9-4-2020

Analytic Provenance for Software Reverse Engineers

Wayne C. Henry

ANALYTIC PROVENANCE FOR SOFTWARE REVERSE ENGINEERS

DISSERTATION

Wayne C. Henry, Maj, USAF

AFIT-ENG-DS-20-S-010

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-DS-20-S-010

ANALYTIC PROVENANCE FOR SOFTWARE REVERSE ENGINEERS

DISSERTATION

Presented to the Faculty

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

Wayne C. Henry, B.S., M.S.

Maj, USAF

July 2020

AFIT-ENG-DS-20-S-010

ANALYTIC PROVENANCE FOR SOFTWARE REVERSE ENGINEERS

DISSERTATION

Wayne C. Henry, B.S., M.S.
Maj, USAF

Committee Membership:

Dr. Gilbert L. Peterson, PhD
Chairman

Dr. Robert F. Mills, PhD
Member

Dr. Michael E. Miller, PhD
Member

Maj Daniel J. Casey, PhD
Member

AFIT-ENG-DS-20-S-010

Abstract

Reverse engineering is a time-consuming process essential to software-security tasks such as malware analysis and vulnerability discovery. During the process, an engineer will follow multiple leads to determine how the software functions. The combination of time and possible explanations makes it difficult for the engineers to maintain a context of their findings within the overall task. Analytic provenance tools have demonstrated value in similarly complex fields that require open-ended exploration and hypothesis vetting. However, they have not been explored in the reverse engineering domain.

This dissertation presents SensorRE, the first analytic provenance tool designed to support software reverse engineers. A semi-structured interview with experts led to the design and implementation of the system. We describe the visual interfaces and their integration within an existing software analysis tool. SensorRE automatically captures user's sensemaking actions and provides a graph and storyboard view to support further analysis. User study results with both experts and graduate students demonstrate that SensorRE is easy to use and that it improved the participants' exploration process.

Acknowledgements

There were many people who helped make this dissertation possible. I would like to express my sincere appreciation to my research advisor, Dr. Gilbert Peterson, for his guidance and support throughout this effort. His insight, experience and willingness to allow me to pursue a topic I was passionate about is greatly appreciated.

This research was made possible through the generous sponsorship from the Air Force Research Laboratory. Thank you for your support and trust in me throughout this endeavor. I would like to thank the developers at Vector 35 and the Visual Storytelling for their outstanding products and technical support.

Last but not least, I extend my most heartfelt appreciation to my wife. This dissertation would not have been possible without your unwavering support.

Wayne C. Henry

Table of Contents

Page

Page

List of Figures

List of Tables

ANALYTIC PROVENANCE FOR SOFTWARE
REVERSE ENGINEERS

I. Introduction

"Solving a problem simply means representing it so that the solution is obvious."

Herbert Simon

Exploring software binaries is a time-consuming and complex process. A single analysis session can consist of hundreds of individual steps. After identifying interesting patterns, analysts may need to explore the relationships between them, generate potential hypotheses explaining those relationships, and find ways to verify those hypotheses. Unfortunately, people have a limited working-memory capacity and cannot hold all of these artifacts simultaneously. They may forget previous findings and relationships or forget how they were derived, making it more difficult to explain their findings. Particularly for long and interrupted analysis sessions, often get lost in the problem space: they are unable to examine their progress, unable to synthesize their discoveries, and unable to decide the next steps effectively.

In other highly exploratory scientific fields (e.g., medical analytics and digital forensics), *analytic provenance* techniques have been explored as a potential solution to these problems. Analytic provenance is a sub-field of visual analytics, capturing both the interactive data exploration process and the accompanying reasoning process, releasing analysts from the burden of keeping track of their discoveries [10]. The provenance data can then be visualized to provide different types of support to users, such as recall, replication, action recovery (undo, redo), collaborative communication,

and presentation of findings.

1.1 Motivation

The Department of Defense (DoD) uses information systems that depend on commercial off-the-shelf software, government off-the-shelf software, and free and open-source software. Securing this diverse technology requires highly skilled reverse engineers to reason out the functionality of software and identify vulnerabilities. This process requires hundreds, if not thousands of hours of manual effort. Scaling up existing approaches to address the size and complexity of modern software packages is not possible given the limited number of experts in the world, much less the DoD [11].

Reverse engineers use program analysis techniques and tools to identify and mitigate vulnerabilities, but this process requires considerable expertise, manual effort, and time. Automated program analysis capabilities can reason over only a few vulnerability classes without human involvement, such as memory corruption or integer overflow, but cannot address the majority of vulnerabilities. Identifying most vulnerabilities requires subtle semantic and contextual information, which is beyond the grasp of modern automation.

As a sponsor of this research, the Defense Advanced Research Projects Agency (DARPA) program Computers and Humans Exploring Software Security (CHESS) aims to develop capabilities to discover and address vulnerabilities in a scalable, timely, and consistent manner [11]. Achieving the necessary scale and timelines in vulnerability discovery will require innovative combinations of automated program analysis techniques with support for advanced computer-human collaboration. Due to the cost and scarcity of expert reverse engineers, such capabilities must be able to collaborate with humans of varying skill levels, even those with no previous experi-

ence or relevant domain knowledge. CHESS is seeking research breakthroughs in the following areas:

- Developing instrumentation to capture and analyze the process by which reverse engineers reason over software artifacts;
- Developing new forms of communication and information sharing between computers and humans;
- Creating techniques that are currently hampered by information gaps and require human insight;
- Generating representations of the information gaps for human collaborators of varying skill levels to reason over.

1.2 Problem Space

Reverse engineering is a time-consuming and complex process [12]. Engineers may examine thousands of machine-language instructions (known as assembly) to understand a software binary [3, 12]. High-level programming languages such as Java or C++ include representations that aid program comprehension through expressive variable names, functions, classes, and objects. This information is lost during compilation. Therefore, assembly instructions must be mentally reconstructed during analysis [3, 13]. These challenges are compounded due to the large volume of assembly code in software binaries. Even experienced reverse engineers face difficulty performing reversing tasks given the sheer quantity of data [14, 15].

Researchers and companies have developed a variety of tools supporting reverse engineers [16, 17, 18, 19]. Unfortunately, these tools focus on exploratory analysis, ignoring more complex analytical tasks (e.g., gathering evidence, forming hypotheses, and presenting results). They provide limited features for organizing findings, forcing

users to track them with external note taking tools. Analytic provenance visualization tools may assist reverse engineers to better explore the program structure, identify patterns of interest, and communicate their results to stakeholders.

Analytic provenance tools, such as SensorRE, are closely related to the field of sensemaking. *Sensemaking* reflects how we make sense of the world so that we can take further actions [20]. More specifically, sensemaking is described as the process of collecting, representing and organizing complex information sets in a way that can help us better understand a problem [21]. For instance, sensemaking can be seen in an everyday process such as finding a cell phone that suits our needs. This process may involve searching for different models, learning domain-specific jargon, and comparing the pros and cons between models.

1.3 Dissertation Overview

This chapter introduced some of the unique challenges present in comprehending binary programs, such as large data sets, program complexity, and tool complexity. The primary research hypothesis is that analytic provenance methods may offer cognitive support beyond what is considered state-of-the-art for reverse engineers during exploration, collaboration, and presentation tasks.

This section describes an analytic visualization solution from surveying reverse engineers through implementing the provenance prototype and evaluating it. The research overview is presented in Figure 1.

The three research phases have individualized research questions. The first is:

- *What visualization needs do reverse engineers have?*

Phase I This phase presents an exploratory study examining expert reverse engineers' discovery processes, methods, and visualization needs. By observing existing users, processes, and tools, the dissertation produces fundamental requirements for

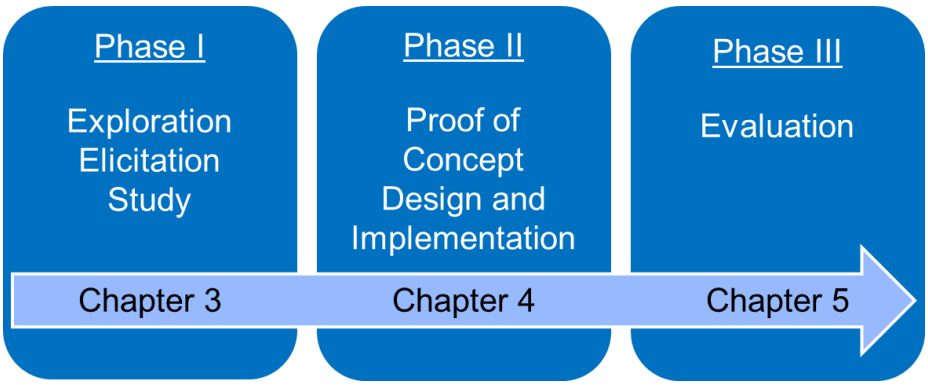


Figure 1. Dissertation overview.

the next generation of solutions. The distilled requirements serve as important design criteria for SensorRE. The engineers reported difficulties managing hypotheses, organizing results, and reporting findings. The results indicate that analytic provenance visualizations may serve as an effective aid, allowing reverse engineers to recall and present formative details of their strategies and decisions.

With an understanding of the domain and capability gaps, the research process shifts to the second research question:

- *How can an analytic provenance tool be designed and implemented to support software reverse engineers?*

Phase II In this phase, an architecture and process for SensorRE is derived that unifies the needs of the users with a technical approach. SensorRE is presented to support reverse engineers' comprehension. The proposed system is a visualization tool integrated with an existing software analysis tool. A cyclic process model is adopted in which analytic provenance data can be used to support sensemaking, as illustrated in Figure 2.

The process starts with a *user* employing a *software analysis system* to solve a reverse engineering problem. During the sensemaking process, both the performed low-level actions (e.g., insert comment, change function, change view), and the produced high-level reasoning artifacts (e.g., findings, assumptions, and hypotheses) are

captured, referred to as *provenance data*. This provenance data should be visualized in a way that can provide support to the ongoing exploration process. The user interacts with both the software analysis system and the *provenance visualization* to solve the problem. These two components communicate with each other to facilitate the interplay between them and the user. The provenance visualization acts as a black box and can be implemented using an information visualization pipeline [22].

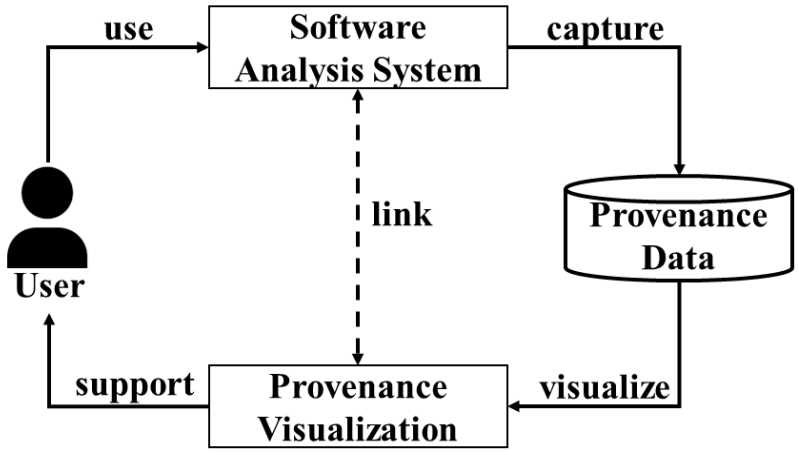


Figure 2. Software reverse engineering analytic provenance model. The two-way dashed line indicates the communication between the provenance visualization and the software analysis system providing sensemaking support [1].

This framework allows users to examine the reasoning relationships between the actions they performed, and it reminds them of what has been done earlier. SensorRE provides an interface for users to assign additional meaning to automatically collected data by spatially grouping actions. SensorRE supports collaboration through the sharing of provenance graphs and the presentation-focused storyboard view.

The third research question then evaluates the prototype and hypothesis:

- *Is the analytic provenance prototype effective at supporting software reverse engineers?*

Phase III Controlled user experiments tested the effectiveness and usability of SensorRE with graduate students trained in reverse engineering as well as with pro-

fessional reverse engineers. The participants were asked to perform routine analysis tasks on a binary including: exploration, validation, collaboration, and presentation of findings. Post-study surveys provide qualitative data for analysis. All participants found the visual representation and interaction of the tool intuitive to use. It helped them to organize information sources, quickly find and navigate to data they wanted, and effectively communicate their findings.

1.4 Research Outline

Toward the overall goal of supporting reverse engineers through the visualization of provenance data, this dissertation contributes:

- **Chapter 2** provides a background and literature review on software reverse engineering, sensemaking, software visualization, and analytic provenance. It also provides background information on SensorRE's implementation technologies.

- **Chapter 3** reports on a user needs study with subject matter experts addressing the first research question. The chapter describes how the interviews were analyzed, represented, and verified, and how it was used to inform the development of the prototype.

- **Chapter 4** presents the design and implementation details for SensorRE. The system is demonstrated through an example usage scenario visualizing the actions a user took analyzing an example program.

- **Chapter 5** describes a formative evaluation of SensorRE with both experts and graduate students. The evaluation examines whether SensorRE improved the participants' ability to collaborate, extend previous findings, and present their results.

- **Chapter 6** discusses the lessons learned during SensorRE design, development, and the user studies. The chapter concludes with implications and areas for future research.

II. Background and Related Work

Chapter 1 introduced cognitive challenges in software reverse engineering and proposed a phased methodology to assist practitioners. Visualizing analytic provenance data may augment the reverse engineer's cognitive processes, enabling the engineer to better recall, replicate, and present formative details of their strategies and decisions.

This chapter begins with a review of the software reverse engineering domain in preparation for addressing the initial research question in Chapter 3. Next, foundational literature in sensemaking and a model supporting the reverse engineers' cognitive processes is reviewed. Analytic provenance literature is then discussed, focusing on the capture, visualization, and utilization of provenance data. An understanding of this topic is necessary for Chapter 4, which presents the SensorRE design and implementation details. Finally, the chapter concludes with a description of the implementation technologies used in the SensorRE tool.

2.1 Software Reverse Engineering

Software reverse engineering is the practice of analyzing a software system to understand its structure, function, and behavior [13]. One classical use of of reverse engineering is to re-document an existing software system whose documentation is lost or lacking [23]. However, practitioners are also in high demand in cyber security to discover software vulnerabilities [24], detect and neutralize malware [25], and protect intellectual property [26]. Each of these specialties relies on an advanced set of tools and processes.

The reverse engineering process can be modeled in three phases: overview, subcomponent scanning, and focused experimentation [27]. Reverse engineers begin by establishing a broad view of the program's functionality. They next use the overview

to prioritize deeper inspection into sub-components (e.g., functions). As the reverse engineers review these sub-components, they develop hypotheses that are tested and answered through manual *static* analysis or *dynamic* execution.

Static analysis involves analyzing a software program before execution [28]. Tools used in static analysis include hex editors, decompilers, and disassemblers. This technique examines the program's structure (e.g., call graphs), file properties, and assembly language contents [29]. Static analysis techniques can be very thorough because they explore all possible execution paths. However, this approach also requires sorting through large quantities of data and therefore can be inefficient [19, 30]. This issue is amplified in programs that deliberately conceal their purpose from the engineer.

Reverse engineers overcome these limitations through dynamic analysis techniques. Dynamic analysis executes the software program with a given input set while recording the run-time execution traces [28]. For many reverse engineering tasks, dynamic analysis offers precise results, sacrificing scalability for a deeper understanding of the executed code path. However, this approach is often incomplete because only one code path is analyzed at a time, and dynamic analysis is dependent on program inputs [18]. Dynamic analysis tools include debuggers, dynamic binary instrumentation tools, and virtualization environments.

Another constraint is that being a skilled reverse engineer requires a great deal of domain knowledge. Reverse engineers need to understand architecture-specific assembly language instructions [31], operating system calls [32], memory stack use [33], process layout [34], and potentially attack and defense techniques [34, 35, 36].

2.2 Reverse Engineering Tools

As with any analysis process, an important need is presenting the right kind of information, in the right amount, at the right level of abstraction. Reverse engineering tools vary in the amount of detail they provide and the kinds of visualizations used. While it is relatively easy to understand a few lines of assembly, the problem is much more difficult when trying to understand thousands of lines of code. In one study, participants analyzing small decompiled code snippets with less than 150 lines required 39 minutes on average to answer common malware analysis questions [37]. Software visualizations have been explored to help reverse engineers solve problems more effectively through intuitive visual representations of the data [38].

Static analysis tools commonly employed in the industry include Binary Ninja [39], IDA Pro [40], and Ghidra [41]. An example of a reverse engineer's complex workspace is shown in Figure 3. These tools support a large number of executable formats for a variety of processors and operating systems. They are also designed for improved functionality and customizability through third-party plugins.

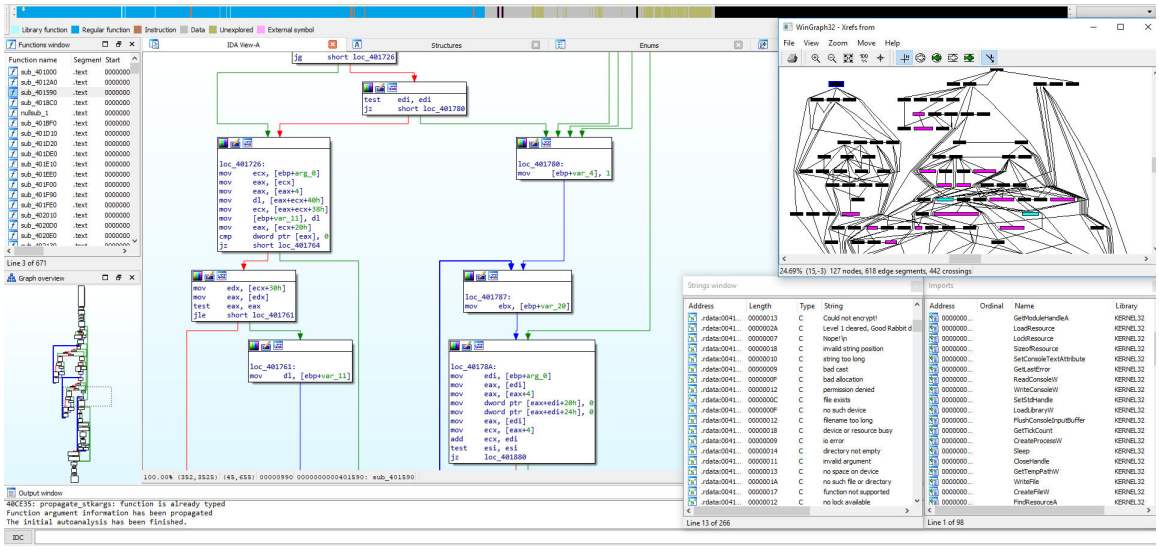
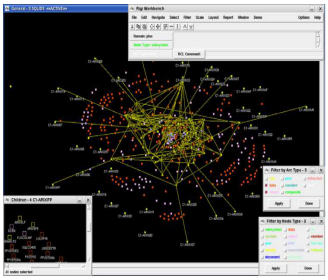


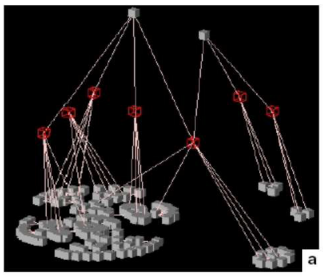
Figure 3. Example workspace in IDA Pro.

There have been many visualization tools developed to support reverse engineers.

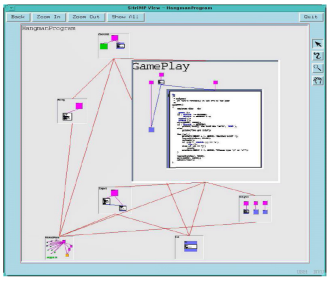
Surveys by Koschke [42], Bassil and Keller [43], and Sim, et al. [44] provide an expansive array of visualization technologies. We highlight a few below.



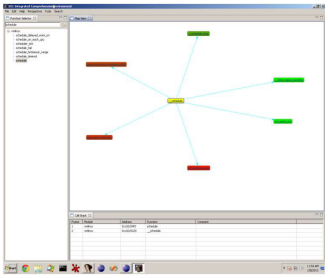
(a) Rigi



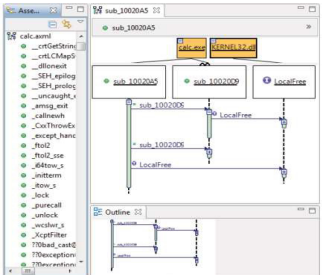
(b) Rigi Extended



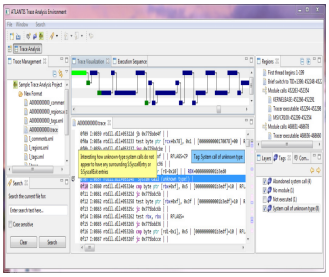
(c) SHriMP



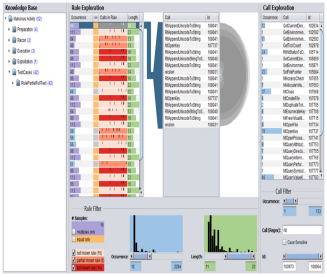
(d) Cartographer



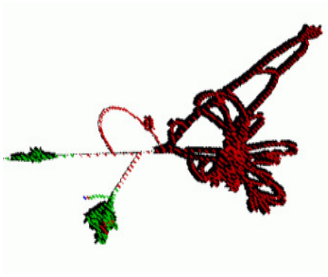
(e) Tracks



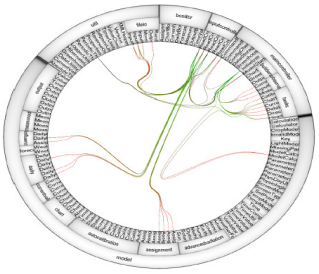
(f) Atlantis



(g) KAMAS



(h) VERA



(i) ExTraVis Circular View

Figure 4. Software Reverse Engineering Visualizations.

Kienle and Muller developed Rigi [45], an interactive reverse engineering visualization environment to explore and visualize source code written in C, C++, and COBOL, as shown in Figure 4a. Rigi supports interactive zooming, layout change, filtering, and summarizing a binary in a multi-window view.

The Rigi environment served as a platform for several other highly-cited software visualization efforts. For example, Rigi extended [46], shown in Figure 4b, added 3-dimensional graphics to the visualization tool. SHriMP (Simple Hierarchical Multi-

Perspective) [47] used fisheye views of nested graphs for Java code, as shown in Figure 4c.

The Rigi system and related efforts are foundational to the field of software visualization. However, because the system required source code or Java bytecode, which preserves package hierarchy information, they are not considered general binary analysis tools [48]. The last official version of Rigi was released in 2003.

More recently, Pucsek developed Cartographer [49] to display binary call graphs as a force-directed graph structure, shown in Figure 4d. This tool supported basic interaction including function name assignment, adding comments to functions, repositioning nodes in the graph, and navigation to associated code in the data source. However, without filtering or layout options, this visualization quickly became unmanageable with larger graphs.

Tracks [19] is a visualization tool using sequence diagrams to display static or dynamic information from software binaries, as shown in Figure 4e. Tracks was built using an open-source sequence diagram tool called Diver [50], in the Eclipse Rich Client Platform (RCP) framework [51]. Similar to Cartographer, Tracks uses socket communication with IDA Pro to allow for greater interactivity, navigation, and control. While sequence diagrams present an intuitive method to relay high-order relations between software artifacts, the horizontal navigation method quickly becomes over-whelming when navigating through many function calls.

Cleary, et al. [52], and later Huang, et al. [16], developed Atlantis, the assembly trace analysis environment, shown in Figure 4f. This interface allows a user to navigate through dynamic analysis data collection. Atlantis is built in the Eclipse RCP environment and provides views to support dynamic analysis through trace, search, regions, and project management views. Future iterations improved the layout and added a register view to observe their behavior during the trace analysis.

To our knowledge, ATLANTIS has not been evaluated for usability and it is not available for public use. Therefore, its ability to improve reverse engineering program comprehension is uncertain.

KAMAS [53] is a knowledge-assisted dynamic analysis system, shown in Figure 4g. The tool visually exposes API system call patterns across a database to assist in malware forensics and classification. KAMAS was developed in Java and borrows traditional programming IDE (Integrated Development Environment) interface design features. The prototype was evaluated through expert review and user study. This system is not available for public use.

Quist and Liebrock [54] released VERA (Visualization of Executables for Reversing and Analysis), a dynamic analysis malware forensics tool, shown in Figure 4h. VERA visualizes basic block sequences as colored loops describing program behavior. VERA supports interaction through zooming, filtering, and panning in both 2-dimensional and 3-dimensional graphs. In a small user study (n=6), VERA was reported to significantly help in the initial analysis of malware. VERA is no longer in development and is not available for public use.

Holten [55] developed ExTraVis, which uses hierarchical edge-bundling in a circular view to display dynamic analysis traces, as shown in Figure 4i. ExTraVis is developed in JHotDraw [56] with OpenGL [57] graphics. The views offer textual details through tooltip hover-on-demand. While the authors only tested the visualization with compiled Java programs, they assert its viability with other programming languages. This research presents an intuitive approach to visualizing hierarchical data and advancements in edge bundling in order to reduce visual clutter. No usability study was conducted on ExTraVis.

2.3 Requirements Elicitations in Reverse Engineering

Relatively few studies have examined the workflow and processes employed by reverse engineers, which has made it difficult for industry to develop software that specifically targets reverse engineers. Treude, et al. explored the work processes of software reverse engineers in a security context [58]. Baldwin, et al. further identified the limitations of visualizations within the reverse engineering domain and developed a methodology to identify associated requirements from two specialized groups of assembly language developers [59]. Through surveys, observations, and interviews with these groups, the researchers identified a range of tooling needs. Kienle and Muller have comprehensive discussions on reverse engineering tool development in terms of requirements, software architectures, and tool evaluation criteria [60, 61]. While these works provide the groundwork for this study, they do not investigate the analytic provenance needs of software reverse engineers.

Meanwhile, in the related hacker community, several empirical studies have explored the tool needs of users, including during reverse engineering tasks [62, 63]. Many of the tools described in such studies were created in an ad-hoc manner and are well-known for being difficult to use [23]. To cope with these difficulties, hackers require high levels of tolerance, patience, and perseverance. While the hackers performed many other activities beyond reverse engineering (e.g., network penetration testing), the progressive methodology employed in these studies for examining the discovery process, methods, and visualization needs influenced the study reported in Chapter 3.

Several studies have also captured and characterized the work practices and analytical processes of individual or collaborative analysis through a qualitative approach. For example, Pirolli and Card studied intelligence analysts and developed a notional model of the analytic processes they follow [2]. In addition, Chin, et al. conducted an

observational case study with professional intelligence analysts in which participants worked on real-world scenarios, either as individual analysts or as an investigative team [64]. The researchers identified processes of intelligence analysts, such as the investigative methodologies they apply, how they collect and triage information, and how they identify patterns and trends. Understanding the analytical processes of reverse engineers, including how they are currently capturing and using provenance, will be helpful for identifying gaps in provenance support technologies.

2.4 Sensemaking

People tend to understand the workings of things by recognizing patterns, making intuitive leaps, validating and refuting hypotheses according to observations, and making adjustments to mental models as needed [2]. For example, reverse engineers try to identify familiar fragments of code based on conventions, expectations, and previous experience. This understanding process is very individualistic and full of trial and error; not everyone builds the same mental model or builds it in the same way or uses the same interpretations.

Recall from Chapter 1 that sensemaking is defined as "the process of collecting, representing and organizing complex information sets in such a way that can help us understand the problem better" [21]. Pirolli and Card [2] presented a sensemaking model that is an iterative process that gradually transforms raw data into reasoning knowledge. The model in Figure 5 describes two major loops of activities: (1) a *foraging loop* that involves processes aimed at seeking information, searching and filtering it and reading and extracting information, and (2) a *sense-making loop* that involves iterative development of a mental model. The sensemaking process can progress upward (from data to knowledge) or downward (from knowledge to data).

The steps in the bottom-up process are summarized as follows:

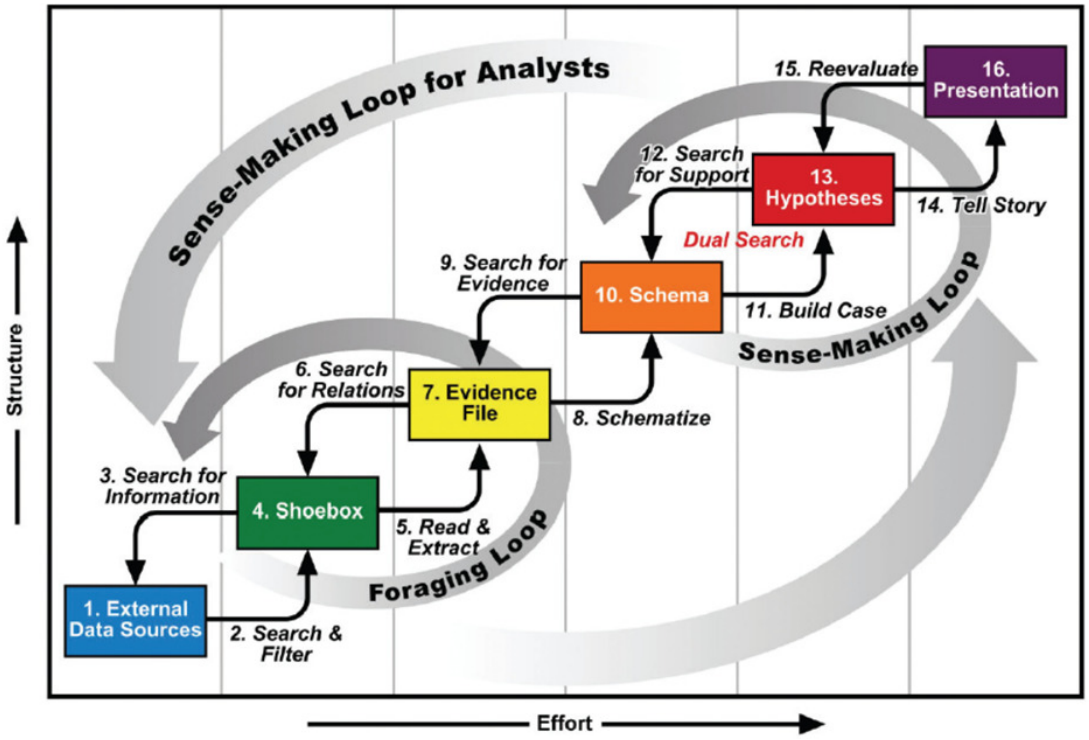


Figure 5. Process model of sensemaking [2].

- *Search and filter.* External data sources, such as classified databases or the web, are searched and filtered to retrieve documents relevant to the task.

- *Read and extract.* These documents are examined to extract pieces of important information that may later be used as evidence.

- *Schematize.* The collected information is organized in a way that aids the analysis. This organization may be executed implicitly in one's mind, using paper and pen, or with support of a complex computer-based system.

