A Task Analysis of Static Binary Reverse Engineering for Security

Megan Nyre-Yu, Karin Butler, Cheryl Bolstad
Sandia National Laboratories
mnyreyu@sandia.gov

Abstract

Software is ubiquitous in society, but understanding it, especially without access to source code, is both non-trivial and critical to security. A specialized group of cyber defenders conducts reverse engineering (RE) to analyze software. The expertise-driven process of software RE is not well understood, especially from the perspective of workflows and automated tools. We conducted a task analysis to explore the cognitive processes that analysts follow when using static techniques on binary code. Experienced analysts were asked to statically find a vulnerability in a small binary that could allow for unverified access to root privileges. Results show a highly iterative process with commonly used cognitive states across participants of varying expertise, but little standardization in process order and structure. A goal-centered analysis offers a different perspective about dominant RE states. We discuss implications about the nature of RE expertise and opportunities for new automation to assist analysts using static techniques.

1. Introduction

Software is ubiquitous in society, controlling anything from the energy grid to medical devices to our social connections. As software scales and changes to meet growing needs, we must ensure that software will protect our sensitive data and continue to perform critical functions with fidelity and reliability. In cybersecurity, a specialized group of defenders uses a unique set of skills to conduct reverse engineering (RE), specifically binary analysis, to find software vulnerabilities and understand malware. For a single program, this process can take days, weeks, or months, presenting a major challenge in scaling analysis as software use grows. As recent events like SolarWinds have demonstrated [1, 2], detecting potentially malicious code is critical for national security. How can we vet software at scale to ensure it is secure?

This problem is tough, and intentional obfuscation of malicious code by bad actors makes it tougher. Whereas source code programmers and analysts use meaningful symbols in the code to help them understand program behaviors, security analysts often do not have access to these meaningful symbols. They must analyze the machine-readable binary code, often stripped of (human-) meaningful elements like variable and function names.

RE binary analysis platforms *do* include automation to help transform binaries back into code, but they are imperfect, and malicious actors often write code that undermines these automations [3-5]. Currently, the process of binary RE relies a great deal on analysts engaging in line-by-line analysis. This baseline study was designed to help build a critical research base for understanding what the humans-in-the-loop (i.e., the binary analysts) might need from their automated tools to improve efficiency and effectiveness in analyzing code and reasoning about its security.

2. Background and prior work

Some research has explored software RE for given applications, like security analysis and software development. Much of the work in software RE is cognitive, requiring observers to infer its presence from other indications. Over the years, source code understanding has come to be explained by hybrid cognitive process models that combine *bottom-up processes* of line-by-line code analysis and chunking into functional units and *top-down processes* that rely on programming domain expertise to guide the analysis [6]. RE also requires abductive reasoning [7]; analysts generate hypotheses from observations, build mental models of the code from the hypotheses, and then look for information to verify or disprove their hypotheses, assumptions, and derived conclusions.

Research that explores the RE process from a cognitive perspective is growing; it has shown that the RE process contains steps that are consistent with the scientific discovery process [8]. Process-related studies identified several 3-phase processes that RE analysts use. Votipka et al. identified 3 phases of overview & prioritization, hypothesis generation, and focused experimentation [9]. Tilley et al. identified 3 phases of

URI: https://hdl.handle.net/10125/79608
978-0-9981331-5-7

data gathering, knowledge generation, and information exploration [10]. These studies generalize across types of RE tasks, while Mullins et al. focused on defining these phases for a specific RE task [11]. They identified 3 phases in vulnerability discovery: reconnaissance, analysis, and patch & proof. All this foundational research starts to build the task structure for different RE objectives, but it does not allow researchers to observe and understand sub-processes, variance across analysts and tools, and opportunities to overcome inefficiencies in different RE tasks. RE literature is also nascent in developing objective measures to infer when these process phases are happening [12], especially when analysts engage in static analysis.

Popular methods for studying the RE process include interviews and think aloud protocols to elicit process knowledge and expertise from experienced individuals. Some studies have collected cognitive information to gain these expert perspectives [8, 9, 11, 13], but many of these studies allow participants to choose an example and walk through what they did or would have done. Unfortunately, people have more difficulty recalling process steps that did not result in useful information; thus, retrospective reports are less likely to capture some of the pain points. Although this approach is valuable in revealing higher level goals and summarized processes, it removes the task context and may not fully generalize to the actual task setting.

3. Study goals and research questions

Our study aimed to investigate basic components of static binary analysis based on observations of RE analysts analyzing a small crafted binary. Our research questions focused on obtaining information about task execution: component steps, tools and techniques, goals and strategies, and information needs.

RQ1: What tools and features do analysts use for static RE? What functionality do they want? Why?

RQ2: Are analysis process states (and sequences) consistent across participants? In what ways?

RQ3: What activities (states) could be automated to free up analysts from performing repetitive and cognitively demanding activities?

