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

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

Abstract—**Binary reverse engineers rely on a combination of 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 has the potential to 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 having no VSA information at all. Our hand-crafted code snippets were presented paired with precise, imprecise, or no VSA information in a blocked design. We recorded participants' eye movements, response times, and accuracy while they answered the problems. Our experiment showed that precise VSA information changed participants' problem-solving strategies and supported faster, more accurate analyses. However, contrary to our predictions, having imprecise VSA information also led to increased accuracy relative to having no VSA information; this seemed to be because participants spent more time working through the code.

I. Introduction

Many static analysis frameworks (e.g., Infer, angr, S2E, BAP) support building automated static binary analyses for tasks like bugfinding, trigger detection, malware analysis, and so on [6], [8], [11]–[13], [29], [30]. However, understanding and using the results of analyses built on such frameworks requires inspection by a human analyst, and that requires some manual reverse engineering to put the results in their proper context. It would be ideal if all the work done by the automated analyses could also support that reverse engineering effort, but it is unclear how to do that effectively.

Many automated static analyses perform some form of value-set analysis (VSA) [2] in the course of their larger analysis, over-approximating what values memory and register locations can take on at runtime [21]; binary analyses particularly lean on VSA because it does not require distinguishing between addresses and integer values [3]. Instead of discarding this VSA information when the analysis completes, we would like to make it available to reverse engineers to help them contextualize the analysis results.

However, these analyses make tradeoffs between precision and scalability while attempting to maintain completeness [14]. Some analysis techniques are very precise, e.g., symbolic execution produces path-, context-, and flow-sensitive VSA information [4]. Other techniques use approximation to gain scalability, often by omitting one or more of these sensitivities. For example, data structure analysis (DSA) accepts imprecision through unification-based tracking and a flow-insensitive analysis [20], and Parfait provides a demand-driven framework that finds bug candidates with fast, imprecise analyses and filters those candidates using slower, precise analyses [9]. The effects of these tradeoffs between approximation and scalability have been well studied within automated workflows, particularly for pointer analysis [17]. But before building a system intended to assist human reverse engineers, we need to understand the effects of these tradeoffs on human reverse engineers within their workflows. We do not want to build a system that ends up impairing human reasoning — and approximate information could be worse for the human than no information at all [22].

To explore the effects of these tradeoffs on human reverse engineers, we designed a human studies experiment (reviewed and approved by our Human Subjects Board) asking people to solve a series of information flow problems. Information flow problems are core to answering many types of security questions (e.g., could this binary leak my password, and how? When could an attacker 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 basic C snippets that looked somewhat like code a reverse engineer might see in a decompiler (e.g., Ghidra or Ida's decompiler, or DREAM [1], [16], [40]), and we defined "PUBLIC" and "SENSITIVE" information in each snippet. Each snippet moved the 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.

Figure 1a shows one of the information flow problems 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

Network and Distributed Systems Security (NDSS) Symposium 2021
21-24 February 2021, San Diego, CA, USA
ISBN 1-891562-66-5
https://dx.doi.org/10.14722/bar.2021.23002
www.ndss-symposium.org

```
note = alloc();
*note = SENSITIVE;
sams = note;
note = alloc();
*note = PUBLIC;
katies = alloc();
*katies = sams;
delivery = *katies;
print(*delivery);
```
(a) Code Snippet

```
note: ->Mem1,
      ->Mem2
sams: ->Mem1
katies: ->Mem3
delivery: ->Mem1

Mem1: SENSITIVE
Mem2: PUBLIC
Mem3: ->Mem1
```
(b) Precise VSA Information

```
note: ->Mem1, ->Mem2
sams: ->Mem1, ->Mem2
katies: ->Mem3
delivery: ->Mem1,
          ->Mem2

Mem1: PUBLIC,
      SENSITIVE
Mem2: PUBLIC,
      SENSITIVE
Mem3: ->Mem1, ->Mem2
```
(c) Imprecise VSA Information

Fig. 1. One code snippet (information flow problem) from our experiment, and its associated precise and imprecise VSA information. (This VSA information uses multiple lines for some of the locations for display purposes. In the experiment, each location's VSA information was given on a single line.) This code snippet **always** prints sensitive information.

our examples used less meaningful names like `eax`.

Recall that 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 at all if they have it? Does the VSA information help? Does the precision of the information affect their performance? To our knowledge there has been no structured investigation into how this kind of imprecision might affect a person's understanding of a program during analysis-assisted reverse engineering.

