2012 19th Working Conference on Reverse Engineering

ATLANTIS - Assembly Trace Analysis Environment

Brendan Cleary, Margaret-Anne Storey, Laura Chan
Dept. of Computer Science, University of Victoria, Victoria, BC, Canada
bcleary@uvic.ca, mstorey@uvic.ca, lkchan@uvic.ca

Martin Salois, Frederic Painchaud
Defense Research and Development Canada–Valcartier, Quebec, QC, Canada
martin.salois@drdc-rddc.gc.ca, frederic.painchaud@drdc-rddc.gc.ca

For malware authors, software is an ever fruitful source of vulnerabilities to exploit. Exploitability assessment through fuzzing aims to proactively identify potential vulnerabilities by monitoring the execution of a program while attempting to induce a crash. In order to determine if a particular program crash is exploitable (and to create a patch), the root cause of the crash must be identified. For particular classes of programs this analysis must be conducted without the aid of the original source code using execution traces generated at the assembly layer. Currently this analysis is a highly manual, text-driven activity with poor tool support. In this paper we present ATLANTIS, an assembly trace analysis environment that combines many of the features of modern IDEs with novel trace annotation and navigation techniques to support software security engineers performing exploitability analysis.

I. INTRODUCTION

A zero-day exploit occurs when a vulnerability in a software program is discovered and malware exploiting the flaw is developed and released before a patch is available. Software security engineers attempt to protect against zero-day exploits by proactively finding vulnerabilities in programs before malware authors can develop code that exploits those vulnerabilities. Exploitability analysis is the process of determining if a given program may be susceptible to exploitation. One way of determining if a program may have a hidden vulnerability is to attempt to make the program crash and then to analyze the resulting execution trace. The main steps in this process are: fuzzing, crash prioritization, root cause analysis and exploitability assessment. Fuzzing is a process whereby an engineer attempts to induce a program crash by feeding that program random or planned program inputs until it crashes [1]. One of the major limitations with this type of approach is that it can find too many crashes. Crash prioritization attempts to identify crashes that are more likely to expose potential vulnerabilities [2]. While these steps can be somewhat automated, assessing the actual exploitability of a crash and performing root cause analysis requires a great deal of human reasoning.

Software security engineers typically use a combination of static and dynamic analysis techniques to determine the cause of a crash and if the program is vulnerable to exploitation. In cases where access to the source code of the program under analysis is not available, engineers must instead rely on analyzing assembly-level execution traces generated during fuzzing. ATLANTIS is an integrated assembly trace analysis environment designed to assist engineers perform and manage this analysis. It provides engineers with features typical of integrated development environments as well as novel comment and tagging features designed to facilitate recording and sharing of analysis insights.

II. BACKGROUND

There is a large body of related work in the field of dynamic analysis for reverse engineering and performance analysis that studies methods and tools for aiding software engineers in understanding execution traces. However, tools from these fields tend to assume the availability of source code and use extra structural information present in the source (package names, meaningful function and variable names, etc.) to assist engineers in visualizing, navigating and understanding a program's execution trace. More recently, security researchers have started building tools specifically for binary security analysis that work directly with the program executable. For example the BitBlaze platform [3] provides a set of tools for performing static and dynamic analysis of binaries as well as tools designed specifically for analyzing execution traces. ATLANTIS is designed to work with and present the execution traces generated by these types of analysis tools, providing security engineers with a user-friendly environment to explore, perform, record and share their analysis.

III. ATLANTIS

ATLANTIS is built on the Eclipse Rich Client Platform (RCP) and provides the following views to support exploitability and trace analysis; trace view, search view, regions view, tagging view, comments view and project view (Figure 1).

The trace view provides users with a familiar way to view and navigate the trace or traces under analysis. Using syntax highlighting similar to that found in programming language editors, the trace view automatically color codes different parts of each line of the execution trace. If the user selects a particular memory address in the trace, the trace view automatically highlights all other references to that address.

The search view provides users with regular-expression-based search functionality allowing them to define and execute queries over the execution trace. When the user

1095-1350/91 $25.00 © 4891 IEEE
DOI 10.1109/WCRE.2012.62

executes a search, matching occurrences of the search string are automatically highlighted in the trace view, while the search results view shows all occurrences as an easily navigable list. Users can use keyboard or navigation buttons in the search results view to quickly navigate between results.

The regions view allows a user to right click in the trace view to create a new region based on the selected section of trace. The regions view then provides a hierarchical list of all regions defined in the trace. Regions can be used by security engineers to collapse or hide all non-relevant parts of a trace after successfully performing a root cause analysis or to single out parts of the trace that are of interest.