Our underlying goals were to illuminate the nature of expertise in RE, to inform efforts integrating new automation into workflows, and to identify automation opportunities. Ultimately, answering these research questions helps inform long-term goals to strategically automate parts of the static binary analysis process and thus address the need for better software security.

This study:

- Provides novel evidence that the RE process is not conducted in clear phases as previously thought. Rather, changing phases of analysis and progress toward an analysis goal are difficult to observe and measure.
- Identifies novel relevance and frequency information about cognitive states employed during RE, particularly regarding information gathering from external tools and resources.
- Validates that RE is an iterative investigative process [9] with consistent process states [13].

4. Methodology

4.1. Task description

During the study session, which lasted up to two hours, participants were asked to determine whether the checks to unlock a device in a C program could be bypassed. Participants read a readme file that described their task and the expected behaviors of the program. The device was described as unlocking under two conditions: if the user-provided password allowed root privileges, or if the time since the last password challenge was less than some threshold (i.e., relating to the timestamp). Participants were constrained to static techniques but could search externally for information (e.g., APIs). Participants were considered successful if they discovered that the program could be controlled through the number of command line arguments; specifically, if more than 5 arguments were passed to *main*, then the device was unlocked without a valid password and regardless of time since authentication.

To better understand the reasoning processes of the participants (e.g., their goals, strategies, hypotheses, information needs, and process order), we used a task analysis with verbal protocols [14]. Participants were asked to think aloud as they tried to find the bypass vulnerability. Concurrent verbal protocols capture self-commentaries *in situ*; delayed reporting can lead to omissions or distortions (e.g., [14]). However, a concurrent think aloud procedure can disrupt some types of cognitive processes, including tasks with high cognitive load and tasks that are so automated that describing them takes effort (e.g., tying your shoes). Interviewers limited interruptions, but when the self-reporting was unclear or when the participant had not spoken for several seconds, interviewers would ask for additional information. Participants were instructed to indicate if the think aloud requirement disrupted their ability to do the task, but no participants did so.

All study procedures were administered remotely through a desktop conferencing application that

allowed for audio and video (screen) recording. The participants used Remote Desktop to access a Linux virtual machine (VM) containing the program under test and providing internet access. To avoid process inconsistency introduced by different RE platforms, we constrained participants to using the well-known RE software IDA Pro 64 (IDA) with the Hex-Rays Decompiler. We chose IDA because we expected to study a larger population; Ghidra, the comparable alternative, was only released publicly in 2019.

4.2. Participants

The population of interest included individuals with two or more years' experience doing software RE of binary code. Given the absence of consensus on what constitutes "expertise" in this area, experienced RE analysts consulting on the study suggested 2 years as our minimum level. The number of people with 2 years of RE experience is low compared to other areas of cyber security. The relatively small population size *N* for binary RE and our choice of qualitative methods suggests that our sample of 12 participants (n = 12) recruited from a large company (> 10,000 employees) is sufficient. This study was reviewed and approved by an Institutional Review Board. Informed consent was obtained from all participants. Participants were compensated for 2 hours of time using a project-specific charge code.

Table 1. Summary of participant experience

Participant	Years in RE	Estimate of programs reverse engineered	Hours per week doing RE tasks	Experience with VR using binary RE
101	11-15	7-10	5-10	N
102*	6-10	26-50	10-20	Y
104*	2-5	7-10	<5	Y
105	2-5	11-25	5-10	Y
106	16-20	11-25	5-10	N
107	6-10	51-100	10-20	Y
108	20+	11-25	<5	N
109	20+	26-50	10-20	Y
110	6-10	26-50	<5	Y
111	6-10	26-50	<5	Y
112*	6-10	26-50	<5	N
113*	6-10	11-25	5-10	Y

*successfully found vulnerability in this study

Table 1 summarizes relevant demographic data about participant experience. Participants reported years of experience ranging from 2 years to more than 20 and of hours per week doing RE ranging from less than 5 to 10-20. When asked to rate their own RE skills on a 5-point Likert scale (1 is beginner, 5 is expert), all participants identified themselves as a 3 or higher. Participants had applied binary RE to vulnerability analysis, malware analysis, embedded systems analysis, and general RE, but not all participants had applied binary RE to vulnerability research (VR).

Of the 12 participants, four (4) were able to find the vulnerability within the available (2 hour) time. Participants who did not find the vulnerability were instructed to stop analysis at the two-hour mark. The average time to find the vulnerability, for those who did, was 1h 13 min (min = 57 min, max = 1h 33 min). The average time for all participants was 1h 14 min, excluding setup, consent, and troubleshooting at the beginning of the session. However, this time-related information is shared to support future studies, not as a result of the study itself. This study was meant to reveal process, not to assess performance, and time-on-task was affected by verbal protocols.

