Effects of Precise and Imprecise Value-Set Analysis (VSA) Information on Manual Code Analysis

Laura Matzen, Michelle A Leger, Geoffrey Reedy
Sandia National Laboratories
lematze@sandia.gov, maleger@sandia.gov, gereedy@sandia.gov

***Abstract*—Binary reverse engineers combine automated and manual techniques to answer questions about software. However, when evaluating automated analysis results, they rarely have additional information to help them contextualize these results in the binary. We expect that humans could more readily understand the binary program and these analysis results if they had access to information usually kept internal to the analysis, like value-set analysis (VSA) information. However, these automated analyses often give up precision for scalability, and imprecise information might hinder human decision making.**

To assess how precision of VSA information affects human analysts, we designed a human study in which reverse engineers answered short information flow problems, determining whether code snippets would print sensitive information. We hypothesized that precise VSA information would help our participants analyze code faster and more accurately, and that imprecise VSA information would lead to slower, less accurate performance than no VSA information. We presented hand-crafted code snippets with precise, imprecise, or no VSA information in a blocked design, recording participants' eye movements, response times, and accuracy while they analyzed the snippets. Our data showed that precise VSA information changed participants' problem-solving strategies and supported faster, more accurate analyses. However, surprisingly, imprecise VSA information also led to increased accuracy relative to no VSA information, likely due to the extra time participants spent working through the code.

I. Introduction

Many static analysis frameworks (e.g., Infer, angr, S2E, BAP) support building automated binary analyses for tasks like bugfinding, trigger detection, and malware analysis [7], [11], [15]–[17], [36], [37]. However, humans must understand the results from these analyses to use them, and that requires some manual reverse engineering (RE) to put the results in their proper context. Can we reuse the work done by the automated analyses to effectively support that RE effort?

Many automated static analyses perform some form of value-set analysis (VSA) [3], over-approximating what values memory and register locations can take on at runtime at each program point [26]. Binary analyses particularly lean on VSA because it does not require distinguishing between addresses and integers [4]. Instead of discarding this VSA information, we would like to make it available to reverse engineers to help them contextualize the analysis results.

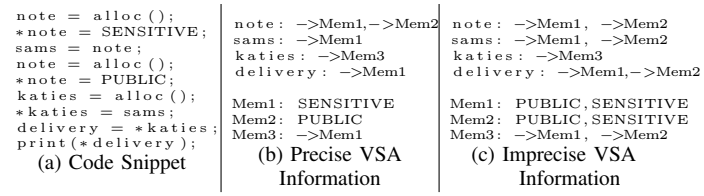
Workshop on Binary Analysis Research (BAR) 2021
21 February 2021, Virtual
ISBN 1-891562-69-X
https://dx.doi.org/10.14722/bar.2021.23002
www.ndss-symposium.org

However, these analyses trade off between precision and scalability [18]. Some analysis techniques are very precise, e.g., symbolic execution produces path-, context-, and flow-sensitive VSA information [5]. Other techniques use approximation to gain scalability, often by omitting some of these sensitivities. For example, data structure analysis (DSA) accepts imprecision through unification-based tracking and flow-insensitivity [25], and Parfait's demand-driven framework finds bug candidates using fast, imprecise analyses but filters candidates using slower, precise analyses [12]. The effects of these tradeoffs between approximation and scalability have been well studied within automated workflows, particularly for pointer analysis [21]. But before building a system to assist human reverse engineers, we need to understand the effects of these tradeoffs on the reverse engineers. We do not want to build a system that ends up impairing human reasoning — and approximate VSA information might do just that [26], [27].

To explore the effects of these tradeoffs on reverse engineers, we designed a human studies experiment asking people to solve a series of information flow problems. Information flow problems are core to many types of security questions (e.g., How could this binary leak a password? When does input influence this critical decision?), and they require reverse engineers to understand how data flows between specific sources and sinks. In our experiment, we crafted 24 simple C snippets that looked somewhat like code from a decompiler (e.g., Ghidra's, Ida's, or DREAM [1], [20], [50]). In each snippet, we defined "PUBLIC" and "SENSITIVE" information, moved this information through memory (sometimes conditionally), and then printed the value of one memory location. We asked participants to tell us whether each snippet would **always**, **sometimes**, or **never** print SENSITIVE information.

Figure 1a shows one information flow problem used in the experiment. In this example, SENSITIVE is written into the allocation to `note`; that memory is pointed to by `sams`, the memory pointed to by `katies`, and `delivery`. The printed memory, referenced through `delivery`, always points to SENSITIVE information, so the answer is **always**. Although this example used whimsical variable names, half of our examples used less meaningful names like `eax`.

We want to understand the effects of having different types of VSA information on reverse engineers' performance in solving simple information flow problems. Will reverse engineers use VSA information if they have it? Does the VSA information help? Does the precision of the information affect their performance? We know of no structured investigation into how this kind of imprecision affects a person's understanding of a program during analysis-assisted RE.



(a) Code Snippet (b) Precise VSA Information (c) Imprecise VSA Information

Fig. 1. One code snippet (information flow problem) from our experiment and its associated precise and imprecise VSA information. This code snippet **always** prints sensitive information.

To answer these questions, we asked our study participants to answer 24 information flow problems in a blocked design across three conditions (presented in a random order): eight problems with only an image of code, eight with images of code and precise VSA information (see Figure 1b), and eight with images of code and imprecise VSA information (see Figure 1c). Our precise and imprecise VSA information reflected that produced by sound may-analyses for the print statement, i.e., both included the correct values, but imprecise VSA information was an over-approximation and also included some incorrect values. Precise VSA information trivially revealed each problem's answer given the print statement. (The print above dereferences `delivery`; in the precise VSA information, `delivery` only points to `Mem1`, which contains SENSITIVE, resulting in an **always** answer.) However, imprecise VSA information never revealed the answer.

We collected eye tracking data to answer our first question (Will people use the information?) and to reveal strategies used.[1] We also collected behavioral data (timing and accuracy) to answer the latter two questions (Does it help? Does precision affect performance?). We hypothesized that participants would have the highest accuracy and fastest response times when they had precise VSA information to aid their analysis. We also hypothesized that imprecise VSA information would produce performance equal to or worse than that with no VSA information. If participants ignored the imprecise VSA information, their performance should be equivalent to the no VSA information condition. Alternatively, if participants spent time trying to understand the imprecise VSA information, which provided very little detail, they should be slower and perhaps less accurate than when they had no VSA information.

In this paper, we describe our experiment to determine whether VSA information (precise or imprecise) can aid a reverse engineer's understanding of a program, present and discuss our results, and briefly touch on implications for static analysis tool builders. Our contributions include:

- presentation and analysis of behavioral and eye tracking data from a human subjects research study, showing that fully precise VSA information *does* improve speed and accuracy of reverse engineers in small information flow problems, and that imprecise VSA information (as compared to no VSA information) improves accuracy but decreases speed;
- an analysis of this data comparing results between less and more experienced participants, showing that VSA information (both precise and imprecise) raised the accuracy of less experienced participants to be comparable to more experienced participants', and that precise VSA information led less experienced participants to adopt strategies similar to those of highly experienced participants; and
- the stimuli and experimental framework used to gather this data, which we release to support reproducibility.

[1] To facilitate eye-tracking data collection, we presented both code and VSA information as static images rather than using a RE platform like Ghidra. Some participants noted that they missed Ghidra's taint highlighting.

II. Related Works

A rich body of work over the past several decades looks to characterize the workflows and cognitive processes in reverse engineering (RE), program understanding, and defect discovery. This work explores both possible and actual interactions between human analysts and automated analyses. Many, but not all, of these studies focus on source code. Here, we mention a few publications specifically related to human studies.

Some of the studies that characterize RE directly are more applied, using metrics related to achieving a realistic RE goal. Example study goals in this category of study include characterizing the processes and reasoning [8] (even without human studies [46]), defining appropriate approaches to RE tasks [29], and identifying effective functional allocation between humans and automated analyses [38]. Our study falls into the other category, using metrics focused on the cognitive processes. Study goals in this category include determining what support humans need for gaining program understanding [42], and, in the programming domain, determining information needs when addressing interruption-caused memory failures [30].