- *Build case.* Multiple hypotheses are generated, and evidence is marshaled to support or disprove them.

- *Tell story.* Discovered cases are presented to some audience of interest.

The sensemaking process has been studied in many different contexts including

information science [65], organizations [66], human-computer interaction [67], intelligence analysis [2, 68], and software reverse engineering [3, 69].

The following section reviews literature related to the study of software reverse engineering.

2.5 Sensemaking in Reverse Engineering

Bryant [3] defined the *sensemaking process* in reverse engineering as "a goal-directed planning-based activity, in which the reverse engineer interacts with an executable program using RE tools to construct a mental model of the functionality of the program." This understanding is continuously improved as one reasons and forages through information to construct a mental model. In this assessment, the model is dynamic, continuously updated via an iterative loop of identifying knowledge gaps, seeking information, and then adjusting the mental model. Figure 6 illustrates the reverse engineering sensemaking process.

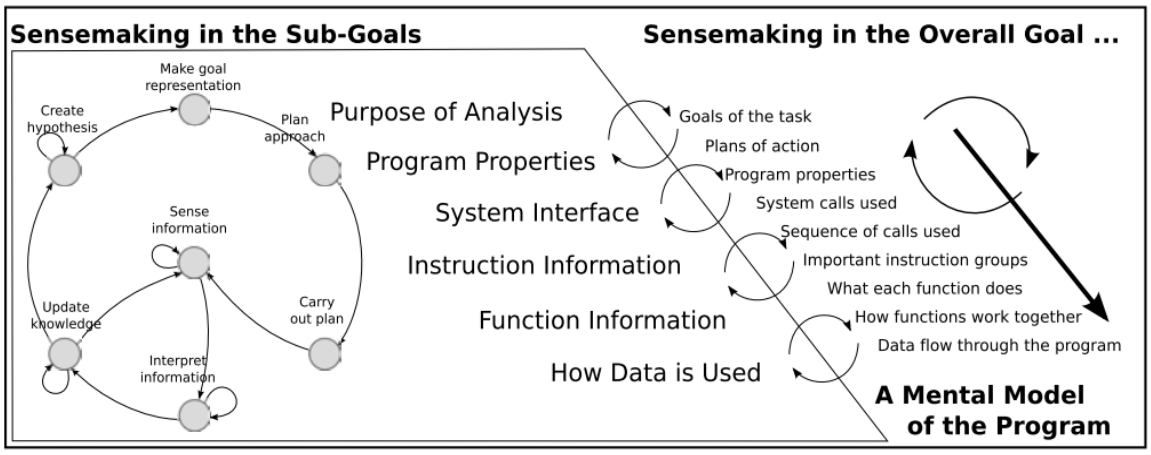


Figure 6. Sensemaking in reverse engineering [3].

Program understanding activities can be somewhat chaotic, with reverse engineers making guesses, questions, and actions using their own experiences. Analysis activities need to become more systematic, but this is difficult to achieve. These difficulties

may stem from the problem that heuristics are often very domain-specific, whereas systematic techniques tend to be quite general. The process of reverse engineering should evolve to make it more repeatable, defined, managed, and optimized [23, 70].

2.6 Provenance

During complex sensemaking tasks, it can be valuable to maintain a history of the data and reasoning involved - referred to as *provenance* information. Provenance plays an important role in many aspects of our daily lives. For example, in everyday shopping, before purchasing a bottle of fruit juice, a customer might like to know about its origin, ingredients, methods used to collect them, processing fruits, and so on. In art, the provenance information of a painting, such as the artist, ownership trail, material, and story behind it, greatly affects its value. In computer systems, the provenance of a piece of data is defined as "the process that led to that piece of data" [71]. It contains input data information, output data, and program configuration used for data processing.

Ragan, et al. [72] characterize the purposes of provenance information as *recall*, *replication*, *action recovery*, *collaborative communication*, *presentation*, and *meta-analysis*. Recall enables awareness of the tasks that have been completed. Replication supports the reproduction of steps to repeat or verify results. Action recovery allows the undo and redo of operations. Collaborative communication supports the sharing of the analysis process with others. Presentation communicates how an analysis was conducted and how findings were determined. Finally, meta-analysis evaluates a user's sensemaking process during an analysis.

Provenance research can be divided into two categories. The first category, *data provenance*, focuses on data derivation history including its source information and the process that produced it. In data-intensive fields such as scientific workflows

and databases, this type of provenance is often emphasized. *Analytic provenance* is the second category focusing on the interactive data exploration and sensory-driven reasoning process.

2.7 Analytic Provenance

Analytic provenance is a visual analysis process often described as "connecting the dots." It is commonly used to provide an overview of the sensemaking process and reveal interesting patterns [73, 74, 75]. Visualization histories automatically record past work, enabling users to easily revisit earlier states of the analysis. There are three stages of analytic provenance as shown in Figure 7. As the user explores the data by interacting with an application, the history of user interactions is *captured*, *visualized*, and then *utilized* by the user to support sensemaking [10]. This section describes a model for capturing, visualizing, and utilizing provenance data.

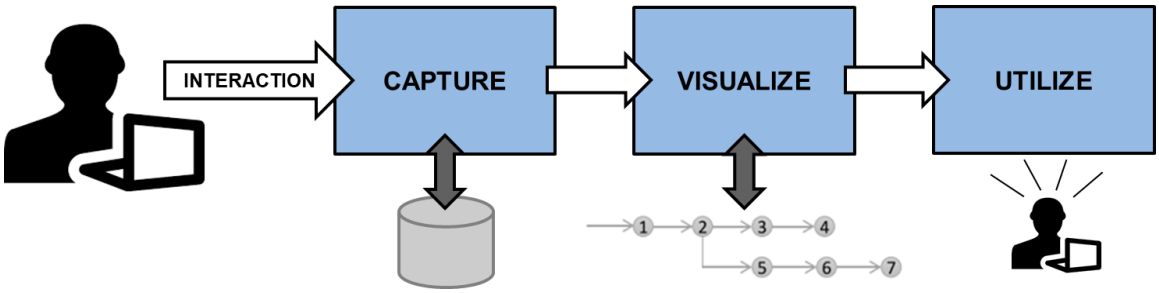


Figure 7. Stages of analytic provenance.

2.7.1 Capture.

From the perspective of human sensemaking, Gotz and Zhou [4] separate semantic richness of information into a four-layer model. Figure 8 illustrates this model with the level of semantic value increasing from bottom to top. The bottom-level consists of low-level user interactions such as mouse clicks and keystrokes, which contain little semantic meaning. The next level up includes *actions*, which are analytic steps such

as querying the database or changing the zooming level of a data visualization. The parameters such as data description and visualization settings are also part of the action layer. Next, *sub-tasks* are the analyses required to achieve the sensemaking goal. At the top-level is the *task*, that is, the overall sensemaking objective.

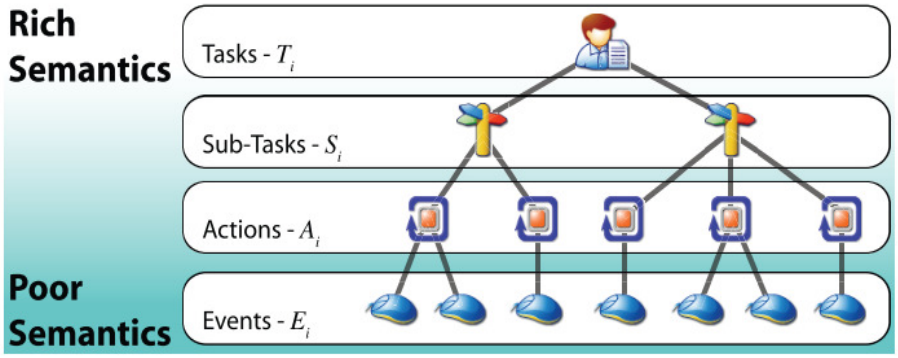


Figure 8. The hierarchical analytic provenance model with increasing semantic richness from bottom to top [4].

This four-layer model is flexible, allowing developers to determine the specific elements they want to capture within each layer for their systems. Capturing lower-level events and actions is relatively straightforward in a visual analytics system. However, such analytic provenance information alone is of limited use [4]. Tasks and sub-tasks provide important clues to the purpose and rationale underlining the sensemaking. They are largely part of the users' thinking, which a visual analytics system does not have direct access to.

The two approaches for capturing high-level analytic provenance data, broadly categorized as *manual* and *automatic* methods [4]. Manual methods rely on users recording their analysis process and sensemaking tasks, while automatic methods attempt to infer the higher-level tasks and sub-tasks from lower-level events and actions. Both methods have strengths and weaknesses. Manual methods tend to be more accurate in their capture, but they can also distract the user from the actual analysis task. In contrast, automatic methods are not obtrusive to the sensemaking

process, but they are limited in their capability of inferring semantics-rich provenance information [4].

2.7.2 Visualization.

Node-link diagrams are a popular choice to show an overview of the sensemaking process [4, 5, 76, 77]. In most cases, nodes represent visualization states and edges are actions that transition the system from one state to another. Besides visualizing the overall sensemaking process, the details of each action are important for recovering the users' thoughts. To provide more context, when a sensemaking step is selected the visual analytics system shows the corresponding visualization state and the action's information [78, 79, 80]. For example, Shrinivasan and van Wijk presented their Aruvi prototype, shown in Figure 9. Aruvi used a horizontal history tree to visualize the captured user actions as states including mouse events, key events, and other input events.

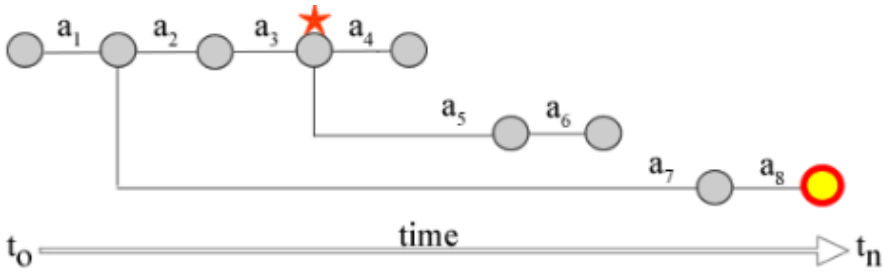


Figure 9. Aruvi prototype history tree showing navigation structure ordered by time [5].

Researchers have explored various methods for capturing history models, visual representations, and operations. Graph-based [81, 82] and tree-based [5, 76] representations have been developed for capturing complex analysis histories. Heer, et al. [83] presented Tableau, a database visualization system used to record interaction histories to support data analysis. Tableau recorded user actions and visualization states as items that could be bookmarked, annotated, and exported. It was primarily

designed to support revisitation and communication. Other notable analytic provenance tools that employ node-link graphs to visually represent the analysis history include: GRASPARC [84], ExPlates [85], GraphTrails [77], VisTrails [76], and CzSaw [78].

KnowledgePearls [6] and CLUE [7] utilize similar interface features to SensorRE, but with different design intentions and domain data. KnowledgePearls records user actions and visualization states during the exploration of biomedical data through a browser-based system, as depicted in Figure 10. The interface also presents the history using a vertical tree layout. CLUE captured user bio-medical provenance data as "Vistories" that can be shared through a storytelling interface. Figure 11 shows a closeup of CLUE's provenance and story views. In contrast, SensorRE visualizes provenance data from an external application. The user can select a sensemaking step and the corresponding visualization state is restored in the reversing application. A linear narrative interface allows users to produce concise stories based on the original exploration.

2.7.3 Utilization.

While there are many possible applications of provenance data, collaboration and storytelling are two major uses that can potentially make a great impact [10].

2.7.3.1 Collaboration.

Traditionally, analytic provenance was used to support an individual's sensemaking process [76]. However, provenance also has many uses in a collaborative context. Analysts can review each other's provenance data to get up to speed on an investigation, to assist in conceptualizing a challenging problem, or as a training aid for sharing best practices. Research suggests that sensemaking activities benefit from

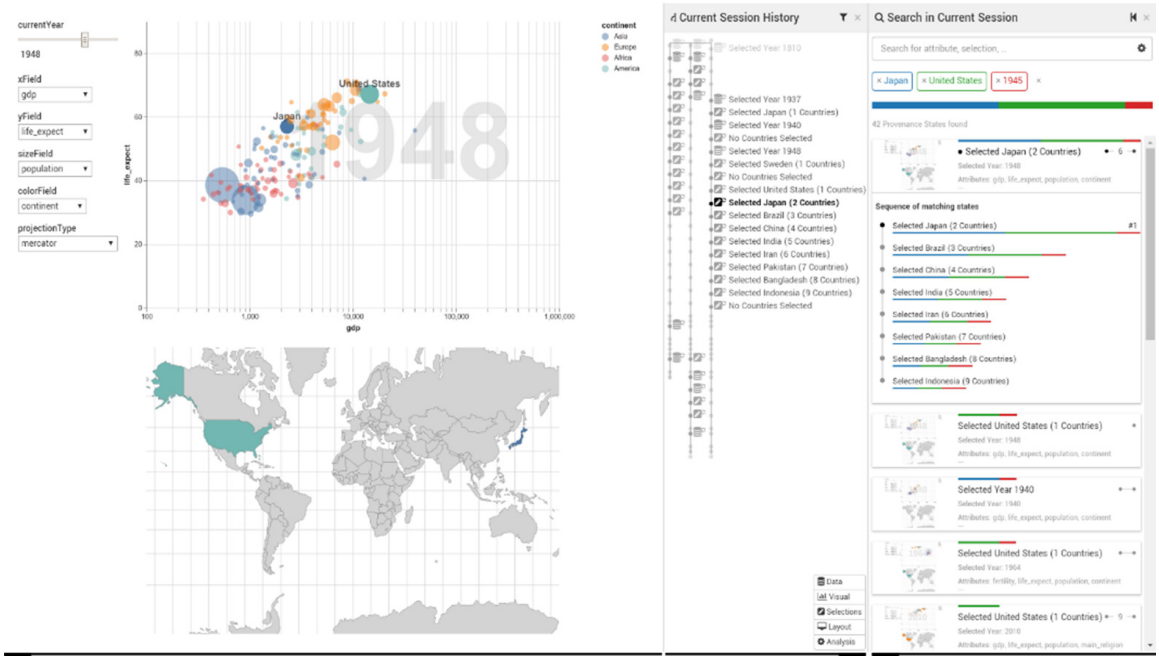


Figure 10. KnowledgePearls prototype. On the left is the application view, in the middle is the provenance graph panel, and on the right is a search side panel [6].

the social aspects of collaboration [66].

Coordination is critical in distributed collaborative analysis. Collaborating analysts need to understand what each person has done and what analysis remains uninvestigated to effectively coordinate their efforts. With distributed collaboration, there is another challenge, known as the "hand-off," in which work started by one collaborator is transferred for continuation by another. Here, the second person needs to learn what has already been done and then chooses where to investigate next. Gaining a good understanding of past work is critical to ensuring effective coordination and minimizing duplicate effort.

Collaborative provenance systems are useful in situations in which members of the team work at different times and in different places. In synchronous collaborative work, real-time shared views and instant-communication attributes can help in building common ground. For instance, CoMotion [86] enables sharing of personal views across the group. Similarly, Cambiera [87] enables an analyst to maintain awareness

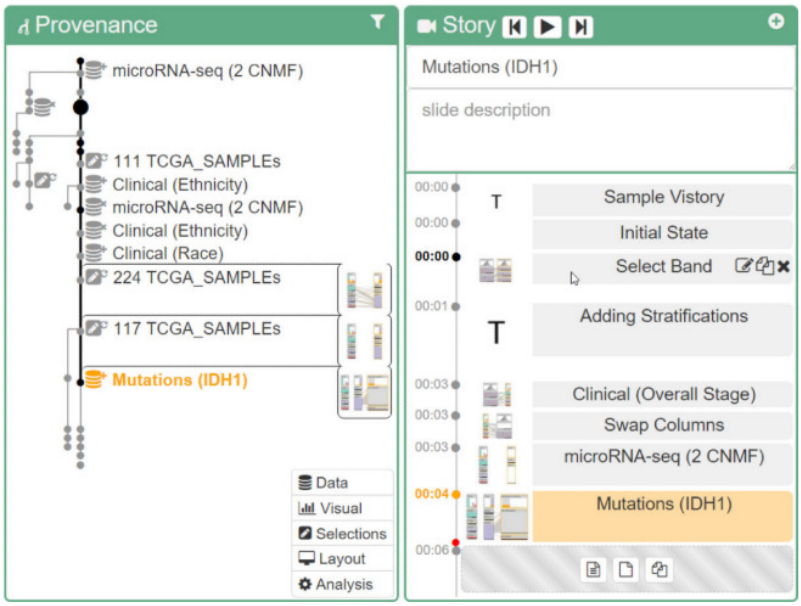


Figure 11. CLUE provenance and story views [7].

of a collaborator's efforts. In an asynchronous context, however, a collaborator must rely on trails of information left behind by the previous analyst.

Asynchronous collaboration tools capture and share findings or hypotheses with a time lag, usually through manual intervention by the user. Sense.us [88], CommentSpace [89], Analytic Trails [90], and ManyEyes [91] are examples of systems that support asynchronous collaboration. Wattenberg, et al. [92] suggested using an information *scent* (i.e., attention pointers that assist a person in navigating the information space) to provide visual cues to potential collaborators to speed up their exploration. Similarly, Willet, et al. [93], incorporated visual cues into interface widgets to help collaborators identify under-explored data.

2.7.3.2 Visual Storytelling.

During the visual exploration process in many fields, including reverse engineering [19, 13], findings are often captured by taking one or multiple screenshots of the visualizations or by creating a screen recording that shows the steps that led to a

discovery. Static images, however, cannot tell the story of a visual discovery, as they cannot convey information about the exploration process. Additionally, results of an analysis often need to be communicated to an audience that lacks technical expertise. Hence, there exists a gap between analysis findings obtained and what is needed to communicate them to a larger audience. This requires results be presented in simpler ways than may typically be used in a sensemaking application. One approach to presenting results is visual narrative construction, during which the user composes findings into a coherent story, a process referred to as visual storytelling [94].

A visual narrative can include raw data, analysis results, visualization, and user notes. Narratives describe the final conclusions in the context of the sensemaking process that led to them, a useful feature for reporting and team collaboration. The DIVA system [95] allows interactive construction of narratives from user annotations and associated visualization states. SchemaLine [96] allows users to create hypotheses or narratives by grouping notes along a timeline.

These tools allow users to construct visual narratives using both story structure and presentation design. Relevant to this dissertation are the ideas concerning the integrated presentation of story content.

2.7.4 Evaluating Analytic Provenance Systems.

A visualization, no matter how novel and interesting, needs to be evaluated to determine whether it meets the design goals and supports the targeted users. Previous work discussed how visual analytics can be evaluated [97, 98, 99]. At present, there are no consensus methods and guidelines for how visual analytics should be evaluated. The prevailing view, however, is that traditional methods and metrics for usability are not sufficient, and novel approaches are required.

Carpendale [100] summarized different quantitative and qualitative approaches

for evaluating information visualizations. Carpendale argued that it takes a variety of methodologies to sufficiently analyze new approaches. Systems should be tested with real users, real tasks, and realistic datasets. We evaluate SensorRE's ability to support reverse engineers completing common tasks with software binaries.

The call for new methods of evaluation is based on several factors. In information visualization, evaluation methods have traditionally consisted of predefined benchmark tests under controlled conditions. Predefined tests, however, are viewed as problematic in this case because visual analytics is exploratory in nature and the set of tasks that users want to perform may not be known beforehand [98]. Furthermore, controlled studies may not effectively represent real situations. Studies with predefined or premature completion times leave little room for insight.

Another problem is that traditional metrics are not well suited to visual analytics [101]. Measures such as performance and accuracy do not necessarily measure the goal of visual analytics. North [74] introduced an insight-based approach to evaluation using controlled experiments both with and without predefined tasks. If tasks are used, they should be more complex than traditional benchmarking tasks or should be tasks that involve uncertainty. Users should be directed to interpret visualizations in articulate written form so their insights can be captured.

Another alternative is to eliminate tasks entirely and study what insights the users gain on their own, letting them explore the data in the way that they choose [74]. In this approach users are initially oriented with the help of starting questions but then left to freely explore the data and report their insights as they discover them. Users verbalize their findings in a think-aloud protocol, enabling the evaluator to capture their understanding. Findings are marked as insight occurrences which can be quantified based on different metrics: complexity, time to generate, errors, insight depth category and so on. During the evaluation, other kinds of usability data can

also be collected, such as which features helped to generate insight and what caused problems for users.

2.8 Implementation Technologies

This section introduces some of the implementation technologies used in developing the analytic provenance tool presented in Chapter 4.

2.8.1 Binary Ninja.

Binary Ninja is a commercial application used in static analysis [39]. Binary Ninja supports a large number of executable formats for a variety of processors and operating systems. Users can also implement lifters for unsupported architectures. Binary Ninja has an extensive application programming interface (API), supporting third-party plugins to improve the application's functionality and customizability for almost every interface element.

Binary Ninja's family of intermediate languages (ILs) are designed to aid in the analysis of computer programs. Each layer works together to provide functionality at different abstraction layers. As the user increases the abstraction level from low, medium, and then high, groups of assembly instructions are transformed into a simplified representation, extracting useful information. For instance, in the medium-level IL all registers are translated into variables, and all variables have types.

An example of the Binary Ninja default graph view is shown in Figure 12. In the main window, adjacent assembly instructions are grouped into basic blocks with entry and exit paths specified by jumps or branches. On the top left, Binary Ninja displays all recognized functions or statically linked operating system calls in the function listing. Below is a cross reference listing, showing references to user-selected functions, easing navigation. The bottom left displays a mini graph with a zoomed-

out view of the function selected in the main window.

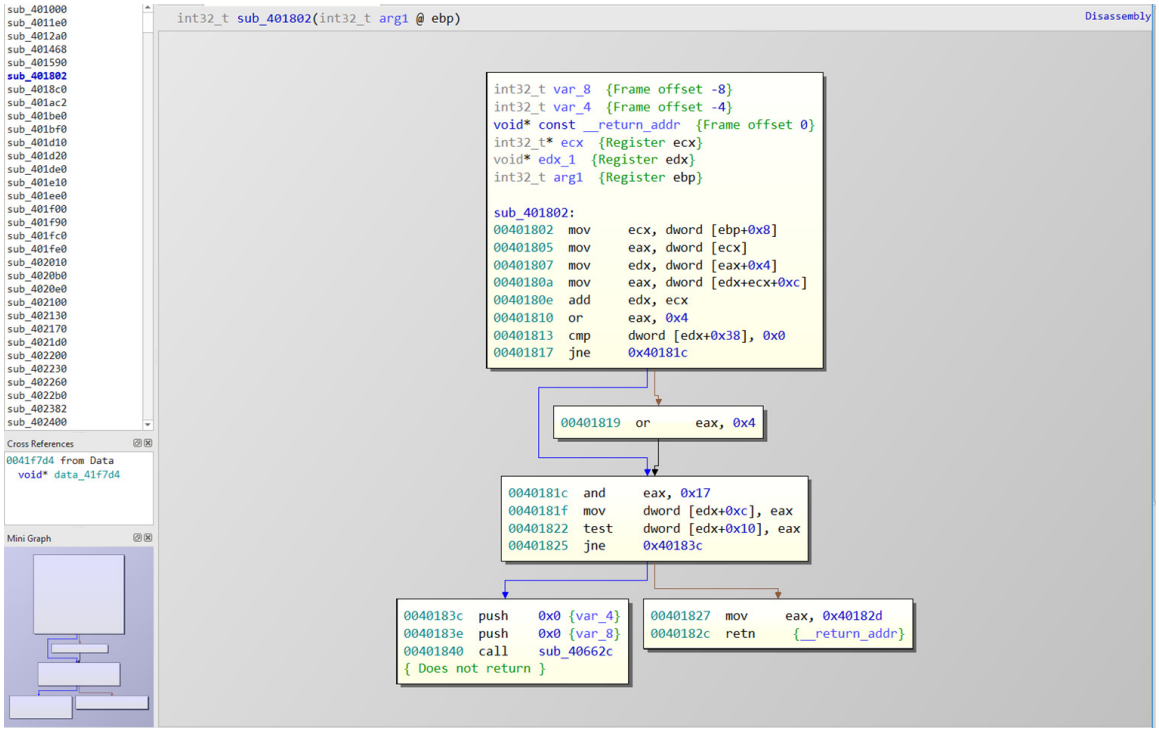


Figure 12. Binary Ninja Reversing Application.

2.8.2 Web Application Platform.

With the emergence of web libraries for interactive visual exploration [102, 103, 104, 105, 106], users are able to explore a wide variety of data from their web browser. Support for animated transitions and event handling gives users the ability to interactively filter data, change views, or dynamically steer computations. This interest has led to modular systems that can be applied to many problem domains.

The Data-Driven Documents (D3) JavaScript library is a graphing library that enables inspection and manipulation of the World Wide Web Consortium (W3C) standard document object model (DOM). The DOM is an object-oriented representation of a web page that can be modified with a scripting language such as JavaScript. D3 is an information visualization (InfoVis) tool that efficiently manipulates docu-

ments based on underlying data. It utilizes scalable vector graphics (SVG) to create a wide array of dynamic visualizations [103]. The use of SVG allows D3 to support partial graph modification, interactive manipulation, transitions, and easy debugging through a web browser's built-in element inspector. These features are not available in many stand-alone graphical development systems, such as Processing [107] or Protovis [108].

Visual Storytelling is an open-source, web-based visualization library used for creating and manipulating provenance graphs [109]. The visualization views are implemented using D3.js through JavaScript and TypeScript interfaces. Visual Storytelling has three modules: *provenance-core*, *provenance-tree-visualization*, and *slide-deck-visualization*. Provenance-core provides tools for managing data structures and tracking interactions in the web application. The provenance-tree-visualization module updates and records the state data within the provenance graph. Slide-deck-visualization offers a D3.js interface to visualize specific provenance graph nodes in a slide-deck interface. SensorRE uses the Visual Storytelling libraries as part of the first analytic provenance system for reverse engineering.

2.9 Chapter Summary

This chapter reviewed the current state of software reverse engineering and provenance techniques. Visualization methods have found practical application for reverse engineers. However, there has been no research into developing analytic provenance tools to support software reverse engineers.

III. User Needs Study

The software reverse engineering industry is dependent upon having useful tools. To be considered 'useful' the tool must meet the needs of the people that will use it. This requires consideration of a number of factors including the capabilities and working patterns of the users, the environments in which the tool will be used, and the system(s) it will be a part of [23]. Understanding these complex factors requires a highly-interactive communication process with the problem owners. The elicitation of requirements is arguably considered the most important step in software development [110].

This chapter presents a focused investigation into the user needs for a new reverse engineering visualization tool. The research question motivating this study is: *What visualization needs do reverse engineers have?* Semi-structured interviews with experienced reverse engineers led to an understanding of their work processes and visualization needs. Iterating on these needs with the users informed the design of an analytic provenance tool. The request for human experimentation is included in Appendix A for reference.

3.1 Methodology

3.1.1 Participants.

The population queried for participation are experienced software reverse engineers. Participants with fewer than two years of hands-on experience in reverse engineering were excluded.

Table 1 shows participant *P1* - *P5*'s background. On average, the participants had 8.6 ($\pm$ 4.6) years of reverse engineering experience in the security sector. Their duty titles included "malware analyst" *(P1, P2)*, "cyber tools analyst" *(P3, P4)*, and

"software analyst" *(P5)*. Two participants have PhDs in Computer Engineering, one with two Masters degrees (Computer Science, MBA), and two have Bachelors degrees (Mathematics and Computer Engineering).

Gaining access to participants who already possess a strong understanding of software reverse engineering has been shown to be a common limitation in related works [3, 23, 47]. However, our population was quite skilled. To get skilled participants, we focused on emailing contacts at national security organizations within our personal and professional networks. These contacts forwarded the request to engineers within their organizations. The participants were under no internal or external pressure to participate.

3.1.2 Procedure.

The in-person semi-structured interviews began with a quick introduction describing the purpose of the interview. Each interview lasted approximately two hours. The interviews were performed off-site due to security limitations but near the participants' work location. The open-ended interview questions were pilot-tested and refined based on feedback from a reverse engineering expert with 12 years experience. The interview protocol provides a brief introduction to the researcher and sponsor, the objectives of the study, instructions to review and sign the consent form, and the break protocol. These questions were designed to inform the researcher on task anal-

Table 1. Participant demographics.

Participant	Duty Title	Exp. (Yrs.)	Education
P1	Malware Analyst	4	MS CS
P2	Malware Analyst	13	BS Math
P3	Cyber Tools Analyst	10	PhD CE
P4	Cyber Tools Analyst	5	BS CE
P5	Software Analyst	11	PhD CE

ysis workflows, tools, challenges, and visualization needs. Each interview discussed the following questions:

- What tasks do software reverse engineers perform?
- What key decisions do you make during the analysis?
- What tools do you regularly use and how do you use them?
- What are the results of analysis and how are they shared with external stakeholders?
- How are findings shared between team members?
- How do time constraints modify your analysis process?

With the permission of the participants, the interviews were audio recorded and extensive notes were taken. The researcher encouraged participants to talk freely about the issues encountered during the binary analysis process. Prompts were used to encourage them to provide more detail. Given the participant-driven nature of exploratory interviews, we received varying levels of responses (e.g., one participant talked at length about teamwork concerns while another focused on workflow challenges). The interview questions consist of six parts as referenced in Appendix B. The ethics approval package is included in Appendix C.

Due to security concerns related to the practical application of the techniques discussed, participants were encouraged to provide generic examples and practices to avoid disclosing national security information.

3.1.3 Data Analysis.

Following the interviews, audio data was transcribed and compared with the researcher's notes for accuracy and completeness. Each response to a question category

is summarized and reported. Researchers followed Creswell's [8] data analysis procedure by iteratively coding and categorizing themes using the data analysis software MaxQDA [111]. We grouped common practices, tools, challenges, and constraints into high-level categories. These categories were refined as we gathered more data.