Participants volunteered for this study knowing that they would be asked to use IDA to do the analysis; this was stated in the recruitment message. During testing, 11 of 12 participants expressed some unfamiliarity with IDA (e.g., not their preferred analysis platform, have not used it recently) or with certain tools or features in IDA. In such cases, participants were urged to refamiliarize themselves with the interface before proceeding. Unfamiliarity with IDA did not appear to prevent participants from discovering the vulnerability; all four analysts who successfully found the vulnerability reported some unfamiliarity with IDA.

4.3. Data processing: qualitative coding

As previously described, audio and screen capture were recorded for each participant's session. These artifacts were analyzed to extract codable units that reflected the reasoning processes of the participants. Audio recordings were transcribed and segmented into units, with screenshots added for context. In addition, certain easily observed actions were identified as important to the reasoning process and captured as analysis units; they included using the cross-reference (x-ref) feature in IDA and using a tool outside of IDA.

Our study was based in Task Analysis; the goal was to better understand the process followed (if any) in static binary RE. We qualitatively analyzed our analysis units [15] using an inductive coding procedure to collaboratively infer the type of cognitive process being used at a given point in time; this approach does not include a predetermined codebook for independent rating. Collaborative coding allowed us to develop new codes to identify emergent process steps. Because we instructed participants to tell us about their reasoning process and information used to make decisions, we expected codes related to goals, strategies, hypotheses,

hypothesis testing, and information gathering. We also included codes that did not strictly fit into the definition of a process but might still be pertinent to discussions about how people do these tasks (e.g., statements about how IDA was working, the participant's expertise with IDA, and the participant's expertise with the type of code).

Our qualitative analysis proceeded in several stages of partitioning and coding with three raters. All three raters collaboratively reviewed the codes for consistent application and conciseness of definition. Thus, the coding process included discussion and debate about each statement and its corresponding code. Because the coding was not conducted independently, inter-rater reliability cannot be calculated. Instead, collaborative coding allowed for agreed interpretation across multiple perspectives [16], especially since the raters were also the observers. In the final stage of coding, the raters honed the code list to seven (7) process-related codes that represent *states* between which analysts transition during binary RE. These cognitive process states then became thematic elements for analyzing patterns in the data.

4.4. Analysis

We used several visualizations (Fig. 1-3) to understand larger patterns in RE processes across participants. First, we created state-based line graphs to visualize transitions between states with respect to sequence, somewhat like 'swimlane' diagrams [17]. Our representation is not time-based, since the x-axis basis is participants' codable units; rather, this shows how participants switched from one cognitive process to another. A different line represents each participant, and the (categorical) y-axis shows the cognitive process employed. The result depicts transitions of states over the task sequence across participants.

We used a chord diagram to show frequency of process steps and patterns of transitions between them. Nodes represent process states, and edges depict direct transitions between states. We found this to be an appropriate way to represent the repetition observed in static binary analysis.

We also identified three consistent goals that most participants pursued, and we analyzed the RE process states used to achieve these goals. These goals were specific to behaviors in the binary under test. First, we identified where participants stated hypotheses related to these goals. Then, we looked for related terms (e.g., "timestamp") in utterances before and after those statements to identify connected activities. To visually compare states across goals, we created radar charts, which have been used to display multivariate, qualitative data in 2-dimensional form [18].

5. Results

Table 2 lists the 7 process-related states identified during qualitative analysis, including definitions and relative frequency. We used these states to categorize participant utterances and actions in this study. As described above, these utterances and actions were observed in sessions lasting from 56 to 98 minutes, indicating that analysts transition between cognitive process states relatively frequently during a static RE task. These states are further discussed in the findings.

Table 2. Static RE cognitive process states

Process States	Definition	Total Utterances
Goal	Information that they are seeking	140
Strategy	How they plan to try to answer a question or get information	253
Hypothesis Generation	Expectation or guess about something in the code (e.g., program behavior); Analyst knows they need investigation/verification	263
Knowledge & Information Gathering	Examination of lines of code/ instructions and understanding what they mean; Looking up information from sources outside tool, like revisiting task description	574
Hypothesis Testing & Assessment	When there is a clear hypothesis generated, the activities followed to test/verify the hypothesis or assumptions; Evaluating a piece of obtained knowledge/information known about the code	145
Binary Annotation	Analyst adds comments or notes, or renames variables or functions	176
Review	Walking through information that has already been gathered, hypothesized, or verified	70

5.1. RQ1: Tools and functionality in RE

Binary analysts rely on many tools to help them to understand binaries. Almost all analysts use an analysis platform like IDA, Ghidra, or Binary Ninja to conduct initial disassembly and organize their analyses. Because we were interested in the similarities in reasoning across analysts when statically analyzing a binary, we constrained participants to using standard IDA Pro and the Hex-Rays Decompiler; no additional static or dynamic tools or plugins were allowed.

A summary of the tools they used, in the analysis platform and externally, can be found in Table 3. All participants began the analysis viewing the start of the **disassembled code** in an IDA View A tabbed window. Although the **Hex-Rays Decompiler** was available, two analysts conducted all of their analysis in the disassembly and the **control flow graph** (CFG) views. All participants used cross-referencing tools (**x-refs**)

that allowed them to see calling functions in some form; ten displayed cross-references through the x-ref view, while the two who only analyzed disassembly relied on the disassembly display of x-refs.