Human studies in this domain have used a range of techniques to gain process insights, including post-hoc analysis or surveys as in [9] (potentially missing automatic decisions), think-aloud protocols or semi-structured interviews as in [45] (potentially changing analysts' approaches), or in situ data collection (potentially losing information about the relevance of actions). Other studies, like ours, take the general approach used in cognitive science research: developing a task that explores the cognitive processes of interest but is controlled enough to support understanding the data. This latter approach can provide interesting insights, even though the stimuli may not be realistic, and it supports the collection of fine-grained physical evidence using tools like fMRI [39] or eye tracking.

Eye tracking data has been used to identify differences between experts and novices during program comprehension tasks, e.g., learning to mitigate software vulnerabilities [14]. [34] provides a review of studies using eye-tracking to study source code understanding. In code comprehension tasks, the relationship between expertise and number or location of fixations is task dependent. When asked to make trustworthiness judgments, experts make more fixations within a piece of code than novices, although their comprehension of the code does not differ [22]. In bug detection, though, experts make more vertical eye movements than novices up-front [31] and achieve faster bug detection later [35], [44]. In code or print statement characterization, experts spend more time viewing task-relevant lines of code, and they tend to focus on *beacons* in the code [13], but novices spend more time reading comments [6]. We are using eye tracking data to evaluate how reverse engineers will use automated analysis results to help

them understand code, and whether experts and novices use those results differently.

Many user studies evaluate existing tooling to explore how automation might support RE. The goals of these studies include comparing specific tools [49], understanding why available tools are not used [23], [40], evaluating the utility of a single tool [2], [28], and discovering information requirements of would-be users [41]. Although our study is similar in that we are evaluating the utility of a single type of information, our evaluation focuses on *potential* information prior to tool development rather than information from an existing tool.

Detailed studies like these have taken us a long way, but we still have much to learn about the cognitive processes supporting RE tasks, and what information and automation would best support RE [10], [43]. Thus, we take another small step in this basic research to understand how binary analyses can help humans better understand programs; specifically, we use eye-tracking to understand in detail how cognitive processing and strategies change in decompiled binary RE given different VSA information approximations.

III. Methods

Prior to data collection, this study was reviewed and approved by the Human Subjects Board at Sandia National Laboratories, in accordance with the Federal Policy for the Protection of Human Subjects.

A. Materials

In this experiment, we asked participants to evaluate a set of 24 information flow problems (reasoning about source to sink relationships). Each problem required analyzing a code snippet that moved PUBLIC and SENSITIVE information through memory (sometimes conditionally) and then printed the value of one memory location. We asked participants to tell us whether each snippet would **always**, **sometimes**, or **never** print SENSITIVE information. The stimuli consisted of these 24 code snippets, each of which could be paired with precise VSA information, imprecise VSA information, or no VSA information. All images of code problems, VSA information, and problem descriptions are available in our supplemental materials.[2] An example of one of the problems, and its layout, is shown in Figure 3.

Each hand-crafted snippet was written in basic C to reflect decompiler code; half of the problems used semantically relevant names like those a reverse engineer might apply (Figure I), and half used less semantically meaningful names more like those a decompiler might apply by default (Figure 3).[3] Our snippets used two C types, pointers and integers; integers could be either an enumeration value of PUBLIC or SENSITIVE, or 0 through 3. Each snippet allocated memory, explicitly initialized memory to PUBLIC or SENSITIVE or to point to other memory, moved values through memory (using dereferences, assignments, and conditional statements testing for equality), and ultimately dereferenced a variable to print the value from one memory location. Because the snippets did not use arithmetic, VSA information in this study devolved to points-to information.

The hand-crafted precise and imprecise VSA information showed *at least* all possible values of variables and memory at the print statement, reflecting results from an over-approximate, may-points-to value-set analysis. Precise VSA information was always sufficient to answer the information flow problem (given the print), and imprecise VSA information was never sufficient to answer the problem. Thus, the memory printed always pointed to both PUBLIC and SENSITIVE in imprecise VSA information, but it only pointed to both in precise VSA information when the answer was **sometimes**.

All problems reflect this structure, but we further classify the problems into four different types, "flow," "path," "field," and "callsite," so named to reflect our inspiration for the problems.[4] We created six problems of each type: three were intended to be easy, and three were intended to be more difficult. More difficult problems used more statements, indirection, or oddly ordered conditionals than easier problems. In each difficulty level, we created one problem that **always** printed SENSITIVE information, one that **never** did, and one that **sometimes** did. Thus, overall, half of the problems were easy, and half were more difficult; and each of the three possible answers was correct for 1/3 of the problems.

Flow problems, like that seen in Figure I, used multiple assignments to the same location (memory or variables) to cause imprecise approximations. **Sometimes** flow problems additionally used a conditional statement.

Path problems additionally used conditional assignments. We define these conditional code regions to be either *Conditional Sources*, where each branch assigns from a different source location to the same destination, or *Conditional Destinations*, where each branch assigns from the same source to a different destination. (Figure 11 shows these two types of code regions in a callsite-inspired problem.) More difficult path problems used multiple related conditionals.

Field problems used at least two levels of memory indirection along with *Conditional Sources*. See Figure 2 for an example of an easy field problem. More difficult field problems used more levels of indirection.

Callsite problems, like that seen in Figure 3, used *Conditional Sources*, paired *Conditional Destinations* (i.e., the conditions between the two were identical), and straightline code between the two with direct information flow from the shared destination of the *Conditional Source* to the shared source of the *Conditional Destination*. More difficult callsite problems presented the paired conditionals in reverse order.

[2]Supplemental materials: https://github.com/maleger/vsa4infoflowstudy

[3]Because novices are less accurate when using semantically meaningless variable names [31], and because we had some imbalance in which problem types had which variable name types, our use of both meaningful and meaningless variable names could have been problematic. However, these differing name types did not appear to be problematic in this case. Labeling variable name types as "letter" or "word," participants averaged 81.4% correct for "letter" variables and 81.2% for "word" variables. Given this, and given that we provided no type information, we further believe that presenting disassembled code rather than source would not impact the shape of our results (although it might slow the process). However, our problems did not give us any insight into the effects of reused variable names.

[4]These names are notional and do not reflect a formal use of flow-, path-, callsite-, and field-sensitivity. However, we constructed imprecise VSA information to reflect the results of an analysis that lacked at least one critical sensitivity (flow- or path-, for these problems), and we constructed the precise VSA information to reflect flow- and path-sensitive results.

```
// assume o is 2
f1 = alloc();
f2 = alloc();
b = alloc();
*b = f1;
**b = SENSITIVE;
*b = f2;
**b = PUBLIC;
if (o == 1) {
    *b = f1; }
if (o == 2) {
    *b = f2; }
res = *b;
print(*res);
```

(a) Code Snippet

```
f1: ->Mem1
f2: ->Mem2
b: ->Mem3
o: 2
res: ->Mem2

Mem1: SENSITIVE
Mem2: PUBLIC
Mem3: ->Mem2
```

(b) Precise VSA Information

```
f1: ->Mem1
f2: ->Mem2
b: ->Mem3
o: ?
res: ->Mem1, ->Mem2

Mem1: SENSITIVE, PUBLIC
Mem2: SENSITIVE, PUBLIC
Mem3: ->Mem1, ->Mem2
```

(c) Imprecise VSA Information

Fig. 2. An example of the easy field problem, where the code **never** prints sensitive data, along with its precise and imprecise VSA information.

