderson. 2013. Rendezvous: A Search Engine for Binary Code. In *Proc. 10th Working Conf. Mining Software Repositories (MSR)*. 329–338.
[17] Dohyeong Kim, William N. Sumner, Xiangyu Zhang, Dongyan Xu, and Hira Agrawal. 2014. Reuse-oriented Reverse Engineering of Functional Components From x86 Binaries. In *Proc. 36th Int. Conf. Software Engineering (ICSE)*. 1128–1139.
[18] Clemens Kolbitsch, Thorsten Holz, Christopher Kruegel, and Engin Kirda. 2010. Inspector Gadget: Automated Extraction of Proprietary Gadgets From Malware Binaries. In *Proc. 31st IEEE Sym. Security & Privacy (S&P)*.
[19] Spiros Mancoridis, Brian S. Mitchell, Yihfarn Chen, and Emden R. Gansner. 1999. Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures. In *Proc. IEEE Int. Conf. Software Maintenance (ICSM)*. 50–59.
[20] Xiaozhu Meng. 2016. Fine-grained Binary Code Authorship Identification. In *Proc. 24th ACM SIGSOFT Int. Sym. Foundations Software Engineering (FSE)*. 1097–1099.
[21] Microsoft. 2007. Visual Studio sample codes. code.msdn.microsoft.com/vstudio. (2007). Retrieved 3/30/2018.
[22] Mark E.J. Newman. 2004. Fast Algorithm for Detecting Community Structure in Networks. *Physical Review E* 69, 6 (2004), 066133.
[23] Beng Heng Ng and Atul Prakash. 2013. Exposé: Discovering Potential Binary Code Re-use. In *Proc. 37th IEEE Annual Computer Software and Applications Conf. (COMPSAC)*. 492–501.
[24] Andreas Noack. 2009. LinLogLayout: Graph Clustering and Force-directed Graph Layout. code.google.com/p/linloglayout. (2009). Retrieved 3/30/2018.
[25] Arzucan Özgür, Levent Özgür, and Tunga Güngör. 2005. Text Categorization with Class-based and Corpus-based Keyword Selection. In *Proc. 20th Int. Sym. Computer and Information Sciences (ISCIS)*. 606–615.
[26] Nathan Rosenblum, Xiaojin Zhu, and Barton P. Miller. 2011. Who Wrote This Code? Identifying the Authors of Program Binaries. In *Proc. 16th European Conf. Research in Computer Security (ESORICS)*. 172–189.
[27] Andreas Sæbjørnsen, Jeremiah Willcock, Thomas Panas, Daniel Quinlan, and Zhendong Su. 2009. Detecting Code Clones in Binary Executables. In *Proc. 18th Int. Sym. Software Testing and Analysis (ISSTA)*. 117–128.
[28] Slashdot Media. 1999. SourceForge. sourceforge.net. (1999). Retrieved 3/30/2018.
[29] Randy Smith and Susan Horwitz. 2009. Detecting and Measuring Similarity in Code Clones. In *Proc. 3rd REF/TCSE Int. Work. Software Clones (IWSC)*. 28–34.
[30] Venkatesh Srinivasan and Thomas Reps. 2014. Recovery of Class Hierarchies and Composition Relationships From Machine Code. In *Proc. 23rd Int. Conf. Compiler Construction (CC)*. 61–84.
[31] Wenhao Wang, Xiaoyang Xu, and Kevin W. Hamlen. 2017. Object Flow Integrity. In *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS)*. 1909–1924.
[32] Xinran Wang, Yoon-Chan Jhi, Sencun Zhu, and Peng Liu. 2009. Behavior Based Software Theft Detection. In *Proc. 16th ACM Conf. Computer and Communications Security (CCS)*. 280–290.
[33] Fangfang Zhang, Yoon-Chan Jhi, Dinghao Wu, Peng Liu, and Sencun Zhu. 2012. A First Step Towards Algorithm Plagiarism Detection. In *Proc. 21st Int. Sym. Software Testing and Analysis (ISSTA)*. 111–121.
[34] Zynamics. 2004. BinDiff. www.zynamics.com/bindiff.html. (2004). Retrieved 3/30/2018.