Table 3. RE tools used by participants

IDA tools/features	External tools/resources
Disassembly	Linux man-pages
Hex-Rays Decompiler	Internet search (general)
Control flow graph (CFG)	Notepad (physical)
Cross-references (x-refs)	Notepad (digital)
Renaming functions or variables	
Function window	
Strings view	

Similarly, all analysts relied on **renaming functions and variables** to capture their understanding of the code and to propagate that information through the code. Six (6) of the participants used the **function window** to navigate to different parts of the code, including returning to the program start and to interesting system calls, like *execve*. Five (5) of the participants either displayed all strings in a window by using the **strings view** or searched through the code for specific strings, e.g., "timestamp".

Although we anticipated IDA feature use, it is noteworthy how frequently participants in our study relied on two types of ***external tools and resources*** (152 utterances, or 26.5% of the knowledge and information gathering category). The first type of resource supported participants' search for information about standard functions (e.g., *umask* and *seteuid*). Participants accessed information through the **Linux man-pages** or using **google search**, often looking for multiple functions. The second type was a **notepad** in either physical or digital form. Five participants relied on these separate note-taking capabilities: three to provide a high-level overview of their understanding of the program and two to calculate buffer offsets or convert values to hexadecimal.

Finally, many participants wanted functionality that was not available in the study. Eleven (11) of the 12 participants stated that they would use **dynamic analysis techniques** to answer certain questions, including running the file and using a debugger to set values to verify control flow, to set incremental breakpoints, and to look for system changes to understand structures and instruction effects.

5.2. RQ2: RE process consistency

One of the goals of this study was to evaluate, across participants, the consistency in the process used to conduct a RE task. In some tasks, the sequence of states that the analyst progresses through is well-defined and consistently ordered. In other tasks, subsets of users might progress through different state sequences. One way to identify sequences in verbal protocol data is to look for process tracing patterns across state transitions. In Figure 1, we present the sequence of state transitions observed for two participants in this study, both of whom identified the vulnerability in the code. This figure shows that state transitions were frequent in analysis sessions, consistent with other research identifying the high cognitive demand associated with this type of work [12, 19]. We also noted continual "revisiting" of past states. Thus, the RE process does indeed appear to be an iterative process as suggested by Votipka et al [9].

To examine the transitions between states in more detail, we created a state transition chord diagram across all participants (Figure 2). We ordered the circumferential axis to reflect a general reasoning process. The proportion of the axis for each state indicates the relative number of times that state was visited across all participants. Link arrows indicate the direction of state transitions, and link thickness indicates the number of transitions from one state to another. Links pointing back to the same state show consecutive state repetition.

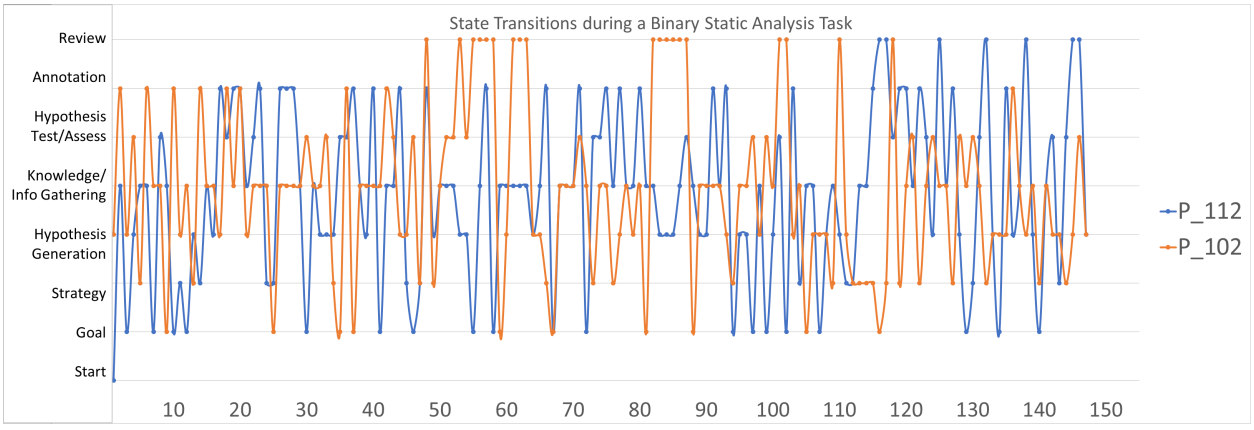

Figure 1. A state sequence graph for two participants conducting static binary analysis

This diagram indicates that, when summed across participants, knowledge & information gathering was the dominant state for static binary analysis; the large proportion of return paths indicates many consecutive steps performing knowledge & information gathering. Goal, strategy, hypothesis generation, and binary annotation states also had strong links into (and out of) knowledge & information gathering. The number of links from the "start" state illustrates the variety of approaches employed initially by different participants.