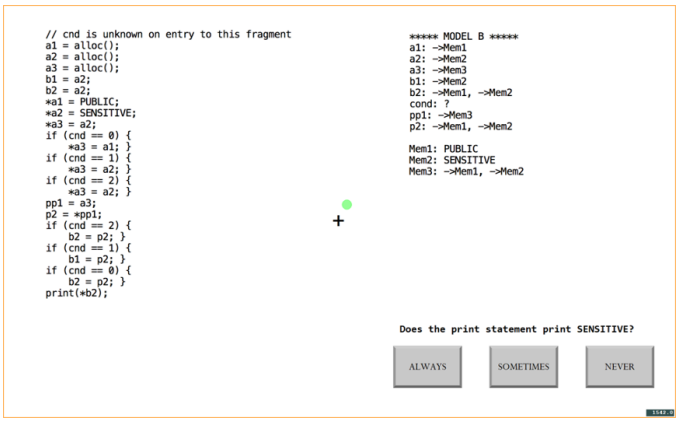


Fig. 3. An example of the problem layout. This example is a callsite problem paired with precise VSA information. The correct answer is **sometimes**.

Pairing each of the 24 snippets with each of the three possible VSA information types, we had a total of 72 stimuli. We organized the stimuli into counterbalanced blocks of trials, where each block had eight different problems paired with the same type of VSA information. Within each block, the stimuli were placed in a fixed random order (i.e., the same order for each participant). We also counterbalanced the order of the blocks: across participants, the block containing each VSA information type appeared equally often first, second, or third. This counterbalancing procedure resulted in nine experimental lists, where every problem appeared in every condition, and every block appeared in every position across lists. Each participant saw one of the nine experimental lists.

B. Procedure

After completing the informed consent process, participants filled out a basic demographic questionnaire, providing their age, highest level of education, and level of experience with C and similar programming languages, information leakage, pointer analysis, and reverse engineering.

Participants sat in a dimly lit, sound-attenuating booth, with their eyes approximately 60 cm from the computer monitor displaying the stimuli. We asked participants to sit as still as possible throughout the experiment and to avoid leaning forward or backward. A Fovio eye tracker recorded participants' eye movements during the task. After participants were seated comfortably, the eye tracker was calibrated using a five point calibration screen. Then participants read the instructions for the task.[5] The instructions explained that the participant would be using code snippets and three types of VSA information (precise, imprecise, or no VSA information) to determine whether each code snippet would **always**, **sometimes**, or **never** print sensitive information. The instructions gave examples to illustrate the nature and format of the precise and imprecise VSA information, and they demonstrated how PUBLIC and SENSITIVE information could be traced through the code snippets to the print. We instructed participants to prioritize accuracy but to answer the questions as quickly as possible.

As described above, we presented the stimuli in blocks of eight problems, displaying the same type of VSA information for all problems in a block. Before each block, a screen of instructions indicated whether the trials in that block would have precise, imprecise, or no VSA information. Participants were given breaks between the blocks.

We used E-Prime 3.0 software to present the experimental stimuli. Each trial began with a fixation cross "+" presented in black 48 point font on a white background. We instructed participants to stare at the cross until the code and VSA information appeared; it was displayed alone on the screen for 2 seconds and remained on the screen when the images appeared. The code always appeared on the left side of the screen, and, when present, the VSA information appeared on the right. Three response buttons, labeled **ALWAYS**, **SOMETIMES**, and **NEVER**, appeared in the lower right-hand corner of the screen (see Figure3). Each stimulus remained on the screen until the participant clicked on one of the response buttons with the mouse. Then, a new screen appeared with a button that said "Continue to next problem." Participants clicked on that button when they were ready to advance to the next trial. Most participants completed the self-paced experiment within 45 minutes, although a handful took longer.

C. Participants

Twenty Sandia employees (four female) participated in the experiment. They were compensated for their time at their normal hourly rate. The mean age of the participants was 32 (range 22-49). Four of the participants held bachelor's degrees, twelve held master's degrees, and four held Ph.Ds.

Of the twenty participants, eleven reported having 8-20 years of experience with C and similar programming languages. All but three of these participants indicated that they had experience with answering questions about information leakage, and all but one indicated that they had experience with pointer analysis. All of them reported that they had experience with reverse engineering.

The other nine participants were less experienced. They reported having 2-5 years of experience with C, but minimal or no experience with answering information leakage questions. One participant rated themselves as being knowledgeable about pointer analysis and two rated themselves as being knowledgeable about reverse engineering. The rest of the participants indicated that they had little or no experience in those areas.

IV. Behavioral Results

Before analyzing our results, we assessed whether the difficulty level, information leakage type, and problem type impacted participants' responses as we intended. If we were

[5]The complete instructions are provided in our supplemental materials.

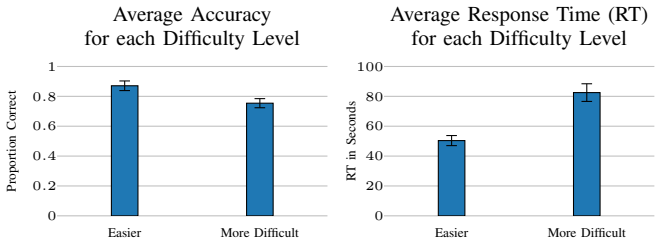


Fig. 4. Average accuracy (left) and response times (right) by difficulty level. Error bars in all figures represent the standard error of the mean.

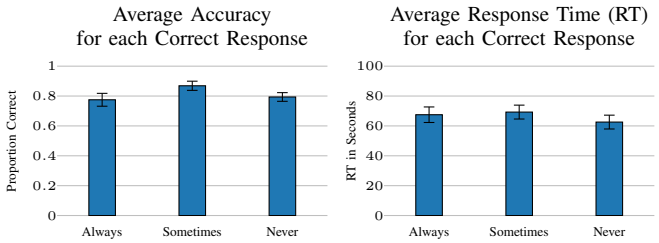


Fig. 5. Average accuracy (left) and response times (RTs, right) for problems where **always**, **sometimes** or **never** was the correct response.

		Participants' Answer		
		Always	**Never**	**Sometimes**
Correct Answer	**Always**	124	9	27
	Never	2	127	31
	Sometimes	5	16	139

TABLE I. COUNT OF PARTICIPANTS' ANSWERS BY CORRECT ANSWER.

successful in stimulus design, participants should have faster, more accurate responses to easier problems than to more difficult problems, and the type of information leakage (i.e., whether the answer was **always**, **sometimes**, or **never**) should not have affected accuracy or response time (RT).

Figure 4 shows the average accuracy and response times (RTs) for each problem difficulty level. We used paired t-tests to determine whether difficulty level impacted participants' performance. As predicted, participants had significantly higher accuracy for easier problems ($t(19) = 3.91, p < 0.001$), and they spent significantly more time on more difficult problems ($t(19) = 7.88, p < 0.001$). These results indicate that our easy and more difficult problems functioned as intended.

Figure 5 shows the average accuracy and RTs for each information leakage condition. These problems were intended to be equivalent in difficulty to one another - problems that **never** printed sensitive information should not be more difficult than problems that **always** or **sometimes** did. One-way ANOVAs [24] showed no significant differences in accuracy ($F(2, 57) = 2.02$, $p = 0.14$) or RTs ($F(2, 57) = 0.51$, $p = 0.60$) across the three conditions. These results show that we successfully developed problems that were similar in difficulty for each information leakage condition.

Although participants' performance did not differ significantly across the information leakage conditions, we observed that participants chose the **sometimes** response most often (see Table I), even though all responses were correct equally often. When the correct answer was **always** or **never**, participants were likely to choose **sometimes** if they answered the problem incorrectly, perhaps answering **sometimes** when they were unsure. In the future, we suggest having an additional response, **unsure**, to distinguish between answers that are confidently **sometimes** and those that are not.

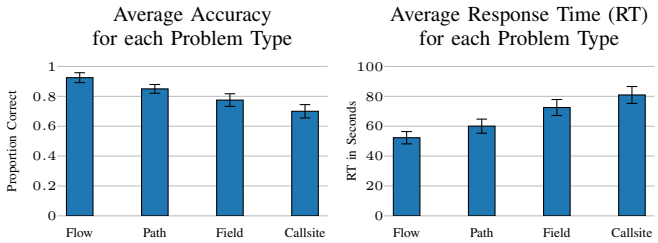


Fig. 6. Average accuracy (left) and RTs (right) by problem type.

