RAMSeS: Rapid Analysis of Mission Software Systems

December 2, 2020

SAND2020-13784 R

Doug Ghormley
Todd Jones
Denis Bueno
Shelley Leger
Tim Loffredo
Geoff Reedy
Sandia National Labs
Albuquerque, NM

ABSTRACT

Over the past few decades, software has become ubiquitous as it has been integrated into nearly every aspect of society, including household appliances, consumer electronics, industrial control systems, public utilities, government operations, and military systems. Consequently, many critical national security questions can no longer be answered convincingly without understanding software, including its purpose, its capabilities, its flaws, its communication, or how it processes and stores data. As software continues to become larger, more complex, and more widespread, our ability to answer important *mission questions* and reason about software in a timely way is falling behind.

Today, to achieve such understanding of third-party software, we rely predominantly on the ability of reverse engineering experts to manually answer each particular mission question for every software system of interest. This approach often requires heroic human effort that nevertheless fails to meet current mission needs and will never scale to meet future needs. The result is an emerging crisis: a massive and expanding gap between the national security need to answer mission questions about software and our ability to do so.

Sandia National Laboratories has established the Rapid Analysis of Mission Software Systems (RAMSeS) effort, a collaborative long-term effort aimed at dramatically improving our nation's ability to answer mission questions about third-party software by growing an ecosystem of tools that augment the human reverse engineer through automation, interoperability, and reuse. Focusing on static analysis of binary programs, we are attempting to identify reusable software analysis components that advance our ability to reason about software, to automate useful aspects of the software analysis process, and to integrate new methodologies and capabilities into a working ecosystem of tools and experts. We aim to integrate existing tools where possible, adapt tools when modest modifications will enable them to interoperate, and implement missing capability when necessary. Although we do hope to automate a growing set of analysis tasks, we will approach this goal incrementally by assisting the human in an ever-widening range of tasks.

We now live in a time where software is running the world.

- Raja Koduri, Intel Senior Vice President

1. The Problem

Over the past few decades, software has become ubiquitous as it has been integrated into nearly every aspect of society, including household appliances, consumer electronics, industrial control systems, public utilities, government operations, and military systems. Consequently, many critical national security questions can no longer be answered convincingly without understanding software, including its purpose, its capabilities, its flaws, its communication, or how it processes and stores data. As software continues to become larger, more complex, and more widespread, our ability to answer important *mission questions* and reason about software in a timely way is falling behind.

Mission questions about software include questions like:

- What network packets could this malware send?
- Which network commands does this software respond to, and what behavior results?
- What network packets should I send on my network to scan for these IoT devices?
- Does this software have a backdoor?
- Does this mobile application have a logic bomb in it?
- How does this malware change my system in ways I may be able to detect?
- Does this software protect my password from disclosure?
- Does this software protect the integrity of my data?
- Would this mitigation block all inputs that could reach a particular vulnerability in this software?

Much of the software for which we would like to answer these questions is produced outside of our control. In such cases, we have to analyze the finished, third-party binary software artifacts themselves (e.g., executables and firmware) by reverse engineering the binary in detail to understand how it works internally, evaluating the control decisions and tracking key data as it flows through the software. Fortunately, to answer any specific mission question, it is rarely necessary to fully reverse engineer every part of a binary.

Today, this reverse engineering process is carried out predominantly by human experts, answering each mission question for each software system of interest. Unfortunately, even for moderately-sized binary artifacts, this process can be labor intensive, often taking months to years to answer such questions. These costs and time frames are often dramatically out of step with mission needs, which may require answers within days or hours. In other cases, the mission may require analyzing millions of software artifacts – a task that will never be solved with existing approaches.

Unfortunately, the effort required to do the necessary reverse engineering is accelerating. The continued growth in software size causes similar growth in the time needed to manually analyze a typical software artifact. This growth in size is compounded by a growth in complexity enabled by advances in software development. At the same time, software systems are significantly decreasing the time between updates, often to mere weeks, meaning the time to reverse engineer the software may greatly exceed the expected lifetime of any useful answers. Finally, as mentioned, the number of national security questions about software is also increasing rapidly.

All of these factors lead to a situation in which the number of software-related questions is growing swiftly but the rate at which human analysts can answer questions about software is slowing. Heroic human efforts fail to meet current mission needs and can never scale to meet future needs. Vital national security questions about third-party software go unanswered, decisions about software are made without adequate knowledge of what the software actually does, and mission objectives are increasingly at risk. The result is an emerging crisis: a massive and expanding gap between the national security need to answer mission questions about software and our ability to do so.