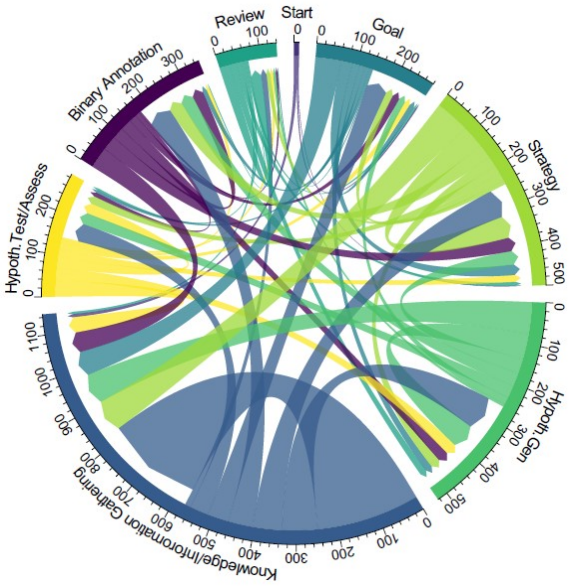

Figure 2. Chord diagram of RE states across participants.

When examining the consistency of sequential processes, we did not detect common patterns across participants. However, we did observe consistency in the goals that participants pursued. We identified three common goals across participants: 1) figuring out how the timestamp works (i.e., the role of elapsed time since the last password challenge), 2) understanding how privileges are set and accessed, and 3) figuring out how password checking works. All participants pursued goals 1 and 3, and 10 out of 12 participants pursued goal 2. We examined consistency by charting the respective cognitive process states associated with pursuing each goal. These data are presented in a radar chart (Figure 3) that shows normalized frequency of states across all participants for the three common goals. The polygon similarity indicates that participants spent about the same proportion of time in each respective process state across different goals.

Here, hypothesis generation was the most visited state (28-40% of utterances across all goals) followed by knowledge & information gathering (21-26%). This differed from the chord diagram (Figure 2) as not all codable units were utilized during the goal-oriented analysis; we only linked utterances with strong, explicit association with a goal. Remaining units are likely related to non-verbalized goals (and thus not identified in our dataset), more abstract goals, or less consistent goals across participants.

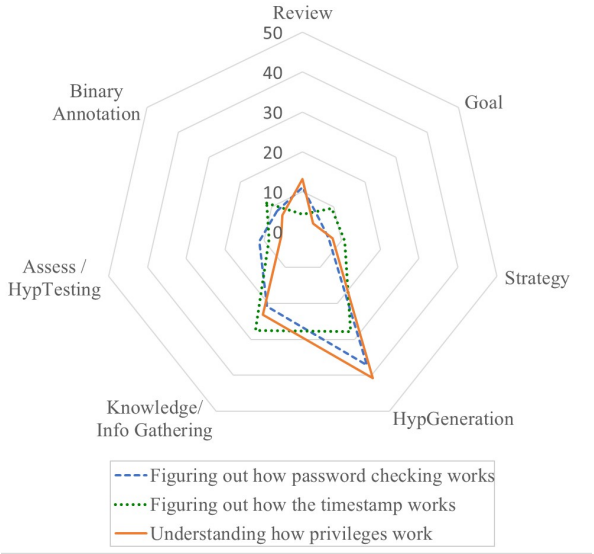

Figure 3. Radar chart of states across analysis goals.

5.3. RQ3: Automatable tasks

Another goal of this study was to identify analyst activities that are good targets for automation. To do this, we identified common repetitive activities. Such activities are cognitively demanding with a low value for binary understanding, like searching.

As shown in Table 2, analysts engaged in a surprising amount of knowledge & information gathering, with external activities accounting for 26.4% of the analysis units in this category. The most frequent goal of external information gathering was to understand how functions were called, what functions do, and what return values mean. Across participants, 99 utterances were about searching for information about binary functions (e.g., *setuid, seteuid, execve, umask*) in reference sources like man pages, stackoverflow.com, and through google searches. Every participant engaged in at least one search for reference information about a function; several participants looked for information on five or more functions, sometimes referencing the same information source more than once. While some searches were quick, all entailed leaving the binary code, conducting

a search, and deciding if the result provided the needed information for the binary under test. As one participant noted when using the man pages, "Unfortunately, this documentation is written for C programmers and not reverse engineers."

Another dominant action in external information gathering was returning to the instruction readme file. These actions accounted for 34 analysis units across participants, with eleven participants returning to the readme at least once and three participants returning six times in the 120-minute session. Participants stated that they were checking the behavior of the program, including the timestamp and password checks.

Both accessing function information and accessing behavior information was critically important for the RE process. Analysts made decisions about how to prioritize their analysis and generated hypotheses about the mapping between code and described behaviors.