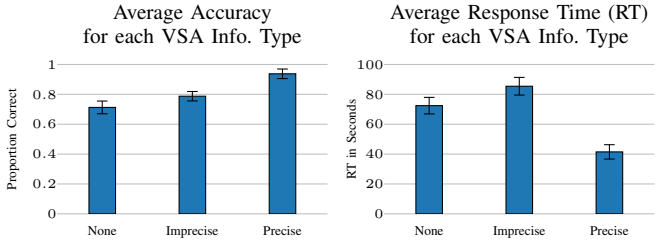


Fig. 7. Average accuracy (left) and RTs (right) by VSA information type.

We also assessed the effect of problem type on participants' performance. Figure 6 shows the average accuracy and RTs for each problem type. One-way ANOVAs showed that problem type had a significant effect on accuracy and RTs (both Fs > 6.49, both ps < 0.001). Paired t-tests showed that participants had significantly higher accuracy and significantly faster RTs for flow problems than for any other problem type (all ts > 1.82, all ps < 0.05). Participants performed next best on path problems, with significantly better accuracy and significantly faster RTs than on field and callsite problems (all ts > 1.90, all ps < 0.04). Participants performed most poorly on field and callsite problems. While the participants' RTs were significantly faster for field problems than for callsite problems ($t(19) = 2.12$, $p = 0.02$), their average accuracy for these two types of problems did not differ significantly ($t(19) = 1.53$, $p = 0.07$, one-tailed). These results showed that flow problems were easiest for participants, while callsite and field problems were the most difficult.

A. *Impact of VSA Information Type*

The results above indicate that our stimuli showed the intended variations in difficulty across conditions. Participants had better performance for easier problems and similar performance across information leakage conditions (although they answered **sometimes** more frequently). The different problem types varied in difficulty, producing a range of performance.

Our primary goal in this experiment was to assess the impact of different VSA information types on participants' accuracy, RTs, and patterns of eye movements. To do this, we first analyzed the impact of each VSA information type (precise, imprecise, and no VSA information) on participants' overall performance, and we then assessed the impact of the different VSA information types on participants' performance across the two difficulty levels and the four problem types.

Figure 7 shows participants' overall accuracy and RTs in the three VSA information conditions. One-way ANOVAs

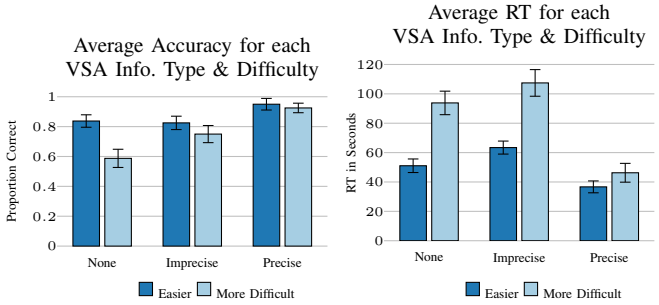


Fig. 8. Average accuracy (left) and response times (RTs, right) by VSA information type and difficulty level.

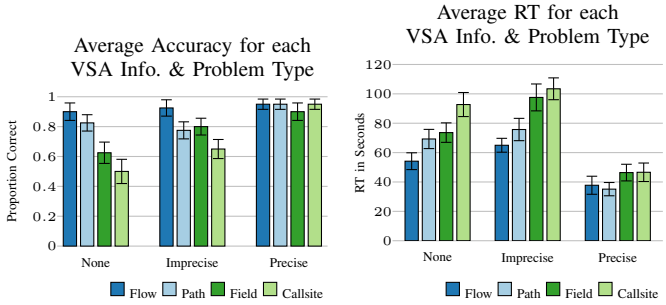


Fig. 9. Average accuracy and RTs by VSA information and problem type.

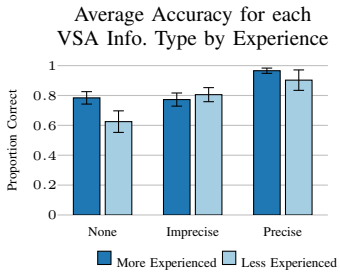


Fig. 10. Average accuracy by VSA information type, participant experience.

showed that VSA information type had a significant effect on the participants' accuracy and RTs (both $Fs > 10.24$, both $ps < 0.001$). Paired t-tests showed that participants were significantly more accurate when given precise VSA information than when given imprecise or no VSA information (both $ts > 3.93$, both $ps < 0.001$). Surprisingly, they were also significantly more accurate when given imprecise VSA information than when given no VSA information ($t(19) = 1.79$, $p = 0.04$, one-tailed). For the RT data, paired t-tests showed that participants responded significantly faster when given precise VSA information than when given imprecise or no VSA information (both $ts > 6.29$, both $ps < 0.001$), and significantly slower when given imprecise VSA information than when given no VSA information ($t(19) = 2.01$, $p = 0.03$, one-tailed). Overall, the results showed that precise VSA information improved both the speed and accuracy of participants' decisions. Contrary to our predictions, imprecise VSA information also led to improved accuracy, but that improvement came at a cost: participants were much slower in the imprecise VSA condition than in any other condition.

Figure 8 shows the average accuracy and RTs for each VSA information type, broken down by problem difficulty. A two-way repeated measures ANOVA showed that accuracy was significantly affected by VSA information type ($F(2, 95) = 15.09$, $p < 0.001$) and by difficulty level ($F(1, 95) = 11.74$, $p < 0.01$). There was also a significant interaction between VSA information type and difficulty level ($F(2, 95) = 4.01$, $p = 0.02$). RTs were similarly affected by VSA information type ($F(2, 95) = 39.03$, $p < 0.001$) and by difficulty level ($F(1, 95) = 59.25$, $p < 0.001$), and the ANOVA again showed significant interaction between the two ($F(2, 95) = 7.29$, $p < 0.01$). When no VSA information was shown, participants had much lower accuracy and much longer RTs on more difficult problems relative to easier problems. Showing participants imprecise VSA information eliminated the difference in accuracy between easier and more difficult problems, but the difference in RTs remained. Showing participants precise VSA information eliminated the accuracy difference and nearly eliminated the RT difference. Paired t-tests confirmed this, showing a significant difference in accuracy between easier and more difficult problems when no VSA information was shown ($t(19) = 4.16$, $p < 0.001$), but not when imprecise or precise VSA information was shown (both $ts < 1$). The average RTs for easier and more difficult problems were significantly different for all three VSA information conditions (all $ts > 2.05$, all $ps < 0.03$), but the magnitude of the difference was much smaller when precise VSA information was shown. This pattern indicates that the presence of VSA information was particularly helpful for more difficult problems, making participants' accuracy on more difficult problems comparable to their accuracy on easier problems.

We also assessed the interplay between problem type and VSA information type (Figure 9). For the accuracy data, a two-way repeated measures ANOVA showed a significant main effect of problem type ($F(3, 209) = 10.52$, $p < 0.001$), a significant main effect of VSA information type ($F(2, 209) = 19.64$, $p < 0.001$), and a significant interaction between the two ($F(6, 209) = 3, 51$, $p < 0.01$). Similarly, the RT data showed a significant main effect of problem type ($F(3, 209) = 17.41$, $p < 0.001$), a significant main effect of VSA information type ($F(2, 209) = 72.85$, $p < 0.001$), and a significant interaction between the two ($F(6, 209) = 2.28$, $p = 0.04$).

One-way ANOVAs comparing the four problem type conditions within each VSA information type condition showed a significant effect of problem type on accuracy and RTs when there was no VSA information or imprecise VSA information (all $Fs > 3.76$, all $ps < 0.02$). However, when participants had precise VSA information, there was no longer a significant effect of problem type for either accuracy ($F(3, 76) = 0.35$) or RTs ($F(3, 76) = 0.36$). Once again, the presence of precise VSA information mitigated the impact of problem difficulty, allowing participants to perform equally well for all four problem types (with no significant differences in accuracy or RTs). We expected that precise VSA information would reduce all problem types to the same kind of question; this confirmed our expectation.