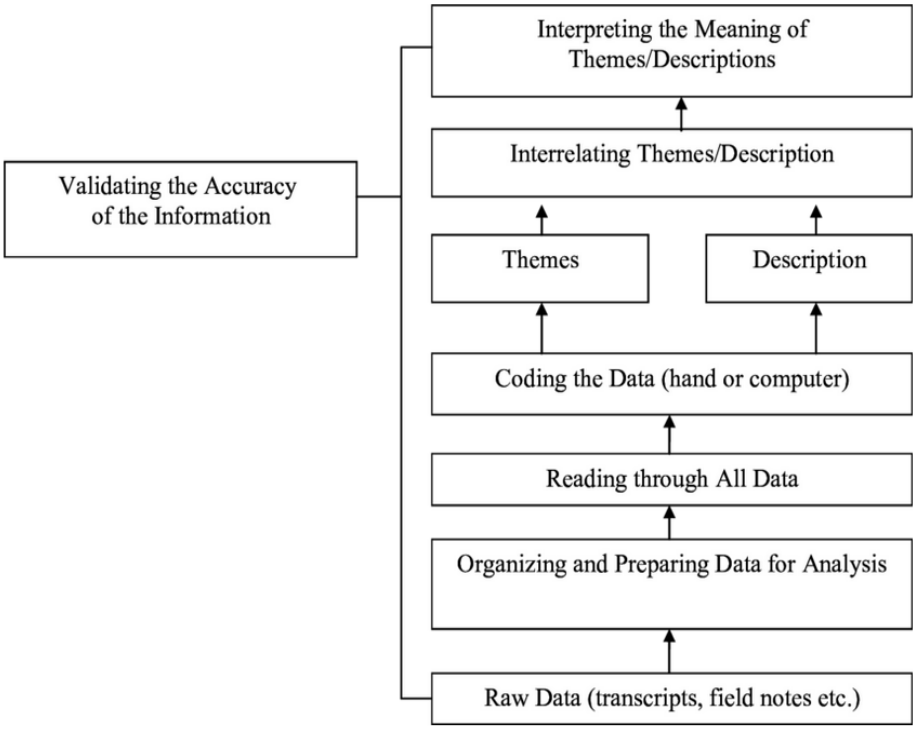


Figure 13. Data Analysis in Qualitative Research [8].

The qualitative data from both the textual and audio recordings are compared using the data analysis software. The results of the study can then be compared to see if there are any design goals not explicitly reported. For example, if the words "trace route" is mentioned noticeably often, we could deduce that it is an area of interest despite not being mentioned as a specific issue. Reviewing the collected data for themes, or *insights*, informed the development of the visualization tool.

3.2 Results & Analysis

Each participant described their typical workflow during the semi-structured interviews. The primary finding is a consistent set of tasks being described. We identified five common processes in the reverse engineering workflow: *metadata, disassembly, dynamic, documentation,* and *collaboration*. The evolution between these themes is captured in Figure 14.

We generally observed the same processes and methods were used by the reverse engineers performing malware analysis as those that were seeking vulnerability discovery or general program understanding. However, there were differences in what the participants prioritized based on their role (e.g. malware analysts focused initially on network calls while others looked at memory management functions). Since the focus is on the high-level processes and methods used, we discuss both groups together in the following sections.

3.2.1 Metadata Analysis.

Metadata analysis was the first procedural step described by all participants. The term *metadata* in this context is static analysis data which provides information about the binary prior to deeper inspection. The engineers perform metadata analysis to form initial impressions on files with little or no prior knowledge, as stated by one subject, *"script as much as we can to point the analyst in the right direction." (P3)* Sources of metadata include the file name, size, header information, file structure, and compiler artifacts. These data sources assist in generating a broad understanding of the binary executable and potentially form hypotheses warranting further examination.

The specific methods and tools used change based on the reverser's objectives, the file under analysis, and how much information was already known about the file. *"[We*

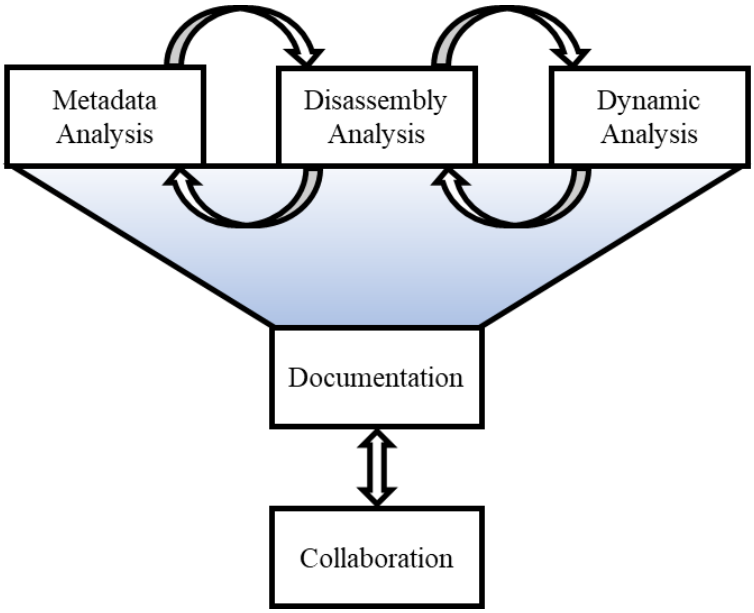


Figure 14. Reverse engineering workflow process.

use] a lot of utilities for understanding the header information for files, sometimes we have files that we're not really sure what they are." (P2) These tools comprise hex editors, binary difference tools, Linux utilities (e.g., file, objdump), packer detection (e.g., PeID), the Windows Sysinternals suite, PE viewers (e.g., CFF Explorer, PE Explorer), and many more depending on the situation. The metadata analysis phase is the quickest of the five phases described, and *"usually completed within an hour." (P2)*

Given the broad range of tools required, the participants described their reluctance with non-scriptable stand-alone tools. Instead, the participants preferred tools that could be integrated into other tools. *"There are too many tools we use all the time that we build scripts to streamline our processes. Ideally, new capabilities could tie into one of the [tools] we already use" (P3).* An analytic provenance tool could be non-intrusively integrated into existing software analysis tools, capturing and then displaying the visualized history data in order to minimize the impact on the reverse engineers' workflow.

3.2.2 Disassembly Analysis.

Each participant indicated the majority of their time is spent statically analyzing code in a disassembler. All participants stated that IDA Pro was their disassembler of choice with Binary Ninja and Ghidra as the next most used. When deciding which disassembler to use, all participants reported cases where they would examine a given binary executable with multiple disassemblers at the same time to piece together their understanding. *"Different tools produce different representations which can be helpful in piecing together the functionality of the binary." (P5)*

In describing the strengths and weaknesses between the disassemblers, the participants highlighted the power of plugins in supplementing the base tool. Participants *P1*, *P2*, and *P4* stated that they use plugins "heavily" during analysis, while P5 only "occasionally." Due to classification concerns, details on the specific plugin capabilities were not shared. However, desirable attributes of the tool's plugin interface were described, including ease of development, API documentation, and active development community. These factors contributed to their choice of IDA Pro as the preferred tool in this phase of analysis, followed by Binary Ninja.

Next, the participants described their analytic process and workflow. The reverse engineers initially examine high-level details such as system calls, strings, or other structural information. The comprehension of these details are further refined through the goal-driven exploration activities.

> *[We are] looking for a handful of things that might indicate what kind of inter-process communication this binary has, either internally or externally. Statically, we want to find out what [the program] interconnects with. How it interacts with the file system, network, or something embedded inside it. (P2)*

Analyzing program interaction through system calls is a critical step in the engineers' workflow, *"API calls or custom programming calls tell us a lot of information about what [the binary] is doing." (P1)* Reverse engineers use the import table and an iterative process of examining the system calls to piece together how the program functions.

3.2.3 Dynamic Analysis.

Dynamic analysis was viewed as an optional but useful analysis step by the participants. Dynamic analysis traces the application's events at run-time, allowing the engineer to inspect stack variables and develop an understanding of the data flow. The participants most often used debuggers in this step and iterated their findings back into the disassembly analysis. *"Dynamic analysis [is] more useful once you already have an understanding of what's happening because then you can look in the stack, you can look at a stack trace and figure out what's there, or stop it at a certain point." (P3)*

Four participants (*P1, P2, P3, P4*) described difficulties performing dynamic analysis, restricting the usefulness of the technique. These difficulties included when portions of the program were missing, when particular anti-debugging or anti-tracing techniques were used, or when the binary's system environment could not be virtually instantiated.

The most frequently used dynamic analysis tool was the debugger (e.g., IDA Pro's integrated debuggers, OllyDbg, or WinDbg), followed by virtual machine emulation tools (e.g., PANDA, Intel PIN), and occasionally fuzzers (e.g., Peech, AFL, and Sully). Participant *P5* described using fuzzers primarily for long-term, exhaustive analyses which was not common-place at their work center.

3.2.4 Documentation.

The reverse engineers discussed three reasons for documenting results: personal note taking, internal team reporting, and external stakeholder collaboration. During the exploration of the binary executables, the participants document just enough information to be able to resume a task and rarely document the paths that were explored without success. The engineers described the following phases of documentation and current limitations in their process:

- **Copying-and-pasting**: To keep a record of their findings, reverse engineers copy and paste images (e.g., graphs) or textual representations of the binary executable into Microsoft PowerPoint, OneNote, or other electronic notebook software. These actions are purely *overhead*, disrupting the researcher's workflow while manually duplicating information that the computer should be able to track automatically.

- **Writing notes**: Most early documentation is written using plain text files, creating small diagrams, and note saving. Notes provide some context for what the researcher is thinking at the present moment, but they are not linked to the binary executable code to which they refer. For instance, *P1* stated, *"I may record that at this [address] I saw something interesting, and may want to follow up later."* Microsoft Word or lightweight text editors such as notepad++, gedit, or vi were commonly used.

- **Organizing notes**: The problem with keeping one central notes file is that it can become hard to search through it, and the problem of keeping many specialized files describing a system can make it hard to sort through when compiling results. Like copying-and-pasting, the burden of coming up with a scheme to organize notes is purely overhead.

The participants expressed a general attitude that intermediate products such as hand-sketched diagrams and comments were "*ad hoc*," "*experimental*," or "*throw-away*." Engineers noted a significant amount of their work ultimately did not validate a useful hypothesis, and so they end up discarding the personal notes. One senior reverse engineer described, "*you go down a lot of dead ends, and you come up with a bunch of hypotheses. 8 out of 10 are dead ends." (P3)* The same engineer went on to say he lacked a process to tell others "*don't look here because I looked here and it's not useful. There are rarely remnants of dead ends*." The reverse engineers intentionally discard intermediate products when the end result does not seem insightful. Recording these analysis paths may prove useful in communicating progress with team members, discussing issues encountered, or as a training tool to learn from past successes and failures.

At the end of analysis, the participants typically record their findings using Microsoft Word including a summary of findings, screenshots, diagrams, and recommendations. Translating the low-level findings into high-level results for consumption by non-technical stakeholders is a constant challenge. One engineer describes this process: *"Typically, what I want to provide is evidence. So I take a lot of screenshots, [and] put them together with diagrams that document how this program works." (P5)* Diagrams are most frequently developed in Microsoft PowerPoint or Visio and imported into the Word document. The length and quality of the reports vary based on the speed, goals, and complexity of the analysis:

> *One report we publish is called a Software Quick Look (SQL). It's a high-level description of the software that gives an understanding of what we think the software does, roughly how it works, and how it could be used ... [The report] could be anywhere from 10 pages to 40 or 50 pages. (P1)*

3.2.5 Collaboration.

The participants reported meeting regularly with other colleagues to discuss long-term projects and immediate next steps. When working on large or complex projects, the participants reported an average team size of 2 - 5 people. Documentation alone is often not enough to understand the work that has been completed by somebody else: *"[I would] look at another engineer's notes, but probably only two or three things would make sense and then I'd still need to jump through [the code] to understand." (P5)* During challenging analyses, sharing intermediate products with other engineers was considered a necessity. All participants agreed that there are many times where it is necessary to ask for assistance or agreement on a finding. As one engineer jokingly states, *"You have a bunch of people that kind of know a little bit about reverse engineering scampering around semi-blindly trying to find cool things that are happening and then make sense of them." (P2)*

When the participants were assigned to teams, they typically work in a co-located facility that allows for face-to-face communication among team members. We found that the engineers did not typically apply specialized collaboration tools in their analysis, but rather relied on basic tools such as shared documents (hand-written or electronic), white boards, and a lesser-used common knowledge repository. Diagrams are most often used to summarize program behavior or highlight specific insights through flow graphs, sequence diagrams, and time-line graphs. *"There are huge needs for improved communication. It's really similar to team-source code development. You don't want to have multiple people working on the same part." (P4)* The participants felt that the specialized collaboration tools were often too primitive to help them. These tools are designed only to provide a common space to collaboratively work together but do not effectively communicate hypotheses or insights during the sensemaking process.

3.3 Discussion & Implications

This exploratory study sought to understand the participants' reverse engineering workflow to find opportunities for tool development. Improved reverse engineering tools will likely always be needed to deal with the rapidly changing software landscape. However, by analyzing the interview data we identified several potential tool concepts that could enhance reverse engineers' investigative and analytic capabilities.

3.3.1 Provenance Capture.

Analytic provenance captures the history and lineage of all of the actions performed by the user during the exploration process. Existing tools and prior research efforts in other heavily researched domains (e.g., intelligence analysis) have focused on supporting the capture of provenance data, but this not a solved problem. Heer and Agrawala [112] note that new visual analytic tools receive better user engagement and acceptance when they are integrated into the users' existing workflow tools. This restricts the selection of the domain-specific tools to ones with API's capable of capturing and storing fine-grained user activity.

Based on the interview data, the captured analytic provenance should be used to reproduce the binary manipulations that are performed by the engineer during analysis. Automated methods are imperfect, however, and there is a lack of clarity on the details of the provenance that needs to be captured. The reversing application and interfaces for future visual analytic systems need to be designed with provenance capture as a significant design focus. In a preliminary assessment of the reverse engineering tools described by the participants, Binary Ninja has the capability to automate the capture of user actions without significant external tooling, making it a potential leading candidate for provenance tool development.

Besides capturing user actions, capturing annotations while performing the analy-

sis would provide important exploration process knowledge. Support for annotations in current reverse engineering tools usually consists of making comments assigned to particular instructions. These comments are usually limited to small hypotheses about the instruction or function under analysis. Some of the participants described the need for better methods for annotation management. Instead of the engineer referring to their handwritten notes, screenshots, or abbreviated comments in the reversing application, the provenance system can automatically record these data collections and the associated context in the program.

3.3.2 Visualization Support.

Participants reported a pressing need for visualization tools supporting workflow tasks, particularly during disassembly analysis. One engineer remarked on the cognitive challenges: *"All of the time in my head I'm trying to build up this graph of how this function calls this other function. We need better methods to keep track of the path activity." (P1)* The other participants agreed, *"[We need] visualizations to help mentally layout and communicate the functionality of the binary." (P3)*

However, such tools must be flexible enough to support tasks with changing goals and needs. Newcomers and experienced software reverse engineers should be equally supported in their efforts to accomplish their tasks. Participant *P5* stated:

> *The vast majority of people need an easy-to-use tool to see some basic things about the file, to hold their hand and walk them through step by step. And it's not any criticism of that person, they just have a very specific set of things that they need to look at.*

Participants frequently construct categories between data items by organizing them spatially in their personal note space. For instance, when determining the

relationship between system calls and functions, participants would review both the function listing and the import table to develop a high level understanding of the binary. In general, participants used scratch paper to record the data flow relationship through node-link diagrams.

Reducing the time necessary to perform routine analysis activities was one need described by all the participants. This early exploration task and accompanying analysis generally took participants less than 24 hours to complete. A visualization tool may be able to assist the engineer by presenting these binary artifacts directly for exploration in a single view.

The level of available time given for an analysis depends on several factors, including the workforce allocation for the task and customer prioritization. Certain projects require a deep understanding of the software and are thoroughly analyzed with little time pressure. In such cases, *"analysis could take anywhere from 6 months to years for complex samples." (P1)* In situations where a quick response is necessary, the engineers will abandon a systematic approach for the sake of time:

> *If I'm being really structured and I have a lot of time, then I'll go through and collect all of the [details], but I'm often multi-tasking with a whole bunch of other various projects. So what I'm doing with the binary I need to do fast. (P3)*

Without a systematic approach, however, reverse engineers risk missing important details and/or critical steps in their analysis. In either case, improved visualization tools should be pursued to improve the speed and accuracy of the analysis.

3.3.3 Workflow Support.

When reverse engineers analyze compiled software, they must organize their findings, hypotheses, and evidence to *"tell a convincing story"*. (P4) The participants described challenges in keeping track of numerous code paths and data files, comparing the results of hypotheses with previously executed attempts, and remembering what they learned from past successes and failures. These problems persist because the current methods for managing and documenting findings require too much user overhead and provide too little *context*. *Low overhead* means that the user can spend more of their time on applying program comprehension skills than on managing notes and findings. *High context* means that the user can directly correlate their past activities and results in the context of the reversing application.

The participants described the process of forming and testing hypotheses to make decisions based on the knowledge gained. This process is often iterative and can account for hours of exploration. Intermediate findings may lead to new strategies and decisions, resulting in a cyclic progression of knowledge building and understanding [3]. The reasoning steps are important to capture in order for the analysis to be explainable, reproducible, and trustworthy [113]. For analysis such as those for national security, these steps may be needed for criminal or intelligence investigations. Therefore, the accuracy and degree of detail of these reasoning steps are of critical importance.

As previously discussed, reverse engineering findings are frequently documented by taking screenshots. Static images, however, cannot convey information about the exploration process. New tooling should not just communicate results, but also describe how these results were derived. The lack of a back-link from the results to the exploration stage and the underlying data makes it difficult (1) to reproduce and verify the findings explained in a report and (2) to extend the exploration to make

new discoveries.

3.4 Limitations and Threats to Validity

A limiting factor was based on the nature of the participants' work. Due to confidentiality concerns, the researchers were not able to freely interview the participants about all aspects of their work or observe them in their daily work environment. This limitation was mitigated by focusing on their work needs and processes, not on specific protected tools, data, or sources. Related concerns also restricted the researchers' ability to use recording devices for three of the interviews. However, the risk of not adequately capturing all answers was mitigated by verifying the results with the participants.

External validity The biggest threat to external validity is that the captured responses may be atypical from other reverse engineering groups. The elicited requirements may be unique due to security limitations, mission goals, or resources of the participants, and therefore may not be representative of the larger population of reverse engineers. This threat is mitigated by adhering to a scripted protocol focusing on their workflow instead of specific techniques unique to their environment.

Internal validity The generality of the interview questions may be affected by researcher bias stemming from personal assumptions. This threat is mitigated by the pilot interview, the open-ended questions, and the follow-up discussions. The pilot interview helped the researchers refine the interview questions and ensure their relevance. Open-ended questions enabled participants to add or elaborate on their own opinions. Furthermore, the findings were confirmed in follow-up communications.

3.5 Chapter Summary

This chapter presented an exploratory study aimed at understanding the processes, tools, challenges, and visualization needs of software reverse engineers to develop requirements for a future tool. Five workflow processes were identified, including metadata analysis, disassembly analysis, dynamic analysis, documentation, and collaboration. Describing these processes led to the identification of specific challenges and visualization needs.

Unfortunately, it appears that no existing workflow tools capture all of the tasks performed by reverse engineers. Task complexity, security challenges, time pressure, and other tool constraints make it impossible to follow a structured heavyweight process. Therefore, future tool support has to be lightweight and flexible.

The participants described a lack of adequate visualization and workflow supportive tools contributing to an increase in analysis complexity, which may also be prevalent in other reverse engineering settings. Analytic provenance tools may address these challenges by enabling the user to revisit the visualization states during the exploration process, validate hypotheses, organize findings, and present their process and results to others.

IV. Design and Implementation

Reverse engineers are in heavy demand due to the growing number of systems, programs, and malicious cyberspace threats. Previous research has indicated that a significant amount of a reverse engineer's time is spent exploring the assembly language in the disassembler, iterating hypotheses until an analysis goal is met. During this process, they may face challenges recalling their analysis path and communicating their findings with others. This research seeks to improve the sensemaking of reverse engineers during the analysis process through analytic provenance methods.

This chapter presents SensorRE, the first analytic provenance tool supporting software reverse engineers. SensorRE automatically captures user actions and provides compact views supporting analysis and presentation tasks. The implementation of the tool is described including the capture, replay, communication, and collaboration mechanisms. Next, the user interface describes the provenance visualizations. The chapter concludes with an example scenario demonstrating SensorRE's use.

4.1 Design Approach

Expert reverse engineers were consulted during the design and implementation of SensorRE. Beginning with the tool needs identified in Chapter 3, the experts helped develop a set of general mission constraints. These constraints were then used throughout the design, building and testing of the prototype.

1. **Existing workflow integration:** the tool should not change the reverse engineers workflow.

2. **Flexibility:** the tool should be lightweight and support multiple CPU architectures (e.g., x86 and ARM) and operating systems.

3. **Collaboration:** the tool should facilitate collaboration between teams of reverse engineers.

4. **Scalability:** the tool should support common sensemaking time frames in the domain by supporting long (2+ hour) and disjointed sessions.

These constraints link directly to the technology platforms and design decisions chosen.

4.2 System Overview

SensorRE is an analytic provenance tool integrated with the existing software reverse engineering platform Binary Ninja. Figure 15 provides an overview of SensorRE system. SensorRE automatically captures the user's provenance data during the binary analysis process. This data is transmitted to the provenance visualization in the user's web browser. The provenance history, which is displayed as nodes on a link diagram, holds the visualization states of the reversing application for recall, replication, or action recovery (undo/redo) activities. Importantly, users should not notice any difference from their normal reverse engineering session (Constraint 1 - Existing workflow integration). The user can treat the provenance visualization as a passive collection of actions taken or they can navigate to any previous states of the analysis.

SensorRE consists of custom Binary Ninja plugin components, a provenance visualization, and intermediary communication modules. Binary Ninja plugin components are written in Python while the visualization and communication modules use Typescript. The plugin is responsible for capturing user action properties, transmitting these properties to a NodeJS server, receiving replay commands from the browser, and formatting the replay commands to be issued to Binary Ninja. Browser components record action properties received as visualization states and store them in

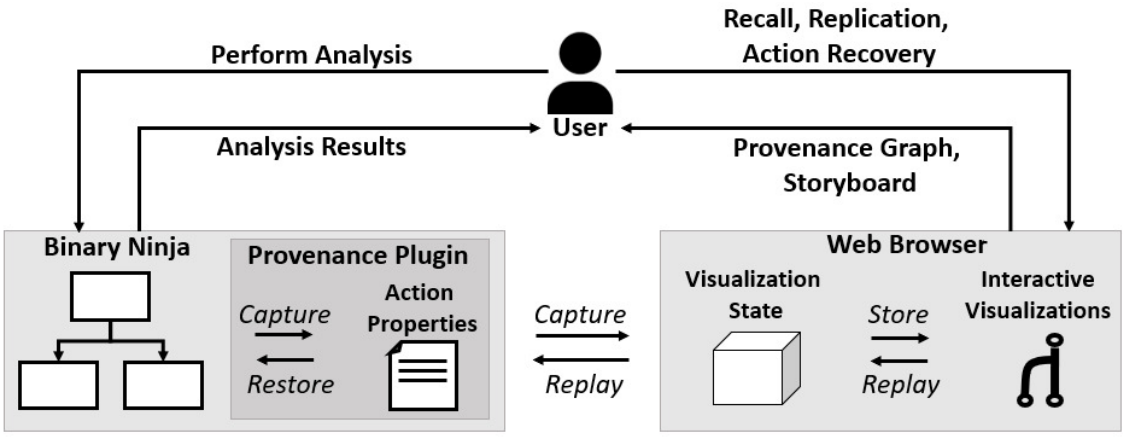


Figure 15. SensorRE System Overview.

an interactive provenance graph for analysis and replay. Appendix D includes the software code listings for all SensorRE components.

SensorRE consists of three user interface views shown in Figure 16: (a) the user's reverse engineering tool, (b) the provenance graph side panel, and (c) the storyboard side panel. On the right side are the provenance graph and storyboard side panels. The provenance graph provides a visualization of all recorded states. Interactions in the application are instantly added to the provenance graph. The user can navigate between previous states of analysis by selecting a node that updates the application view. The storyboard view allows users to communicate their provenance discoveries (e.g., hypotheses, tasks, subtasks) for reproducibility and presentation. The interface is integrated with the reversing tool, enabling users to switch from exploration to storytelling seamlessly.

A demonstration of SensorRE is presented here:
`https://www.youtube.com/watch?v=aPB8T19VSKk`.

4.2.1 System Design.

The researchers began by comparing existing reversing tools (e.g., IDA Pro, Binary Ninja, and Ghidra) for their ability to support capture and replay mechanisms. Binary

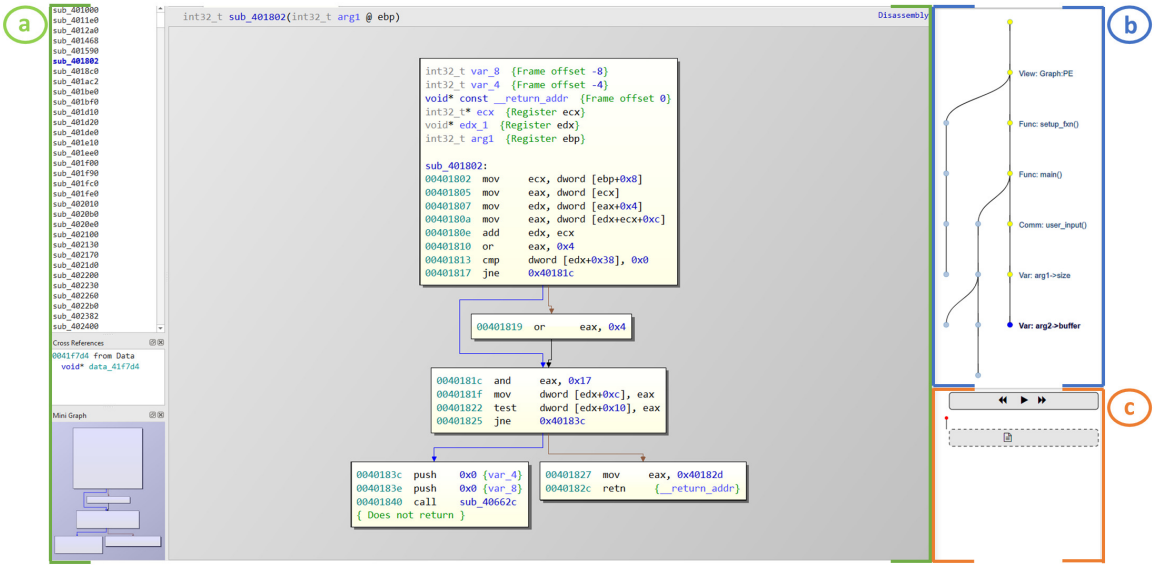


Figure 16. SensorRE prototype (a) Binary Ninja application, (b) a captured provenance graph, (c) story board panel.

Ninja stood out as the leader in accessible application Programmable Interface (API) control and automation. While future provenance solutions may be possible with IDA Pro and Ghidra, the same level of user provenance data and API control necessary was not readily available. Binary Ninja is a static analysis platform that contains a Python-based API. Users can develop third-party plugins to extend the platform's functionality.

Utilizing the analysis capabilities of Binary Ninja ensured that our solution is adaptable to a wide range of CPU architectures and operating systems, meeting Constraint 2 - Flexibility. Binary Ninja includes disassembly support for x86 32-bit, x86 64-bit, ARMv7, Thumb2, ARMv8, PowerPC, MIPS, 6502, and others developed by community. The following sections describe the design and implementation of SensorRE's capture and replay capabilities.

4.2.1.1 Designing Capture.

User actions must be captured with meaningful granularity. It is crucial to determine which key actions are to be captured, when they are captured, and how they will be stored. Instead of recording low-level events (e.g., mouse clicks, keyboard presses) SensorRE captures users' sensemaking *actions*. An action is an atomic and semantically meaningful activity performed by the user. These actions contain rich context for the user's sensemaking activity within the application. In the context of the Gotz and Zhou model [4], this level of capture is classified as *visual* and *knowledge* insight actions, which are defined as follows:

- **Knowledge insights:** The manipulation of new knowledge created by the user as a result of knowledge synthesis. These insights are captured when there are any changes to the following: variables, functions, read-only data (e.g., strings), or type information.
- **Visual insights:** The explicit markings of visual objects related to their derived insights. SensorRE captures these insights through user comments and highlighting activities. These are visual indicators of the user connecting the dots.

Figure 17 summarizes the action types captured in SensorRE.

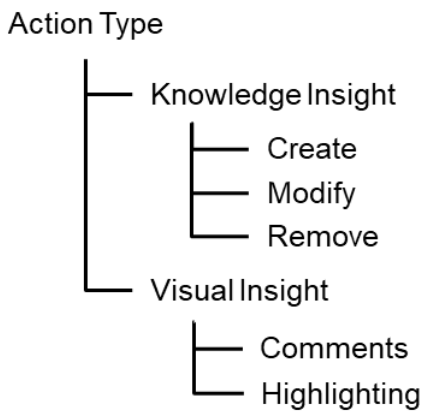


Figure 17. The *action* types captured by SensorRE.

4.2.1.2 Capture Mechanism.

The detection and recording of all "insight" action types are automatically captured within the Binary Ninja application in real-time. A custom plugin passively monitors for application events and supports communication between Binary Ninja and SensorRE system components.

All insight action captures are automated within the Binary Ninja application API. Binary Ninja's *BinaryDataNotification* class supports 14 different types of analysis actions (depicted in Figure 18), including data modifications, function updates, and type definitions. Each action triggers a callback function which performs state data collection. State data recorded includes the action's binary address, current view, action type, previous state, and the new state. The *current view* references the main window view: graph disassembly, linear disassembly, hex editor, type, or strings. SensorRE references the specific BinaryDataNotification *action type* triggered by the callback function. The *previous state* (prior to the user modification) and *new state* (post-user modification) are also used by SensorRE to maintain the running history of user interactions and their relation to each other to construct the provenance graph.

SensorRE's capture plugin formats messages into JavaScript Object Notation

```
class BinaryDataNotification:
  __init__()
  data_inserted(view, offset, length)
  data_removed(view, offset, length)
  data_var_added(view, var)
  data_var_removed(view, var)
  data_var_updated(view, var)
  data_written(view, offset, length)
  function_added(view, func)
  function_removed(view, func)
  function_update_requested(view, func)
  function_updated(view, func)
  string_found(view, string_type, offset, length)
  string_removed(view, string_type, offset, length)
  type_defined(view, name, type)
  type_undefined(view, name, type)
```

Figure 18. Binary Ninja's BinaryDataNotification class [9].

(JSON). Each message to the web browser visualization is saved to and served by a minimal NodeJS web server residing on the local host. The JSON message contains information identifying the message type, current state, previous state, and visual properties.

Performing actions in the application creates a new visualization state. The state data is stored in the provenance graph as metadata. This enables the user to return to any previous state within Binary Ninja by clicking a node. For example, modifying a local variable triggers the *function_updated* API, which leads to an update of the provenance graph. The user may return by clicking on the corresponding node, reverting Binary Ninja to the prior state. A JSON message is created (see Figure 19).

