Poster: Risk-Based Attack Surface Approximation

Christopher Theisen
North Carolina State University
Department of Computer Science
890 Oval Drive, Campus Box 8206 Raleigh, NC 27695
crtheise@ncsu.edu

Laurie Williams
North Carolina State University
Department of Computer Science
890 Oval Drive, Campus Box 8206 Raleigh, NC 27695
lawilli3@ncsu.edu

ABSTRACT
Proactive security review and test efforts are a necessary component of the software development lifecycle. Since resource limitations often preclude reviewing, testing and fortifying the entire code base, prioritizing what code to review/test can improve a team's ability to find and remove more vulnerabilities that are reachable by an attacker. One way that professionals perform this prioritization is the identification of the attack surface of software systems. However, identifying the attack surface of a software system is non-trivial. *The goal of this poster is to present the concept of a risk-based attack surface approximation based on* ***crash dump stack traces*** *for the prioritization of security code rework efforts*. For this poster, we will present results from previous efforts in the attack surface approximation space, including studies on its effectiveness in approximating security relevant code for Windows and Firefox. We will also discuss future research directions for attack surface approximation, including discovery of additional metrics from stack traces and determining how many stack traces are required for a good approximation.

CCS Concepts
D.2.8 [**Software Engineering**]: Metrics – *Process Metrics, Product Metrics.*

Keywords
Stack traces; attack surface; crash dumps; security; metrics

1. INTRODUCTION
According to security expert Dan Geer, "measuring deployable systems' attack surfaces is nontrivial [3]". Determining what exactly what code (i.e. functions/methods) is actually on the whole attack surface from the source (input at an external interface; output through data) to the sink (vulnerability) involves tools such as a slicer [11]. As a result, the attack surface metric has mainly been used for comparing the attack surface of different versions of the same system [5].

Our issue with current approaches to reasoning about attack surfaces is *scalability* and *usability*. If we want to consider the attack surface of very large systems (e.g. Microsoft Windows™), then we must offer methods that can be practically applied to very large software systems that can be used during the development process. Further, if we want to use knowledge of the attack surface to guide prioritizing code-level vulnerability location and removal, then we need to trace from the attack surface into the affected parts of the system. As we discuss below in Section 2.1, certain attack surface methods do not offer visibility into the internal structure of the code.

The goal of this poster is to present the concept of a risk-based attack surface approximation based on ***crash dump stack traces*** *for the prioritization of security code rework efforts*. We treat the appearance of code in crash dump stack traces as a metric for inclusion on an approximation of the attack surface of a software system. We hypothesize that stack traces from crash dumps can be a useful metric for identifying security related code because crash dump stack traces represent code that was being executed when the system was under stress or performing unexpected actions, which matches the state of the system when a vulnerability becomes known. If a high percentage of discovered vulnerabilities are on the attack surface approximation, and the computation time of the approximation is much lower than computing the ground truth of the attack surface for the system, and the approximation contains only a subset of the ground truth, we can consider the approximation a *risk-based* attack surface approximation because it is an efficient means of determining the part of the system that is most likely to contain vulnerabilities.

Risk-based attack surface approximation is a useful approach since it is scalable (it uses a data source that is already collected-in-the-large from complex software systems; i.e. the crash dump stack traces). Also, the results from research into risk-based attack surface approximation can be quickly and widely used in industry since many organizations collect crash dump stack traces. Unlike other approaches, it does not require complex language-specific code analysis tools since it is a report of the dependencies between binaries, files or methods. Hence, risk-based attack surface approximation can be applied to systems that are combinations of different programming languages. Additionally, the approach is scalable, allowing the user of the approach to use any number of stack traces for analysis as appropriate for the software system.

2. RELATED WORK
In this section, we explore worked related to research into attack surfaces and using stack traces and crash dumps for metric generation.

2.1 Attack Surface
The attack surface of an application, as defined by the Open Web Application Security Project (OWASP) [1] is as follows:

1) The sum of all paths for data/commands into and out of the application.
2) The code that protects these paths (including resource connection and authentication, authorization, activity logging, data validation, and encoding).
3) All valuable data used in the application, including secrets and keys, intellectual property, critical business data, personal data, and personally identifiable information (PII).