6. Discussion

6.1. Discussion of findings

6.1.1. Themes of cognitive processes during vulnerability analysis. Our findings corroborate prior research conclusions about RE tasks that use only static techniques. Bryant [8] investigated binary RE using interview-based methods on a smaller population. His qualitative analysis resulted in similar codes related to process, such as creating goals or plans, forming hypotheses, and searching for information. We believe these themes are central to research that seeks to understand cognitive process during RE tasks.

As expected, our findings also align with previous research on the goals and strategies of analysts using static techniques for source code vulnerability analysis. Smith et al [20] identified many of the same questions that our participants asked. For example, "where is this used in the code?", "where is the method being called?", "how can I get calling information?", "where does this information/data go?", "where is the data coming from?", and "what is the context of this vulnerability/code?". Thus, there is some similarity in cognitive processes across different use cases of static analysis (on binaries and source code, and for reverse engineers and developers).

We noted that the volume of utterances related to planning was greater than for execution or review. There were more verbalizations of goals, strategies, hypothesis generation, and information gathering than of hypothesis testing, assessment, and review. One possible reason for this is that static binary analysis is a complex task that may be difficult to describe through verbalization, and not all participants may be adept at verbalizing their thoughts. However, we conducted different types of data analyses that resulted in different dominant states; *knowledge & information gathering* was dominant when looking at the data as a whole (Figure 2), but *hypothesis generation* was dominant when looking across specific goals (Figure 3). We note that anchoring goal-oriented analysis around specific hypotheses could inflate the dominance of hypothesis generation. These findings are also consistent with previous research finding that program understanding is a hybrid of cognitive processes, with top-down, hypothesis generation driving the analysis and bottom-up, line-by-line analysis and chunking processes playing a large role [12]. Further, these findings suggest that when analysts engage in different modes of processing they are biased to report, or engage in, that type of processing, a finding that may have implications for the efficiency of analysis and for how to design automation.

6.1.2. RE as a complex, non-linear, highly variable process. Foundational work in understanding RE workflows has depicted software analysis as a "3-phase process" (discussed in Section 2). However, our study revealed the novel observation that the *static binary analysis process is not clearly broken down into procedural phases*, nor are these phases indicative of the actual process flow. That is, the 3-phase approach nicely summarizes progress towards a goal in an abstract way but does not clearly depict the iteration and backtracking we observed.

Our findings indicate that the RE process is ad hoc, repetitive, and iterative. It is a cycle that repeats as new leads for investigation are identified or removed from scope. Instead of a 3-phase process with clear milestones and transitions [11], we observed a range of process states that are visited frequently and iteratively during a given RE task, with little indication of how much progress the analyst was making toward a larger goal. Moreover, there was a high degree of variation across participants and their process sequences, indicating that the RE process is less formal or organized than previous representations suggest [11].

We also observed that the RE tools did not support the cyclical nature of this complex task. Bryant [8] described RE tasks as "large world" problems in which the goal is not always well-defined at the beginning but becomes clearer at the end. However, RE tools do not provide strong support for changing goals. Also, their usability is still a problem; they require high familiarity or "tool expertise" [21] to do a relatively simple task efficiently. Though efficiency was not measured in this study, participants noted that lesser familiarity with IDA affected their speed despite having expertise in

the task (RE). Thus, we infer that platform familiarity would impact their ability to find the vulnerability *within a time limit*. Although analysis efficiency is a valid concern in RE, since our time limit was artificial and abnormally short, it is irrelevant to our results. Specifically, although unfamiliarity with IDA could impact which features and resources were used, it should not affect the cognitive states presented here.

6.1.3. Overhead tasks: knowledge & information gathering. Our approach allowed us to identify novel relevance and frequency information about cognitive states employed in the RE process. We observed frequent knowledge & information gathering activities across all participants, accounting for 31.9% of all coded units. Our coding differentiated between "internal" (within IDA) and "external" (outside IDA) knowledge & information gathering activities: internal activities accounted for 23.5% of all coded units and external activities accounted for 8.5% of all units.

Applying principles from industrial engineering, we can identify overhead, or non-value added, parts of the process. These overhead tasks can be characterized and mitigated or removed to increase process efficiency. Knowledge and information are critical to reasoning and code comprehension in RE tasks, but the act of searching for and retrieving that information is not adding value; as in other investigation-based cybersecurity settings [22], this makes knowledge & information gathering a good candidate for automation.

In general, foraging activities that lead analysts away from the environment where they apply their collected information is overhead. The "distance" from the environment is like movement or transportation in physical processes; further, search and retrieval add a concurrent cognitive load that could disrupt primary task performance. Reducing or removing distance by bringing that external information to the analysis environment could help increase the speed of analysis.

Again, although it had not been identified as a significant activity in previous studies about RE, we observed that participants frequently referenced external resources throughout the task. These resources included existing documentation about the program (readme), man-pages, and internet searches about specific functions, making these types of information good candidates for automated information retrieval.