4.2.1.3 Designing Replay.

The Binary Ninja plugin uses the eXtensible Markup Language (XML) Remote Procedure Call (RPC) protocol to receive and process messages from the web browser visualization. A SimpleXMLRCPServer module that is bundled in Binary Ninja's de-

```
{
   "type":  "var_name_update",
   "func_addr":  "0x401000",
   "function":  "_start",
   "index":  "0",
   "var_name_new":  "var_14_test",
   "var_name_old":  "var_14",
   "var_type_new":  "int32_t",
   "var_type_old":  "int32_t",
   "view":  "Graph:PE"
}
```


Figure 19. JSON messaging format.

fault Python installation instantiates the client-server communication from the web browser to the plugin [114]. XML messages conform to the Binary Ninja API specifications. For instance, the user may interact with the provenance graph to update the Binary Ninja analysis to a specific view state. The SimpleXMLRPCServer module receives a message from the client in the browser and issues *binaryview.file.navigate* within the Binary Ninja, moving the user's cursor to the view and address that were provided as arguments.

SensorRE provides options to import and export the provenance data supporting persistence over multiple sessions and sharing between users. At any time during the analysis, users can save their analysis history or load a previously saved history. When new provenance data is loaded, SensorRE automatically advances the Binary Ninja application to the last saved state. This provides an interesting usage scenario. When one user loads a provenance graph from another user, the provenance graph automates the re-building of the saved analysis. Each step can then be inspected individually or as a whole to promote reproducibility and replication. Table 2 presents the implemented XML-RPC functions and brief descriptions.

Table 2. XML-RPC messages used by SensorRE.

Function Name	Description
FuncVar	Set local variable name and type
FuncName	Set function name address to string
FuncType	Set function type
FuncNameType	Set function name and type
Jump	Move to the address pointed by 'addr'
MakeComm	Modify comment at the location 'addr'
SetColor	Set the color pointed by 'addr'
DefineFunc	Define a function at the location 'addr'
UndefineFunc	Undefine a function at the location 'addr'
WriteData	Write data to address
AddType	Add type to binary
RemoveType	Remove type from binary

4.2.2 Communication Modules.

SensorRE supports bidirectional communication between Binary Ninja and the web browser visualization. Messages are passed using JSON over Transmission Control Protocol (TCP) sockets. A modular design supports the software independence between the Binary Ninja and the web browser modules. Moreover, this structure enables SensorRE to be compatible with other software analysis tools with only minor changes.

SensorRE transmits data over a predetermined port on the loopback interface. This approach cuts down on network traffic and is beneficial in malware analysis, since the machines are typically separated from all networks to promote security. However, the browser does not have to be co-located with the software analysis tool.

Communication modules are non-blocking, supporting an improved user experience using TypeScript's Async/Await construct. When an action occurs in the application, it sends an asynchronous message to the provenance graph. The graph constantly listens and responds to such messages. This approach ensures that the view is responsive to user interactions even when awaiting further input.

Figure 20 depicts a sequence diagram representing a day in the life of the SensorRE system. The user first initiates Binary Ninja, the plugin (which spawns an XMLRPC server), web server, and web browser. Next, the web server monitors for interaction events from the Binary Ninja plugin. As the file contents change, unique JSON messages are transmitted over a TCP socket to the provenance visualization in the web browser. When the user selects a provenance node on the graph, XMLRPC messages are transmitted to the server which issues Binary Ninja API command corresponding to the user's interaction. Navigating between multiple nodes on the provenance graph issues each state's corresponding action sequentially in a non-blocking queue.

4.2.3 Collaboration Support.

Collaboration has become an important focus in reverse engineering analysis. Users frequently share with another person or a small group [3]. When many users analyze a specific binary, they can build collective knowledge and identify common paths of inquiry.

SensorRE supports asynchronous collaboration through the sharing of provenance graphs (Constraint 3 - Collaboration). Users can import and export provenance data to support continuity over multiple sessions. At any time, users can save their session or load a previously saved history.

In a collaborative session, a user may load an existing analysis and extend the findings, as depicted in Figure 21. When a user first interacts with Binary Ninja, their actions are recorded and stored. Later, a second user loads the provenance graph from the first user, automating the re-building of the saved analysis. Each step can then be inspected individually or as a whole to promote reproducibility and replication.

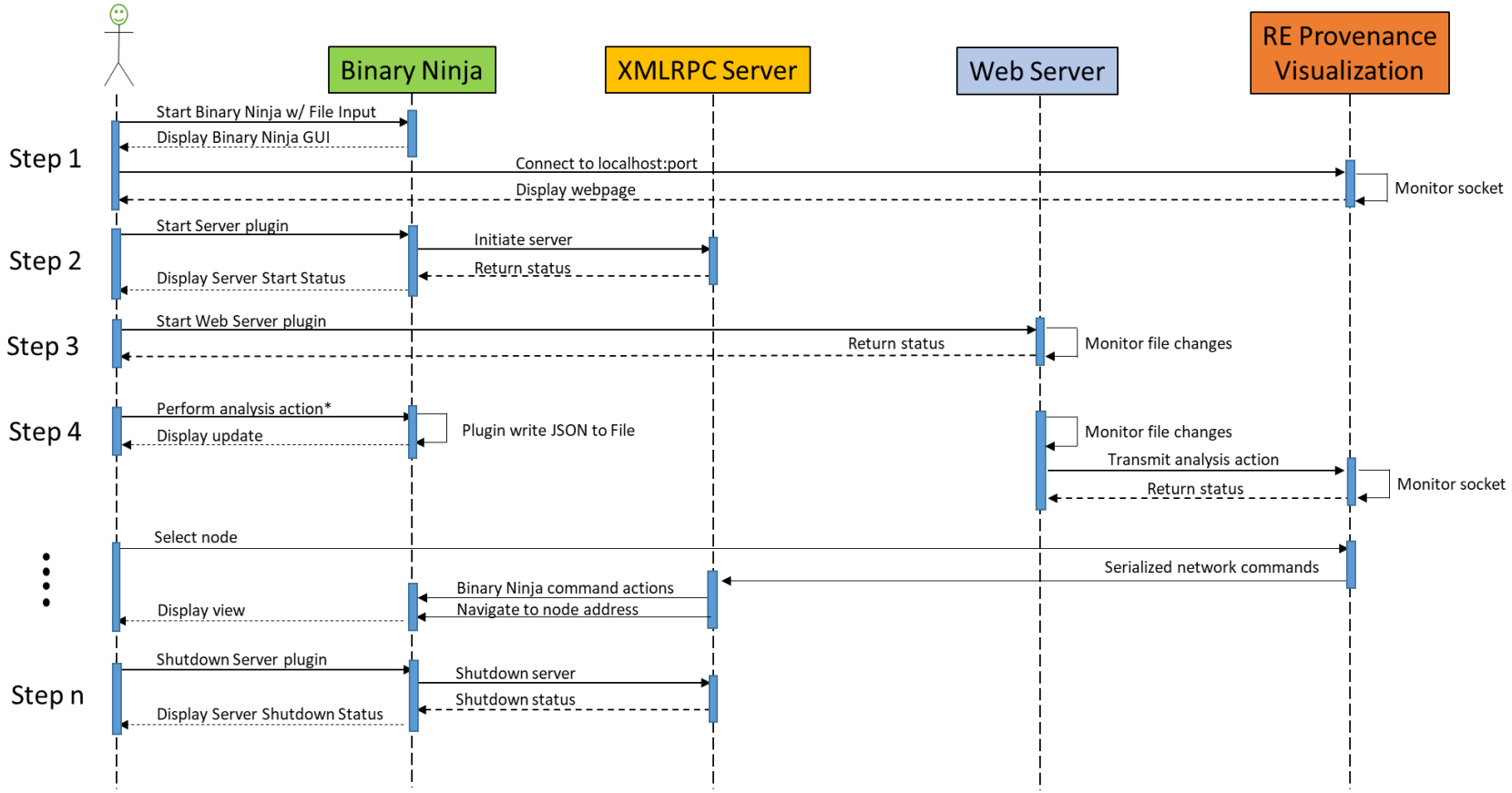


Figure 20. SensorRE day in the life sequence diagram.

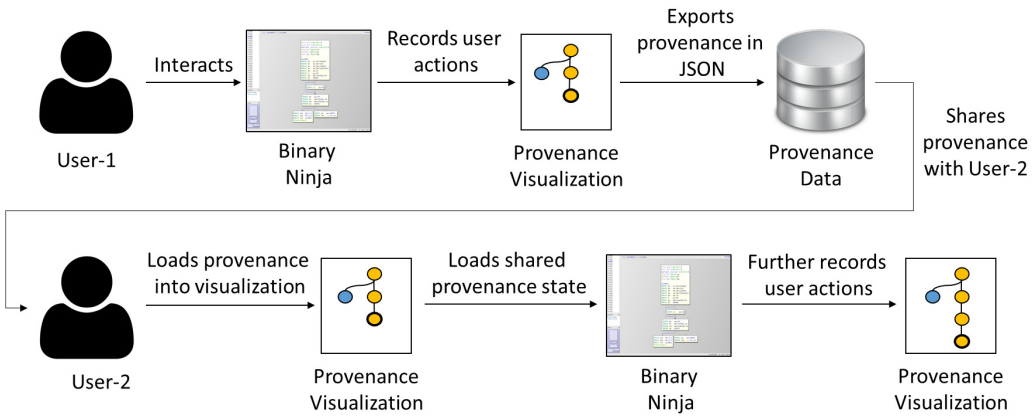


Figure 21. Two users leverage the provenance framework to asynchronously collaborate on work in the Binary Ninja application.

4.3 User Interface

The user interface plays an important role in the usability of the captured provenance data. The interface permits navigation, browsing, and annotation capabilities. SensorRE provides the reverse engineer with a flexible user interface in the form of a provenance view and storyboard view. These views are dynamically constructed using the D3.js and Visual Storytelling libraries [115].

4.3.1 Provenance View.

The provenance view depicted in Figure 22 provides a scalable visual history of analysis actions. The graph offers an overview of the provenance by displaying all captured states in a vertical tree. The view is also utilized for navigating and selecting states. Users interact with the provenance graph by left-clicking a node. Selecting a node triggers the corresponding state navigation in the application.

The provenance graph consists of three components: nodes, edges, and branches. A node consists of the visualization state. An edge represents actions that transform one state into another. Branches are pivot points where the user tried different hypotheses.

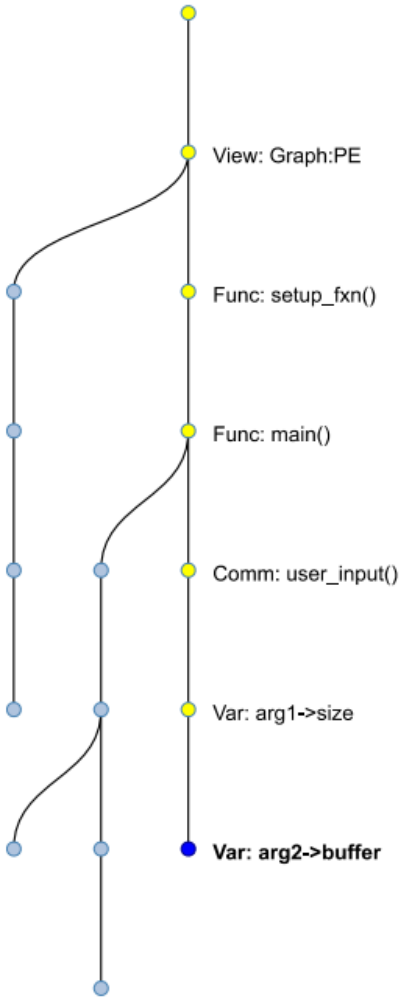


Figure 22. Provenance graph view.

When an action occurs or the user selects a state, the tree layout changes such that the currently selected node and its ancestor nodes are right-aligned. The remaining nodes and branches are then aligned on the left side. Maximizing space for node labels through layout and re-alignments addresses design Constraint 4 - Scalability.

4.3.2 Story Board View.

Reverse engineers communicate results with other engineers, managers, intelligence analysts, and other interested stakeholders. Translating low-level findings into

high-level results for stakeholders can be problematic due to their differing goals and technical expertise. These different perspectives can cause misunderstandings [33].

Presenting provenance information is a frequent challenge reported in visual analytics studies [116, 117]. As provenance graphs grow in size with each captured interaction, they become increasingly difficult to understand. Presenting the findings through storytelling is therefore an effective strategy.

SensorRE presents a storyboard visualization allowing users to create a narrative based on personalized annotations and captured analysis states. Annotations allow users to record their thinking, providing semantically rich information. The storyboard view can also be used to prepare a post-analysis replay of captured provenance data. Figure 23 displays the storyboard view loaded with a single state.

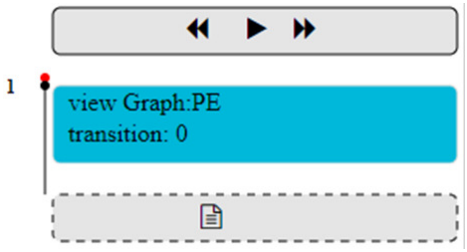


Figure 23. Story board view.

The menu panel along the top includes rewind, play, and fast forward buttons. Selecting rewind navigates the story board to the state that directly precedes the active state. Selecting play initiates an interactive session where the set of actions loaded in the storyboard are played sequentially. Selecting fast forward advances the active storyboard to a subsequent loaded state.

The duration that the application spends on each action can be modified by selecting the bottom edge of the state and dragging, shown in Figure 24. By default, each state is set to view for one second. The state duration is shown along the left hand side of the vertical timeline in seconds. The transition duration between states can be adjusted by selecting the top edge of a state and dragging, as shown in Figure

25. The default transition duration is zero seconds. The transition duration is shown within each state in seconds. Modifying the state and transition duration creates a more personalized story telling presentation of the findings.

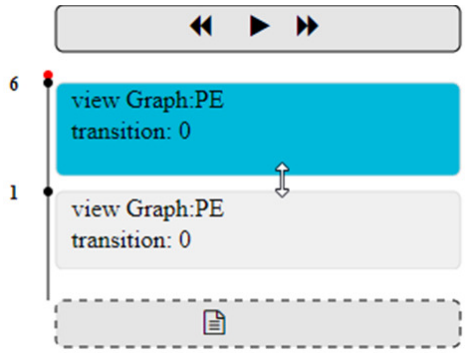


Figure 24. SensorRE story board state duration change.

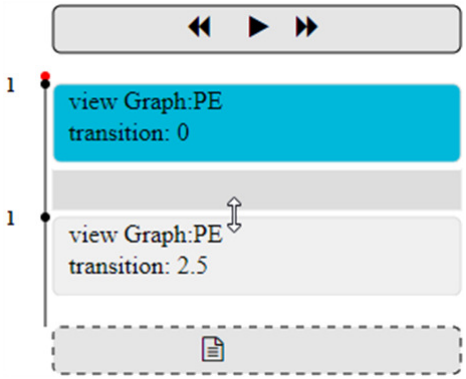


Figure 25. SensorRE story board transition duration change.

4.4 Usage Scenario

The functionality and effectiveness of SensorRE is described in a typical scenario in which an analyst is attempting to solve a simple "crackme" binary by deducing the embedded password. The provenance graph updates during the scenario are captured in Figure 26.

An initialized provenance graph is displayed in Figure 26 (a). Next, the user begins analyzing the binary by dissecting a high-level function. The user hypothesizes

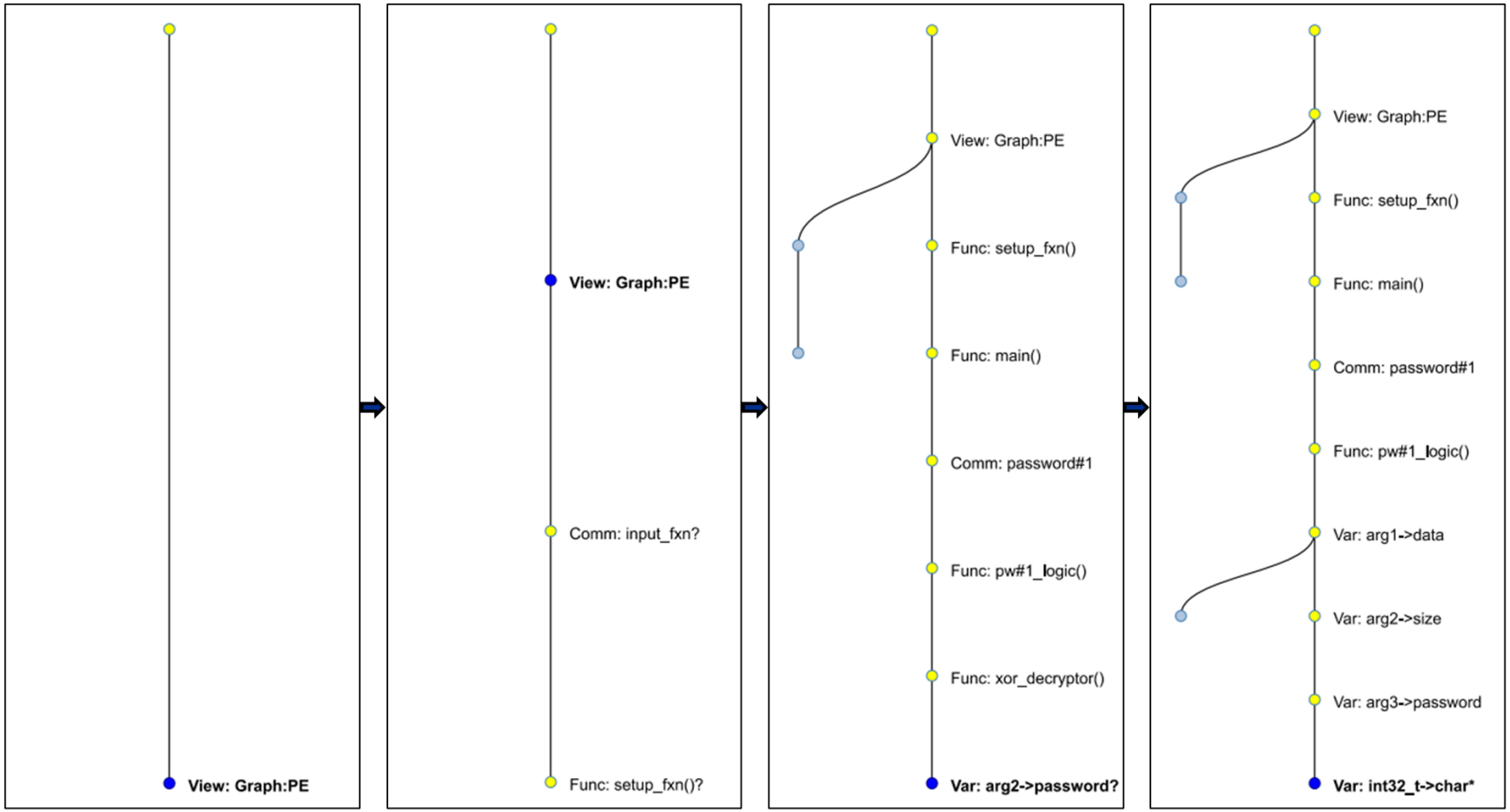


Figure 26. Provenance graphs during usage scenario.

that the function handles user input and annotates the hypothesis by creating a comment in the Binary Ninja application (*"Comm: input_fxn?"*) and renaming the function label (*"Func: setup_fxn()?"*) (see Figure 26 [b]). Later in the analysis, the user determines that the prior hypothesis was incorrect when they find the actual setup function, so the user backtracks to the *"View: Graph:PE"* node (reverting the previous actions) and modifies the actual function name appropriately. From the setup function, the user also identifies and renames the main function (*"Func: main()"*), where the program starts its execution.

Next, the user studies the strings in the binary (*"View: Strings"*) and identifies an entry of interest with a comment annotation *"Comm: password#1"*. The user navigates to the calling function using the string's cross reference and examines how the local variables are being used. The user hypothesizes that argument 1 contains a yet-to-be determined data buffer and that argument 2 may be the password of interest. Therefore, the user modifies the variable name labels corresponding with the hypothesis, as depicted in Figure 26 (c). As the user continues to examine how the arguments used by the function, the hypothesis is reconsidered. Argument 2 is understood to be the size of the password buffer. Renaming the variable creates a new primary branch in the provenance tree which records the alternate hypothesis in-case the user wants to re-visit their analysis.

The analyst identifies that the password of interest is contained in argument 3 *"Var: arg3–>password"* and that the variable type is *char** instead of the assembler default type of *int32_t*. The final state of the usage scenario is depicted in Figure 26 (d).

At the end of the session, the user develops a story board to present the findings to the stakeholders (Figure 27). Each action state along the main branch is added to the story board in order. The user then adjusts the transition times between each

state to present the results in a clear manner, and adds customized annotations to the specific views. Finally, the results are replayed with those stakeholders in a remote or local setting.

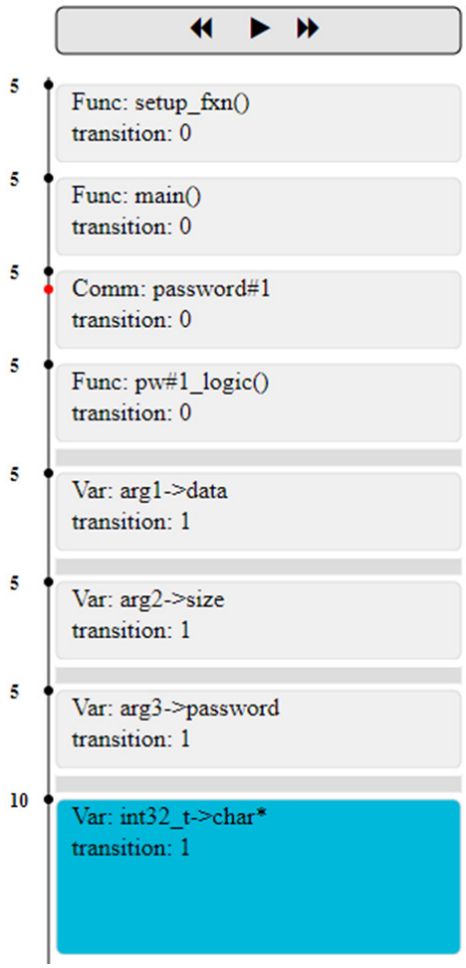


Figure 27. Story board scenario results.

In the above example, the user started the analysis by investigating high-level details in the reversing application. During the course of analysis, the user considered several other aspects of the binary before forming the initial hypothesis action. This corresponds to the user's information interests. These interests vary over time as the user gains familiarity with the piece of software. During the exploration process, users' evolving interests are captured in an action trails. The knowledge from the

analyst's session is encoded as a network structure with nodes representing actions and links representing associations among the actions.

4.5 Chapter Summary

This chapter presented SensorRE an analytic provenance tool supporting software reverse engineers. The user needs study presented in Chapter 3 led to the development of mission constraints for the system. Next, we discussed SensorRE's overview including its integration with an existing reverse engineering application, Binary Ninja. The capture, replay, and communication modules were discussed using screenshots and descriptions, as well as how these features were implemented. SensorRE's provenance graph and storyboard views are implemented in a web browser, making the prototype lightweight and portable.

We ended the chapter by demonstrating SensorRE through an example usage scenario. The user analyzed a "crackme" program while the prototype captured and visualized their actions as provenance data. Then, the user then created a storyboard of their actions and hypotheses.

This chapter addresses our second research question: *How can an analytic provenance tool be designed and developed to support software reverse engineers?* This encompasses Phase 3 of the dissertation. The next chapter describes user-centered evaluations of the proof-of-concept SensorRE tool.

V. Evaluation

In the previous chapter, SensorRE was introduced as an analytic provenance tool supporting the reverse engineers' cognitive processes. SensorRE was designed to aid recall, replication, collaboration, and presentation activities. This chapter addresses the final research question: *Is the analytic provenance prototype effective at supporting software reverse engineers?*

This chapter presents two studies evaluating the prototype with subject matter experts and graduate students. Each participant is observed performing reverse engineering tasks while screen capture and think-aloud data are collected. Survey results from the participants are analyzed within their respective studies.

5.1 Study Design

As the first analytic provenance system for software reverse engineers, there is no baseline to compare. Instead, the researchers examine the system's usability and its effect on the participants' processes during a series of practical reversing tasks. The participants were asked to reconstruct a program's functionality by figuring out how the program is structured from its assembly language representation and the accompanying provenance graphs.

Since analytic provenance systems for reverse engineers have not been previously developed, sensemaking research in the domain guided the selection of tasks. Bryant [3] described how complex reverse engineering tasks (e.g., vulnerability analysis, malware analysis, or software protection) require significant domain knowledge which limits what could be learned in a sensemaking study. Such tasks may have additional concerns due to classification or legal restrictions. As such, the tasks selected for this study were designed to highlight the impact of the provenance system.

We considered numerous studies evaluating provenance systems when defining our specific tasks [5, 118, 119]. These studies explored characteristics of provenance systems during domain-specific challenges at different levels of detail. *Concrete tasks* are well-defined, low-level operational tasks representative of typical actions that a practitioner would perform. They mark an efficient method for evaluating the usability of the interface. However, analyzing only concrete tasks are not sufficient for understanding a user's comprehension. *Abstract tasks* are discovery-focused and seek to elicit high-level comprehension.

Participants were asked to complete both concrete and abstract tasks. Concrete tasks were graded with a prepared answer key with only one correct answer per question. In contrast, abstract tasks evaluated the higher-level understanding of the scenario and therefore did not have binary right or wrong answers.

Before performing the studies, the researchers performed experimental process reviews (including the tutorial and scenarios), and made revisions accordingly. To minimize the risk of bias, we asked two reverse engineering professionals with an average of 7 years of experience to inspect our approach. The experts provided feedback on issues encountered and we made revisions accordingly. To reduce external confounding factors, these tests used the same equipment and location.

5.2 Selection of Participants

Four subject matter experts participated in the study (listed in Table 3). Each expert works in the field of software reverse engineering related to cybersecurity with an average of 11 years of experience. Two of the participants were members of the interviews for the user-centered elicitation process (Chapter 3). All experts identified themselves as currently working within the federal government with prior experience in the private sector ranging from three to 10 years.

Table 3. Expert reverse engineer participants demographic survey.

Participant	Age	Education	Experience
E1	20-29	MSc	6-10 years
E2	30-39	PhD	11-15 years
E3	20-29	MSc	6-10 years
E4	40-49	MSc	16-20 years

Participants in the second study are listed in Table 4. Eleven university students were recruited from Computer Science and Cyberspace Operations graduate programs. We recruited only students who had successfully completed a graduate-level software reverse engineering course. This ensured the students possessed the minimum skills required [3, 19]. In the demographic survey, all participants reported they were familiar with the reverse engineering process but lacked experience (i.e., they knew the concepts, but had not practiced outside of coursework).

Table 4. Graduate student participant demographic survey.

Participant	**Degree**	**Education**	**RE Experience**
P1	Comp Sci	PhD	>5 years
P2	Comp Sci	Masters	<1 year
P3	CSO	Masters	<1 year
P4	CSO	Masters	1-2 years
P5	CSO	Masters	1-2 years
P6	CSO	PhD	<1 year
P7	Comp Sci	Masters	<1 year
P8	Comp Sci	Masters	<1 year
P9	Comp Sci	Masters	<1 year
P10	Comp Sci	PhD	<1 year
P11	CSO	Masters	<1 year

5.3 Apparatus and Materials

A desktop computer with Windows 10, a 19" LCD display, and standard peripheral devices (keyboard and mouse) was used for this study. The screen capture

software OBS: Open Broadcasting Software recorded user activities. The executable was loaded into Binary Ninja v1.2.192 with an adjacent Chrome browser running the provenance visualization.

A *crackme* program was selected for this study because they are available with a free-use license. They also range in size and difficulty, with the tasks readily understandable and solvable within an hour. This choice of the dataset allows the researcher to tailor the difficulty to the participants' experience. *White Rabbit* is a level-2 difficulty crackme. The program was developed in C/C++ for the Windows operating system. It is 6.7 MB in size, and has approximately 30,000 assembly instructions and 750 functions.

5.4 Procedure

The procedure for each user experiment is outlined in Figure 28. There were four phases to both studies: 1) the experimental setup, 2) the training phase, 3) the task observation phase, and 4) the post-study survey phase.

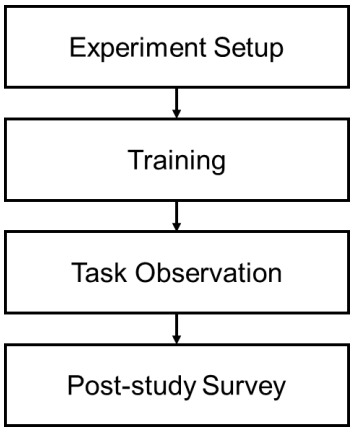


Figure 28. Phases of tool evaluation.

5.4.1 Experimental Setup.

In any experiment, properly controlled conditions are necessary to obtain reliable results. A written protocol was followed describing the researcher's actions during each phase of the experiment. The protocol specifies how to introduce the users to the experiment, provides instructions on setting up the workstation, and how to interact with participants if they encounter an issue. This protocol ensured that the experiments proceeded smoothly and consistently, reducing the likelihood of mishaps that might affect user performance.

5.4.2 Training Phase.

The training session was designed to build the participants' familiarity with the tools before the evaluation. The protocol included a description of the motivations driving the development of the tool, demonstrating Binary Ninja interface navigation, and showing the participant a short video demonstration of the prototype. Participants were provided as much time as necessary to complete the training and ask questions. The training phase took approximately 10 minutes for each participants.

First, the researcher asked participants to read a brief background paper on visual analytics and provenance systems, shown in Appendix E. This ensured that all participants possessed a basic understanding of provenance and the motivations for the research.

Next, in a guided session with the researcher, participants were familiarized with the basic operation of the Binary Ninja application. Participants were expected to have prior experience with the IDA Pro disassembler in the x86 environment, but not with Binary Ninja. Therefore, this training session was necessary to ensure that the participants possessed a baseline level of knowledge with Binary Ninja needed to carry out the tasks. Lastly, the participants were asked to watch a video demonstrating

the SensorRE tool during which they were instructed on how to interact with the provenance graph and storyboard views.

All participants received identical training material to ensure uniformity. During the training session, to ensure understanding, the participants were encouraged to ask as many questions as they wanted.

5.4.3 Task Phase.

The users were given the task descriptions and the questions they needed to answer in print form, refer to Appendix E. It should be noted that the scenarios described below do not cover all possible use cases for analytic provenance systems in reverse engineering. However, the scenarios presented are representative of the types of supportive tasks applicable to the provenance prototype.

During the experiment, the facilitator actively listened to the concerns, challenges, problems, and findings raised during the participants' analysis. The facilitator initiated a dialogue when the participants elicited verbal or non-verbal frustration (e.g., participants shaking their head or verbalizing "I'm stuck."). The participants were free to use any features of SensorRE and to ask questions during the scenarios.