To answer these questions, we asked our study participants to answer the 24 information flow problems in a blocked design across three conditions (presented in a random order): eight problems with only an image of the code, eight problems with images of the code and precise VSA information at the print statement (see Figure 1b), and eight problems with images of the code and imprecise VSA information at the print statement (see Figure 1c). Our precise and imprecise VSA information reflected that produced by sound may-analyses, i.e., both included the correct values, but imprecise VSA information was an over-approximation and also included some wrong values. Given precise VSA information, each problem's answer could be determined without referencing code other than the print statement. (The print above dereferences `delivery`; in the precise VSA information, `delivery` points to only `Mem1`, which only contains SENSITIVE information, resulting in an **always** answer.) However, given imprecise VSA information, the answer could not be determined without referencing the code.

We collected eye tracking data to answer our first question (will people use the information?) and to reveal strategies used during the exercise.[1] We also collected behavioral data (timing and accuracy) to help us answer the latter two questions (does it help, and does precision affect performance?). We hypothesized that participants would have the highest accuracy and the fastest response times when they had precise VSA information to aid their analysis. We also hypothesized that the imprecise VSA information would produce performance that was equal to or worse than having no VSA information at all. 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, we would expect them to be slower and perhaps less accurate than when they had no VSA information to work with.

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 having VSA information (both precise and imprecise) raised the accuracy of less experienced participants to be comparable with more experienced participants, and that VSA information led less experienced participants to adopt strategies that were similar to those commonly used by highly experienced participants; and
- the stimuli and experimental framework that we used to gather this data, which we intend to release to support reproducibility.

II. Related Works

Several related papers characterize why *source code* analysts and developers don't use the static analysis tools available to them [18], how to make the static analysis tools more effective and customizable for the users [23], or what information users are looking for [33]. Others evaluate usability of static analysis tools and reverse engineering support tools [32], [39]. Some evaluate and compare behavioral data about workflows end-to-end, drawing conclusions about appropriate approaches to reverse engineering tasks [24] or how a human analyst might effectively assist the automated analysis [31]. Still others observe reverse engineers' general processes and cognitions to determine what supports they need in given tasks [5], [7], [34], [36]. Some recent work uses eye tracking to understand comprehension or to assess how expertise affects process in software engineering and software understanding [10], [25], [28], [35], but these also focus on source code comprehension. We are not aware of any prior studies using eye-tracking to focus on detailed cognitive processing in decompiled binary reverse engineering.

III. Methods

This study was reviewed and approved by the Human Subjects Board at Sandia National Laboratories.

[1]Because we collected eye-tracking data to answer whether the reverse engineers used the VSA information at all, we had to present both code and VSA information as static images rather than within the context of a reverse engineering platform like Ghidra. Some participants noted that they missed the ability to see variable highlighting.

A. Materials

In this experiment, we asked participants to evaluate a set of 24 information flow problems (reasoning about source to sink relationships) instantiated in code snippets. Each snippet 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 and text 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), and ultimately dereferenced a variable to print the value from one memory location. Arithmetic was not included in any of the snippets, conditional statements compared two values for equality to determine which branch to execute.

The hand-crafted precise and imprecise VSA information showed *at least* all possible values of variables and memory at the print statement; this information reflected 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 statement, and imprecise VSA information was never sufficient to answer the information flow problem given the print statement. Thus, the memory printed always pointed to one of PUBLIC or SENSITIVE in precise VSA information, and it always pointed to both in imprecise VSA information.

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 problems were intended to be easy, and three were intended to be more difficult; more difficult problems use more statements, indirection, or oddly ordered conditional statements 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; each of the three possible answers was correct for 1/3 of the problems.

[2]We intend to release our supplemental materials on GitHub.

[3]This naming did not appear to be a confound, although there is some imbalance in which problem types have which variable types. Labeling variable name types as "letter" or "word," we saw that the average was 81.4% correct for "letter" variables and 81.2% for "word" variables.