6.2. Implications of findings

6.2.1. What is the nature of expertise in RE? One goal of human-centered approaches is to characterize how the work is currently done, preferably by "experts". By sampling from the population of "experts", the resulting processes, pain points, and cognitive needs are likely to be the ones that, if implemented and accommodated in training or new analysis systems, will result in the most performance gains for novices.

In this study, we were interested in the processes and reasoning of individuals that conduct RE tasks but were unsure how to evaluate "expertise". As is standard practice, we used multiple questions to try to assess "expertise". These questions included self-ratings of expertise, reports of numbers of years of experience and hours per week doing RE, and number of systems reverse engineered. Each of these measures has some drawbacks. Self-ratings require knowledge of the comparison group. Numbers of years of experience may not reflect skilled performance, and number of prior systems reverse engineered does not account for system complexity.

Expertise, particularly when measured through performance in an applied domain, may be better conceptualized as a multidimensional construct [21]. In our study, although analysts reported having at least 2 years of RE experience, their familiarity with the operating system for which the program was written constrained their ability to reason about this binary. To a lesser extent, participants reported that their lack of familiarity with the analysis platform (IDA) impeded their speed of reasoning.

The knowledge required to support expert RE differ for *source code* and *binary code*. Analysts discovering code behaviors in *binary code*, especially without symbols, cannot rely on programming knowledge and skill in the same way as those discovering behaviors in *source code*, and most RE analysts do not regularly program in assembly language. Binary RE relies on specialized, binary-specific knowledge, including about assembly instructions, operating systems, and programming languages. This task is extremely difficult [7], particularly when binary behaviors are intentionally obfuscated.

6.2.2. Opportunities for automation. Though the discussion above touched on several ideas, we discuss one automation opportunity in more detail. One pattern we observed was that almost all analysts returned to the readme file to reconsider and remember known behaviors of the program. The analyst's understanding of known behaviors is a critical element of RE and, thus, this specific activity is not a target for automation. However, these types of memory failures and reference needs will be greater when the program is larger and when the sources of information about program behaviors are more dispersed. How might this activity be organized differently to support better memory about program behaviors and to aid more efficient analysis scoping? Despite the readme giving

guidance about what functions were in and out of scope, several analysts went back and verified what was stated in the readme during the exploration.

This finding prompts an empirical question: how time-consuming is this activity for larger programs? Supporting analysts' need to reference background information, whether a readme file or other information, is similar to delivering reporting from automated analyses so that information is more accessible during code review. Automatically marking certain code or functions as out-of-scope, or providing easy access to detailed usage information in-line with the code, may help significantly.

7. Limitations and future research

Our research has several limitations. First, the reduced analysis scope (our small, two-hour activity) did not allow us to observe any difficulties with complex code navigation across a large code base. Analysis on large binaries can span days to months; this study could not address cognitive demands and strategies that come into play across those timeframes.

The use of verbal protocols limited our ability to capture specific analysis details. Participants may not have access to knowledge about their thinking, may not report all their thoughts and activities, and may report incorrectly about why they took particular actions. For example, if participants were in a goal state but failed to verbalize that, we could not identify it as a consistent, cross-participant goal. However, omissions or misrepresentations are less likely to happen during a verbal protocol than during an interview.

This study was meant to provide guidelines for developing time-based data collection approaches so that less inference is needed to analyze performance and conclude, e.g., what participants are thinking and where they are having trouble. While we were able to identify more granular steps of the RE process than previous efforts, there are still questions about what performance means in this domain and how to measure it. Future research could focus on specific research questions about performance (e.g., finite analysis elements, measures) and use objective, even experimental, methods to answer them.

This paper did not present specific information relevant to subtasks in RE, such as information needed to test specific hypotheses. However, this is an outstanding research question that we are exploring.

Though we expected to recruit more participants for this qualitative study, our final sample size was small and homogeneous; all participants were recruited from a single company. We also did not collect data for each participant across multiple tasks and thus cannot conclude if individuals consistently approach static binary analysis across different binaries (for the same objective), across different objectives (e.g., malware analysis), or across different analysis platforms (e.g., Ghidra). Future research could investigate this gap using within-subjects designs to capture variance between similar tasks and across different objectives.

8. Conclusions

Our task analysis of a reverse engineering (RE) task identified which process states analysts use. However, our findings reveal a lack of cross-subject consistency in analysis process order and structure. Future research is needed to understand within-subjects consistency and how much these states vary over target of analysis, task, and platform, tool, and technique availability.

In our efforts to better understand the RE process and the expertise required to execute it, we found indications of specialized subsets of skills within the larger RE task space based on role (e.g., developer or security analyst), task (e.g., vulnerability analysis), and target of analysis (e.g., binary, C source). Moreover, additional expertise is needed with respect to the analysis platforms and tools being utilized.

Our findings highlight opportunities to automate subtasks in RE, particularly those that are overhead, such as searching for information within and outside the analysis platform. Additional analyst support could target, e.g., overview diagrams, decision support, and iterative hypothesis generation and testing.