The first scenario evaluated the tool's support for *validating collaborator findings*. It asked users to answer a set of questions based on an existing provenance graph developed by an imaginary colleague. The questions required the participant to inspect the provided provenance graph for clues. The graph consisted of approximately 15 steps, including branches, that were captured during the colleague's analysis. Participants were encouraged to inspect the graphs and binary using Binary Ninja but not to extend or modify them. After the participants submitted answers to the questions, they were able to view the correct solutions to the tasks.

Scenario 2 evaluated the tool's support for *extending a collaborator's analysis*.

Participants were asked to complete the partial analysis from Scenario 1. The participants were asked to identify the encryption routine and rename the calling function (as opposed to the default naming scheme). The chosen name would be used to communicate the functionality back to the original collaborator. As participants completed this exploration task their findings were automatically recorded in the provenance tool. There were no right or wrong answers to this abstract task.

In Scenario 3, participants *present their results* from Scenario 2 using SensorRE's storyboard. The participants selected the steps in the provenance graph visualization and added them to the storyboard. The scenario was complete when the user presented the facilitator with the finished product.

Since SensorRE is a prototype, software errors during the experiment were possible. Equipment or software failures during the experiment were mitigated by having a virtual backup system in a standby state. If the study ends prematurely due to a software error, the time for that task was restarted once the researcher sets the backup system up for that specific task. If the subject's participation ends prematurely, the results up to that point were evaluated for potential inclusion in the data analysis.

5.4.4 Post-Study Survey Phase.

The post-study surveys assessed common usability study metrics such as learnability, ease of use, useful features, features missing, and limitations to adoption. The responses are meant to gather feedback on how to improve SensorRE in the future. All participants were asked to answer the following survey questions:

1. What was particularly useful about the tool?

2. Do you think the tree diagram is a useful representation of the provenance graph?

3. Are there other artifacts you would like to add to the provenance graph?

4. Are there any features missing?

5. Are there any limitations of the system which would hinder its adoption?

In addition to the open-ended textual responses, the graduate student participants submitted usability feedback through questionnaires adapted from the Computer System Usability Questionnaire (CSUQ) [120]. The expert participants were asked to provide qualitative feedback on SensorRE but were not given the usability questionnaire. The CSUQ contained the following five questions on a 7-point Likert scale in Figure 29.

	Statement	Strongly Disagree	Disagree	Slightly Disagree	Neutral	Slightly Agree	Agree	Strongly Agree
1	This tool was easy to use.							
2	This tool was easy to learn.							
3	This tool could improve the recall of findings, strategies and methods.							
4	This tool could improve validating findings from others.							
5	This tool could improve communicating findings among teams of reverse engineers.							

Figure 29. Likert-scale questionnaire.

5.5 Data Analysis

The participants' think aloud data, recorded screen activities, and facilitator notes were used to create a detailed timeline of the participants' actions. The timeline is the researchers interpretation and might not perfectly reflect the participant's internal reasoning process. The coded data was transcribed with a conscious effort to minimize human bias. The resulting transcript represents the ground truth of each participant's actions during the experiment.

Data analysis in a mixed-methods design study consists of analyzing both qualitative and quantitative data. Based on a review of related research, this is the approach that is most frequently used to evaluate provenance prototypes in other domains [1, 5, 7, 121, 122, 123]. Qualitative data sources include the researcher's observations of verbal and non-verbal data during the problem-solving process, the participants' interactions with SensorRE recorded through screen capture, and open-ended task and survey responses. Quantitative data sources include completion times for each scenario, task errors, and Likert-scale scores from the post-study survey.

5.6 Results and Observations

5.6.1 User Study 1 - Experts.

During the interviews, each of the four experts agreed that the provenance view helped them clearly see their analysis process. Experts found the real-time recording and visualization of findings provided a quick, context rich view of data that is otherwise masked in reversing applications. Expert 3 stated that action logging in reversing applications is often very rudimentary, and commented that the visualization and context of actions provided by SensorRE would be of immediate benefit. The provenance graph view helped visualize their hypotheses, notes, and assertions.

Expert 1 said: "*The ability to step through and replay the analysis of a binary is extremely useful. Especially, when you can export a file of your analysis and share it with someone else.*" Expert 4 referred to the provenance graph as "*a cleaner view of my thinking process.*" The expert commented that, he frequently navigates between multiple parts of a binary, occasionally resulting in getting "*stuck down the rabbit hole*" and saw immediate potential in using SensorRE to reduce the burden of mentally tracking each step.

The experts expressed confidence that the tool helped them to be more thorough, systematic, and organized. Expert 2 noted that the graphical representation was a great starting point to get an overview of the binary and to determine where analysis might still be needed. The sequence in the provenance view was important, since it represents the workflow. The expert wanted to rearrange, group together, and purge certain states to create an optimum analysis workflow template. This was particularly important since the expert has to frequently re-evaluate earlier hypotheses during the course of analysis and didn't want to overly clutter the visualization. Reducing the quantity of visual artifacts in the provenance graph would improve system scalability. Expert 3 liked the level of detail provided by the provenance graph because it helped him quickly remember the context of his notes.

During the collaboration case, the experts loaded a saved provenance graph and then navigated through the analysis, validating the results. The experts verbalized details about how the collaborator analyzed the binary, as well as assessing the accuracy of their findings. Experts 1 and 4 stated that they have to regularly audit team members' analyses, and complimented the system for its transparency and ease of navigation. Expert 1 was compelled to share a previous experience with an alternate collaboration tool which he tried out but found the tool integration lacking. With SensorRE, the expert commented that he was able to quickly navigate between

collaborator's analysis steps with no discernible slow-down or latency. Our findings suggest that SensorRE aided the experts' reconstruction of their thought process with almost immediate benefit.

Towards the end of the interview, the experts suggested a few improvements to the system. Expert 1 suggested that "*more collaborative features should be added*," expressing a lack of collaborative reverse engineering tools. The expert stressed that collaborative tools are desperately needed for reverse engineering teams. He suggested the development of a shared provenance graph between team members, so long as the inputs are easily distinguishable. Expert 4 described the story board panel as "*a nice feature*" but suggested adding annotations to each node for additional context.

The experts found the provenance display simple yet effective in providing provenance data. However, several experts recommended adding scalability features. Experts 1 and 3 suggested a grouping feature for similar nodes (i.e., actions residing within a single function) for space saving. Expert 2 recommended adding an input string field for text searching. Other potentially beneficial features suggested included semantic zooming or tagging in the provenance display for different analysis tasks. How to design and implement such new features without making the interface overly complex and reducing its usability is a challenging problem, and is left for future work.

Overall, all four experts reported that they liked the tool and found it easy to use. The experts also stated they would consider using it their workplace, and one asked us after the study when the tool would be released for use.

5.6.2 User Study 2 - Graduate Students.

Analysis Tasks. The researcher's focused on the accuracy of the task, how many queries were viewed, and how quickly the participants resolved each task. Tables 5

6 report the observed metrics from each participant. Most of the participants completed the exploration tasks, correctly identifying the relevant functions and forming a general understanding of the program. From the experiments, the researchers drew a few key observations.

Table 5. Observed results of the user study.

		P1	P2	P3	P4	P5
Scenario 1	Accuracy	100%	50%	100%	0%	100%
	Time Taken	18:13	15:40	16:26	20:32	17:10
	Number of queries	18	10	11	15	7
Scenario 2	Time Taken	7:35	8:39	10:20	10:41	9:47
Scenario 3	Time Taken	6:21	4:50	5:42	7:18	4:30

Table 6. Observed results of the user study (continued).

		P6	P7	P8	P9	P10	P11
Scenario 1	Accuracy	100%	100%	0%	100%	50%	100%
	Time Taken	18:08	16:22	18:50	12:18	16:27	15:41
	Number of queries	12	16	22	12	10	15
Scenario 2	Time Taken	14:28	12:24	18:28	8:29	9:11	8:24
Scenario 3	Time Taken	4:52	4:27	5:15	5:04	6:12	5:48

In scenario 1, three participants (P1, P4, P8) began their exploration by using Binary Ninja to search for the functions of interest. The remaining eight participants went directly to the provenance graph to trace the results of the fictional collaborator, resulting in slightly faster completion times. These participants used the graph to review the embedded hypotheses and their relative connections within the binary. Four participants (P2, P4, P8, P10) incorrectly answered at least one of the concrete tasks; however, their poor accuracy on the concrete, graded tasks did not impede their completion of the abstract tasks. Multiple participants commented on the immediate benefits provided by SensorRE:

Making connections between my findings is the most difficult part. It was great to be able to visualize all of my changes step by step - it clearly shows the thought process of the engineer. (P8)

It is usually difficult to connect current information to what I read previously. There is just too much data to sort through. Displaying the history of either my own analysis or someone else's is really useful. It helps me switch back and forth and piece together the story. (P1)

During scenario 2, three of the participants (P4, P6, P8) reported that they did not know where to begin resulting in relatively slower performance. The remaining eight participants correctly identified the software function and completed the task without assistance. The strategy eventually used by all participants was Shneiderman's InfoVis mantra: "overview first, filter, and detail on demand" [124]. When examining the performance of those who did well, we found they spent more time reading the task description and organizing information before exploring the binary. Following the experiment, participants reported how reviewing the provenance graph improved their sensemaking.

Exploring the provenance graph was really helpful. At first, I started looking at each item in sequence starting from the first element, but at some point I jumped around to the 'comment' nodes, and found those were much more helpful to make sense of the binary. The overall list of findings helped me organize what parts of the binary were responsible for different activities and complete the task quickly. (P9)

Multiple participants noted the usefulness of the provenance graph when assessing new information. Those participants who initially attempted to figure out the task without the assistance of SensorRE completed the task, but were more frustrated:

I find that I'm always trying to manage too much information when reversing.

Working through a binary, I have to keep revisiting previous points because I need to understand how these pieces work together. It can be really easy to get off track and lose focus. (P11)

In scenario 3, the participants constructed a presentation of their results using SensorRE's storyboard view. They selected the steps in the provenance graph and added them to the view. All participants correctly used the storyboard to present their results. P3 remarked during the scenario that the feature was "very useful for creating a coherent flow of logic."

Survey Results. Figure 30 displays the results of the 7-point Likert-scale questionnaire. There were nearly universal positive opinions about the usefulness of SensorRE for understanding the provenance of existing analyses. Ninety percent of the participants agreed or strongly agreed that the tool was easy for them to use. Unfortunately, during P1's experiment, an anomaly during setup required restarting the scenario resulting in a relatively low usability rating on Q1 (4 - Neutral). However, all of the participants strongly agreed that SensorRE was easy to learn, improved processes, and could improve the validation of collaborators' findings. In addition, 72% found the prototype would improve the communication of findings among teams. P4 stated, "*the provenance graph makes it easy to quickly assess where you or someone else is in their analysis,*" and "*the interface is simple (not too distracting or busy), as well as intuitive to use.*" At the conclusion of the experiment, eight of the participants expressed that SensorRE would have really helped them in their graduate reverse engineering course studies.

As a proof of concept, SensorRE can still be improved. Participant P1 experienced a software malfunction that required restarting the scenario. As such, P1's feedback described the need for improving Binary Ninja integration. Another participant (P3) stated that the visualization lacked certain features, such as time-stamps and user-

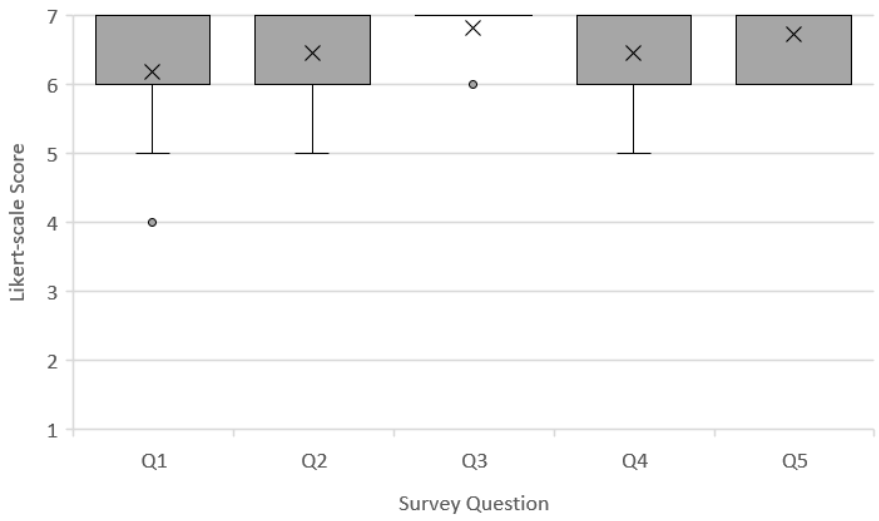


Figure 30. The average ratings (marked X) for the Likert-scale questions (1 - Strongly disagree, 7 - Strongly agree).

selected labels for nodes. Three participants (P7, P10, P11) also felt the asynchronous design could prove difficult for multiple users collaborating in parallel.

5.7 Limitations and Threats to Validity

Designing empirical studies to evaluate program comprehension tools is a challenging endeavor. There is tension between evaluating the tool in the users' traditional work setting and the desire to test in a controlled environment. Due to security concerns with the experts' work location, we were required to perform the experiment off-site.

Iterative design. The software development life-cycle in an industrial or commercial context typically requires multiple iterations on the design and implementation with users to meet their objectives. To the extent possible during development, we solicited feedback on the tool by both subject matter experts in reverse engineering, and human factors researchers.

External validity. The subject's participation in the experiment is entirely volun-

tary among the participant pools. Although using students for empirical studies is standard practice [125, 126], this can affect the generalizability of the experiment to other populations. This threat was mitigated by including experts in the evaluation.

Internal validity. An internal threat to validity is that participants are aware of the researchers' observations at all times, which may introduce a *Hawthorne effect* [127]. This occurs when participants improve or modify an aspect of their behavior simply because they know they are being studied. We attempt to mitigate this threat through the introductory protocol, establishing the tool and not the user as the focus of the study.

5.8 Chapter Summary

This chapter detailed the results of two studies testing the efficacy of the SensorRE tool with expert and graduate student reverse engineers. All participants found the visual representation and interaction with the tool intuitive to use. SensorRE helped the participants organize findings, quickly navigate to the provenance data they wanted, and effectively communicate their findings.

This chapter concludes the last phase of our research, answering the final question: *Is the developed analytic provenance tool effective at supporting software reverse engineers?* The next chapter summarizes the contributions of this dissertation, discusses the implications, presents future research directions.

VI. Conclusions

Reverse engineering is a cognitively challenging activity involving reconstructing assembly instructions into meaningful representations. Whereas high-level programming languages have expressive representations (e.g., variable names, functions, classes, and objects), in assembly these representations are stripped during the compilation process. Further complicating the analysis process is the large volume of assembly code in software binaries. Even experienced reverse engineers may face difficulty reversing given the sheer amount of data.

This dissertation developed and evaluated a fully functional analytic provenance tool supporting the cognitive processes of reverse engineers. Two studies confirmed the efficacy of the tool through qualitative analysis of the post-study survey data. Overall, this research confirmed the hypothesis that an analytic provenance tool offers cognitive support reverse engineers during exploration, collaboration, and presentation tasks. This chapter summarizes the research contributions, discusses implications and lessons learned, and concludes with a presentation of future research opportunities.

6.1 Research Contributions

Chapter 3 presented a case study that explored the visualization problem space with experienced reverse engineers. The study investigated the current visualization needs, tools, timelines, and collaboration needs through semi-structured interviews. By doing so, it identified a current capability gap in binary exploration: lack of workflow support. When reverse engineers analyze a complex binary, they need to organize their findings, hypotheses, and evidence. They also need to share that information with their collaborators, and coordinate activities among team members.

The interviewed participants currently document their results through screenshots and raw notes. Static images, however, cannot convey information about the exploration process. New tools should not just communicate results, but also describe how these results were derived. Given the separation of tools, it is inefficient for the engineer to work back from an artifact being developed for presentation to the exploration stage. The lack of a back-link from the results to the exploration stage and the underlying data makes it difficult to reproduce and verify the findings explained in a report and to extend an exploration to make new discoveries. Based on the interview data, analytic provenance techniques were explored to address these problems.

Chapter 4 contributes a novel analytic provenance tool for software reverse engineers, SensorRE. SensorRE provides facilities to capture, store, use, and share reverse engineering provenance data for a given binary. The visualization uses a tree layout that maintains branches for previously failed hypotheses. It also provides for fluid interaction with the Binary Ninja reverse engineering platform, supporting users in performing various sensemaking activities.

Chapter 5 contributes a user-centered methodology for evaluating the SensorRE tool. The evaluation focuses on the tool's support for validating and extending collaborator findings, reducing repetitive analyses, and presenting findings. SensorRE helped the participants organize their workflow, quickly navigate to the provenance data they wanted, and effectively communicate their findings.

6.2 Implications and Lessons Learned

SensorRE was designed with the goal of improving reverse engineers' access to provenance data. However, SensorRE is still in its infancy with much to be explored. The participants provided valuable feedback on the effectiveness of our prototype and how it could be further improved. This section discusses implications and lessons

learned to aid future researchers.

6.2.1 Tool Integration.

Integrating independently developed tools is a challenging endeavor. Developers must assess the degree of integration required as well as the tools' automated scripting potential. For reverse engineering tools such as the ones described in this research, the following patterns provide strong return on investment:

Modularity: SensorRE is modularly developed, separating the visualization from the messaging and application components. This early design decision allowed for iterative development in each component area, while reducing overall complexity.

To investigate our research goals efficiently we leveraged an existing reversing analysis tool instead of developing our own. Ideally the provenance visualizations could be fully integrated into the reversing platform to further reduce the impact on the user's workflow. Unfortunately, reversing platforms are highly-specialized tools currently limited in their graphing capabilities. We briefly explored developing the provenance visualization as a stand-alone application (e.g., OpenGL). This idea was dismissed because deploying a new application into the users' existing workflow becomes more complex and risks lowering the adoptability of the tool [128].

Common API: Among static reversing platforms including IDA Pro and Ghidra, Binary Ninja stood out as the leader in accessible API control and automation. While similar solutions may be possible with other applications, the same level of user provenance data is not readily available. Common APIs enable access to shared data. Applications with restricted or negligible APIs would hinder tool developers.

SensorRE's design is specific to static analysis provenance data. Extending the system to other reversing techniques may be possible. For example, visualized provenance data collected during dynamic analysis may prove possible for future develop-

ment.

6.2.2 Challenges Building Tools.

Visual analytic tool building is a complex design activity. A critical success factor is creating visual and interaction models that fit the users' mental map. Examining interactions with the visualization is not sufficient to comprehend the user's intent [80]. Characterizing user goals and tasks should be considered as a part of the study: examining why a user executed a task, the purpose of the task, and how the user executed it. Future research could categorize reverse engineering tasks into patterns for cognitive analysis mapping.

Lightweight systems are considerably simpler to build, deploy, and use. When visualization requirements increase, so do the requirements of the input capture. Several iterations of the GUI design were explored prior to selecting the node-link layout common in other provenance applications [6, 5, 7]. In examining exiting toolkits, Visual Storytelling emerged in ease of use, flexibility, and layout quality. While originally developed for tracking provenance in web applications, its design allowed for externally provided provenance data. The researchers developed the modules necessary to support bi-directional links with the reversing application. The following points highlight some of the difficulties encountered:

Visual clutter: Graphs are often used for representing software structures such as functions, data types, and subsystems. However, as the complexity of software systems increase, so too does the visual clutter. Tools should quickly and succinctly answer the user's questions to speed up the process of program comprehension. The SensorRE prototype adopted a minimalist design to reduce interface complexity, especially when multiple branches were opened. Some participants commented that the level of information shown was helpful. They could easily see an overview of

the analysis while hiding irrelevant branch data. For future work, we plan to add additional features to reduce clutter including filtering, dynamic search, and visual markers for nodes.

Visual scalability: We addressed scalability in the design of the tool in several ways. We created a visually scalable solution by supporting vertical scrolling, hiding labels in non-active branches, and by re-aligning trees based on the user's selection. While the examples exhibit a small number of nodes, there are circumstances in which the number is much higher. Scalable solutions are still a challenge in visual analytics research [10]. A potential solution could be to incorporate a hierarchical arrangement of nodes while providing the means to collapse or expand parts according to the user's needs.

6.2.3 Evaluating Provenance Tools.

Provenance tools can be benchmarked through user studies by comparing tools against each other [129, 130], or measuring its usability [73]. Since SensorRE is the first reverse engineering provenance tool there were no systems to compare it to. Instead, we evaluated the system through usability metrics. Certain reverse engineering specialties, such as malware analysts, benefit extensively from knowledge gained through their interactions and annotations [53]. By providing a general provenance tool, the externalized knowledge can be studied to improve user performance among multiple specialties.

In highly-studied provenance domains, technical performance in user tasks (e.g., speed or accuracy) can be compared between systems using standard datasets. However, in reverse engineering, standard datasets are lacking [3, 23]. Measuring the tool's usefulness is helpful but complex since usefulness is highly context specific and prone to bias [2]. Therefore, research into standardized datasets for the reverse engineering

domain should be investigated.

6.3 Future Work

This research showed that analytic provenance tools can effectively support reverse engineers' sensemaking process. We suggest the following future research directions to have a high impact supporting reverse engineers through analytic provenance:

- Analyzing provenance histories may not only be used to communicate findings but to serve as a training aid or performance-critiquing tool to identify efficient lines of reasoning. Provenance histories may contribute to our understanding of common analysis patterns. Larger-scale analysis of these patterns may improve our understanding of the sensemaking process and suggest enhanced interface designs supporting the domain. With a large corpus of history data, machine learning approaches may be useful in identifying optimal analysis paths for the human reverse engineer working with unknown binaries.

- SensorRE captures all user actions in real time, but what if the sensemaking actions could be predicted before they are completed? Machine learning approaches may prove useful in identifying optimal analysis paths in unknown binaries. This could significantly improve the reverse engineering landscape by leveraging automation to improve the human-computer interface.

- The evaluation of SensorRE only scratches the surface for collaborative communication among analysts. Further research is needed to explore the rich context of the reverse engineers' sensemaking process and how they collaborate in teams. Analysis patterns may reveal cognitive profiles that could be useful for team selection, bringing together analysts with different styles to complement each other when put together as a unit.

- The collaboration method provided by SensorRE is asynchronous. However, the system could be further expanded to support synchronous collaboration, wherein the user's analysis is automatically updated to a live database for sharing with other users. Further research could examine synchronous communication approaches closely and potentially merge the capability into the existing provenance tool.

- Binary Ninja provided an ideal platform for interface development due to its powerful and flexible API. However, there are a variety of other popular reverse engineering platforms such as IDA Pro, Radare 2.0, and Ghidra. Developing modules supporting these platforms should attract more research effort.

- Although the SensorRE provenance graph and story board views support multiple detail levels, the design and implementation of a truly scalable provenance visualization was not the main focus here and is therefore open for future research.

6.4 Closing Remarks

Software reverse engineers are more in demand now than ever before. New cognitive assistance tools are needed to rapidly train and support these engineers to counter growing malicious threats. This research presented SensorRE, the first analytic provenance system designed for software reverse engineers. A user needs study with subject matter experts helped shape the design and implementation details of the system. The resulting prototype automatically captures, manages, and visualizes reverse engineering provenance data in a common disassembly tool. Combining the quantitative and qualitative results from both experts and graduate student user studies revealed that all participants found the prototype easy to use. The system

helped the participants organize their workflow, quickly navigate to the provenance data they wanted, and effectively communicate their findings.

Analyzing provenance histories can contribute to our understanding of common analysis patterns and foster the creation of enhanced interface designs. Provenance histories may also prove useful as a training aide or performance critiquing tool to identify efficient lines of reasoning. SensorRE has the potential to provide a new and powerful approach for supporting reverse engineers.

Appendix A. Request for Human Experimentation

7 Aug 2018

MEMORANDUM FOR AFIT EXEMPT DETERMINATION OFFICIAL

FROM: AFIT/ENG
2950 Hobson Way
Wright Patterson AFB OH 45433-7765

SUBJECT: Request for exemption from human experimentation requirements (32 CFR 219, DoDD 3216.2 and AFI 40-402) for Requirements Elicitation in Software Reverse Engineering

1. The purpose of this study is to gain a better understanding of the software reverse engineering environment and work practices to inform the design of a future visualization tool. We are trying to learn more about the typical workflow in their environment. This information will be useful in designing better reverse engineering tools or visualizations that address current needs of reverse engineers. The results are intended to be published in peer-reviewed venues as well as in the doctoral dissertation.

2. This request is based on the Code of Federal Regulations, title 32, part 219, section 101, paragraph (b)(2) Research activities that involve the use of educational tests (cognitive, diagnostic, aptitude, achievement), survey procedures, interview procedures, or observation of public behavior unless: (i) Information obtained is recorded in such a manner that human subjects can be identified, directly or through identifiers linked to the subjects; and (ii) Any disclosure of the human subjects' responses outside the research could reasonably place the subjects at risk of criminal or civil liability or be damaging to the subjects' financial standing, employability, or reputation.

3. The following information is provided to show cause for such an exemption:

a) Equipment and facilities: The equipment for this study will include a personal laptop and audio recording of the subject's responses for analysis by the researcher. The interviews will take place in person at the Air Force Institute of Technology in a pre-reserved conference room.

b) Subjects: The requirements elicitation interview will consist of up to 15 volunteers solicited by the researcher. The target population have existing voluntary activities collaborating with AFIT researchers and are under no internal or external

pressure to participate. Participants may include a mix of active duty military, DoD civilians, and potentially DoD contractors. Subjects are expected to vary in skill level from journeyman (two to five years) up to subject matter experts (typically five to seven years). Subjects with fewer than two years of hands-on experience in reverse engineering will be excluded.

Factors such as age, sex, race, or job designation will not be used to include or exclude candidates.

c) Timeframe: The study will be conducted over the period of two months (not including data analysis). Individual interview sessions are expected to be between one to two hours. After two hours, the interview session will be terminated.

d) Data collected: Demographic data will be collected including the subject's educational background and experience in the reverse engineering domain. This information will only be used to outline potential sources of bias or excluding criteria in the research.

I understand that any names and associated data I collect must be protected at all times, only be known to the researchers, and managed according to AFIT interview protocol. All interview data will only be handled by the following researchers: Major Wayne Henry, Dr. Gilbert Peterson. At the conclusion of the study, all data will be turned over to Dr. Peterson and all other copies will be destroyed.

Research data will include audio and written recordings of each participant as he or she responds to the interview. The interview protocol is attached.

e) Risks to subjects: The risk to subjects includes the potential accidental disclosure of collected data on their background and experience. Steps will be taken to protect the subject's identity including masking participants' names and avoiding reference to their specific organization. The data will not provide any other purpose besides establishing the subject's experience in the target domain. If the participant inadvertently releases personally identifiable information during the interview, it will be sanitized by the researcher.

f) Informed consent: All subjects will be self-selected to volunteer to participate in the interview. No adverse action is taken against those who choose not to participate. Subjects are made aware of the nature and purpose of the research, sponsors of the research, and disposition of the results. A copy of the Privacy Act Statement of 1974 is presented for their review.

g) Adverse Impact: If a subject's future response reasonably places them at risk of criminal or civil liability or is damaging to their financial standing, employability

or reputation, I understand that I am required to immediately file an adverse event report with the IRB office.

4. If you have any questions about this request, please contact Major Wayne Henry – Phone 785-3636, ext. 6146; E-mail – wayne.henry@afit.edu.

//SIGNED//
GILBERT L. PETERSON, Ph.D.
Faculty Advisor, AFIT/ENG
Principal Investigator

//SIGNED//
WAYNE C. HENRY, Major, USAF
Graduate Student, AFIT/ENG

Attachments: 1. Interview Protocol

Appendix B. User Needs Survey Interview Protocol

Interview Protocol

Investigator: Maj Wayne Henry, AFIT/ENG
Date:

Introduction:

Hello, my name is Maj Wayne "Chris" Henry. This research is related to doctoral work at the Air Force Institute of Technology's Department of Electrical and Computer Engineering.

You have been selected to speak with us today because you have been identified as someone with working knowledge in the field of software reverse engineering in the security field. This interview will focus on gaining a better understanding of your environment and work practices. Our study does not aim to evaluate your specific techniques or any protected information. Instead, we are trying to learn more about the workflow to better understand the processes used by reverse engineers in a security setting. This information will be useful to inform the design of a future visualization capability.

To facilitate our note-taking, we would like to audio record our conversations today. Your information will be held confidential and you may stop at any time you feel uncomfortable with the question. Given the security environment where you work, if at any time you feel you cannot answer the question without revealing classified information, please refrain from answering.

We have planned this interview to last approximately one to two hours. During this time, we have several questions that we would like to cover. If you need a break, please let me know and we will take a 10- minute break. If time begins to run short, it may be necessary to interrupt you in order to push ahead and complete the line of questioning. Thank you for agreeing to participate.

Interview Questions

1. Interviewee Background:
   (a) How long have you been practicing software reverse engineering?
   (b) What is your highest degree?
   (c) What is your field of study?
   (d) How would you rate your skill level in software reverse engineering?

2. General Program Understanding:

   (a) Can you describe your typical reverse engineering workflow focusing on key decision points you make in your analysis?

   (b) What questions are you trying to satisfy?

   (c) What are your goals?

   (d) What key characteristics are you looking for in the binary?

3. Tools:

   (a) What tools do you regularly use (primary and secondary)?

   (b) Do you use program summary tools (API Monitor, CFF Explorer, BinDiff, Lighthouse) in your workflow?

   (c) What do you consider the best features about these tools?

   (d) What deficiencies or challenges do you have with tools you commonly use?

   (e) What tools are most needed? (wish list)

4. Visualizations:

   (a) What visualizations do you regularly use?

   (b) How do they help to solve current tasks?

   (c) What visualization features are most useful?

   (d) What deficiencies or challenges do you have with current visualizations? (worst features)

   (e) Where are visualizations most needed? (wish list)

   (f) What phase of the workflow?

   (g) What challenges or usage barriers exist for visualization tools in reverse engineering? (adoption)

   (h) Are visualizations more useful for novices than experts? (limitations)

5. Time constraints:

   (a) How important are time constraints in your analysis?

   (b) What impact does available time have on your processes or workflow?

   (c) How much time does a typical assessment take?

   (d) How long does an initial assessment require? What do you look for?

6. Results and Collaboration:

   (a) What products are created from your analysis (reports, other programs, etc.)?

(b) How important is collaboration in reverse engineering?

(c) What is the size of a typical team?

(d) What artifacts or resources are common for transferring knowledge between the team?

(e) What impediments to collaboration exist in the environment?

Appendix C. Ethics Approval - User Needs Survey

This appendix provides the approval for the exemption request for human experimentation requirements protocol number REN2018031R from the Air Force Institute of Technology. The study was approved on 31 August, 2018.