2. The Solution

Although training a larger reverse engineering workforce would help in some ways, this is a linear solution to an exponential problem and cannot meet our nation's need. Instead, the solution must be informed by the trends that have created the problem – trends in the software development community. The software development community has been radically increasing the ability of any given developer to rapidly and reliably create larger and more complex software artifacts. Except in very limited cases, developers have not been replaced by automation; rather, they leverage a growing ecosystem of automated tools, and they leverage increasing interoperability between software systems to effectively reuse solutions from the growing pool of options. That is, developers are made far more productive through increasing trends in automation, interoperability, and reuse.

To keep up with developers, a solution to the reverse engineering problem must pursue parallel efforts in automation, interoperability, and reuse. Rather than attempting to replace reverse engineers with fully-automated systems, we believe that we need to make the human reverse engineer far more productive through automated assistance.

Fortunately, we need not start from scratch. For decades, the academic community has been laying the groundwork in automated static software analyses for a variety of other purposes, including compilation, optimization, verification, debugging, and code refactoring. Although we can leverage a considerable and growing body of academic research and commercial investment, significant applied research is needed to discover how to effectively apply these techniques to augment reverse engineers answering real-world national security mission questions.

We believe that a long-term, national-scale partnership is needed to start closing the gap between the national mission need and our ability to reason about software.

3. The RAMSeS Approach

To address this need, Sandia National Labs has established a multi-sponsor, multi-activity program called the Rapid Analysis for Mission Software Systems (RAMSeS) program.

Historically, most software analysis activities have been tool-focused (creating a specific tool to do a specific analysis for a specific system of interest), technique-focused (e.g., symbolic execution, fuzzing, code coverage, abstract interpretation), or question-focused (with a team learning to answer a specific mission question about a specific system of interest, regardless of the tools and techniques they use). In practice, these activities have not coordinated fully enough to create reusable, interoperable tools which can provide effective automated support for human reverse engineers facing a wide range of questions. These approaches have proven insufficient to keep up with the trends in software described above and the resulting national security software analysis needs.

In contrast, RAMSeS is part of a fast-growing community of researchers focusing on the foundational, broadly-applicable capabilities needed to answer a wide range of mission questions quickly, avoiding the limitations inherent in an exclusive focus on a single tool, technique, or mission question. We believe that this community of researchers, which includes many distinguished academic, industry, and government groups, is taking the approach necessary to accomplish revolutionary advancement in software analysis.

In pursuit of the goal of advancing the nation's ability to rapidly answer a wide range of high-level mission questions about real-world, third-party software, RAMSeS is specifically exploring statical analysis approaches (without running the software) to supporting the human reverse engineer with increasingly automated software analysis. We are exploring how existing capabilities in software analysis, human factors, reverse engineering, formal methods, data science, and high-performance computing can be enhanced and combined to provide human analysts with an ecosystem of automated or semi-automated software analysis tools to rapidly answer new mission questions about third-party software. We are working to identify reusable software analysis components that advance our ability to reason about software, to automate useful aspects of the software analysis process, and to integrate new methodologies and capabilities into a working ecosystem of tools and experts. We aim to integrate existing tools where possible, adapt tools when modest modifications will enable them to interoperate, and implement missing capability when necessary.

To provide initial focus, we selected three specific mission questions as proxies. We are investigating the core capabilities needed to answer them end-to-end, favoring approaches that contribute to a flexible ecosystem of tools that can apply to other mission questions.

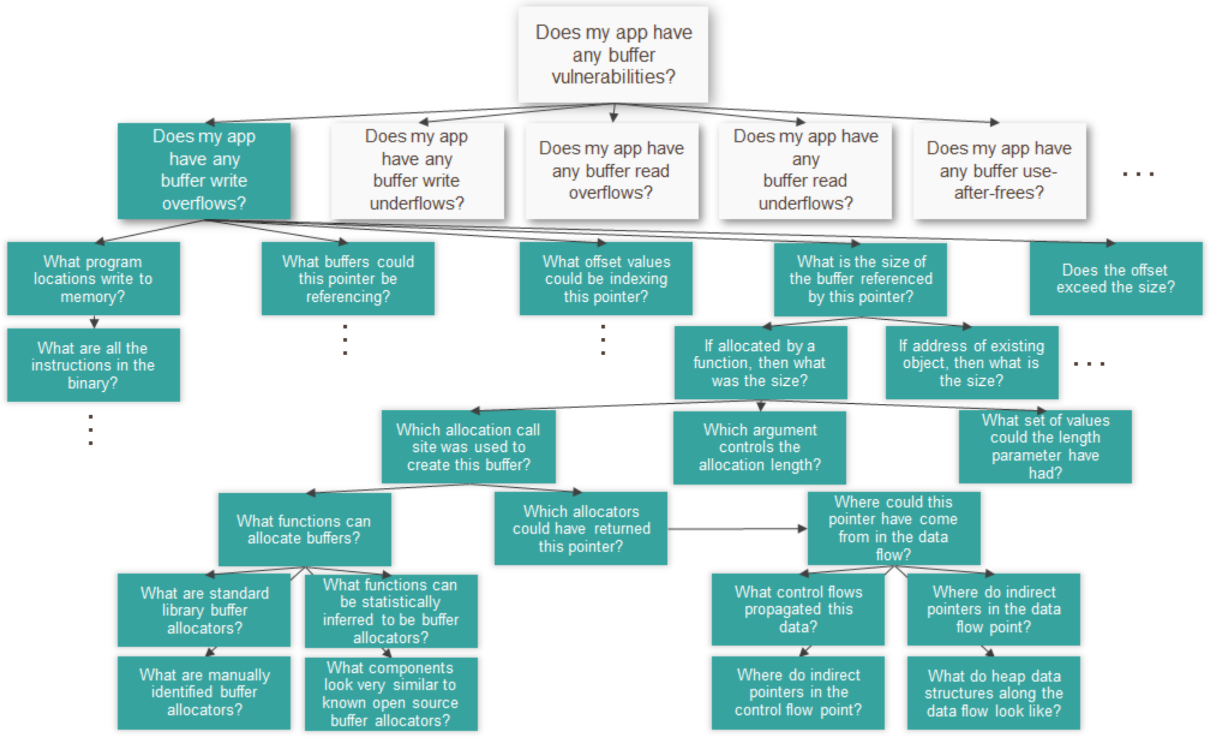
Our three proxy mission questions are:

1. Network Protocol Extraction (NPE). Given a binary, extract information about the network messages it could send or receive.
2. Indicators of Compromise (IOC). Given a binary, identify artifacts the binary could create that would indicate it had run on the system (e.g., files, registry keys, named pipes).
3. Hidden Trigger Detection (HTD). Given a binary, identify locations in the code that may execute unwanted behavior under very narrow circumstances.

These are challenging and interesting research questions, but they are merely starting points. We plan to expand and change our exemplar mission questions over time to aid our pursuit of a practical software analysis ecosystem that helps human analysts answer mission questions about a wide range of real software.

We believe that investment in four core avenues of research is necessary to answer such missions questions:

1. Transforming a high-level mission question into a set of specific technical problems that could be



Does my app have any buffer vulnerabilities?
Does my app have any buffer write overflows?
Does my app have any buffer write underflows?
Does my app have any buffer read overflows?
Does my app have any buffer read underflows?
Does my app have any buffer use-after-frees?
. . .
What program locations write to memory?
What buffers could this pointer be referencing?
What offset values could be indexing this pointer?
What is the size of the buffer referenced by this pointer?
Does the offset exceed the size?
What are all the instructions in the binary?
If allocated by a function, then what was the size?
If address of existing object, then what is the size?
Which allocation call site was used to create this buffer?
Which argument controls the allocation length?
What set of values could the length parameter have had?
What functions can allocate buffers?
Which allocators could have returned this pointer?
Where could this pointer have come from in the data flow?
What are standard library buffer allocators?
What functions can be statistically inferred to be buffer allocators?
What control flows propagated this data?
Where do indirect pointers in the data flow point?
What are manually identified buffer allocators?
What components look very similar to known open source buffer allocators?
Where do indirect pointers in the control flow point?
What do heap data structures along the data flow look like?

Figure 1: An example fragment of a mission question breakdown. This illustrates just one way out of many that the question could be decomposed.

solved by existing or adapted software analysis tools and techniques.

2. Accelerating the reverse engineering process by injecting externally available sources of knowledge.
3. Bridging the semantic gap between machine instructions and meaningful, domain-relevant explanations of program behavior.
4. Forming an ecosystem of tools supporting automation, interoperability, and reuse across a wide range of mission questions

We discuss each of these avenues in more detail below.

3.1. From Mission Question to Software Analysis

Mission questions are typically too broad and complicated to answer directly. We must first decompose them into a set of technical sub-questions each sufficiently focused that we can apply traditional software analysis techniques to solve it. One way to do this is to incrementally break down a high-level mission question into successively finer-grained components specifying the information or activities needed to answer each level. We call this a *mission question breakdown*. Figure 1 shows a fragment of a mission question breakdown.

On RAMSeS, we are investigating how to effectively translate large, complicated mission questions into approachable software analysis questions.

3.2. Injecting External Knowledge Sources

Software does not exist in a vacuum. It interfaces with its environment (including operating systems, hardware, and other applications), incorporates various libraries, follows typical design patterns, implements various standards, and parses according to specifications. There is an ocean of information that *could* help a reverse engineer understand a code base more quickly, were it usefully available (e.g., open source software repositories, text-based internet standards documents, and hardware data-sheets). Also, reverse engineers may have mission-specific information about the uses and limitations of a piece of software or its environment. Such information can make an automated analysis far more effective – often moving the analysis from infeasible to practical.