B. *Impact of Experience*

To assess the impact of the participants' prior experience with information leakage questions, pointer analysis, and reverse engineering on their performance, we compared the aver-

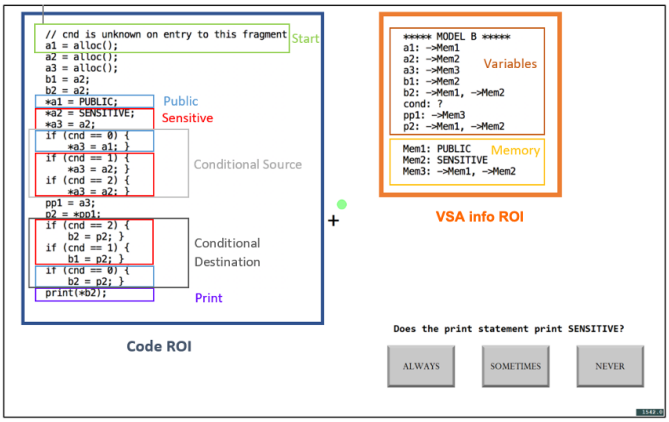


Fig. 11. An example of the ROIs for one stimulus. Blue regions within the Code ROI are Public ROIs, and red regions are Sensitive ROIs.

age accuracy of participants who had professional experience in those areas (11 participants) to those who did not (9 participants) for each VSA information type. A two-way ANOVA with VSA information type and experience level as factors showed a significant interaction between VSA information type and experience ($F(2, 36) = 3.29$, $p < 0.05$). Post-hoc t-tests showed that more experienced participants had significantly higher accuracy than less experienced participants when given no VSA information ($t(13) = 1.91$, $p = 0.04$). However, the difference between the two groups was eliminated when either precise VSA information or imprecise VSA information was available (both ts < 1). As shown in Figure 10, the presence of either type of VSA information enabled the less experienced participants to perform at the same level as the more experienced participants.

V. Eye Tracking Results

Due to a technical problem impacting eye tracking data from five participants, only 15 participants (eight experienced and seven less experienced) were included in the eye tracking analysis. Human eye movements consist of saccades (fast eye movements from one location to another) and fixations (where the eyes are relatively stationary). Virtually all visual information processing occurs during fixations [32], [33]. In our analysis, we used the default algorithm in the EyeWorks software to calculate fixations, using one degree of visual angle for the spatial parameter (estimated at 52 pixels based on the distance from the participants' eyes to our computer monitor). We divided each stimulus into regions of interest (ROIs) for the eye tracking analysis. We defined two high-level ROIs: one for the code, on the left side of the screen, and one for the VSA information, on the right side. We also conducted a more fine-grained analysis, subdividing the code and VSA information ROIs into smaller regions. We split the VSA information region into two ROIs: Variables and Memory. We identified seven possibly overlapping subdivisions in the Code ROI: Start, Memory Initialization, Conditional Source, Conditional Destination, Sensitive, Public, and Print. Not every stimulus had all of these seven ROIs, and each line of code was included in all relevant ROIs. Figure 11 shows example ROIs for one stimulus. A more detailed explanation of the ROIs is provided in the supplemental materials.

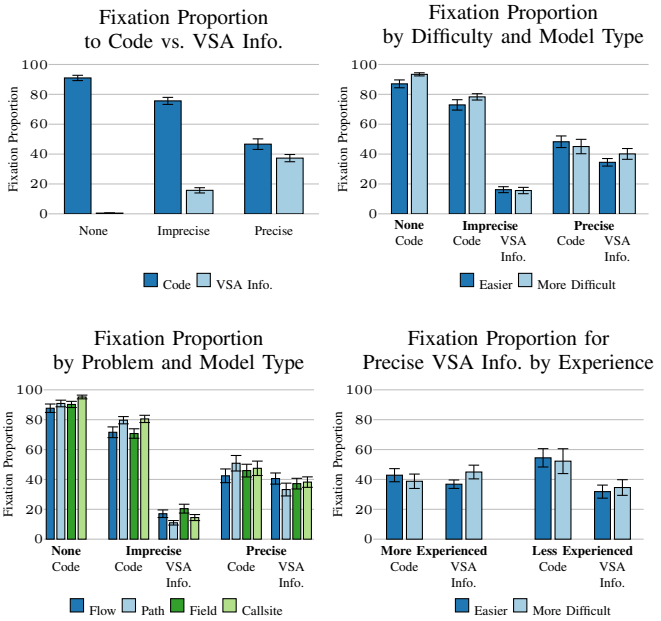


Fig. 12. Average proportion (percentage) of fixations to the Code and VSA Information Regions of Interest (ROIs) for each VSA information type condition, additionally broken down by problem difficulty, by problem type, and, for the precise VSA information condition only, by experience.

A. Fixations to the Code and VSA Information Regions of Interest

For our high-level analysis, we calculated the total number of fixations for each trial, then calculated the average proportion of fixations to the Code and VSA Information ROIs. Changes to the distribution of fixations to the different ROIs reflect changes in the participants' information processing strategies, since people tend to fixate on the information that they consider to be most relevant to their current task [19], [48]. Figure 12a shows the results of this analysis. As expected, participants did not fixate in the VSA Information ROI when no VSA information was present. When the imprecise VSA information was shown, approximately 18% of participants' fixations (on average) were in the VSA Information ROI, and when the precise VSA information was shown, approximately 37% of participants' fixations were in the VSA Information ROI. This shows that participants did use the VSA information, and that they relied more heavily on the precise VSA information than on the imprecise VSA information.

We found that problem difficulty had no impact on how much participants fixated on the imprecise VSA information (Figure 12b). When given precise VSA information, participants had a numerically higher proportion of fixations to the VSA information for more difficult problems than for easier problems. However, this difference did not reach statistical significance ($t(14) = 1.43$, $p = 0.08$).

Incorporating participants' experience levels, our analysis showed that the more experienced participants drove this trend toward fixating more on the precise VSA information for difficult problems. As shown in Figure 12d, the less experienced participants looked at the Code ROI more often than they looked at the Precise VSA Information ROI. This difference was significant for easier problems ($t(11) = 3.00$,

$p < 0.001$) and marginally significant for more difficult problems ($t(10) = 1.80$, $p = 0.05$). In contrast, the more experienced participants had more similar proportions of fixations to the Code and VSA Information ROIs. For easier problems, they had a slightly higher proportion of fixations to the Code ROI than to the VSA Information ROI. This pattern switched for more difficult problems, with experienced participants having a higher proportion of fixations to the VSA Information ROI than to the Code ROI. This switch indicates that the more experienced participants tended to rely more on the VSA information for the more difficult problems.

When we considered the different problem types, we observed again that when precise VSA information was available, participants had a lower proportion of fixations to the Code ROI and a higher proportion of fixations to the VSA Information ROI (Figure 12c). The problem type did not impact the proportion of fixations to the precise VSA information ($F(3, 56) = 0.58$); participants devoted 33-40% of their fixations to the VSA Information ROI across all four problem types. However, problem type did have an impact on the proportion of fixations to the imprecise VSA information ($F(3, 56) = 2.99$, $p < 0.04$).

B. Fixations to the Smaller ROIs Within the Code and VSA Information Regions

We next examined the patterns of fixations to the smaller ROIs that marked specific features within the code or VSA information. The VSA Information ROI contained Variables and Memory ROIs (see Figure 11). We did not observe any differences in the proportions of fixations to those ROIs across any of the experimental conditions. For all conditions in which VSA information was present, participants devoted approximately equal proportions of fixations to the Variables ROI and the Memory ROI.