DEPARTMENT OF THE AIR FORCE
AIR FORCE INSTITUTE OF TECHNOLOGY
WRIGHT-PATTERSON AIR FORCE BASE OHIO

31 August 2018

MEMORANDUM FOR GILBERT L. PETERSON, PHD

FROM: William A. Cunningham, Ph.D.
AFIT IRB Research Reviewer
2950 Hobson Way
Wright-Patterson AFB, OH 45433-7765

SUBJECT: Approval for exemption request from human experimentation requirements (32 CFR 219, DoDD 3216.2 and AFI 40-402) for your study on Requirements Elicitation in Software Reverse Engineering, package number REN2018031R Peterson.

1. Your request was based on the Code of Federal Regulations, title 32, part 219, section 101, paragraph (b) (2) Research activities that involve the use of educational tests (cognitive, diagnostic, aptitude, achievement), survey procedures, interview procedures, or observation of public behavior unless: (i) Information obtained is recorded in such a manner that human subjects can be identified, directly or through identifiers linked to the subjects; and (ii) Any disclosure of the human subjects' responses outside the research could reasonably place the subjects at risk of criminal or civil liability or be damaging to the subjects' financial standing, employability, or reputation.

2. Your study qualifies for this exemption because you are not collecting sensitive data, which could reasonably damage the subjects' financial standing, employability, or reputation. Further, the demographic data you are utilizing and the way that you plan to report it cannot realistically be expected to map a given response to a specific subject.

3. This determination pertains only to the Federal, Department of Defense, and Air Force regulations that govern the use of human subjects in research. Further, if a subject's future response reasonably places them at risk of criminal or civil liability or is damaging to their financial standing, employability, or reputation, you are required to file an adverse event report with this office immediately.

WILLIAM A CUNNINGHAM, PH.D.
AFIT Exempt Determination Official

Appendix D. Software Listings

Binary Ninja Plugin - autocollect.py

```
1  Copyright 2020 Wayne C. Henry
2
3  Licensed under the Apache License, Version 2.0 (the "License"); you may
       not use this file except in compliance with the License. You may
       obtain a copy of the License at
4
5      http://www.apache.org/licenses/LICENSE-2.0
6
7  Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
       implied. See the License for the specific language governing
       permissions and limitations under the License.
8
9  import threading
10 import sys
11 import gc
12 import os, sys
13 import binaryninja as bn
14 import ctypes
15 import json
16 from binaryninja import scriptingprovider
17 import tempfile
18 import time
19 import difflib
20 from collections import defaultdict, OrderedDict
21 import collections
22
23 current_addr = 0x401000
```

```
24 current_view = "Graph:PE"
25 var_state = None
26 func_name = None
27 comment_state = None
28 highlight_state = None
29 data_state = None
30 dict_funcs = None
31 func_type = None
32 eventfunc2  = time.time()
33
34 def serialize(obj):
35     """JSON serializer for objects not serializable by default json code
       """
36
37     return obj.__dict__
38
39 def printJSONFile(data):
40
41     fullpath = "jsondata.json"
42
43     json_dump = json.dumps(data, sort_keys=True)
44
45     try:
46         jf = open(fullpath, "a+")
47         jf.write(json_dump + "\n")
48         jf.close()
49     except IOError:
50         print("ERROR: Unable to open/write to {}".format(fullpath))
51     return
52
53 class OrderedSet(collections.Set):
54     def __init__(self, iterable=()):
```

```
55          self.d = collections.OrderedDict.fromkeys(iterable)
56      def __len__(self):
57          return len(self.d)
58      def __contains__(self, element):
59          return element in self.d
60      def __iter__(self):
61          return iter(self.d)
62
63  def type_lookup(var_type):
64      type_list = ['int16_t', 'int24_t', 'int32_t', 'char', 'void', '
            uint16_t','uint24_t','uint32_t',
65          'float8','float16','float24','float', 'double','float72','long
                double',
66          'void*','void* const','void* volatile','void&','int32_t*']
67
68      if (type_list.count(var_type) > 0):
69          print("found var: {}".format(var_type))
70          return True
71      else:
72          return False
73
74
75  def setValue(bip, bv):
76      global current_addr, selChanged, current_view, var_state, func_name,
              comment_state, highlight_state, data_state, dict_funcs,
            func_type
77      valueChanged = current_addr != bip.current_addr
78      if (valueChanged):
79          print("valueChanged")
80          update_ns(bip, bv)
81      current_addr = bip.current_addr
82      current_view = bv.file.view
```

```
83      try:
84          if (bv.file.view == "Graph:PE" or bv.file.view == "Linear:PE"):
85              var_state = bip.current_func.vars
86              func_name = bip.current_func.symbol.name
87              func_type = bip.current_func.return_type
88              comment_state = bip.current_func.comments
89              highlight_state = bip.current_func.get_instr_highlight(
                    current_addr)
90              dict_funcs = func_types(bv)
91
92          if (bv.file.view == "Hex:PE"):
93              data_state = bv.read(bip.current_addr,1)
94      except Exception as e:
95          print("Found setValue exception {}".format(e))
96
97  def update_ns(bip, bv):
98      """Updates the namespace of the running kernel with the binja magic
            variables"""
99
100     global current_addr, current_view
101
102     print("[*] Printing view updates!")
103     current_addr = hex(int(bip.current_addr))
104     current_view = bv.file.view
105
106     return
107
108 def func_types(bv):
109     s = []
110     tup_master = ()
111     for func in bv.functions:
112         tup_temp = (str(func)[11:-1], str(func.return_type))
```

```
113             s.append(tup_temp)
114             tup_temp2 = (str(func)[11:-1], str(func.name))
115             s.append(tup_temp2)
116 
117         d = defaultdict(list)
118         for k, v in s:
119             d[k].append(v)
120         return d
121 
122 def diff_func_types(a, b):
123         # Change function type
124         address = 0
125         diff_change = 0
126         set_diff = None
127 
128         for i in a:
129             diff = set(a[i]) - set(b[i])
130             if (len(diff) > 0):
131                 address = i
132                 diff_change = diff
133                 set_diff = set(a[i])
134 
135         return address, diff_change, set_diff
136 
137 def start_watch(bv):
138 
139         obj = [o for o in gc.get_objects() if isinstance(o,
                scriptingprovider.PythonScriptingInstance.InterpreterThread)]
140         if len(obj) == 1:
141             bip = obj[0]
142         else:
143             raise Exception("Couldn't find scriptingprovider. Sure you are
```

```
            in the right kernel?")
144
145     setValue(bip, bv)
146     threading.Timer(1, start_watch, [bv]).start()
147
148 def func_updated(bv, function):
149     global eventfunc2, var_state, current_addr, func_name, comment_state
            , highlight_state, dict_funcs
150     data = OrderedDict()
151     temp_name = 0
152     temp_type = 0
153
154     try:
155         if (eventfunc2 + 1 < time.time()):
156             #Check for var name collision
157             if (str(bv.get_functions_containing(current_addr)[0]) != str
                    (function)):
158                 var_state = function.vars
159
160             # Local Var name/type change
161             for item,var in enumerate(var_state):
162                 if (str(var_state[item].name) != str(function.vars[item
                        ].name)) and temp_name == 0:
163                     print("[] Name change: {} {}".format(function.vars[
                            item].name, item))
164                     var_type_new, var_name_new, index = function.vars[
                            item].type, function.vars[item].name, item
165                     var_type_old, var_name_old = var_state[item].type,
                            var_state[item].name
166                     temp_name = 1
167                 if (str(var_state[item].type) != str(function.vars[item
                        ].type)) and temp_type == 0:
```

```
168                     print("[] Type change: {} {}".format(function.vars[
                            item].type, item))
169                     var_type_new, var_name_new, index = function.vars[
                            item].type, function.vars[item].name, item
170                     var_type_old, var_name_old = var_state[item].type,
                            var_state[item].name
171                     temp_type = 1
172
173                 # Local Var name/type change
174                 if (temp_name == 1 and temp_type == 1):
175                     print("[*] Var_Updated: func:{} func_addr:{}
                            var_name_new:{} var_type_new:{} var_name_old:{}
                            var_type_old:{}"
176                           .format(function.symbol.name, str(function)
                                  [11:-1], var_name_new, var_type_new,
                                  var_name_old, var_type_old))
177                     data = {
178                         'type': 'var_updated',
179                         'function': str(function.symbol.name),
180                         'func_addr': str(function)[11:-1],
181                         'var_name_new': str(var_name_new),
182                         'var_type_new': str(var_type_new),
183                         'var_name_old': str(var_name_old),
184                         'var_type_old': str(var_type_old),
185                         'index': str(index),
186                         'view': current_view
187                     }
188                     printJSONFile(data)
189                 elif temp_type == 1:
190                     data = {
191                         'type': 'var_type_updated',
192                         'function': str(function.symbol.name),
```

```
193                     'func_addr': str(function)[11:-1],
194                     'var_name_new': str(var_name_new),
195                     'var_type_new': str(var_type_new),
196                     'var_name_old': str(var_name_old),
197                     'var_type_old': str(var_type_old),
198                     'index': str(index),
199                     'view': current_view
200                 }
201                 printJSONFile(data)
202             elif temp_name == 1:
203                 data = {
204                     'type': 'var_name_updated',
205                     'function': function.symbol.name,
206                     'func_addr': str(function)[11:-1],
207                     'var_name_new': str(var_name_new),
208                     'var_type_new': str(var_type_new),
209                     'var_name_old': str(var_name_old),
210                     'var_type_old': str(var_type_old),
211                     'index': str(index),
212                     'view': current_view
213                 }
214                 printJSONFile(data)
215             var_state = function.vars
216
217             #Function name change (from a call)
218             dict_funcs_new = func_types(bv)
219             new_key_change, new_key_diff, new_set_diff = diff_func_types
                    (dict_funcs_new, dict_funcs)
220             if (new_key_diff > 0):
221
222                 print('[*] Updating function name {name}'.format(name=
                        function.symbol.name))
```

```
223                 print("func_new: {} {} {}".format(new_key_change,
                        new_key_diff, new_set_diff))
224                 old_key_change, old_key_diff, old_set_diff =
                        diff_func_types(dict_funcs, dict_funcs_new)
225                 print("func_old: {} {} {}".format(old_key_change,
                        old_key_diff, old_set_diff))
226
227                 if (old_key_change != 0 and new_key_change != 0):
228                     # Name change
229                     print("keydiff: {} {}".format(old_key_diff,
                            new_key_diff))
230                     if (str(old_key_diff) != str(new_key_diff)):
231                         print("**** Name updated****")
232                         if (str(list(new_set_diff)[1]) == str(list(
                                old_set_diff)[1])):
233                             data = {
234                                 'type': 'func_name_updated',
235                                 'func_addr': str(old_key_change),
236                                 'function_name_new': str(new_key_diff)
                                        [6:-3],
237                                 'function_name_old': str(old_key_diff)
                                        [6:-3],
238                                 'function_type_new': str(list(
                                        new_set_diff)[1]),
239                                 'function_type_old': str(list(
                                        old_set_diff)[1]),
240                                 'view': current_view
241                             }
242                         else:
243                             data = {
244                                 'type': 'func_name_updated',
245                                 'func_addr': str(old_key_change),
```

```
246                         'function_name_new': str(new_key_diff)
                                [6:-3],
247                         'function_name_old': str(old_key_diff)
                                [6:-3],
248                         'function_type_new': str(list(
                                new_set_diff)[1]),
249                         'function_type_old': str(list(
                                old_set_diff)[0]),
250                         'view': current_view
251                     }
252                 # Name and Type change
253                 elif (len(list(new_key_diff)) > 1):
254                     print("**** Name and Type change ****")
255                     if (type_lookup(list(old_key_diff)[0])):
256                         # First element in old set is the type
257                         data = {
258                             'type': 'func_name_type_updated',
259                             'func_addr': str(old_key_change),
260                             'function_name_new': str(list(
                                    new_set_diff)[0]),
261                             'function_name_old': str(list(
                                    old_key_diff)[1]),
262                             'function_type_new': str(list(
                                    new_set_diff)[1]),
263                             'function_type_old': str(list(
                                    old_set_diff)[0]),
264                             'view': current_view
265                         }
266                     else:
267                         # Second element in old set is the type
268                         data = {
269                             'type': 'func_name_type_updated',
```

```
270                             'func_addr': str(old_key_change),
271                             'function_name_new': str(list(
                                    new_set_diff)[0]),
272                             'function_name_old': str(list(
                                    old_key_diff)[0]),
273                             'function_type_new': str(list(
                                    new_set_diff)[1]),
274                             'function_type_old': str(list(
                                    old_set_diff)[1]),
275                             'view': current_view
276                         }
277
278                     # Type change
279                     else:
280                         print("**** Type updated****")
281                         data = {
282                             'type': 'func_type_updated',
283                             'func_addr': str(old_key_change),
284                             'function_name_new': str(list(new_set_diff)
                                    [0]),
285                             'function_name_old': str(list(old_set_diff)
                                    [0]),
286                             'function_type_new': str(list(new_set_diff)
                                    [1]),
287                             'function_type_old': str(list(old_set_diff)
                                    [1]),
288                             'view': current_view
289                         }
290                     dict_funcs = dict_funcs_new
291                     eventfunc2 = time.time()
292                     printJSONFile(data)
293
```

```
294
295             #Comment state change
296             if ((comment_state != function.comments)):
297                 address, comment, comment_text = None, None, None
298                 comment_state_len = len(comment_state)
299                 new_comment_len = len(function.comments)
300                 print("Comment change {} {}".format(comment_state_len,
                        new_comment_len))
301                 print("comment_state: {}".format(comment_state))
302                 print("function.comments: {}".format(function.comments))
303                 # Added
304                 if (comment_state_len < new_comment_len or
                        comment_state_len == new_comment_len):
305                     for item in function.comments.items():
306                         if item not in comment_state.items():
307                             address = item[0]
308                             comment = item[1]
309                             comment_text = "comment_changed"
310                             print("[*] Comment changed: {}".format(
                                    comment))
311
312
313                     if not comment_state.values():
314                         data = {
315                             'type': comment_text,
316                             'func': str(function)[11:-1],
317                             'addr': hex(int(address)),
318                             'comment_new': comment,
319                             'comments_old': "",
320                             'view': current_view
321                         }
322                     else:
```

```
323                         data = {
324                             'type': comment_text,
325                             'func': str(function)[11:-1],
326                             'addr': hex(int(address)),
327                             'comment_new': comment,
328                             'comments_old': comment_state.values().pop
                                    (0),
329                             'view': current_view
330                         }
331
332                 # Removed comment
333                 elif (comment_state_len > new_comment_len):
334                     for item in comment_state.items():
335                         if item not in function.comments.items():
336                             address = item[0]
337                             comment = item[1]
338                             comment_text = "comment_removed"
339                             print("[*] Comment removed: {}".format(
                                    comment))
340                             data = {
341                                 'type': comment_text,
342                                 'func': str(function)[11:-1],
343                                 'addr': hex(int(address)),
344                                 'comment_new': "",
345                                 'comments_old': comment,
346                                 'view': current_view
347                             }
348                     comment_state = function.comments
349                     printJSONFile(data)
350
351             # Highlight change:
352             if (str(highlight_state) != str(function.get_instr_highlight
```

```
                (current_addr))):
353             print("[*] Highlight change: {} {}".format(hex(int(
                    current_addr)), function.get_instr_highlight(
                    current_addr)))
354             print("highlight_state: {}".format(highlight_state))
355             print("get_instr_highlight: {}".format(function.
                    get_instr_highlight(current_addr)))
356
357             color_old, color_new = color_matching(str(
                    highlight_state), str(function.get_instr_highlight(
                    current_addr)))
358
359             data = {
360                 'type': "highlight",
361                 'func': str(function)[11:-1],
362                 'addr': hex(int(current_addr)),
363                 'color_new': color_old,
364                 'color_old': color_new,
365                 'view': current_view
366             }
367             highlight_state = function.get_instr_highlight(
                    current_addr)
368             printJSONFile(data)
369
370         else:
371             print("Skipping func_updated1")
372     # else:
373     #     print("Skipping func_updated2")
374     except Exception as e:
375         print("Exception: skipping func_update: {}".format(e))
376
377 def color_matching(color_old, color_new):
```

```
378     colors_dict = {'none':'NoHighlightColor', 'black':'
            BlackHighlightColor', 'blue':'BlueHighlightColor',
379                    'cyan':'CyanHighlightColor', 'green': '
                           GreenHighlightColor', 'magenta': '
                           MagentaHighlightColor',
380                    'orange': 'OrangeHighlightColor', 'red': '
                           RedHighlightColor', 'white': '
                           WhiteHighlightColor',
381                    'yellow': 'YellowHighlightColor'}
382
383     _old_color=color_old.split(':')[1].split()[0][:-1]
384     _new_color=color_new.split(':')[1].split()[0][:-1]
385
386     print(_old_color, _new_color)
387     old_color = colors_dict.get(_old_color, color_old)
388     new_color = colors_dict.get(_new_color, color_new)
389
390     return old_color, new_color
391
392 def func_added(bv, function):
393     global eventfunc2
394
395     if (eventfunc2 + 2 < time.time()):
396         data = OrderedDict()
397         print("[*] Function Added: {}".format(function.symbol.name))
398         data = {
399             'type': 'func_added',
400             'function': function.symbol.name,
401             'func_addr': str(function)[11:-1],
402             'view': current_view
403         }
404         eventfunc2 = time.time()
```

```
405         printJSONFile(data)
406
407 def func_removed(bv, function):
408     global eventfunc2
409
410     if (eventfunc2 + 1 < time.time()):
411         data = OrderedDict()
412         print("[*] Function Removed: {}".format(function.symbol.name))
413         data = {
414             'type': 'func_removed',
415             'function': function.symbol.name,
416             'func_addr': str(function)[11:-1],
417             'view': current_view
418         }
419         eventfunc2 = time.time()
420         printJSONFile(data)
421
422 def data_written(bv, address, length):
423     global eventfunc2, data_state#, eventfunc
424
425     if (eventfunc2 + 1 < time.time()):
426         data = OrderedDict()
427         print('[*] Data Written <0x{name:x}> {length}'.format(name=
                address, length=length))
428
429         data_new = bv.read(address, 1)
430         print("new data: {} old data: {}".format(data_new, data_state))
431         data = {
432             'type': 'data_written',
433             'address': "0x{:x}".format(int(address)),
434             'length': str(length),
435             'data_old': str(data_state),
```

```
436             'data_new': str(data_new),
437             'view': current_view
438         }
439 
440         printJSONFile(data)
441         eventfunc2 = time.time()
442 
443     else:
444         pass
445 
446 event = time.time()
447 
448 def type_defined(bv, name, type):
449     global event, eventfunc2
450 
451     if (event + 1 < time.time()):
452         if (eventfunc2 + 1 < time.time()):
453             data = OrderedDict()
454             print('[*] Type Defined')
455             event = time.time()
456             data = {
457                 'type': 'type_defined',
458                 'name': str(name),
459                 'type_defined': str(type),
460                 'view': current_view
461             }
462 
463             printJSONFile(data)
464 
465     else:
466         print('[*] Type Defined - skipping')
467         pass
```

```
468
469 def type_undefined(bv, name, type):
470     global event, eventfunc2
471
472     if (event + 1 < time.time()):
473         if (eventfunc2 + 1 < time.time()):
474             data = OrderedDict()
475             print('[*] Type Undefined')
476             print(event)
477             data = {
478                 'type': 'type_undefined',
479                 'name': str(name),
480                 'type2': str(type),
481                 'view': current_view
482             }
483
484             printJSONFile(data)
485
486     else:
487         print('[*] Type Undefined - skipping')
488         pass
489
490 def data_var_added(bv, var):
491     global eventfunc2
492
493     if (eventfunc2 + 1 < time.time()):
494         data = OrderedDict()
495         print('[*] Data_var_added')
496         data = {
497             'type': 'data_var_added',
498             'var': str(var),
499             'view': current_view
```

```
500         }
501         eventfunc2 = time.time()
502         printJSONFile(data)
503 
504     else:
505         print('[*] Data Var Added - skipping')
506         pass
507 
508 def data_var_removed(bv, var):
509     global eventfunc2
510 
511     if (eventfunc2 + 1 < time.time()):
512         data = OrderedDict()
513         print('[*] Data_var_removed')
514         data = {
515             'type': 'data_var_removed',
516             'var': str(var),
517             'view': current_view
518         }
519         eventfunc2 = time.time()
520         printJSONFile(data)
521     else:
522         print('[*] Data Var Removed - skipping')
523         pass
```

Binary Ninja Plugin - __init__.py

```
1 Copyright 2020 Wayne C. Henry
2 
3 Licensed under the Apache License, Version 2.0 (the "License"); you may
    not use this file except in compliance with the License. You may
    obtain a copy of the License at
```

```
4
5       http://www.apache.org/licenses/LICENSE-2.0
6
7   Unless required by applicable law or agreed to in writing, software
        distributed under the License is distributed on an "AS IS" BASIS,
        WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
        implied.  See the License for the specific language governing
        permissions and limitations under the License.
8
9   from collections import OrderedDict
10  from SimpleXMLRPCServer import SimpleXMLRPCRequestHandler,
        SimpleXMLRPCServer, list_public_methods
11  import threading, string, inspect, copy, socket, xmlrpclib
12  import binaryninja as bn
13  import time
14  import autocollect
15
16  HOST, PORT = "0.0.0.0", 1337
17  DEBUG = True
18  HL_NO_COLOR = bn.HighlightStandardColor.NoHighlightColor
19  HL_BP_COLOR = bn.HighlightStandardColor.RedHighlightColor
20  HL_CUR_INSN_COLOR = bn.HighlightStandardColor.GreenHighlightColor
21
22  started = False
23  t = None
24  _current_instruction = 0
25  func_hold = "test"
26  type_flag = 0
27
28  PAGE_SZ = 0x1000
29
30  def expose(f):
```

```
31    "Decorator to set exposed flag on a function."
32    f.exposed = True
33    return f
34
35  def is_exposed(f):
36    "Test whether another function should be publicly exposed."
37    return getattr(f, 'exposed', False)
38
39  def ishex(s):
40    return s.startswith("0x") or s.startswith("0X")
41
42  class RequestHandler(SimpleXMLRPCRequestHandler):
43    rpc_paths = ("/RPC2",)
44
45    def do_OPTIONS(self):
46      self.send_response(200)
47      self.end_headers()
48
49    def end_headers(self):
50      self.send_header("Access-Control-Allow-Headers",
51                "Origin, X-Requested-With, Content-Type, Accept")
52      self.send_header("Access-Control-Allow-Origin", "*")
53      SimpleXMLRPCRequestHandler.end_headers(self)
54
55  def start_service(host, port, bv):
56    print("[+] Starting service on {}:{}".format(host, port))
57    server = SimpleXMLRPCServer((host, port),
58                    requestHandler=RequestHandler,
59                    logRequests=False,
60                    allow_none=True)
61    server.register_introspection_functions()
62    server.register_instance(Bookmark(server, bv), allow_dotted_names=True
```

```
        )
63    print("[+] Registered {} functions.".format( len(server.
          system_listMethods()) ))
64    while True:
65      if hasattr(server, "shutdown") and server.shutdown==True: break
66      server.handle_request()
67    return
68
69  def start_server(bv):
70    global t, started
71    t = threading.Thread(target=start_service, args=(HOST, PORT, bv))
72    t.daemon = True
73    print("[+] Creating new thread {}".format(t.name))
74    t.start()
75
76    started = True
77    return
78
79  def stop_server(bv):
80    global t
81    t.join()
82    t = None
83    print("[+] Server stopped")
84    return
85
86  def server_start_stop(bv):
87    if t is None:
88      start_server(bv)
89      bn.show_message_box("Serv","Service successfully started, you can
            now connect to it",
90                bn.MessageBoxButtonSet.OKButtonSet, bn.MessageBoxIcon.
                    InformationIcon)
```

```
91      register_stuff(bv)
92    else:
93      try:
94        cli = xmlrpclib.ServerProxy("http://{:s}:{:d}".format(HOST, PORT))
95        cli.shutdown()
96      except socket.error:
97        pass
98      stop_server(bv)
99      bn.show_message_box("Serv", "Service successfully stopped",
100               bn.MessageBoxButtonSet.OKButtonSet, bn.MessageBoxIcon.
                     InformationIcon)
101   return
102
103 class Bookmark:
104   """
105   Top level class where exposed methods are declared.
106   """
107
108   def __init__(self, server, bv, *args, **kwargs):
109     self.server = server
110     self.view = bv
111     self.base = bv.entry_point & ~(PAGE_SZ-1)
112     self._version = ("Binary Ninja", bn.core_version)
113     self.old_bps = set()
114     return
115
116   def _dispatch(self, method, params):
117     """
118     Plugin dispatcher
119     """
120     func = getattr(self, method)
121     if not is_exposed(func):
```

```
122         raise NotImplementedError('Method "%s" is not exposed' % method)
123
124       if DEBUG:
125         print("[+] Executing %s(%s)" % (method, params))
126       return func(*params)
127
128     def _listMethods(self):
129       """
130       Class method listing (required for introspection API).
131       """
132       m = []
133       for x in list_public_methods(self):
134         if x.startswith("_"): continue
135         if not is_exposed( getattr(self, x) ): continue
136         m.append(x)
137       return m
138
139     def _methodHelp(self, method):
140       """
141       Method help (required for introspection API).
142       """
143       f = getattr(self, method)
144       return inspect.getdoc(f)
145
146     @expose
147     def shutdown(self):
148       """ shutdown() => None
149       Cleanly shutdown the XML-RPC service.
150       Example: binaryninja shutdown
151       """
152       self.server.server_close()
153       print("[+] XMLRPC server stopped")
```

```
154     setattr(self.server, "shutdown", True)
155     return 0
156
157   @expose
158   def version(self):
159     """ version() => None
160     Return a tuple containing the tool used and its version
161     Example: binaryninja version
162     """
163     return self._version
164
165   def begin_undo(self):
166     print("[+] Begin Undo")
167     return self.view.begin_undo_actions()
168
169   def commit_undo(self):
170     print("[+] Commit Undo")
171     return self.view.commit_undo_actions()
172
173   @expose
174   def Undo(self):
175     """ Undo() => None
176     Undo most recent action
177     Example: binaryninja Undo
178     """
179     autocollect.eventfunc2 = time.time()
180     return self.view.undo()
181
182   @expose
183   def Redo(self):
184     """ Redo() => None
185     Redo most recent action
```

```
186     Example: binaryninja Redo
187     """
188     self.begin_undo()
189     autocollect.eventfunc2 = time.time()
190     self.redo_helper()
191     return self.commit_undo()
192
193   def redo_helper(self):
194     return self.view.redo()
195
196   def var_lookup(self, var_type):
197     arch = self.view.arch
198     type_dict = {'int16_t': bn.Type.int(2), 'int24_t':bn.Type.int(3), '
          int32_t':bn.Type.int(4), 'char':bn.Type.char(),
199             'void':bn.Type.void(), 'uint16_t':bn.Type.int(2,0),'uint24_t
                ':bn.Type.int(3,0),'uint32_t':bn.Type.int(4,0),
200             'float8':bn.Type.float(1),'float16':bn.Type.float(2),'
                float24':bn.Type.float(3),'float':bn.Type.float(4),
201             'double':bn.Type.float(8),'float72':bn.Type.float(9),'long
                double':bn.Type.float(10),
202             'void*':bn.Type.pointer(arch, bn.Type.void(),False, False,
                False),
203             'void* const':bn.Type.pointer(arch,bn.Type.void(),True,
                False, False),
204             'void* volatile':bn.Type.pointer(arch, bn.Type.void(),False,
                 True, False),
205             'void&':bn.Type.pointer(arch, bn.Type.void(),False, False,
                True),
206             'int32_t*':bn.Type.pointer(arch, bn.Type.int(4),False, False
                , False)}
207
208     new_type = type_dict.get(var_type)
```

```
209     return new_type
210
211   @expose
212   def FuncVar(self, func_address, var_type, var_name, index):
213     """ FuncVar(str func_address, string type (int32_t), string var_name
          , int var index
214     s.FuncVar('0x401000', 'uint32_t', 'var_14', 0)
215     """
216     autocollect.eventfunc2 = time.time()
217     func = self.view.get_function_at(int(func_address,16))
218     _var_type = self.var_lookup(var_type)
219     self.Jump(func_address,'Graph:PE')
220     return func.create_user_var(func.vars[int(index)], _var_type,
          var_name)
221
222   @expose
223   def FuncName(self, address, funcName):
224     """ SetFunc(int address, string funcName) => None
225     Set Function name address to string
226     s.FuncName('0x40102c', "new")
227     """
228     autocollect.eventfunc2 = time.time()
229     self.Jump(address,'Graph:PE')
230     func = self.view.get_function_at(int(address,16))   #Only works for
          main function, not called
231     func.name = funcName
232
233   @expose
234   def FuncNameType(self, address, funcName, var_type):
235     """ SetFunc(int address, string funcName) => None
236     Set Function name address to string
237 import xmlrpclib
```

```
s = xmlrpclib.ServerProxy('http://localhost:1337')
s.FuncNameType('0x40102c', "new",'uint32_t')
    """
    autocollect.eventfunc2 = time.time()
    self.Jump(address,'Graph:PE')
    func = self.view.get_function_at(int(address,16))   #Only works for
        main function, not called
    func.name = funcName
    temp_var = self.var_lookup(var_type)
    func.return_type = temp_var

  @expose
  def FuncType(self, address, var_type):
    """ s.FuncType('0x401021','int32_t')
    """
    autocollect.eventfunc2 = time.time()
    func = self.view.get_function_at(int(address,16))
    temp_var = self.var_lookup(var_type)
    func.return_type = temp_var

  @expose
  def Jump(self, address, view):
    """ Jump(int addr) => None
    Move the EA pointer to the address pointed by `addr`.
    s.Jump('0x4049de','Graph:PE')
    """
    self.view.file.navigate(view, int(address,16))

  @expose
  def MakeComm(self, address, comment, function):
    """ MakeComm(str addr, string comment) => None
    Add a comment at the location `address`.
```

```
269     s.MakeComm('0x401019', "Important call here!", '0x401000')
270     """
271     autocollect.eventfunc2 = time.time()
272     func = self.view.get_function_at(int(function,16))
273     self.Jump(address,'Graph:PE')
274     return func.set_comment_at(int(address,16), comment)
275
276   def do_command(self, cmd):
277     print(cmd)
278     return eval(cmd)
279
280   @expose
281   def SetColor(self, address, color):
282     """ SetColor(int addr, string color) => None
283     Set the location pointed by `address` with `color`.
284     Example:  s.SetColor('0x401000', 'CyanHighlightColor')
285     """
286     autocollect.eventfunc2 = time.time()
287     start_addr = self.view.get_previous_function_start_before(int(
            address,16))
288     func = self.view.get_function_at(start_addr)
289     if func is None: return
290     color_new = "bn.HighlightStandardColor."+color
291     func.set_user_instr_highlight(int(address,16),eval(color_new))
292
293   @expose
294   def DefineFunc(self, address):
295     """ str address must be the address of the function
296     s.DefineFunc('0x401000')
297     """
298     autocollect.eventfunc2 = time.time()
299     self.view.create_user_function(int(address,16))
```

```
300       self.view.file.navigate('Graph:PE', int(address,16))
301
302     @expose
303     def UndefineFunc(self, address):
304       """ str address must be the address of the function
305       s.UndefineFunc('0x401000')
306       """
307       autocollect.eventfunc2 = time.time()
308       func = self.view.get_function_at(int(address,16))
309       self.view.remove_user_function(func)
310       self.view.file.navigate('Linear:PE', int(address,16))
311
312     @expose
313     def WriteData(self, address, data):
314       self.view.file.navigate('Hex:PE', int(address,16))
315       self.view.write(int(address, 16), data)
316
317     @expose
318     def AddType(self, name, var_type):
319
320       _var_type = self.var_lookup(var_type)
321       self.view.define_user_type(name, _var_type)
322
323     @expose
324     def RemoveType(self, name):
325       self.view.undefine_user_type(name)
326
327 #Register Notifications
328 class myNotification(bn.BinaryDataNotification):
329     def __init__(self, view):
330       self.view = view
331       pass
```

```
332
333   def data_written(self, view, offset, length):
334     print("data_written: ", view, offset, length)
335     autocollect.data_written(view, offset,length)
336     pass
337   def data_inserted(self, view, offset, length):
338     print("data_inserted: ", view, offset, length)
339     pass
340   def data_removed(self, view, offset, length):
341     print("data_removed: ", offset, length)
342     pass
343   def function_added(self, view, func):
344     print("function_added: ", func)
345     autocollect.func_added(view, func)
346     pass
347   def function_removed(self, view, func):
348     print("function_removed: ", func)
349     autocollect.func_removed(view, func)
350     pass
351   def function_updated(self, view, func):
352     # print("function_updated")
353     autocollect.func_updated(view, func)
354     pass
355   def data_var_added(self, view, var):
356     autocollect.data_var_added(view, var)
357     print("var_added: ", var)
358     pass
359   def data_var_removed(self, view, var):
360     autocollect.data_var_removed(view, var)
361     print("var_removed: ", var)
362     pass
363   def data_var_updated(self, view, var):
```

```
364         print("var_updated: ", var)
365         pass
366     def string_found(self, view, string_type, offset, length):
367         print("string_found: ", string_type, offset, length)
368         pass
369     def string_removed(self, view, string_type, offset, length):
370         print("string_removed: ", string_type, offset, length)
371         pass
372     def type_defined(self, view, name, type):
373         global type_flag
374         autocollect.type_defined(view, str(name), type)
375         print("type_defined: ", name, type)
376         pass
377     def type_undefined(self, view, name, type):
378         global type_flag
379         autocollect.type_undefined(view, str(name), type)
380         print("type_undefined: ", name, type)
381         pass
382
383 def on_complete(self):
384     print("Analysis Complete")
385
386 def register_stuff(bv):
387     notification = myNotification(bv)
388     bv.register_notification(notification)
389     autocollect.start_watch(bv)
390
391 bn.PluginCommand.register("Binja Start/Stop XML Server", "Start/Stop XML
        Server.", server_start_stop)
```