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.
Copyright is held by the owner/author(s).
HotSoS '16, April 19-21, 2016, Pittsburgh, PA, USA
ACM 978-1-4503-4277-3/16/04.
http://dx.doi.org/10.1145/2898375.2898388

4) The code that protects these data (including encryption and checksums, access auditing, and data integrity and operational security controls).

Some examples of resources that comprise a system's attack surface include the following [8]: open ports; services available on the inside of the firewall; code that processes incoming data; user interface forms and fields, especially web forms; HTTP headers and cookies; application program interfaces; files; databases; and run-time arguments. An attacker can use these resources to attack the system.

Howard et al. [5] describe the attack surface along three dimensions: (1) targets and enablers; (2) channels and protocols, and (3) access rights. *Targets and enabler*s refer to the assets attackers want to access, and the resources they use to access them. *Channels and protocols* are the messaging structure software uses to pass messages to one another. Attackers use these messages to gain access to assets. *Access rights* give legitimate users of a software system access to data. Attackers attempt to gain these access rights for themselves by posing as a legitimate user or bypassing access checks altogether. Howard's analysis, while providing a strong definition for attack surface, is more theoretical and does not provide a practical toolset for evaluating the attack surface of a variety of systems.

2.2 Stack Traces and Crash Dumps

The use of crash reporting systems, including stack traces from the crashes, is becoming a standard industry practice [2,12]. Bug reports contain information to help engineers replicate and locate software defects. An increasing number of empirical studies use bug reports and crash reports to cluster bug reports according to their similarity and diversity, e.g. Podgurski et al. [9] were among the first to take this approach. Not all crash reports are precise enough to allow for clustering. Guo et al. [4] used crash report information to predict which bugs will get fixed. Bettenburg et al. [13] assessed the quality of bug reports to suggest better and more accurate information helping developers to fix the bug.

Additional research has explored how vulnerabilities specifically can be identified and fixed by using data from crashes, Huang et al. [6] used crash reports to generate new exploits. Kim et al. [7] analyzed security bug reports to predict "top crashes," or those few crashes that account for the majority of crash reports, before new software releases.

3. MOTIVATION

Stack traces identify code that was loaded in memory at the time of the crash. Typically used for debugging, traces define the logical path between an external input and the system crash.

Through the parsing of crashes, we can build a set of artifacts that users have had influence over. While bugs can appear in any artifact, the subset of code appearing in these stack traces has important security implications. Malicious users can only exploit vulnerabilities that they can access. If a bad check on incoming data could result in a buffer overflow attack in a specific function, but no outside user can ever pass input to that bad check, then that vulnerability has a lower priority unless code or configuration changes cause it to be exposed. By de-prioritizing code unlikely to be on the execution path, we can limit the scope of what security professionals have to review by having them focus on files that outside users can pass input to. An example of this type of prioritization shown on a system call graph can be seen in Figure 1.

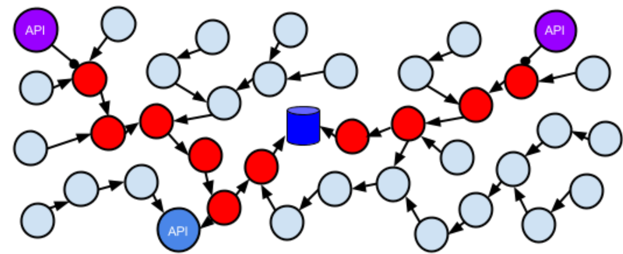

Figure 1: A visualization of the attack surface of a system.

4. METHODOLOGY

To create our attack surface approximation, we first parse a collection of stack traces from the software system we are analyzing. These stack traces are chosen from a set period of time. For the sake of practicality, a random sampling of stack traces from the system is used. We also collect data on the historical security vulnerabilities for the system in the same time period the stack traces were chosen from.

For stack trace parsing, we isolate the binary, file, or function that is seen on each line of each stack trace, recording each one that was seen and how many times it has been seen in a stack trace. The list of code artifacts seen in stack traces and the frequency in which they were seen is stored for analysis.

We collect a series of metrics for use in the evaluation of our attack surface approximation. First, we count the number of files included on the attack surface. We then count the number of those files that have security vulnerabilities. After we have these two counts, we can calculate the percentage of files that are considered on the attack surface approximation at that point, along with the percentage of files with security vulnerabilities that were captured by the approximation.