[4]These names are notional only and *do not* reflect any formal use of flow-, path-, callsite-, or field-sensitivity.

```
// 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

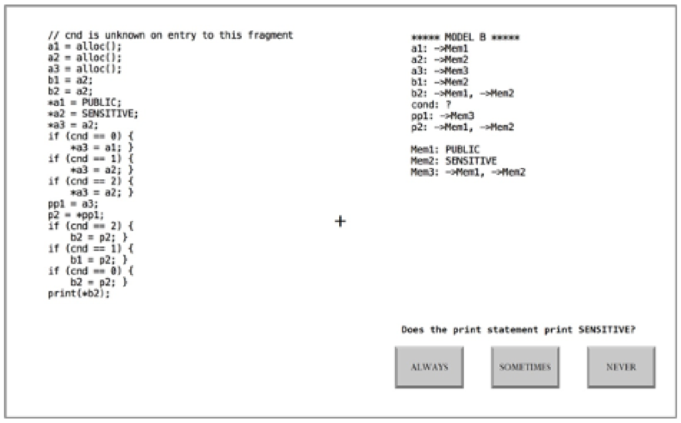
```
f1: ->Mem1
f2: ->Mem2
b: ->Mem3
o: 2
res: ->Mem2

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.



```
// cnd is unknown on entry to this fragment
a1 = alloc();
a2 = alloc();
a3 = alloc();
b1 = a2;
b2 = a2;
*a1 = PUBLIC;
*a2 = SENSITIVE;
*a3 = a2;
if (cnd == 0) {
    *a3 = a1; }
if (cnd == 1) {
    *a3 = a2; }
if (cnd == 2) {
    *a3 = a2; }
pp1 = a3;
p2 = *pp1;
if (cnd == 2) {
    b2 = p2; }
if (cnd == 1) {
    b1 = p2; }
if (cnd == 0) {
    b2 = p2; }
print(*b2);
```

```
***** MODEL B *****
a1: ->Mem1
a2: ->Mem2
a3: ->Mem3
b1: ->Mem2
b2: ->Mem1, ->Mem2
cond: ?
pp1: ->Mem3
p2: ->Mem1, ->Mem2

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

Does the print statement print SENSITIVE?

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

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

Path problems additionally use 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 location, or *Conditional Destinations*, where each branch assigns from the same source location to a different destination location. (Figure 11 shows these two types of code regions in a callsite problem.) The more difficult path problems use multiple related conditionals.

Field problems use 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 use more levels of indirection.

Callsite problems, like that seen in Figure 3, use *Conditional Sources*, paired *Conditional Destinations* (i.e., the conditions between the two are 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 present the paired conditionals in reverse order to each other.

Pairing each of the 24 snippets with each of the three possible VSA information types for that problem, 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: within any given experimental list, the block containing each VSA information type appeared equally often in the first, second, or third position. The result of the counterbalancing procedure was a set of 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 their level of experience with C and similar programming languages, information leakage, pointer analysis, and reverse engineering.

Participants were seated in a dimly lit, sound-attenuating booth, with their eyes approximately 60 cm from the computer monitor on which the stimuli were displayed. Participants were asked 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 statement. Participants were instructed to prioritize accuracy, but to answer the questions as quickly as possible.

As described above, the stimuli were presented in blocks of eight problems such that all of the problems in the same block were paired with the same type of VSA information. Each block was preceded by a screen of instructions that indicated whether the trials in that block would have precise, imprecise, or no VSA information. Participants were given breaks between the blocks.