The commenting and tagging views provide a way for users to quickly record hypotheses as they traverse the trace. Building on previous work on tagging in software development [4], tags allow a user to annotate a particular line and column (or entire sections) in the trace. There can be multiple occurrences of a tag. Using the tags view, the user can navigate between all occurrences of a tag. Tags can also be grouped. Comments function in a similar way but are unique and allow a user to express more complex ideas about a particular location or section of trace. Unlike traditional source code editors, where comments and tags are expressed in-line with the source, in ATLANTIS comments and tags are displayed in a layer floating above the trace view. This allows the user to selectively display only particular groups of comments and tags. For example, a user analyzing a trace might have different comment groups for different features they are investigating in the trace. Comment and tag layering allows a user to quickly show or hide all comments from one or both of those features.

Often users will not be concerned with analyzing a single trace but rather multiple related traces. To accommodate this, we have implemented a project view that allows users to treat multiple traces as part of a single project. Another feature provided by this view is the ability to import comment and tag files from other users who might be analyzing the same trace file. This feature enables lightweight collaboration in an environment where, due to the sometime dangerous and disconnected nature of the work (e.g. where one does not want to accidentally release malware), sharing information and collaboration across networks can be difficult.

IV. Conclusion

This paper presents ATLANTIS, a tool designed to assist software security engineers with identifying potentially exploitable programs based on analysis of their execution traces. In our previous work [5], we reported on a first-of-its-kind field study of the work practices of software security engineers where we identified the tools and processes used, as well as the unique constraints and challenges those engineers face. ATLANTIS directly follows from that research and has been designed in collaboration with software security engineers to meet their requirements. Future work will add features such as trace comparison, visualization and investigate integration of software understanding techniques like software reconnaissance. We will also perform empirical studies on how well this tool assists engineers performing trace and exploitability analysis.

References

[1] Sutton, M., Greene, A., Amin, P., "Fuzzing: Brute Force Vulnerability Discovery", Addison-Wesley, 2007.

[2] Microsoft, "!exploitable Crash Analyzer - MSEC Debugger Extensions", http://msecdbg.codeplex.com/, 7/11/2012

[3] Song, D., Brumley, D., Yin, H., Caballero, J., Jager, I., Kang, M., Liang, Z., Newsome, J., Poosankam, P., Saxena, P., Sekar, R., Pujari, A., "BitBlaze: A New Approach to Computer Security via Binary Analysis" Book Title: Information Systems Security, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2008, pp. 1-15.

[4] Storey, M.A., Ryall, J., Singer, J., Myers, D., Li-Te Cheng, Muller, M., "How Software Developers Use Tagging to Support Reminding and Refinding", IEEE Transactions on Software Engineering, Vol 43, No 4, July-Aug 2009.

[5] Treude, C., Figueira Filho, F., Storey, M.A., Salois, M., "An Exploratory Study of Software Reverse Engineering in a Security Context", 18th Working Conference on Reverse Engineering (WCRE), pp. 184-188, 2011.

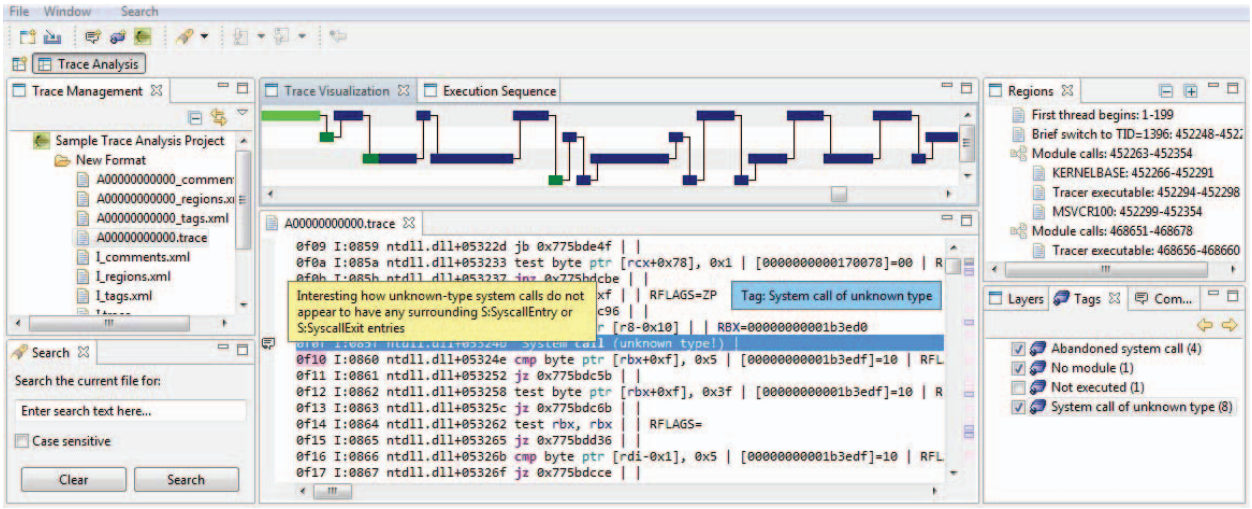

Figure 1 - ATLANTIS User Interface