JavaScript - fileChange.py

```
1  Copyright 2020 Wayne C. Henry
2
3  Licensed under the Apache License, Version 2.0 (the "License"); you may
       not use this file except in compliance with the License. You may
       obtain a copy of the License at
4
5      http://www.apache.org/licenses/LICENSE-2.0
6
7  Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
       implied.  See the License for the specific language governing
       permissions and limitations under the License.
8
9  import { XmlRpcRequest } from "./mimic";
10
11 import {
12      ActionFunctionRegistry,
13      ProvenanceGraph,
14      ProvenanceTracker,
15      ProvenanceGraphTraverser,
16      ReversibleAction,
17      IrreversibleAction,
18      StateNode,
19      Action,
20      isReversibleAction,
21    } from '@visualstorytelling/provenance-core';
22
23
24 class FileChangeApp {
25      public method: string = "Starting...";
```

```
26
27      public async FuncNameUpdated(address: string, funcName: string){
28          this.method = "FuncNameUpdate";
29          console.log("FileChangeApp " + this.method);
30          const call = "FuncName";
31          let request = new (XmlRpcRequest as any)("http://localhost:1337/
                RPC2", call);
32          request.addParam(address);
33          request.addParam(funcName);
34          let response = await request.send();
35          console.log(response);
36      }
37      public async funcType(address: string, var_type: string){
38          this.method = "FuncTypeUpdate";
39          console.log("FileChangeApp " + this.method);
40          const call = "FuncType";
41          let request = new (XmlRpcRequest as any)("http://localhost:1337/
                RPC2", call);
42          request.addParam(address);
43          request.addParam(var_type);
44          let response = await request.send();
45          console.log(response);
46      }
47      public async funcNameType(address: string, funcName: string,
            var_type: string){
48          this.method = "FuncNameTypeUpdate";
49          console.log("FileChangeApp " + this.method);
50          const call = "FuncNameType";
51          let request = new (XmlRpcRequest as any)("http://localhost:1337/
                RPC2", call);
52          request.addParam(address);
53          request.addParam(funcName);
```

```
54          request.addParam(var_type);
55          let response = await request.send();
56          console.log(response);
57      }
58      public async LocalVarUpdate(funcAddress: string, var_type: string,
            var_name:string, index:string){
59          this.method = "LocVarUpdate";
60          const call = "FuncVar";
61          let request = new (XmlRpcRequest as any)("http://localhost:1337/
                RPC2", call);
62          request.addParam(funcAddress);
63          request.addParam(var_type);
64          request.addParam(var_name);
65          request.addParam(index);
66          let response = await request.send();
67          console.log(response);
68          console.log("FileChangeApp " + this.method);
69      }
70      public async CommentUpdated(address: string, message: string,
            functionAddr: string) {
71          this.method = "CommentUpdated";
72          const call = "MakeComm";
73          let request = new (XmlRpcRequest as any)("http://localhost:1337/
                RPC2", call);
74          request.addParam(address);
75          request.addParam(message);
76          request.addParam(functionAddr)
77          let response = await request.send();
78          console.log(response);
79          console.log("FileChangeApp " + this.method);
80      }
81      public async highlight(address: string, color: string) {
```

```
82          this.method = "highlight";
83          const call = "SetColor";
84          let request = new (XmlRpcRequest as any)("http://localhost:1337/
                RPC2", call);
85          request.addParam(address);
86          request.addParam(color);
87          let response = await request.send();
88          console.log(response);
89          console.log("FileChangeApp " + this.method);
90      }
91      public async defineFunc(address: string) {
92          this.method = "defineFunc";
93          const call = "DefineFunc";
94          let request = new (XmlRpcRequest as any)("http://localhost:1337/
                RPC2", call);
95          request.addParam(address);
96          let response = await request.send();
97          console.log(response);
98          console.log("FileChangeApp " + this.method);
99      }
100     public async undefineFunc(address: string) {
101         this.method = "undefineFunc";
102         const call = "UndefineFunc";
103         let request = new (XmlRpcRequest as any)("http://localhost:1337/
                RPC2", call);
104         request.addParam(address);
105         let response = await request.send();
106         console.log(response);
107         console.log("FileChangeApp " + this.method);
108     }
109     public async dataWritten(address: string, data: string) {
110         this.method = "dataWritten";
```

```
111         const call = "WriteData";
112         let request = new (XmlRpcRequest as any)("http://localhost:1337/
                RPC2", call);
113         request.addParam(address);
114         request.addParam(data);
115         let response = await request.send();
116         console.log(response);
117         console.log("FileChangeApp " + this.method);
118     }
119     public async addType(typeName: string, typeDefined: string) {
120         this.method = "addType";
121         const call = "AddType";
122         let request = new (XmlRpcRequest as any)("http://localhost:1337/
                RPC2", call);
123         request.addParam(typeName);
124         request.addParam(typeDefined);
125         let response = await request.send();
126         console.log(response);
127         console.log("FileChangeApp " + this.method);
128     }
129     public async removeType(typeName: string) {
130         this.method = "removeType";
131         const call = "RemoveType";
132         let request = new (XmlRpcRequest as any)("http://localhost:1337/
                RPC2", call);
133         request.addParam(typeName);
134         let response = await request.send();
135         console.log(response);
136         console.log("FileChangeApp " + this.method);
137     }
138     public async view(address: string, viewMode: string) {        //Jump
            action for Revert
```

```
139         this.method = "view";
140         const call = "Jump";
141         let request = new (XmlRpcRequest as any)("http://localhost:1337/
                RPC2", call);
142         request.addParam(address);
143         request.addParam(viewMode);
144         let response = await request.send();
145         console.log(response);
146         console.log("FileChangeApp " + this.method);
147     }
148 }
149
150 export class DataTypes {
151     //JSON Types:  func_name_updated
152     public datablock: string;
153     public type: string;
154     public view: string;
155
156     //JSON Types:  func_name_updated
157     public oldFuncName: string;
158     public newFuncName: string;
159     public address: string;
160     public oldFuncType: string;
161     public newFuncType: string;
162
163     // newcomment
164     public newComment: string;
165     public oldComment: string;
166     public functionAddr: string;
167
168     // Var_type updates
169     public functionName: string;
```

```
170     public index: string;
171     public newVarName: string;
172     public oldVarName: string;
173     public newVarType: string;
174     public oldVarType: string;
175
176     // data_written
177     public dataNew: string;
178     public dataOld: string;
179     public length: string;
180
181     // highlight
182     public colorNew: string;
183     public colorOld: string;
184
185     // Types
186     public typeName: string;
187     public typeDefined: string;
188
189     // View
190     public oldview: string;
191     public oldaddress: string;
192
193     constructor(datablock: string) {
194         this.datablock = datablock;
195         this.parseJSON(this.datablock);
196     }
197
198     private parseJSON(data: string) {
199         let obj:JSON = JSON.parse(data);
200         // func_name_updated
201         if ((Object.values(obj)[5] == 'func_name_updated') ||
```

```
202             (Object.values(obj)[5] == 'func_type_updated') ||
203             (Object.values(obj)[5] == 'func_name_type_updated')) {
204             this.oldFuncName = Object.values(obj)[2];
205             this.type = Object.values(obj)[5];
206             this.newFuncName = Object.values(obj)[1];
207             this.address = Object.values(obj)[0];
208             this.oldFuncType = Object.values(obj)[2];
209             this.newFuncType = Object.values(obj)[3];
210             this.view = Object.values(obj)[6];
211             console.log(this.type + ' ' + this.newFuncName + ' ' + this.
                    newFuncType);
212         }
213         // comment_changed || comment_removed
214         else if ((Object.values(obj)[4] == 'comment_changed') ||
215                  (Object.values(obj)[4] =='comment_remove')) {
216             this.address = Object.values(obj)[0]
217             this.newComment = Object.values(obj)[1];
218             this.oldComment = Object.values(obj)[2];
219             this.functionAddr = Object.values(obj)[3];
220             this.type = Object.values(obj)[4];
221             this.view = Object.values(obj)[5];
222             console.log(this.type + ' ' + this.newComment + ' ' + this.
                    oldComment + ' ' + this.address);
223         }
224         // 'var_type_updated' || 'var_updated' || 'var_name_updated'
225         else if ((Object.values(obj)[3] == 'var_type_updated') ||
226                  (Object.values(obj)[3] == 'var_updated') ||
227                  (Object.values(obj)[3] == 'var_name_updated')) {
228             this.functionAddr = Object.values(obj)[0];
229             this.functionName = Object.values(obj)[1];
230             this.index = Object.values(obj)[2];
231             this.type =  Object.values(obj)[3];
```

```
232             this.newVarName = Object.values(obj)[4];
233             this.oldVarName = Object.values(obj)[5];
234             this.newVarType = Object.values(obj)[6];
235             this.oldVarType = Object.values(obj)[7];
236             this.view = Object.values(obj)[8];
237             console.log(this.type + ' ' + this.functionAddr + ' ' + this
                    .newVarName + ' ' + this.newVarType);
238         }
239         // data_written
240         else if (Object.values(obj)[4] == 'data_written') {
241             this.type = Object.values(obj)[4];
242             this.address = Object.values(obj)[0];
243             this.dataNew = Object.values(obj)[1];
244             this.dataOld = Object.values(obj)[2];
245             this.length = Object.values(obj)[3];
246             this.view = Object.values(obj)[5];
247             console.log(this.type + ' ' + this.address + ' ' + this.
                    dataNew);
248         }
249         // highlight
250         else if (Object.values(obj)[4] == 'highlight') {
251             this.address = Object.values(obj)[0];
252             this.colorNew = Object.values(obj)[1];
253             this.colorOld = Object.values(obj)[2];
254             this.functionAddr = Object.values(obj)[3];
255             this.type = Object.values(obj)[4];
256             this.view = Object.values(obj)[5];
257             console.log(this.type + ' ' + this.address + ' ' + this.
                    colorNew);
258         }
259         // func_removed || func_added
260         else if (Object.values(obj)[2] == 'func_removed' ||
```

```
261                     Object.values(obj)[2] =='func_added') {
262                 this.functionAddr = Object.values(obj)[0];
263                 this.functionName = Object.values(obj)[1];
264                 this.type = Object.values(obj)[2];
265                 this.view = Object.values(obj)[3];
266                 console.log(this.type + ' ' + this.functionAddr + ' ' + this
                        .functionName);
267             }
268             // type_defined || type_undefined
269             else if (Object.values(obj)[1] == 'type_defined' ||
270                     Object.values(obj)[1] == 'type_undefined') {
271                 this.typeName = Object.values(obj)[0];
272                 this.type = Object.values(obj)[1];
273                 this.typeDefined = Object.values(obj)[2];
274                 this.view = Object.values(obj)[3];
275                 console.log(this.type + ' ' + this.typeName + ' ' + this.
                        typeDefined);
276             }
277             // func_type_updated
278             else if (Object.values(obj)[0] == 'bv.file.view') {
279                 this.view = Object.values(obj)[0];
280                 this.oldview = Object.values(obj)[1];
281                 this.oldaddress = Object.values(obj)[2];
282                 this.type = Object.values(obj)[3];
283                 this.address = Object.values(obj)[4];
284                 console.log(this.view + ' ' + this.oldview + ' ' +
285                             this.address + ' ' + this.oldaddress);
286             }
287
288     }
289 }
290
```

```
291 export class FileChange {
292     private graph: ProvenanceGraph;
293     private registry: ActionFunctionRegistry;
294     private tracker: ProvenanceTracker;
295     private traverser: ProvenanceGraphTraverser;
296     private readonly app: FileChangeApp;
297 
298     constructor(
299         graph: ProvenanceGraph,
300         registry: ActionFunctionRegistry,
301         tracker: ProvenanceTracker,
302         traverser: ProvenanceGraphTraverser,
303     ) {
304         this.graph = graph;
305         this.registry = registry;
306         this.tracker = tracker;
307         this.traverser = traverser;
308 
309         this.app = new FileChangeApp();
310 
311         this.registry.register('FuncNameUpdate', this.app.
                FuncNameUpdated, this.app);
312         this.registry.register('FuncTypeUpdate', this.app.funcType, this
                .app);
313         this.registry.register('FuncNameTypeUpdate', this.app.
                funcNameType, this.app);
314         this.registry.register('CommentUpdated', this.app.CommentUpdated
                , this.app);
315         this.registry.register('LocVarUpdate', this.app.LocalVarUpdate,
                this.app);
316         this.registry.register('dataWritten', this.app.dataWritten, this
                .app);
```

```
317         this.registry.register('highlight', this.app.highlight, this.app
                );
318         this.registry.register('defineFunc', this.app.defineFunc, this.
                app);
319         this.registry.register('undefineFunc', this.app.undefineFunc,
                this.app);
320         this.registry.register('addType', this.app.addType, this.app);
321         this.registry.register('removeType', this.app.removeType, this.
                app);
322         this.registry.register('view', this.app.view, this.app);
323     }
324
325     public async makeActionAndApply(
326         reversible: boolean,
327         label: string,
328         doAction: string,
329         doArguments: any[],
330         undoAction?: string,
331         undoArguments?: any[],
332       ): Promise<StateNode> {
333         let method: Action;
334         const intermediate: Action = {
335           do: doAction,
336           doArguments,
337           metadata: {
338             createdBy: 'me',
339             createdOn: 'now',
340             tags: [],
341             userIntent: doAction,
342           },
343         };
344         if (reversible) {
```

```
345             method = {
346             ...intermediate,
347             undo: undoAction,
348             undoArguments,
349           } as ReversibleAction;
350         } else {
351             method = {
352             ...intermediate,
353           } as IrreversibleAction;
354         }
355
356         const node = await this.tracker.applyAction(method);
357         node.label = label;
358         return node;
359     }
360
361     public currentState(): string {
362         return this.app.method;
363     }
364
365     public async setupBasicGraph() {
366         const intermediateNode = await this.makeActionAndApply(
367             true,
368             'View: Graph:PE',
369             'view',
370             ['0x401000', 'Graph:PE'],
371             'view:',
372             ['0x401000', 'Graph:PE'],
373         );
374         await this.makeActionAndApply(true, 'Comm: input_fxn?', '
            CommentUpdated',['0x402db4','Starting here','0x402cc0'], '
            CommentUpdated', ['0x402db4','','0x402cc0'],)
```

```
375         await this.makeActionAndApply(true, 'Func: setup_fxn()?', '
                CommentUpdated',['0x403c90','arg1->???','???','0x403c90'], '
                CommentUpdated',['0x403c90','','0x403c90']);
376         await this.traverser.toStateNode(intermediateNode.id);
377         await this.makeActionAndApply(true, 'Func: setup_fxn()','
                FuncNameUpdate',['0x402cc0', 'setup_fxn()'], 'FuncNameUpdate
                ', ['0x402cc0', 'sub_402cc0']);
378         const testNode = await this.makeActionAndApply(true, 'Func: main
                ()','FuncNameUpdate',['0x403cd0', 'main()'], 'FuncNameUpdate
                ', ['0x403cd0', 'sub_403cd0']);
379         await this.makeActionAndApply(true, 'Comm: password#1', '
                CommentUpdated',['0x40354a','password#1','0x4034d0'], '
                CommentUpdated', ['0x40354a','','0x4034d0']);
380         await this.makeActionAndApply(true, 'Func: pw#1_logic()','
                FuncNameUpdate',['0x4034d0', 'pw#1_logic()'], '
                FuncNameUpdate', ['0x4034d0', 'sub_4034d0']);
381         const testNode4 = await this.makeActionAndApply(true, 'Func:
                xor_decryptor()','FuncNameUpdate',['0x403c90', '
                xor_decryptor()'], 'FuncNameUpdate', ['0x403c90', '
                sub_403c90']);
382         await this.makeActionAndApply(true, 'Comm: password#2', '
                CommentUpdated',['0x4036f0','password#2_logic?','sub_4036f0'
                ], 'CommentUpdated', ['0x4036f0','','0x4034f0']);
383         await this.traverser.toStateNode(testNode4.id);
384         await this.makeActionAndApply(true, 'Var: arg2->size', '
                LocVarUpdate',['0x403c90','int32_t','size','6'], '
                LocVarUpdate',['0x403c90','int32_t','arg2','6']);
385         await this.makeActionAndApply(true, 'Var: arg3->password', '
                LocVarUpdate',['0x403c90','int32_t','password','7'], '
                LocVarUpdate',['0x403c90','int32_t','arg3','7']);
386     }
387 }
```

JavaScript - index.ts

```
1  Copyright 2020 Wayne C. Henry
2
3  Licensed under the Apache License, Version 2.0 (the "License"); you may
       not use this file except in compliance with the License. You may
       obtain a copy of the License at
4
5      http://www.apache.org/licenses/LICENSE-2.0
6
7  Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
       implied.  See the License for the specific language governing
       permissions and limitations under the License.
8
9  import { XmlRpcRequest } from "./mimic";
10 import { FileChange,
11          DataTypes
12 } from './fileChange';
13
14 import {
15   ProvenanceGraph,
16   ProvenanceTracker,
17   ProvenanceGraphTraverser,
18   ActionFunctionRegistry,
19   ProvenanceSlide,
20   ProvenanceSlidedeck,
21   ProvenanceSlidedeckPlayer,
22   serializeProvenanceGraph,
```

```
23    restoreProvenanceGraph,
24    SerializedProvenanceGraph
25  } from '@visualstorytelling/provenance-core';
26
27  import { ProvenanceTreeVisualization } from '@visualstorytelling/
        provenance-tree-visualization';
28  import { SlideDeckVisualization } from '@visualstorytelling/slide-deck-
        visualization';
29  import 'normalize.css';
30  import './style.scss';
31  import '@visualstorytelling/slide-deck-visualization/dist/bundle.css';
32  import * as io from "socket.io-client";
33
34  const visDiv: HTMLDivElement = document.getElementById('vis') as
        HTMLDivElement;
35  const saveDivBtn: HTMLButtonElement = document.getElementById(
36      'Save',) as HTMLButtonElement;
37
38  let graph = new ProvenanceGraph({ name: 'FileChange', version: '1.0.0'
        });
39  const registry = new ActionFunctionRegistry();
40  const tracker = new ProvenanceTracker(registry, graph);
41  const traverser = new ProvenanceGraphTraverser(registry, graph);
42
43  let player: ProvenanceSlidedeckPlayer<ProvenanceSlide>;
44  const playBtn: HTMLButtonElement = document.getElementById(
45    'play',
46  ) as HTMLButtonElement;
47
48  // Setup named pipe with server
49  console.log("Try to logon...");
50  var socket = io.connect('http://localhost:8082');
```

```
51
52 socket.on("connected", function(data: any) {
53     console.log("Connected User?", data.accept);
54 });
55
56 var requestFile = socket.on("fileChanged", async (data: string) => {
57     console.log("typeof data: " + typeof data);
58     if (data && data.length !== 0 ) {
59         console.log(data);
60         let newNode = new DataTypes(data);
61         // Node operations:
62         if (newNode.type == 'comment_changed' || newNode.type == '
               comment_remove' ) {
63             const node = await tracker.applyAction({
64                 do: 'CommentUpdated',
65                 doArguments: [newNode.address, newNode.newComment,
                       newNode.functionAddr],
66                 undo: 'CommentUpdated',
67                 undoArguments: [newNode.address, newNode.oldComment,
                       newNode.functionAddr],
68                 metadata: {
69                   createdBy: 'me',
70                   createdOn: 'now',
71                   tags: [],
72                   userIntent: 'comment',
73                 },
74             }, true);
75             node.label = "Comment: "+ newNode.address;
76         }
77         else if (newNode.type == 'view') {
78             const node = await tracker.applyAction({
79                 do: 'view',
```

```
80                doArguments: [newNode.address, newNode.view],
81                undo: 'view',
82                undoArguments: [newNode.oldaddress, newNode.oldview],
83                metadata: {
84                  createdBy: 'me',
85                  createdOn: 'now',
86                  tags: [],
87                  userIntent: 'view',
88                },
89            }, true);
90            node.label = "view " + newNode.view + " " + newNode.address;
91        }
92        else if (newNode.type == 'func_name_updated') {
93          console.log("Func_name_updated");
94            const node = await tracker.applyAction({
95                do: 'FuncNameUpdate',
96                doArguments: [newNode.address, newNode.newFuncName],
97                undo: 'FuncNameUpdate',
98                undoArguments: [newNode.address, newNode.oldFuncName],
99                metadata: {
100                 createdBy: 'me',
101                 createdOn: 'now',
102                 tags: [],
103                 userIntent: 'func_name_updated',
104               },
105           }, true);
106           node.label = "FuncName: " + newNode.newFuncName;
107       }
108       else if (newNode.type == 'func_name_type_updated') {
109         console.log("Func_name_type_updated");
110           const node = await tracker.applyAction({
111               do: 'FuncNameTypeUpdate',
```

```
112                 doArguments: [newNode.address, newNode.newFuncName,
                        newNode.newFuncType],
113                 undo: 'FuncNameTypeUpdate',
114                 undoArguments: [newNode.address, newNode.oldFuncName,
                        newNode.oldFuncType],
115                 metadata: {
116                   createdBy: 'me',
117                   createdOn: 'now',
118                   tags: [],
119                   userIntent: 'func_name_type_updated',
120                 },
121             }, true);
122             node.label = "FuncNameType: " + newNode.newFuncName + " " +
                    newNode.newFuncType;
123         }
124         else if (newNode.type == 'func_type_updated') {
125           console.log("Func_type_updated");
126             const node = await tracker.applyAction({
127                 do: 'FuncTypeUpdate',
128                 doArguments: [newNode.address, newNode.newFuncType],
129                 undo: 'FuncTypeUpdate',
130                 undoArguments: [newNode.address, newNode.oldFuncType],
131                 metadata: {
132                   createdBy: 'me',
133                   createdOn: 'now',
134                   tags: [],
135                   userIntent: 'func_type_updated',
136                 },
137             }, true);
138             node.label = "FuncType: " + newNode.newFuncType;
139         }
140         else if (newNode.type == 'func_removed') {
```

```
141             const node = await tracker.applyAction({
142                 do: 'undefineFunc',
143                 doArguments: [newNode.functionAddr],
144                 undo: 'defineFunc',
145                 undoArguments: [newNode.functionName, newNode.functionAddr
                        ],
146                 metadata: {
147                   createdBy: 'me',
148                   createdOn: 'now',
149                   tags: [],
150                   userIntent: 'func_removed',
151                 },
152             }, true);
153             node.label = "UndefineFunc: "+newNode.functionAddr;
154           }
155           else if (newNode.type == 'func_added') {
156             const node = await tracker.applyAction({
157                 do: 'defineFunc',
158                 doArguments: [newNode.functionName, newNode.functionAddr],
159                 undo: 'undefineFunc',
160                 undoArguments: [newNode.functionAddr],
161                 metadata: {
162                   createdBy: 'me',
163                   createdOn: 'now',
164                   tags: [],
165                   userIntent: 'func_added',
166                 },
167             }, true);
168             node.label = "DefineFunc: "+newNode.functionName;
169           }
170           else if (newNode.type == 'type_defined'){
171             const node = await tracker.applyAction({
```

```
172             do: 'addType',
173             doArguments: [newNode.typeName, newNode.typeDefined],
174             undo: 'removeType',
175             undoArguments: [newNode.typeName],
176             metadata: {
177               createdBy: 'me',
178               createdOn: 'now',
179               tags: [],
180               userIntent: 'type_defined',
181             },
182         }, true);
183         node.label = "type_defined "+newNode.typeName;
184         }
185         else if (newNode.type == 'type_undefined'){
186           const node = await tracker.applyAction({
187             do: 'removeType',
188             doArguments: [newNode.typeName],
189             undo: 'addType',
190             undoArguments: [newNode.typeName, newNode.typeDefined],
191             metadata: {
192               createdBy: 'me',
193               createdOn: 'now',
194               tags: [],
195               userIntent: 'type_undefined',
196             },
197         }, true);
198         node.label = "type_undefined "+newNode.typeName;
199         }
200         else if (newNode.type == 'data_written'){
201           const node = await tracker.applyAction({
202             do: 'dataWritten',
203             doArguments: [newNode.address, newNode.dataNew],
```

```
204             undo: 'dataWritten',
205             undoArguments: [newNode.address, newNode.dataOld],
206             metadata: {
207               createdBy: 'me',
208               createdOn: 'now',
209               tags: [],
210               userIntent: 'data_written',
211             },
212           }, true);
213           node.label = "DataWritten: "+newNode.dataNew;
214           }
215           else if (newNode.type == 'highlight'){
216             const node = await tracker.applyAction({
217               do: 'highlight',
218               doArguments: [newNode.address, newNode.colorNew],
219               undo: 'highlight',
220               undoArguments: [newNode.address, newNode.colorOld],
221               metadata: {
222                 createdBy: 'me',
223                 createdOn: 'now',
224                 tags: [],
225                 userIntent: 'highlight',
226               },
227             }, true);
228             node.label = "Highlight: "+newNode.address;
229           }
230           else if ((newNode.type == 'var_type_updated') || (newNode.type
                == 'var_updated') || (newNode.type == 'var_name_updated')){
231             const node = await tracker.applyAction({
232               do: 'LocVarUpdate',
233               doArguments: [newNode.functionAddr, newNode.newVarType,
                    newNode.newVarName, newNode.index],
```

```
234             undo: 'LocVarUpdate',
235             undoArguments: [newNode.functionAddr, newNode.oldVarType,
                    newNode.oldVarName, newNode.index],
236             metadata: {
237               createdBy: 'me',
238               createdOn: 'now',
239               tags: [],
240               userIntent: 'varNameType_updated',
241             },
242           }, true);
243           node.label = "LocVarUpdate: " + newNode.newVarName + " " +
                  newNode.newVarType;
244         }
245     }
246 });
247
248 function download(content:any, fileName:any, contentType:any) {
249   var a = document.createElement("a");
250   var file = new Blob([content], {type: contentType});
251   a.href = URL.createObjectURL(file);
252   a.download = fileName;
253   a.click();
254 }
255
256 saveDivBtn.addEventListener('click', async () => {
257   const serializableGraph = serializeProvenanceGraph(graph);
258   const actionsDiv = JSON.stringify(serializableGraph);
259   download(actionsDiv, 'json.txt', 'application/json');
260 });
261
262 document.getElementById("file-input").addEventListener("change", async (
      e) => {
```

```
263   var file = (<HTMLInputElement>e.target).files[0];
264   var reader = new FileReader();
265   reader.onload = file => {
266       var contents: any = file.target;
267       this.text = contents.result;
268       console.log(this.text.toString());
269
270       var graphLoad:SerializedProvenanceGraph = JSON.parse(this.text);
271       graph = restoreProvenanceGraph(graphLoad);
272   };
273     reader.readAsText(file);
274 }, false);
275
276 const fileChange = new FileChange(graph, registry, tracker, traverser);
277
278 let provenanceTreeVisualization: ProvenanceTreeVisualization;
279
280 fileChange.setupBasicGraph().then(() => {
281   provenanceTreeVisualization = new ProvenanceTreeVisualization(
282     traverser,
283     visDiv,
284   );
285
286   const slideDeck = new ProvenanceSlidedeck(
287     { name: 'fileChange', version: '1.0.0' },
288     traverser,
289   );
290
291   const provenanceSlidedeckVis = new SlideDeckVisualization(
292     slideDeck,
293     document.getElementById('slidedeck_root') as HTMLDivElement,
294   );
```

```
295
296   player = new ProvenanceSlidedeckPlayer(
297     slideDeck.slides as ProvenanceSlide[],
298     (slide) => (slideDeck.selectedSlide = slide),
299   );
300
301   if (playBtn){
302     playBtn.addEventListener('click', () => {
303       player.setSlideIndex(slideDeck.slides.indexOf(slideDeck.
              selectedSlide));
304       player.play();
305     });
306   }
307 });
```

HTML - index.html

```
1 Copyright 2020 Wayne C. Henry
2
3 Licensed under the Apache License, Version 2.0 (the "License"); you may
    not use this file except in compliance with the License. You may
    obtain a copy of the License at
4
5    http://www.apache.org/licenses/LICENSE-2.0
6
7 Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    implied.  See the License for the specific language governing
    permissions and limitations under the License.
8
9 <!DOCTYPE html>
```

```
10 <html>
11   <head>
12     <title>Reverse Engineering Provenance</title>
13     <script src="https://code.jquery.com/jquery-3.3.1.min.js" type="text
           /javascript"></script>
14     <link rel="icon" href="/favicon.ico">
15     <meta http-equiv="Content-Type" content="text/html; charset=utf-8"
           />
16
17   </script>
18   </head>
19   <body>
20
21     <button id="Save">Save</button>
22     <input type="file" id="file-input">
23     <div id="vis"></div>
24     <div id="slidedeck_root"></div>
25
26   </body>
27
28   <script>
29     (function () {
30       var isChrome = /Chrome/.test(navigator.userAgent) && /Google Inc/.
             test(navigator.vendor);
31       if (!isChrome) {
32         alert('This application was designed to be used in Google Chrome
               .\nRunning the application in other browsers might result in
                performance issues or other misbehavior.');
33       }
34     })();
35   </script>
36
```

```
37 </html>
```

Minimal JS server - fileupdate.js

```
1
2  Copyright 2020 Wayne C. Henry
3
4  Licensed under the Apache License, Version 2.0 (the "License"); you may
       not use this file except in compliance with the License. You may
       obtain a copy of the License at
5
6     http://www.apache.org/licenses/LICENSE-2.0
7
8  Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
       implied.  See the License for the specific language governing
       permissions and limitations under the License.
9
10 var PORT = 8082;
11 var io = require("socket.io").listen(PORT);
12 var fs = require("fs");
13 Tail = require('tail').Tail; // https://github.com/lucagrulla/node-tail
14
15 console.log("dir", __dirname);
16 const logfile = 'jsondata.json';
17 var temp = '';
18
19 try{
20 io.sockets.on('connection', function(socket) {
21     console.log("Connected!");
22     socket.emit('connected', { accept: true});
```

```
23
24      console.log("Trying to send the content to a client...");
25      console.log("dir", __dirname);
26
27      tail = new Tail(logfile);
28      tail.on("line", function(data) {
29          if (data != temp) {          // Watch for repeated entries
30              console.log("Content:", data);
31              socket.emit("fileChanged", data );
32          }
33          temp = data;
34      })
35      tail.on("error", function(error) {
36          console.log('ERROR: ', error);
37      })
38  });
39  } catch(error)
40  {
41      console.log('Error: ', error);
42  }
```

Appendix E. Ethics Approval - Evaluation

This appendix provides the data collection materials for the SensorRE evaluation.