The experimental stimuli were presented using E-Prime 3.0 software. Each trial began with a fixation cross "+" presented in black 48 point font on a white background. The fixation cross was displayed alone on the screen for 2 seconds, and participants were instructed to stare at the cross while it was on the screen. The cross remained on the screen when the image containing the code and VSA information appeared. The code always appeared on the left side of the screen, and, when present, the VSA information always appeared on the right. Three response buttons, labeled **ALWAYS**, **SOMETIMES**, and **NEVER**, were shown 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. After a response button was clicked, a new screen appeared with a button that said "Continue to next problem." Participants clicked on the button when they were ready to advance to the next trial. Most participants completed the experiment within 45 minutes, although a handful took longer due to the self-paced nature of the task.

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

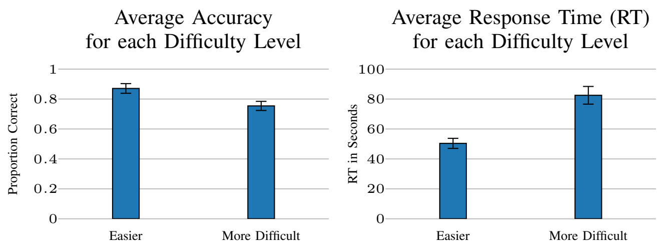


Fig. 4. The average accuracy (left) and response times (right) for problems of each level of difficulty. The error bars in all figures represent the standard error of the mean.

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 performed a quick self-check to assess whether the difficulty level, information leakage type, and problem type impacted participants' responses as we intended. If we were successful in designing appropriate stimuli, 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. These showed that, as predicted, participants had significantly higher accuracy for the easier problems than for the more difficult problems ($t(19) = 3.91$, $p < 0.001$), and they spent significantly more time on the difficult problems than on the easier problems ($t(19) = 7.88$, $p < 0.001$). These results indicate that our easy and more difficult problems functioned as intended.

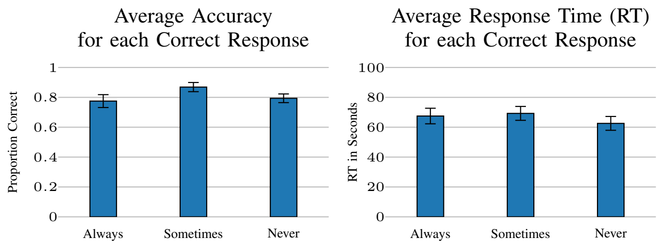


Fig. 5. The average accuracy (left) and response times (right) for problems where **always**, **sometimes** or **never** was the correct response.

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

TABLE I. COUNT OF PARTICIPANTS' ANSWERS VERSUS THE CORRECT ANSWER.

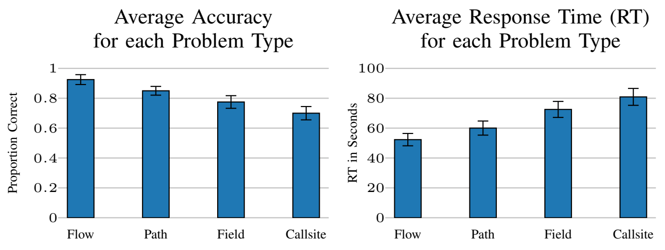


Fig. 6. The average accuracy (left) and response times (right) for each problem type.

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 - we did not want problems that **never** printed sensitive information to be more difficult than the problems that **always** or **sometimes** did. One-way ANOVAs 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 three information leakage conditions, we observed that participants chose the **sometimes** response more often than **always** or **never**, even though all three responses were correct equally often. Table I shows how often the participants selected each response type. When the correct answer was **always** or **never**, participants were likely to choose **sometimes** if they answered the problem incorrectly; this pattern indicates that the participants may have used the **sometimes** response in cases where they were unsure of the correct answer. In the future, we suggest having an additional response, **unsure**, to distinguish between answers that are confidently **sometimes** and those that are not.

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 the participants' accuracy ($F(3, 76) = 6.54$, $p < 0.001$) and RTs ($F(3, 76) = 6.50$, $p < 0.001$). Paired t-tests showed that the participants had significantly higher accuracy and significantly faster RTs for

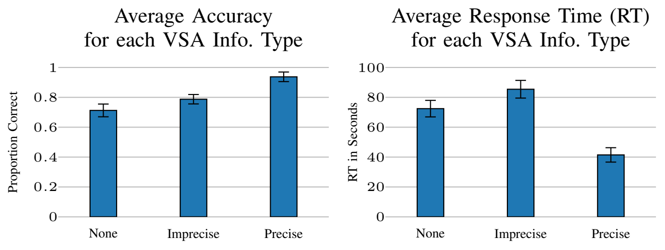


Fig. 7. The average accuracy (left) and response times (right) for each VSA information type.