9. Acknowledgments

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

We would like to formally acknowledge several individuals for their contributions to this effort, including Michelle Leger, John Ziegler, Sam Mulder, Ryan Vrecenar, and D.J. Beyette.

10. References

[1] US-CERT, "Malware Analysis Report (AR21-039A) - SUNBURST," Apri 15, 2021. [Online]. Available: https://us-cert.cisa.gov/ncas/analysis-reports/ar21-039a

[2] FireEye, "Highly Evasive Attacker Leverages SolarWinds Supply Chain to Compromise Multiple Global Victims With SUNBURST Backdoor," in *FireEye Threat Research* vol. 2021, ed, 2020.

[3] S. Banescu, C. Collberg, V. Ganesh, Z. Newsham, and A. Pretschner, "Code obfuscation against symbolic execution attacks," presented at the Proceedings of the 32nd Annual Conference on Computer Security Applications, Los Angeles, California, USA, 2016.

[4] S. Sebastio *et al.*, "Optimizing symbolic execution for malware behavior classification," *Computers & Security,* vol. 93, p. 101775, 2020.

[5] N. Kuzurin, A. Shokurov, N. Varnovsky, and V. Zakharov, "On the concept of software obfuscation in computer security," presented at the Proceedings of the 10th international conference on Information Security, Valparaíso, Chile, 2007.

[6] A. von Mayrhauser and A.M. Vans, "Comprehension processes during large scale maintenance," presented at the Proceedings of the 16th international conference on Software engineering, Sorrento, Italy, 1994.

[7] K.A. Weigand and R. Hartung, "Abduction's role in reverse engineering software," in *2012 IEEE National Aerospace and Electronics Conference (NAECON)*, 25-27 July 2012, pp. 57-62, 2012.

[8] A.R. Bryant, "Understanding How Reverse Engineers Make Sense of Programs from Assembly Language Representations," Doctor of Philosophy (PhD) Dissertation, Department of Electrical and Computer Engineering, Air Force Institute of Technology, 2012.

[9] D. Votipka, S. Rabin, K. Micinski, J. S. Foster, and M.L. Mazurek, "An Observational Investigation of Reverse Engineers' Process and Mental Models," presented at the Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems, Glasgow, Scotland Uk, 2019.

[10] S.R. Tilley, S. Paul, and D.B. Smith, "Towards a framework for program understanding," in *WPC '96. 4th Workshop on Program Comprehension*, 29-31 March 1996 1996, pp. 19-28, 1996.

[11] R. Mullins, D. Kelliher, B. Nargi, M. Keeney, and N. Schurr, "Challenges and Opportunities in Collaborative Vulnerability Research Workflows," *Proceedings of the Human Factors and Ergonomics Society Annual Meeting,* vol. 64, no. 1, pp. 420-424, 2020.

[12] M.A.D. Storey, F.D. Fracchia, and H.A. Müller, "Cognitive design elements to support the construction of a mental model during software exploration," *Journal of Systems and Software,* vol. 44, no. 3, pp. 171-185, 1999.

[13] A.R. Bryant, R F. Mills, G.L. Peterson, and M.R. Grimaila, "Software Reverse Engineering as a Sensemaking Task," *Journal of Information Assurance & Security,* Article vol. 6, no. 6, pp. 483-494, 2011.

[14] B. Kirwan and L.K. Ainsworth, *A guide to task analysis: the task analysis working group*. CRC press, 1992.

[15] S B. Merriam and E. Tisdell, "Qualitative research : a guide to design and implementation," 2016.

[16] F. Cornish, A. Gillespie, and T. Zittoun, "The SAGE Handbook of Qualitative Data Analysis," London: SAGE Publications Ltd, 2014.

[17] R. Damelio, "The Basics of Process Mapping," 2011.

[18] J.M. Chambers, W.S. Cleveland, B. Kleiner, and P.A. Tukey, *Graphical Methods for Data Analysis*. Wadsworth International Group, 1983.

[19] J.A. Cowley and F.L. Greitzer, "Organizational Impacts to Cybersecurity Expertise Development and Maintenance," *Proceedings of the Human Factors and Ergonomics Society Annual Meeting,* vol. 59, no. 1, pp. 1187-1191, 2015.

[20] J. Smith, B. Johnson, E. Murphy-Hill, B. Chu, and H.R. Lipford, "Questions developers ask while diagnosing potential security vulnerabilities with static analysis," presented at the Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, Bergamo, Italy, 2015.

[21] S.K. Garrett, B.S. Caldwell, E.C. Harris, and M.C. Gonzalez, "Six dimensions of expertise: a more comprehensive definition of cognitive expertise for team coordination," *Theoretical Issues in Ergonomics Science,* vol. 10, pp. 93-105, 2009.

[22] M. Nyre-Yu, "Determining System Requirements for Human-Machine Integration in Cyber Security Incident Response," Doctor of Philosophy, School of Industrial Engineering, Purdue University, 2019.