When looking at the smaller ROIs within the Code ROI, we considered each problem type separately because some ROIs were specific to certain problem types. The results for each problem type are shown in Figure 13. As expected, we observed a reduced proportion of fixations to each of the ROIs in the code region when VSA information was present. In most cases, the magnitude of the reduction was similar for all ROIs, producing similar distributions of fixations across all VSA information types. For example, in flow problems, the Sensitive ROI always had the highest proportion of fixations and the Public ROI always had the second highest proportion of fixations, regardless of VSA information type. This indicates that the presence of VSA information decreased participants' fixations to the ROIs in the code region in general, rather than having a disproportionate impact on some ROIs over others.

We saw one exception to this general pattern. In callsite and path problems, participants had a higher proportion of fixations to the Conditional Source ROIs than to the Conditional Destination ROI when given no VSA information or imprecise VSA information. That pattern reversed when they were given precise VSA information; participants then had numerically higher proportions of fixations to the Conditional Destination ROI than to the Conditional Source ROIs. In these two problem types, the relationship between the conditional source and the conditional destination is critical to determining whether or

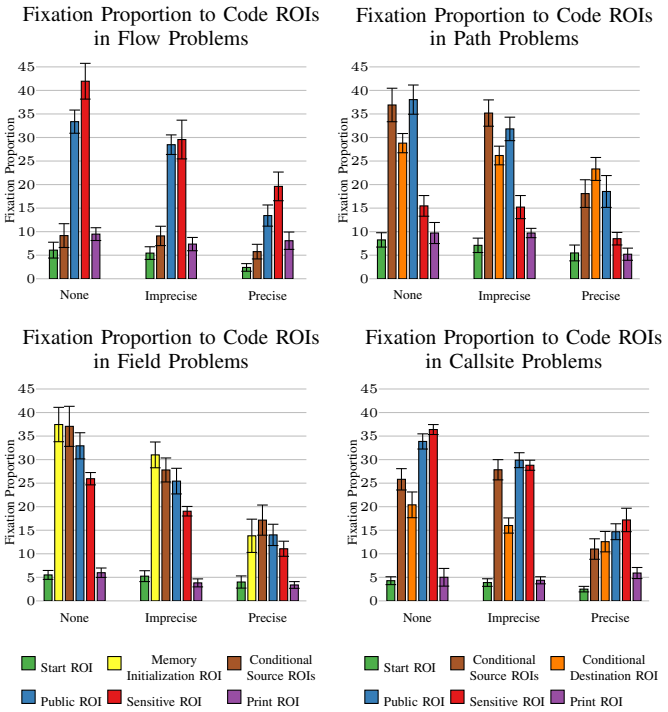


Fig. 13. Average proportion of fixations to the line-by-line ROIs in the Code region for the four problem types. Callsite problems are shown in (a), field problems in (b), flow problems in (c), and path problems in (d).

not the code will print sensitive information. The fact that participants spent less time studying the Conditional Source ROIs when precise VSA information was present indicates that they were using the information from the VSA information when reasoning about these problems. With the assistance of the VSA information, they did not need to spend as much time studying the information in the Conditional Source ROIs. Instead, they could study the VSA information and then check their understanding of the VSA information against the information in the Conditional Destination ROI. The VSA information reduced the overall amount of time needed to understand the code, but it also changed how participants interacted with specific parts of the code.

C. Order of Fixations to ROIs

We next assessed the order in which participants fixated on various regions in the code to determine whether their level of experience or the type of VSA information presented had an impact on their strategies. Given precise VSA information, participants could have answered the question of whether the code **always**, **sometimes**, or **never** printed sensitive information by looking only at the print statement and the VSA information, ignoring the rest of the code. The eye tracking data revealed that some participants did just that, as shown in Figure 14. A trace of all of the gaze points on this trial shows that this participant started at the fixation cross, looked at the VSA information, then the print statement in the code, then back to the VSA information, and finally to the correct response button at the bottom of the screen before clicking on that button.

To assess how often participants used this strategy, we analyzed the time between trial onset and each participant's

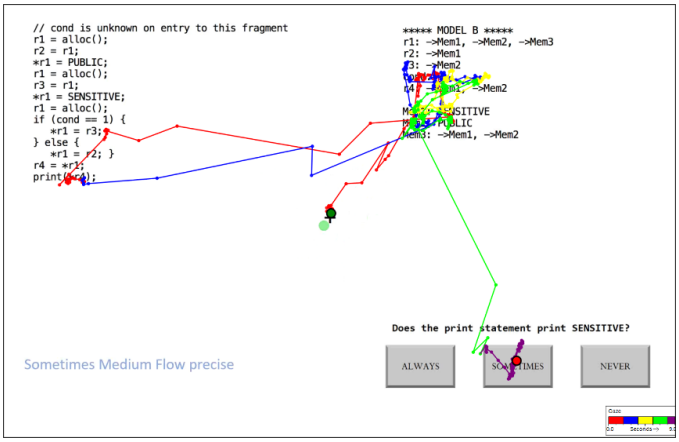


Fig. 14. Gaze data for one participant on one trial. The green circle shows the position of the participant's gaze at the beginning of the trial, and the color coding shows how the participant's eye movements unfolded over time.

first fixation on the VSA Information ROI, the Start ROI in the code region, and the Print ROI in the code region. If participants read the code from top to bottom, we would expect a shorter time to first fixation for the Start ROI relative to the Print ROI. If a participant started from the Print ROI and worked backwards through the code, we would expect the Print ROI to have a shorter time to first fixation than the Start ROI.

We found that most participants generally read the code from top to bottom, fixating on the Start ROI before the Print ROI. Given no VSA information, the average time between trial onset and the first fixation on the Start ROI was 6.44 seconds ($SD = 6.55s$), while the average was 19.5 seconds for the Print ROI ($SD = 11.28s$). Only three participants fixated on the Print ROI before the Start ROI (as determined by average time to first fixation). All three of those participants were from the highly experienced group.

Given imprecise VSA information, the average time to first fixation across all participants was shortest for the Start ROI at 7.98 seconds ($SD = 6.41s$), slightly longer for the VSA Information ROI at 9.49 seconds ($SD = 7.18$), and longest for the Print ROI at 19.27 seconds ($SD = 17.20$). This indicates that participants often started reading the code from the top, glancing back and forth between the code and the VSA information as they worked their way to the bottom. However, the overall averages obscure the fact that different participants used different strategies. While seven participants followed the pattern observed when averaging across all participants, eight did not. A group of four participants consistently fixated on the VSA Information ROI first, then on the Print ROI, and later on the Start ROI. Once again, all four were in the highly experienced group. Another two highly experienced participants tended to fixate on the Print ROI first, then on the Start ROI, and later on the VSA Information ROI, after working their way backwards through the code. Finally, two participants, both among the least experienced, had unique patterns. One of these tended to fixate on the VSA Information ROI first, then on the Start ROI, and then on the Print ROI. The other tended to fixate on the Print ROI first, then on the VSA Information ROI, and later on the Start ROI.

The average times to first fixation in our three regions of interest were quite different when participants were shown precise VSA information. Here, when averaging across all participants, the average time to first fixation was lowest for the VSA Information ROI at 4.86 seconds ($SD = 2.97$), somewhat longer for the Start ROI at 9.83 seconds ($SD = 9.32s$), and longest for the Print ROI at 15.14 seconds ($SD = 17.12s$). A total of eleven participants (six highly experienced and five less experienced) consistently fixated on the VSA Information ROI first. Seven of those eleven, including five of the highly experienced participants, tended to fixate on the Print ROI next and the Start ROI last (although one highly experienced participant never fixated on the Start ROI at all in trials with precise VSA information), while the other four participants tended to fixate on the Start ROI before the Print ROI. Another group of three participants tended to fixate on the Start ROI first, the VSA Information ROI later, and the Print ROI last. Finally, one of the less experienced participants tended to fixate on the Start ROI first, the Print ROI next, and the VSA Information ROI last.

One takeaway from the differences across VSA information conditions is that precise VSA information made the less experienced participants' eye movements more like those of the highly experienced participants. When given no VSA information or imprecise VSA information, none of the less experienced participants consistently used a strategy of starting from the print statement and working backwards through the code. In addition, when given imprecise VSA information, none of the less experienced participants started by looking at the VSA Information, whereas half of the more experienced participants started there. Yet when given precise VSA information, the majority of participants (11 of 15) tended to use a strategy where they started from the VSA Information ROI, and two of the less experienced participants adopted the experts' strategy of looking at the Print ROI before looking at the Start ROI.