On RAMSeS, we hope to mine such sources for meaningful information to incorporate into the reverse engineering process.

3.3. Bridging the Semantic Gap

System designer approaches to software development differ vastly from reverse engineer approaches to analysis. The system designer thinks in high-level, semantically rich terms, whereas the reverse engineer has to sift through low-level artifacts to reconstruct each piece of relevant semantic information. For example, while the system designer thinks in terms of application-level components like “network manager”, “security manager”, “display subsystem”, “logger”, and “configuration manager”, the reverse engineer has to reconstruct those components from an unorganized list of functions; the system designer sees meaningful names for data structures and fields, but the reverse engineer often sees only pointer operations with offset values; the system designer thinks about data in semantically rich terms like “security token”, “password”, and “encryption key”, but the reverse engineer sees arrays of uninterpreted bytes. These gaps between the reverse engineer’s view and the system designer’s view are *semantic gaps*[1].

Most tools are not designed to assist human analysts in their higher-level reasoning other than through manual annotation or displaying debug information when available. Consequently, such semantic gaps must be bridged in the brain of the analyst. Unfortunately, analysts spend much of the reverse engineering process drowning in a sea of low-level information and suffering from cognitive overload.

RAMSeS is researching ways to automatically recover useful semantic abstractions and make them available to human analysts.

3.4. A Tool Ecosystem

On RAMSeS, as we’ve considered our selected proxy mission questions, we have made a few observations. First, today there is no single tool capable of providing an end-to-end solution to any of the mission questions we have considered.

Second, the state of the practice in reverse engineering lags significantly behind the academic and open source state of the art in automated software analysis. This lag results in part from the considerable applied research necessary to realize state of the art advances for practical problems. Typical reverse engineers cannot readily perform that applied research, keep up with the multiple releases per year, incorporate the results into their tool of choice, all while accomplishing their mission tasks.

[1]We borrow the phrase from the world of virtual machine introspection, which faces a similar problem [CN01].

And third, many elements are repeated or very similar across multiple mission questions. That is, a substantial set of core software analysis capabilities are foundational to answering many mission questions.

These observations together have led the RAMSeS team to choose to focus on cultivating a flexible tool ecosystem. We envision a human reverse engineer creating a mission capability to answer a specific mission question by assembling software analysis components into an end-to-end analysis, using a mission question breakdown to identify and select appropriate components, manually intervening wherever the tooling is insufficient to the job.

4. Summary

RAMSeS is engaged in a long-term research effort with a goal of dramatically improving our nation's ability to answer mission questions about third-party software by cultivating an ecosystem of tools that augment the human reverse engineer through automation, interoperability, and reuse. Focusing on static analysis of binary programs, we are working to identify reusable software analysis components that advance our ability to reason about software, to automate useful aspects of the software analysis process, and to integrate new methodologies and capabilities into a working ecosystem of tools and experts. We aim to integrate existing tools where possible, adapt tools when modest modifications will enable them to interoperate, and implement missing capability when necessary. Although we do hope to automate a growing set of analysis tasks, we will approach this goal incrementally by assisting the human in an ever-widening range of tasks. Our goal is to grow our ecosystem by producing prototype tools that build on each other and transition these prototype tools to appropriate groups for productization, operations, and maintenance.

5. A Note on Decidability

Many problems in computer science are provably undecidable in the general case or infeasible in particular cases. We are not trying to overcome these limitations. Rather, we note that despite these challenges, practical progress has been made by commercial activities such as Infer [CDD^{+}15] from Facebook and SLAM [BCLR04] from Microsoft. By focusing on specific, relevant automated assistance to the human reverse engineer, we believe that practical progress can be made for a broad range of mission questions.

REFERENCES

[BCLR04] Thomas Ball, Byron Cook, Vladimir Levin, and Sriram K. Rajamani. Slam and static driver verifier: Technology transfer of formal methods inside microsoft. In Eerke A. Boiten, John Derrick, and Graeme Smith, editors, *Integrated Formal Methods*, pages 1–20, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[CDD$^+$15] Cristiano Calcagno, Dino Distefano, Jérémy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter W. O'Hearn, Irene Papakonstantinou, Jim Purbrick, and Dulma Rodriguez. Moving fast with software verification. In Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods - 7th International Symposium, NFM 2015, Pasadena, CA, USA, April 27-29, 2015, Proceedings*, volume 9058 of *Lecture Notes in Computer Science*, pages 3–11. Springer, 2015.

[CN01] Peter M Chen and Brian D Noble. When virtual is better than real. In *Proceedings of the eighth workshop on hot topics in operating systems*, pages 133–138. IEEE, 2001.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.