Visualizing the variational callgraph of the Linux Kernel: An approach for reasoning about dependencies [POSTER]

Momin M. Malik, Jürgen Pfeffer, Gabriel Ferreira, Christian Kästner
Institute for Software Research
Carnegie Mellon University

ABSTRACT

Software developers use #ifdef statements to support code configurability, allowing software product diversification. But because functions can be in many executions paths that depend on complex combinations of configuration options, the introduction of an #ifdef for a given purpose (such as adding a new feature to a program) can enable unintended function calls, which can be a source of vulnerabilities. Part of the difficulty lies in maintaining mental models of all dependencies. We propose analytic visualizations of the *variational callgraph* to capture dependencies across configurations and create visualizations to demonstrate how it would help developers visually reason through the implications of diversification, for example through visually doing change impact analysis.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*; D.2.5 [**Software Engineering**]: Testing and Debugging—*code inspections and walkthroughs*; H.1.2 [**Information Systems**]: User/Machine Systems—*human factors*

Keywords

visualization, configuration complexity, Linux Kernel, callgraph, #ifdef, dependencies, vulnerabilities

1. INTRODUCTION

The presence of #ifdefs in code requires developers to expend fare more effort to maintain the code [5][11]. Specifically, developers struggle to keep a complete and accurate mental model of dependencies, often relying on default values [2] or enabling as many configuration options as possible when configuring a new software product [4]. Both are risky, but the latter even more so because it increases the attack surface of software systems and, potentially, the number of undesired interactions among features [7].

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

HotSoS '16 April 19-21, 2016, Pittsburgh, PA, USA

© 2016 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-4277-3/16/04.

DOI: http://dx.doi.org/10.1145/2898375.2898398

Graphs are widely and successfully used not only as formal models but also to help visually reason about dependencies [9], for example in UML diagrams to represent the artifacts of software systems [10], graphical models to represent dependencies between variables [8] or sociograms to navigate social dependencies [3]. Examples of visual reasoning on networks includes finding bridges, tracing paths, and assessing structure [6]. We propose to carry out a similar task, aiding developers by visually representing the possible sequence of function execution paths that may be affected by modifying, inserting, or removing an #ifdef.

2. APPROACH

A callgraph is a software representation that provides information about which functions call which other functions, with the nodes being functions and the edges being calls. In a given configuration of the Linux Kernel, which we use as our focus, only some function definitions and calls will be included. In a variational callgraph [1], we represent all nodes and edges in order to also capture cross-configuration dependencies. Our variational callgraph consists of about 250,000 nodes and 660,000 edges. For a test application, we focus on vulnerabilities, as vulnerabilities are one of the most important consequences for failing to manage dependencies. We mined information on vulnerabilities from the National Vulnerability Database (NVD), which assigns unique CVE identifiers to vulnerabilities, and assigned these cases to specific nodes (functions in the Linux Kernel). We identified vulnerabilities reported from 1999 to 2015, and used these to collect information on about 1,100 functions that were modified, inserted, or removed to fix vulnerabilities.

3. VISUALIZATION FOR CALLGRAPHS

Callgraphs are frequently dense, with neither global patterns nor individual connections being visible when plotted [12]. While one possibility is using community or hierarchical structure to reduce the visual complexity of the graph and convey overall structure, it no longer is possible to the ability to analyze specific dependencies as is our interest. Instead, in order to convey all and only the relevant dependency information, we first focus on immediate neighborhoods of functions of interest (such as a function that a developer might modify, or a function with a known vulnerability), and second visually representing the local position of a function in the overall graph structure relative to other functions. Examples of use would be examining a specific neighborhood of a function or functions of interest, identifying neighbors and finding functions that will potentially be

immediately impacts, selecting out structurally important functions for careful syntactic examination and testing, and having marked edges or side-by-side comparisons for change impact analysis.

4. ACKNOWLEDGMENTS

This work is supported in part by the National Security Agency. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the National Security Agency or the U.S. government. G. Ferreira is supported by a doctoral research grant from CAPES, Brazil (BEX 13713/13-2). M. M. Malik is supported by an award from the ARCS Foundation.

5. REFERENCES

[1] G. Ferreira, C. Kästner, J. Pfeffer, and S. Apel. Characterizing complexity of highly-configurable systems with variational call graphs. In *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security*, HotSoS '15, 2015.

[2] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. The most dangerous code in the world: Validating SSL certificates in non-browser software. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CSS '12, pages 38–49, 2012.

[3] B. Hogan, J. A. Carrasco, and B. Wellman. Visualizing personal networks: Working with participant-aided sociograms. *Field Methods*, 19(2):116–144, 2007.

[4] A. Kurmus, R. Tartler, D. Dorneanu, B. Heinloth, V. Rothberg, A. Ruprecht, W. Schröder-Preikschat, D. Lohmann, and R. Kapitza. Attack surface metrics and automated compile-time OS kernel tailoring. In *Proceedings of the 20th Annual Network and Distributed System Security Symposium*, NDSS 2013, 2013.

[5] D. Le, E. Walkingshaw, and M. Erwig. #ifdef confirmed harmful: Promoting understandable software variation. In *2011 IEEE Symposium on Visual Languages and Human-Centric Computing*, VL/HCC '11, pages 143–150, 2011.

[6] C. McGrath, J. Blythe, and D. Krackhardt. The effect of spatial arrangement on judgments and errors in interpreting graphs. *Social Networks*, 19(3):223–242, 1997.

[7] A. Nhlabatsi, R. Laney, and B. Nuseibeh. Feature interaction: The security threat from within software systems. *Progress in Informatics*, 1(5):75–89, 2008.

[8] J. Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, 2nd edition, 2009.

[9] A. Perer and B. Shneiderman. Integrating statistics and visualization: Case studies of gaining clarity during exploratory data analysis. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 265–274, 2008.

[10] J. Rumbaugh, I. Jacobson, and G. Booch. *Unified Modeling Language Reference Manual*. Pearson Higher Education, 2nd edition, 2004.

[11] S. Schulze, J. Liebig, J. Siegmund, and S. Apel. Does the discipline of preprocessor annotations matter?: A controlled experiment. In *Proceedings of the 12th International Conference on Generative Programming: Concepts & Experiences*, GPCE '12, pages 65–74, 2013.

[12] L. Šubelj and M. Bajec. Software systems through complex networks science: Review, analysis and applications. In *Proceedings of the First International Workshop on Software Mining*, SoftwareMining '12, pages 9–16, 2012.