VI. Discussion

In our experiment, we found that providing participants with precise VSA information improved their speed and accuracy in assessing information flow through code. We also found that providing imprecise VSA information improved participants' accuracy relative to having no VSA information, but it decreased participants' speed. The speed-accuracy tradeoff is a common finding in research on cognition [47], so it is likely that the additional time spent on imprecise VSA problems led to the accuracy improvement.[6]

We also found an interaction between problem difficulty and the three VSA information conditions. Participants performed reasonably well on easier code problems when given no VSA information, but they struggled for more difficult problems, spending much more time on each problem and providing incorrect responses an average of 40% of the time. When given imprecise VSA information, participants were still slower for more difficult problems than for easier problems,

[6]We have noticed that reverse engineers often avoid using imprecise automated analyses for support when they can perform a task manually. We wonder if reverse engineers make this trade-off because their tools do not meet their information needs, like developers [23], or because their analyses are time-limited and they can identify and correct classification errors later. This remains interesting future work.

but the difference in average accuracy was eliminated. When given precise VSA information, participants had equally good accuracy for both easy and difficult problems, and the difference in response times (RTs) was nearly eliminated as well.[7]

We saw a similar pattern when considering different problem types (flow, path, field, and callsite problems). Precise VSA information was particularly beneficial for the more difficult problem types. We observed significant effects of problem type on participants' accuracy and RTs when there was no VSA information or imprecise VSA information, but providing precise VSA information eliminated these differences, leading to near-ceiling accuracy across all four types of problems. Recall that our stimuli were fully counterbalanced, so the same code problems appeared equally often with each type of VSA information across all participants. This allows us to conclude that the differences between VSA information types are driving these results.

Our analysis of participants' eye movements provided further evidence that the precise VSA information was helpful to participants, particularly for more difficult problems. The eye tracking data confirm that participants used the VSA information, particularly the precise VSA information. All participants had a higher proportion of fixations to the VSA Information region of interest (ROI) when given precise VSA information than when given imprecise VSA information. For more experienced participants, the proportion of fixations to the precise VSA information grew even higher when the problems were more difficult. We also observed that the presence of precise VSA information could change which parts of the code participants looked at most. For callsite and path problems, participants had a high proportion of fixations to the Conditional Source ROIs when there was no VSA information or imprecise VSA information. When precise VSA information was shown, the relative proportion of fixations to the Conditional Source ROIs dropped substantially, while the relative proportion of fixations to the Conditional Destination ROIs increased. This pattern indicates that participants may have changed their strategies for reasoning about the code when precise VSA information was available.

Finally, our analysis of the order of participants' fixations showed that more experienced participants often used different strategies for approaching the code problems than less experienced participants. Some of the most experienced participants tended to start from the print statement and work their way up the code when given no VSA information. When given imprecise VSA information, the majority of the more experienced participants used this strategy, looking at the print statement first, referring to the VSA information, and then working their way through the rest of the code. When given precise VSA information, the majority of participants looked at the VSA information first, then checked the contents of the VSA information against the print statement. In this case, several of the less experienced participants adopted this strategy as well, leading to more similar patterns of eye movements for the more and less experienced participants. This echoes the behavioral results, which showed that the VSA information helped the less experienced participants to improve their accuracy to the point where it was comparable to that of the more experienced participants.

Although we greatly simplified this binary reverse engineering task to fit the constraints of a small human study using eye-tracking, we believe that the cognitive aspects observed are similar to those in realistic tasks. However, additional research is needed to determine whether our results generalize to more complex VSA information beyond points-to information (i.e., that obtained for more complex code snippets).

VII. Conclusion

Overall, the results of this experiment show that fully precise memory VSA information improves the speed and accuracy of reverse engineers reasoning about information flow problems. Precise VSA information was particularly helpful for more difficult problems and for less experienced participants. Given precise memory VSA information, participants changed the ways in which they worked through the problems, and several of the less experienced participants adopted the same strategies as the more experienced participants. Participants did not rely as heavily on imprecise VSA information, but even that VSA information was helpful in improving accuracy. Participants spent the most time on problems with imprecise VSA information, and their accuracy there was significantly higher than when there was no VSA information available. Like precise VSA information, imprecise VSA information was particularly helpful for the more difficult problems and for the less experienced participants, although it did not improve the participants' accuracy as much as precise VSA information did.

Our findings show that aiming for more precision in VSA information is useful for reverse engineers, and also that full precision is not necessary for improving their accuracy. However, having imprecise VSA information does slow reverse engineers. In situations limited by the human reverse engineers' time where accuracy does matter, fully precise analyses like symbolic execution may be more appropriate. In situations limited by human time where accuracy is not as important, it may be better to have no supporting analyses than an imprecise analysis. Further study is necessary to understand which situations require accuracy in spite of the long time scales required by realistic reverse engineering tasks today.

Acknowledgments

We would like to thank Breannan Howell, David Hannasch, Denis Bueno, our editors and reviewers, and the RAMSeS team for their comments and support; we would also like to thank the reverse engineers who made time to be our study's subjects.

This work was supported by the Laboratory Directed Research and Development program at Sandia National Laboratories, a multimission laboratory managed and operated by National Technology and 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.

[7] RTs remained relatively consistent for each VSA information condition regardless of block order, indicating no significant learning effect.

REFERENCES

[1] N. S. Agency, "Ghidra - software reverse engineering framework," https://www.nsa.gov/resources/everyone/ghidra/, 2019.

[2] M. Angelini, G. Blasilli, L. Borzacchiello, E. Coppa, D. C. D'Elia, C. Demetrescu, S. Lenti, S. Nicchi, and G. Santucci, "Symnav: Visually assisting symbolic execution," in *2019 IEEE Symposium on Visualization for Cyber Security (VizSec)*, 2019, pp. 1–11.

[3] G. Balakrishnan and T. Reps, "Analyzing memory accesses in x86 executables," in *International conference on compiler construction*. Springer, 2004, pp. 5–23.

[4] ——, "Wysinwyx: What you see is not what you execute," *ACM Trans. Program. Lang. Syst.*, vol. 32, no. 6, Aug. 2010. [Online]. Available: https://doi.org/10.1145/1749608.1749612

[5] R. Baldoni, E. Coppa, D. C. D'Elia, C. Demetrescu, and I. Finocchi, "A survey of symbolic execution techniques," *ACM Comput. Surv.*, vol. 51, no. 3, 2018.

[6] T. Beelders and J.-P. du Plessis, "The influence of syntax highlighting on scanning and reading behaviour for source code," in *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists*, ser. SAICSIT '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2987491.2987536

[7] D. Brumley, I. Jager, T. Avgerinos, and E. J. Schwartz, "Bap: A binary analysis platform," in *Computer Aided Verification*, G. Gopalakrishnan and S. Qadeer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 463–469.

[8] A. R. Bryant, R. Mills, G. Peterson, and M. Grimaila, "Software reverse engineering as a sensemaking task," 2012.

[9] K. Butler, M. Leger, D. Bueno, C. Cuellar, M. Haass, T. Loffredo, G. Reedy, and J. Tuminaro, *Creating a User-Centric Data Flow Visualization: A Case Study*, 06 2019, pp. 174–193.

[10] G. Canfora, M. Di Penta, and L. Cerulo, "Achievements and challenges in software reverse engineering," *Commun. ACM*, vol. 54, no. 4, p. 142–151, Apr. 2011. [Online]. Available: https://doi.org/10.1145/1924421.1924451

[11] V. Chipounov, V. Kuznetsov, and G. Candea, "The s2e platform: Design, implementation, and applications," *ACM Trans. Comput. Syst.*, vol. 30, no. 1, pp. 2:1–2:49, Feb. 2012. [Online]. Available: http://doi.acm.org/10.1145/2110356.2110358

[12] C. Cifuentes and B. Scholz, "Parfait: designing a scalable bug checker," in *SAW '08: Proceedings of the 2008 workshop on Static analysis*. New York, NY, USA: ACM, 2008, pp. 4–11.