5. CURRENT RESULTS

Table 1 shows the results of our risk-based attack surface approximation approach as applied to Windows 8 [10]. In this study, we categorized stack traces into two different types: user mode crashes and kernel mode crashes. User mode crashes took place while the system was operating on user programs, while kernel mode crashes took place while the system was executing Windows-specific internal code. By combining both sets of crashes, we covered 94.6% of the vulnerabilities seen in our study's time period with 48.4% of the codebase. This indicates that security professionals could focus security hardening and review efforts on roughly half of the Windows 8 codebase while still identifying 95% of the potential vulnerabilities. In addition this feasibility study on Windows 8, risk-based attack surface approximation is also being applied to the newer versions of Windows (8.1, 10) and Mozilla Firefox for future studies.

6. POSTER CONTENT

The goal of the poster as presented in section 1 is to present the concept of a risk-based attack surface approximation based on

Table 1: Results of our risk-based attack surface approximation analysis for Windows 8 [10]

	User Mode	Kernel Mode	Kernel and User Mode
% binaries	40.2%	7.1%	48.4%
%vulnerabilities	66.7%	40.6%	94.6%

crash dump stack traces for the prioritization of security code rework efforts. Our poster at HoTSoS would be divided into four main sections: an overview of the motivation behind risk-based attack surface approximation, the methodology to perform risk-based attack surface approximation on a generic software system, a presentation of the current results of the approach, and a discussion on future directions for this research. We hope to inspire discussion with HoTSoS attendees on how our new metrics could be used by practitioners to help prioritize their security efforts in finding vulnerabilities before customers or malicious parties find them in the field.

7. ACKNOWLEDGMENTS

We would like to thank the Realsearch group at North Carolina State University for their feedback on this work, and Microsoft Research for their assistance in starting the first efforts in this research space.

8. REFERENCES

[1] Bird, J. and Manico, J. OWASP Attack Surface Analysis Cheat Sheet. *Open Web Application Security Project*, 2015. https://www.owasp.org/index.php/Attack_Surface_Analysis_Cheat_Sheet.

[2] Dang, Y., Wu, R., Zhang, H., Zhang, D., and Nobel, P. ReBucket: A method for clustering duplicate crash reports based on call stack similarity. *Proceedings - International Conference on Software Engineering*, (2012), 1084–1093.

[3] Geer, D.E. Attack surface inflation. *IEEE Security and Privacy 9*, 4 (2011), 85–86.

[4] Guo, P.J., Zimmermann, T., Nagappan, N., and Murphy, B. Characterizing and predicting which bugs get fixed. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10*, (2010), 495.

[5] Howard, M., Pincus, J., and Wing, J.M. Measuring Relative Attack Surfaces. *Computer Security in the 21st Century*, CMU-TR-03-169 (2005), 109–137.

[6] Huang, S.K., Huang, M.H., Huang, P.Y., Lu, H.L., and Lai, C.W. Software crash analysis for automatic exploit generation on binary programs. *IEEE Transactions on Reliability 63*, 1 (2014), 270–289.

[7] Kim, D., Wang, X., Kim, S., Zeller, A., Cheung, S.C., and Park, S. Which crashes should i fix first?: Predicting top crashes at an early stage to prioritize debugging efforts. *IEEE Transactions on Software Engineering 37*, 3 (2011), 430–447.

[8] Manadhata, P.K. and Wing, J.M. An attack surface metric. *IEEE Transactions on Software Engineering 37*, 3 (2011), 371–386.

[9] Podgurski, A., Leon, D., Francis, P., et al. Automated support for classifying software failure reports. *25th International Conference on Software Engineering, 2003. Proceedings.*, (2003), 465–475.

[10] Theisen, C., Herzig, K., Morrison, P., Murphy, B., and Williams, L. Approximating Attack Surfaces with Stack Traces. *IEEE/ACM 37th IEEE International Conference on Software Engineering*, (2015).

[11] Thome, J., Shar, L.K., and Briand, L. Security slicing for auditing XML, XPath, and SQL injection vulnerabilities. *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, (2015), 553–564.

[12] Wang, S., Khomh, F., and Zou, Y. Improving bug localization using correlations in crash reports. *IEEE International Working Conference on Mining Software Repositories*, (2013), 247–256.

[13] Zimmermann, T., Premraj, R., Bettenburg, N., Just, S., Schröter, A., and Weiss, C. What makes a good bug report? *IEEE Transactions on Software Engineering 36*, (2010), 618–643.