Reverse Engineering Provenance Prototype Evaluation

Demographic Survey:

1. What is your age?
    a. 18-24
    b. 25-34
    c. 35-44
    d. 45-54
    e. 55 or older

2. What is the highest level of education you have completed?
    a. High School degree
    b. Associate's degree
    c. Bachelor's degree
    d. Master's degree
    e. Doctorate degree

3. What is your field of study?

4. How much experience do you have with software reverse engineering?
    a. No experience
    b. Less than one year
    c. Between one to three years
    d. Greater than three years

5. What educational experience do you have in software reverse engineering? (Circle all that apply)
    a. N/A
    b. Certificates achieved
    c. Courses completed
    d. Degrees

    Explain your response:

Reverse Engineering Provenance Prototype Evaluation

6. How would you subjectively rate your skill level in reverse engineering (x86)?
    a. No Knowledge
    b. Novice
    c. Intermediate
    d. Advanced

Reverse Engineering Provenance Prototype Evaluation

Case Study Tasks

Instructions: **Please *r*ead each scenario description carefully and complete the tasks by answering the questions or using the provenance tool. If you have any questions or issues completing the tasks, please ask the researcher.**

Scenario 1: Auditing a collaborator's analysis

The first scenario presents the SensorRE provenance graph in a state where a collaborator has already made several relevant discoveries about the particular binary. You are asked to answer a set of statements based on the existing provenance graph. The statements are related to the overall function of the software executable. You are encouraged to inspect the graph and associated binary using Binary Ninja but **not** to extend or modify them.

What functions has the collaborator examined according to the provenance graph?

What system calls are made in the previously examined functions of the binary?

Scenario 2: Extending the collaborator's analysis

In this scenario, extend the previous analysis from Scenario 1. First, locate the function which calls the Windows encryption routine. Rename the calling function to be more representative of the activity within (change *sub_XXXXX*). Choose a name which would be used to communicate the behavior of the function back to the original collaborator.

Using annotations in the provenance graph, record (1) what you expected to see from the data (hypotheses), and (2) what you actually saw and thought (finding).

Reverse Engineering Provenance Prototype Evaluation

Scenario 3: Present results using the Storyboard view

In this task, use the Storyboard view to present the active branch of the provenance graph. Select the applicable steps in the graph and add them to the storyboard in order and modify the presentation timing. Show the researcher the finished product.

Reverse Engineering Provenance Prototype Evaluation

Post-Task Survey

Instructions: **Reflecting on the tasks you were asked to complete, please answer each statement with a rating from Strongly Disagree to Strongly Agree.**

	Statement	Strongly Disagree	Disagree	Slightly Disagree	Neutral	Slightly Agree	Agree	Strongly Agree
1	This tool was easy to use.							
2	This tool was easy to learn.							
3	This tool could improve the recall of findings, strategies and methods.							
4	This tool could improve validating findings from others.							
5	This tool could improve communicating findings among teams of reverse engineers.							

(Continued)

Reverse Engineering Provenance Prototype Evaluation

1. What was particularly useful about the tool?

2. Do you think the tree diagram is a useful representation of the provenance graph?

3. Are there other artifacts you would like to add to the provenance graph?

4. Are there any features missing?

5. Are there any limitations of the system which would hinder its adoption?

Bibliography

1. P. H. Nguyen, "Visualization of analytic provenance for sensemaking," Ph.D. dissertation, Middlesex University, 2017.

2. P. Pirolli and S. Card, "The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis," in *Proceedings of the International Conference on Intelligence Analysis*, vol. 5, 2005, pp. 2–4.

3. A. R. Bryant, "Understanding how reverse engineers make sense of programs from assembly language representations," PhD Dissertation, Air Force Institute of Technology, 2012.

4. D. Gotz and M. X. Zhou, "Characterizing users' visual analytic activity for insight provenance," *Information Visualization*, vol. 8, no. 1, pp. 42–55, 2009.

5. Y. B. Shrinivasan and J. J. van Wijk, "Supporting the analytical reasoning process in information visualization," *Proceeding of the Conference on Human factors in Computing Systems*, p. 1237, 2008.

6. H. Stitz, S. Gratzl, H. Piringer, T. Zichner, and M. Streit, "KnowledgePearls: Provenance-Based Visualization Retrieval," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 120–130, 2019.

7. S. Gratzl, A. Lex, N. Gehlenborg, N. Cosgrove, and M. Streit, "From visual exploration to storytelling and back again," in *Computer Graphics Forum*, vol. 35, no. 3. Wiley Online Library, 2016, pp. 491–500.

8. J. W. Creswell and V. L. P. Clark, *Designing and conducting mixed methods research*. Thousand Oaks, CA: Sage Publications, 2017.

9. Binary Ninja Python API Documentation. 2019. [Online]. Available: https://api.binary.ninja/

10. K. Xu, S. Attfield, T. Jankun-Kelly, A. Wheat, P. H. Nguyen, and N. Selvaraj, "Analytic provenance for sensemaking: A research agenda," *IEEE Computer Graphics and Applications*, vol. 35, no. 3, pp. 56–64, 2015.

11. DARPA - Computers and Humans Exploring Software Security. 2018. [Online]. Available: https://www.darpa.mil/program/computers-and-humans-exploring-software-security

12. J. Cowley, "Job analysis results for malicious-code reverse engineers: A case study," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2014-TR-002, 2014.

13. M. K. Tennor, "Reverse engineering cognition," MITRE, Tech. Rep. 15-2630, 2015.

14. A. Telea, H. Byelas, and L. Voinea, "A framework for reverse engineering large C++ code bases," *Electronic Notes in Theoretical Computer Science*, vol. 233, pp. 143–159, 2009.

15. V. Fix, S. Wiedenbeck, and J. Scholtz, "Mental representations of programs by novices and experts," in *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 1993, pp. 74–79.

16. H. N. Huang, E. Verbeek, D. German, M.-A. Storey, and M. Salois, "Atlantis: Improving the Analysis and Visualization of Large Assembly Execution Traces," in *International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2017, pp. 623–627.

17. M. Polino, A. Scorti, F. Maggi, and S. Zanero, "Jackdaw: Towards automatic reverse engineering of large datasets of binaries," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2015, pp. 121–143.

18. D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena, "Bitblaze: A new approach to computer security via binary analysis," in *International Conference on Information Systems Security*. Springer, 2008, pp. 1–25.

19. J. Baldwin, P. Sinha, M. Salois, and Y. Coady, "Progressive user interfaces for regressive analysis: Making tracks with large, low-level systems," in *Proceedings of the Australasian User Interface Conference*. Australian Computer Society, Inc., 2011, pp. 47–56.

20. D. Snowden, "Multi-ontology sense making: a new simplicity in decision making," *Journal of Innovation in Health Informatics*, vol. 13, no. 1, pp. 45–53, 2005.

21. D. M. Russell, R. Jeffries, and L. Irani, "Sensemaking for the rest of us," in *Sensemaking Workshop at CHI*, 2008.

22. S. K. Card, J. D. Mackinlay, and B. Shneiderman, *Readings in Information Visualization: Using Vision to Think*. San Francisco: Morgan Kaufmann Publishers, 1999.

23. H. A. Müller, J. H. Jahnke, D. B. Smith, M.-A. Storey, S. R. Tilley, and K. Wong, "Reverse engineering: A roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, 2000, pp. 47–60.

24. D. Brumley, I. Jager, T. Avgerinos, and E. J. Schwartz, "BAP: A binary analysis platform," in *International Conference on Computer Aided Verification*. Springer, 2011, pp. 463–469.

25. J. Kinable and O. Kostakis, "Malware classification based on call graph clustering," *Journal in Computer Virology*, vol. 7, no. 4, pp. 233–245, 2011.

26. C. S. Collberg and C. Thomborson, "Watermarking, tamper-proofing, and obfuscation-tools for software protection," *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 735–746, 2002.

27. D. Votipka, S. Rabin, K. Micinski, J. S. Foster, and M. L. Mazurek, "An observational investigation of reverse engineers' process and mental models," in *Extended Abstracts of the Conference on Human Factors in Computing Systems*, 2019, pp. 1–6.

28. F. Nielson, H. R. Nielson, and C. Hankin, *Principles of Program Analysis*. Berlin, Heidelberg: Springer, 2015.

29. G. Conti and E. Dean, "Visual forensic analysis and reverse engineering of binary data," in *Proceedings of Black Hat USA*, 2008.

30. A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Proceedings of the Computer Security Applications Conference*. IEEE, 2007, pp. 421–430.

31. E. Eilam, *Reversing: Secrets of Reverse Engineering*. Indianapolis, IN: Wiley Publishing Inc., 2011.

32. A. S. Tanenbaum and A. S. Woodhull, *Operating systems: design and implementation*. Englewood Cliffs, NJ: Prentice-Hall Professional, 1987, vol. 2.

33. J. Trümper, "Visualization techniques for the analysis of software behavior and related structures," PhD Dissertation, University of Potsdam, 2014.

34. B. Blunden, *The Rootkit Arsenal: Escape and evasion in the dark corners of the system*. Burlington, MA: Jones & Bartlett Publishers, 2012.

35. E. Skoudis and L. Zeltser, *Malware: Fighting malicious code*. Englewood Cliffs, NJ: Prentice Hall Professional, 2004.

36. G. Hoglund and J. Butler, *Rootkits: Subverting the Windows Kernel*. Addison-Wesley Professional, 2006.

37. K. Yakdan, S. Dechand, E. Gerhards-Padilla, and M. Smith, "Helping johnny to analyze malware," *IEEE Security & Privacy (Oakland), San Jose, CA*, 2016.

38. T. Munzner, "A nested process model for visualization design and validation," *Transactions on Visualization & Computer Graphics*, no. 6, pp. 921–928, 2009.

39. Binary Ninja. 2020. [Online]. Available: https://binary.ninja/

40. IDA Pro Disassembler. 2019. [Online]. Available: https://hex-rays.com/

41. Ghidra software reverse engineering framework. 2019. [Online]. Available: https://github.com/NationalSecurityAgency/ghidra

42. R. Koschke, "Software visualization in software maintenance, reverse engineering, and re-engineering: A research survey," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 15, no. 2, pp. 87–109, 2003.

43. S. Bassil and R. K. Keller, "Software visualization tools: Survey and analysis," in *International Workshop on Program Comprehension*. IEEE, 2001, pp. 7–17.

44. S. E. Sim, C. L. Clarke, and R. C. Holt, "Archetypal source code searches: A survey of software developers and maintainers," in *Proceedings of the International Workshop on Program Comprehension*. IEEE, 1998, pp. 180–187.

45. H. M. Kienle and H. A. Müller, "Rigi: An environment for software reverse engineering, exploration, visualization, and redocumentation," *Science of Computer Programming*, vol. 75, no. 4, pp. 247–263, 2010.

46. A. Telea, A. Maccari, and C. Riva, "An open visualization toolkit for reverse architecting," in *Proceedings of the International Workshop on Program Comprehension.* IEEE, 2002, pp. 3–10.

47. M.-A. Storey, C. Best, and J. Michand, "SHriMP Views: An Interactive Environment for Exploring Java Programs," in *Proceedings of the International Workshop on Program Comprehension.* IEEE, 2001, pp. 111–112.

48. D. L. Moise, K. Wong, and D. Sun, "Integrating a reverse engineering tool with microsoft visual studio. net," in *Proceedings of the Eighth European Conference on Software Maintenance and Reengineering.* IEEE, 2004, pp. 85–92.

49. D. W. Pucsek, "Visualization and analysis of assembly code in an integrated comprehension environment," Master's thesis, University of Victoria, 2013.

50. C. Bennett, D. Myers, M.-A. Storey, D. M. German, D. Ouellet, M. Salois, and P. Charland, "A survey and evaluation of tool features for understanding reverse-engineered sequence diagrams," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 20, no. 4, pp. 291–315, 2008.

51. Eclipse Open Source Rich Client Platform. 2018. [Online]. Available: https://www.eclipse.org/community/rcpos.php

52. B. Cleary, M.-A. Storey, L. Chan, M. Salois, and F. Painchaud, "Atlantis - Assembly Trace Analysis Environment," in *Working Conference on Reverse Engineering.* IEEE, 2012, pp. 505–506.

53. M. Wagner, A. Rind, N. Thür, and W. Aigner, "A Knowledge-assisted Visual Malware Analysis System: Design, Validation, and Reflection of KAMAS," *Computers & Security*, vol. 67, pp. 1–15, 2017.

54. D. A. Quist and L. M. Liebrock, "Visualizing compiled executables for malware analysis," in *Proceedings of the International Workshop on Visualization for Cyber Security.* IEEE, 2009, pp. 27–32.

55. D. Holten, "Hierarchical edge bundles: Visualization of Adjacency Relations in Hierarchical Data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 741–748, 2006.

56. JHotDraw. 2018. [Online]. Available: http://www.jhotdraw.org/

57. OpenGL - The Industry Standard for High Performance Graphics. 2018. [Online]. Available: https://opengl.org/

58. C. Treude, F. Figueira Filho, M.-A. Storey, and M. Salois, "An exploratory study of software reverse engineering in a security context," in *Proceedings of the Working Conference on Reverse Engineering (WCRE)*, 2011, pp. 184–188.

59. J. Baldwin, A. Teh, E. Baniassad, D. Van Rooy, and Y. Coady, "Requirements for tools for comprehending highly specialized assembly language code and how to elicit these requirements," *Requirements Engineering*, vol. 21, no. 1, pp. 131–159, 2016.

60. H. M. Kienle and H. A. Müller, "The tools perspective on software reverse engineering: requirements, construction, and evaluation," in *Advances in Computers.* Elsevier, 2010, vol. 79, pp. 189–290.

61. H. M. Kienle and H. A. Muller, "Requirements of software visualization tools: A literature survey," in *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*. IEEE, 2007, pp. 2–9.

62. M. Ceccato, P. Tonella, C. Basile, B. Coppens, B. De Sutter, P. Falcarin, and M. Torchiano, "How professional hackers understand protected code while performing attack tasks," in *Proceedings of the 25th International Conference on Program Comprehension*. IEEE, 2017, pp. 154–164.

63. T. C. Summers, "How hackers think: A mixed method study of mental models and cognitive patterns of high-tech wizards," PhD Dissertation, Case Western Reserve University, 2015.

64. G. Chin Jr, O. A. Kuchar, and K. E. Wolf, "Exploring the analytical processes of intelligence analysts," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2009, pp. 11–20.

65. B. Dervin, *An Overview of Sense-making Research: Concepts, Methods, and Results to Date*. Dervin, Brenda, 1983.

66. K. E. Weick, *Sensemaking in organizations*. Sage, 1995, vol. 3.

67. D. M. Russell, M. J. Stefik, P. Pirolli, and S. K. Card, "The cost structure of sensemaking," in *Proceedings of the Conference on Human factors in Computing Systems*. ACM, 1993, pp. 269–276.

68. G. Klein, J. K. Phillips, E. L. Rall, and D. A. Peluso, "A data-frame theory of sensemaking," in *Proceedings of the International Conference on Naturalistic Decision Making*. New York, NY, USA: Lawrence Erlbaum, 2007, pp. 113–155.

69. A. Telea, O. Ersoy, and L. Voinea, "Visual analytics in software maintenance: Challenges and opportunities," in *Proceedings of EuroVAST, EuroVis*, 2010, pp. 65–70.

70. W. S. Humphrey, *Managing the software process*. Addison-Wesley Longman Publishing Co., Inc., 1989.

71. L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers *et al.*, "The open provenance model core specification (v1.1)," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 743–756, 2011.

72. E. D. Ragan, A. Endert, J. Sanyal, and J. Chen, "Characterizing provenance in visualization and data analysis: an organizational framework of provenance types and purposes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 1, pp. 31–40, 2016.

73. P. H. Nguyen, K. Xu, A. Wheat, B. W. Wong, S. Attfield, and B. Fields, "Sensepath: Understanding the sensemaking process through analytic provenance," *IEEE transactions on visualization and computer graphics*, vol. 22, no. 1, pp. 41–50, 2015.

74. C. North, R. Chang, A. Endert, W. Dou, R. May, B. Pike, and G. Fink, "Analytic provenance: process+ interaction+ insight," in *Extended Abstracts on the Conference for Human Factors in Computing Systems*, 2011, pp. 33–36.

75. K. A. Cook and J. J. Thomas, *Illuminating the path: The research and development agenda for visual analytics*. Pacific Northwest National Lab, Richland, WA, 2005.

76. L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, C. E. Scheidegger, C. T. Silva, and H. T. Vo, "Vistrails: Enabling interactive multiple-view visualizations," in *IEEE Visualization*. IEEE, 2005, pp. 135–142.

77. C. Dunne, N. Henry Riche, B. Lee, R. Metoyer, and G. Robertson, "Graphtrail: Analyzing large multivariate, heterogeneous networks while supporting exploration history," in *Proceedings of the Conference on Human Factors in Computing Systems*, 2012, pp. 1663–1672.

78. N. Kadivar, V. Chen, D. Dunsmuir, E. Lee, C. Qian, J. Dill, C. Shaw, and R. Woodbury, "Capturing and supporting the analysis process," in *IEEE Symposium on Visual Analytics Science and Technology*, 2009, pp. 131–138.

79. W. A. Pike, J. Stasko, R. Chang, and T. A. O'Connell, "The science of interaction," *Information Visualization*, vol. 8, no. 4, pp. 263–274, 2009.

80. T. Blascheck, M. John, K. Kurzhals, S. Koch, and T. Ertl, "VA2: A visual analytics approach for evaluating visual analytics applications," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 1, pp. 61–70, 2015.

81. T. Jankun-Kelly, K. L. Ma, and M. Gertz, "A model for the visualization exploration process," in *Proceedings of the Conference on Visualization*. IEEE, 2002, pp. 323–330.

82. Y. Kim, K. Wongsuphasawat, J. Hullman, and J. Heer, "Graphscape: A model for automated reasoning about visualization similarity and sequencing," in *Proceedings of the Conference on Human Factors in Computing Systems*, 2017, pp. 2628–2638.

83. J. Heer, J. Mackinlay, C. Stolte, and M. Agrawala, "Graphical histories for visualization: Supporting analysis, communication, and evaluation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1189–1196, 2008.

84. K. Brodlie, A. Poon, H. Wright, L. Brankin, G. Banecki, and A. Gay, "Grasparc: A problem solving environment integrating computation and visualization," in *Proceedings of the Conference on Visualization*. IEEE Computer Society, 1993, pp. 102–109.

85. W. Javed and N. Elmqvist, "Explates: Spatializing interactive analysis to scaffold visual exploration," in *Computer Graphics Forum*, vol. 32, no. 3pt4. Wiley Online Library, 2013, pp. 441–450.

86. M. C. Chuah and S. F. Roth, "Visualizing common ground," in *Proceedings of the International Conference on Information Visualization*. IEEE, 2003, pp. 365–372.

87. P. Isenberg and D. Fisher, "Collaborative brushing and linking for co-located visual analytics of document collections," in *Computer Graphics Forum*, vol. 28, no. 3. Wiley Online Library, 2009, pp. 1031–1038.

88. J. Heer, F. B. Viégas, and M. Wattenberg, "Voyagers and voyeurs: Supporting asynchronous collaborative information visualization," in *Proceedings of the Conference on Human Factors in Computing Systems*. ACM, 2007, pp. 1029–1038.

89. W. Willett, J. Heer, J. Hellerstein, and M. Agrawala, "Commentspace: structured support for collaborative visual analysis," in *Proceedings of the Conference on Human Factors in Computing Systems*. ACM, 2011, pp. 3131–3140.

90. J. Lu, Z. Wen, S. Pan, and J. Lai, "Analytic trails: supporting provenance, collaboration, and reuse for visual data analysis by business users," in *Conference on Human-Computer Interaction*. Springer, 2011, pp. 256–273.

91. F. B. Viegas, M. Wattenberg, F. Van Ham, J. Kriss, and M. McKeon, "Manyeyes: A site for visualization at internet scale," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1121–1128, 2007.

92. M. Wattenberg and J. Kriss, "Designing for social data analysis," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 4, pp. 549–557, 2006.

93. W. Willett, J. Heer, and M. Agrawala, "Scented widgets: Improving navigation cues with embedded visualizations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1129–1136, 2007.

94. Y. Chen, J. Alsakran, S. Barlowe, J. Yang, and Y. Zhao, "Supporting effective common ground construction in asynchronous collaborative visual analytics," in *IEEE Conference on Visual Analytics Science and Technology (VAST)*. IEEE, 2011, pp. 101–110.

95. R. Walker, A. Slingsby, J. Dykes, K. Xu, J. Wood, P. H. Nguyen, D. Stephens, B. W. Wong, and Y. Zheng, "An extensible framework for provenance in human terrain visual analytics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 12, pp. 2139–2148, 2013.

96. P. H. Nguyen, K. Xu, R. Walker, and B. W. Wong, "Schemaline: Timeline visualization for sensemaking," in *International Conference on Information Visualisation*. IEEE, 2014, pp. 225–233.

97. R. Chang, C. Ziemkiewicz, T. M. Green, and W. Ribarsky, "Defining insight for visual analytics," *IEEE Computer Graphics and Applications*, vol. 29, pp. 14–17, 2009.

98. C. North, "Toward measuring visualization insight," *IEEE Computer Graphics and Applications*, vol. 26, no. 3, pp. 6–9, 2006.

99. J. S. Yi, Y. ah Kang, J. T. Stasko, J. A. Jacko *et al.*, "Toward a deeper understanding of the role of interaction in information visualization," *IEEE Transactions on Visualization & Computer Graphics*, no. 6, 2007.

100. S. Carpendale, "Evaluating information visualizations," in *Information Visualization*. Springer, 2008, pp. 19–45.

101. C. Plaisant, G. G. Grinstein, and J. Scholtz, "Visual-analytics evaluation," *IEEE Computer Graphics and Applications*, vol. 29, no. 3, pp. 16–17, 2009.

102. Crossfilter: Fast multidimensional filtering for coordinated views. 2019. [Online]. Available: https://square.github.io/crossfilter/

103. D3.js. 2018. [Online]. Available: https://d3js.org/

104. dc.js: Dimensional charting javascript library. 2019. [Online]. Available: https://dc-js.github.io/dc.js/

105. Plotly Technologies Inc. Collaborative data science. 2019. [Online]. Available: https://plot.ly/

106. Vega: A visualization grammar. 2019. [Online]. Available: https://vega.github.io/vega/

107. Processing. 2019. [Online]. Available: https://processing.org/

108. Protovis. 2019. [Online]. Available: https://mbostock.github.io/protovis/

109. Visual storytelling. 2019. [Online]. Available: https://github.com/VisualStorytelling

110. H. F. Hofmann and F. Lehner, "Requirements engineering as a success factor in software projects," *IEEE Software*, no. 4, pp. 58–66, 2001.

111. MaxQDA: Qualitative Data Analysis Software. 2018. [Online]. Available: https://www.maxqda.com/

112. J. Heer and M. Agrawala, "Design considerations for collaborative visual analytics," *Information visualization*, vol. 7, no. 1, pp. 49–62, 2008.

113. N. Chinchor and W. A. Pike, "The science of analytic reporting," *Information Visualization*, vol. 8, no. 4, pp. 286–293, 2009.

114. XMLRPC server and client modules. 2019. [Online]. Available: https://docs.python.org/3/library/xmlrpc.html

115. S. Verhoeven, T. Klaver, H. Stitz, M. van Meersbergen, and P. Pushpanjali, "Visual storytelling provenance core," Mar. 2019.

116. K. Madanagopal, E. D. Ragan, and P. Benjamin, "Analytic provenance in practice: The role of provenance in real-world visualization and data analysis environments," *IEEE Computer Graphics and Applications*, 2019.

117. J. Heer and B. Shneiderman, "Interactive dynamics for visual analysis," *Queue*, vol. 10, no. 2, p. 30, 2012.

118. A. Rind, W. Aigner, M. Wagner, S. Miksch, and T. Lammarsch, "User tasks for evaluation: Untangling the terminology throughout visualization design and development," in *Proceedings of the Workshop on Beyond Time and Errors: Novel Evaluation Methods for Visualization*, 2014, pp. 9–15.

119. O. ElTayeby and W. Dou, "A survey on interaction log analysis for evaluating exploratory visualizations," in *Proceedings of the Workshop on Beyond Time and Errors: Novel Evaluation Methods for Visualization*, 2016, pp. 62–69.

120. J. R. Lewis, "IBM Computer Usability Satisfaction Questionnaires: Psychometric evaluation and instructions for use," *International Journal of Human-Computer Interaction*, vol. 7, no. 1, pp. 57–78, 1995.

121. J. Stasko, C. Görg, and Z. Liu, "Jigsaw: supporting investigative analysis through interactive visualization," *Information Visualization*, vol. 7, no. 2, pp. 118–132, 2008.

122. M.-A. Storey, K. Wong, P. Fong, D. Hooper, K. Hopkins, and H. A. Muller, "On designing an experiment to evaluate a reverse engineering tool," in *Proceedings of the Working Conference on Reverse Engineering*. IEEE, 1996, pp. 31–40.

123. Y.-a. Kang, C. Gorg, and J. Stasko, "How can visual analytics assist investigative analysis? Design implications from an evaluation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 5, pp. 570–583, 2011.

124. B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *Proceedings of the Symposium on Visual Languages*. IEEE, 1996, pp. 336–343.

125. T. Munzner, "Process and pitfalls in writing information visualization research papers," in *Information Visualization*. Springer, 2008, pp. 134–153.

126. M. Di Penta, R. K. Stirewalt, and E. Kraemer, "Designing your next empirical study on program comprehension," in *Proceedings of the 15th International Conference on Program Comprehension*. IEEE, 2007, pp. 281–285.

127. R. H. Franke and J. D. Kaul, "The hawthorne experiments: First statistical interpretation," *American Sociological Review*, pp. 623–643, 1978.

128. I. Zayour and T. C. Lethbridge, "Adoption of reverse engineering tools: a cognitive perspective and methodology," in *Proceedings of the Working Conference on Reverse Engineering (WCRE)*, 2001, pp. 245–255.

129. E. D. Keim, J. Kohlhammer, and G. Ellis, "Mastering the information age: Solving problems with visual analytics," *Eurographics Association*, vol. 2, 2010.

130. D. Reniers, L. Voinea, O. Ersoy, and A. Telea, "The solid* toolset for software visual analytics of program structure and metrics comprehension: From research prototype to product," *Science of Computer Programming*, vol. 79, pp. 224–240, 2014.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
07–21–2020	Dissertation	Sept 2017 — July 2020

4. TITLE AND SUBTITLE

ANALYTIC PROVENANCE FOR
SOFTWARE REVERSE ENGINEERS

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Major Wayne C. Henry

5d. PROJECT NUMBER

HR0011940828

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-DS-20-S-010

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory Information Directorate (AFRL/RI)
525 Brooks Rd
Rome Lab AFB, NY 13441
DSN 587-4517
Email: amanda.ozanam@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Reverse engineering is a time-consuming process essential to software-security tasks such as malware analysis and vulnerability discovery. During the process, an engineer will follow multiple leads to determine how the software functions. The combination of time and possible explanations makes it difficult for the engineers to maintain a context of their findings within the overall task. Analytic provenance tools have demonstrated value in similarly complex fields that require open-ended exploration and hypothesis vetting. However, they have not been explored in the reverse engineering domain.

This dissertation presents SensorRE, the first analytic provenance tool designed to support software reverse engineers. A semi-structured interview with experts led to the design and implementation of the system. We describe the visual interfaces and their integration within an existing software analysis tool. SensorRE automatically captures user's sensemaking actions and provides a graph and storyboard view to support further analysis. User study results with both experts and graduate students demonstrate that SensorRE is easy to use and that it improved the participants' exploration process.

15. SUBJECT TERMS

Reverse Engineering, Visualization, Analytic Provenance, Requirements Elicitation, Collaboration

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Gilbert L. Peterson, AFIT/ENG
U	U	U	UU	187	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636 x4281: gilbert.peterson@afit.edu