[13] M. Crosby, J. Scholtz, and S. Wiedenbeck, "The roles beacons play in comprehension for novice and expert programmers," in *PPIG*, 2002.

[14] D. Davis and F. Zhu, "Understanding and improving secure coding behavior with eye tracking methodologies," in *Proceedings of the 2020 ACM Southeast Conference*, ser. ACM SE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 107–114. [Online]. Available: https://doi.org/10.1145/3374135.3385293

[15] D. Distefano, M. Fähndrich, F. Logozzo, and P. W. O'Hearn, "Scaling static analyses at facebook," *Commun. ACM*, vol. 62, no. 8, p. 62–70, Jul. 2019. [Online]. Available: https://doi.org/10.1145/3338112

[16] Y. Fratantonio, A. Bianchi, W. Robertson, E. Kirda, C. Kruegel, and G. Vigna, "Triggerscope: Towards detecting logic bombs in android applications," 05 2016, pp. 377–396.

[17] GrammaTech, "Codesonar for binaries," https://www.grammatech.com/codesonar-sast-binary.

[18] N. Grech, G. Fourtounis, A. Francalanza, and Y. Smaragdakis, "Shooting from the heap: Ultra-scalable static analysis with heap snapshots," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 198–208. [Online]. Available: https://doi.org/10.1145/3213846.3213860

[19] J. M. Henderson, "Human gaze control during real-world scene perception," *Trends in Cognitive Sciences*, vol. 7, no. 11, pp. 498 – 504, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364661303002481

[20] Hex-Rays, "Ida hex-rays decompiler," https://www.hex-rays.com/products/decompiler/.

[21] M. Hind, "Pointer analysis: Haven't we solved this problem yet?" *ACM SIGPLAN/SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, 07 2001.

[22] S. A. Jessup, S. M. Willis, G. M. Alarcon, and M. A. Lee, "Using eye-tracking data to compare differences in code comprehension and code perceptions between expert and novice programmers," in *54th Hawaii International Conference on System Sciences*, Conference Proceedings.

[23] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. IEEE Press, 2013, p. 672–681.

[24] J. Kaufmann and A. Schering, *Analysis of Variance ANOVA*, 09 2014.

[25] C. Lattner, A. Lenharth, and V. Adve, "Making context-sensitive points-to analysis with heap cloning practical for the real world," in *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 278–289. [Online]. Available: https://doi.org/10.1145/1250734.1250766

[26] J. Lin, L. Jiang, Y. Wang, and W. Dong, "A value set analysis refinement approach based on conditional merging and lazy constraint solving," *IEEE Access*, vol. 7, pp. 114 593–114 606, 2019.

[27] E. Loftus, D. Miller, and H. Burns, "Semantic integration of verbal information into a visual memory," *Journal of experimental psychology. Human learning and memory*, vol. 4, pp. 19–31, 02 1978.

[28] R. Mangal, X. Zhang, A. V. Nori, and M. Naik, "A user-guided approach to program analysis," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 462–473. [Online]. Available: https://doi.org/10.1145/2786805.2786851

[29] T. Nosco, J. Ziegler, Z. Clark, D. Marrero, T. Finkler, A. Barbarello, and W. M. Petullo, "The industrial age of hacking," in *USENIX Security Symposium*, 2020.

[30] C. Parnin and S. Rugaber, "Programmer information needs after memory failure," in *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, 2012, pp. 123–132.

[31] N. Peitek, J. Siegmund, and S. Apel, "What drives the reading order of programmers? an eye tracking study," in *Proceedings of the 28th International Conference on Program Comprehension*, ser. ICPC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 342–353. [Online]. Available: https://doi.org/10.1145/3387904.3389279

[32] K. Rayner, "Eye movements in reading and information processing: 20 years of research." *Psychological bulletin*, vol. 124 3, pp. 372–422, 1998.

[33] ——, "The 35th sir frederick bartlett lecture: Eye movements and attention in reading, scene perception, and visual search," *Quarterly Journal of Experimental Psychology*, vol. 62, no. 8, pp. 1457–1506, 2009, pMID: 19449261. [Online]. Available: https://doi.org/10.1080/17470210902816461

[34] Z. Sharafi, Y.-G. Guéhéneuc, and Z. Soh, "A systematic literature review on the usage of eye-tracking in software engineering," *Elsevier Journal of Software and Information Technology (IST)*, 07 2015.

[35] B. Sharif, M. Falcone, and J. I. Maletic, "An eye-tracking study on the role of scan time in finding source code defects," in *Proceedings of the Symposium on Eye Tracking Research and Applications*, ser. ETRA '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 381–384. [Online]. Available: https://doi.org/10.1145/2168556.2168642

[36] Y. Shoshitaishvili, R. Wang, C. Salls, N. Stephens, M. Polino, A. Dutcher, J. Grosen, S. Feng, C. Hauser, C. Kruegel, and G. Vigna, "Sok: (state of) the art of war: Offensive techniques in binary analysis," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 138–157.

[37] Y. Shoshitaishvili, R. Wang, C. Hauser, C. Kruegel, and G. Vigna, "Firmalice - automatic detection of authentication bypass vulnerabilities in binary firmware," in *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015.

[38] Y. Shoshitaishvili, M. Weissbacher, L. Dresel, C. Salls, R. Wang, C. Kruegel, and G. Vigna, "Rise of the hacrs: Augmenting autonomous cyber reasoning systems with human assistance," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer*

and Communications Security, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 347–362. [Online]. Available: https://doi.org/10.1145/3133956.3134105

[39] J. Siegmund, C. Kästner, S. Apel, C. Parnin, A. Bethmann, T. Leich, G. Saake, and A. Brechmann, "Understanding understanding source code with functional magnetic resonance imaging," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 378–389. [Online]. Available: https://doi.org/10.1145/2568225.2568252

[40] J. Smith, L. Do, and E. Murphy-Hill, "Why can't johnny fix vulnerabilities: A usability evaluation of static analysis tools for security," in *SOUPS @ USENIX Security Symposium*, 2020.

[41] J. Smith, B. Johnson, E. Murphy-Hill, B. Chu, and H. R. Lipford, "Questions developers ask while diagnosing potential security vulnerabilities with static analysis," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 248–259. [Online]. Available: https://doi.org/10.1145/2786805.2786812

[42] 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," *J. Syst. Softw.*, vol. 44, no. 3, pp. 171–185, Jan. 1999. [Online]. Available: http://dx.doi.org/10.1016/S0164-1212(98)10055-9

[43] M. K. Tennor, "Reverse engineering cognition," 2015.

[44] R. Turner, M. Falcone, B. Sharif, and A. Lazar, "An eye-tracking study assessing the comprehension of c++ and python source code," in *Proceedings of the Symposium on Eye Tracking Research and Applications*, ser. ETRA '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 231–234. [Online]. Available: https://doi.org/10.1145/2578153.2578218

[45] 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 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI EA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–6. [Online]. Available: https://doi.org/10.1145/3290607.3313040

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

[47] W. A. Wickelgren, "Speed-accuracy tradeoff and information processing dynamics," *Acta Psychologica*, vol. 41, no. 1, pp. 67 – 85, 1977. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0001691877900129

[48] J. Wolfe, "Guided search 2.0 a revised model of visual search," *Psychonomic Bulletin and Review*, vol. 1, pp. 202–238, 06 1994.

[49] K. Yakdan, S. Dechand, E. Gerhards-Padilla, and M. Smith, "Helping johnny to analyze malware: A usability-optimized decompiler and malware analysis user study," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 158–177.

[50] K. Yakdan, S. Eschweiler, E. Gerhards-Padilla, and M. Smith, "No more gotos: Decompilation using pattern-independent control-flow structuring and semantic-preserving transformations," in *NDSS*, 2015.