the flow problems than for any other type of problem (all ts$>$ 1.82, all ps$<$ 0.05). The participants' next best performance was on the path problems, which had significantly better accuracy and significantly faster RTs than the field and callsite problems (all ts$>$ 1.90, all $ps < 0.04$). The participants' performance was poorest for the field and callsite problems. While the participants' RTs were significantly faster for the field problems than for the callsite problems ($t(19) = 2.12$, $p = 0.02$), the average accuracy of their responses to these two types of problems did not differ significantly ($t(19) = 1.53$, $p = 0.07$, one-tailed). These results showed that the flow problems were the easiest for the participants, while the 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 they had 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 showed that VSA information type had a significant effect on the participants' accuracy ($F(2, 57) = 10.25$, $p < 0.001$) and RTs ($F(2, 57) = 17.21$, $p < 0.001$). Paired t-tests showed that participants were significantly more accurate when given precise VSA information than when given imprecise ($t(19) = 3.94$, $p < 0.001$) or no VSA information ($t(19) = 5.91$, $p < 0.001$). 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 VSA informations ($t(19) = 8.05$, $p < 0.001$) or no VSA information ($t(19) = 6.30$, $p < 0.001$), 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

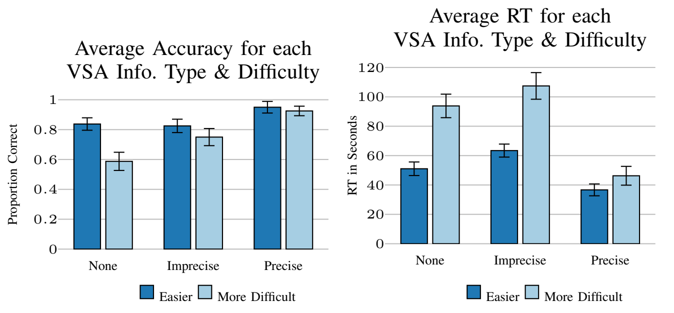


Fig. 8. The average accuracy (left) and response times (right) for each VSA information type and difficulty level.

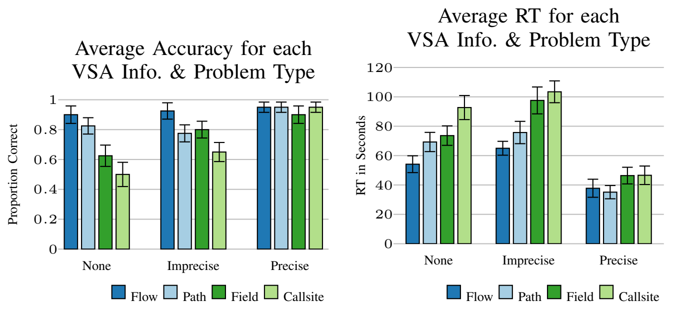


Fig. 9. The average accuracy (left) and response times (right) for each VSA information type and problem type.

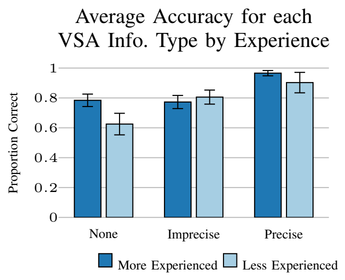


Fig. 10. The average accuracy of more and less experienced participants for each VSA information type.

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.

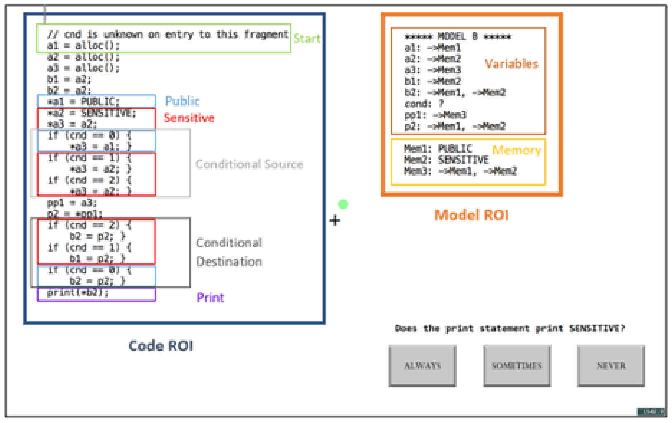
Figure 8 shows the average accuracy and RTs for each VSA information type, broken down by problem difficulty. A two-way repeated measures ANOVA showed that accuracy was significantly affected by VSA information type ($F(2,95) = 15.09$, $p < 0.001$) and by difficulty level ($F(1,95) = 11.74$, $p < 0.01$). 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 response times on the more difficult problems relative to the 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 between easier and more difficult problems. Paired t-tests confirmed this pattern, showing a significant difference in accuracy between the easier and more difficult problems when no VSA information was shown ($t(19) = 4.16$, $p < 0.001$), but not when imprecise ($t(19) = 0.92$) or precise ($t(19) = 0.81$) VSA information was shown. 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 the more difficult problems, making participants' accuracy on the difficult problems comparable to their accuracy on the easier problems.

We also assessed the interplay between problem type and VSA information type (Figure 9). For participants' accuracy, 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 task performance, we compared the average 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 of participants was eliminated when either precise VSA information ($t(9) =$



```
// cnd is unknown on entry to this fragment
a1 = alloc();
a2 = alloc();
a3 = alloc();
b1 = a2;
b2 = a2;
*a1 = PUBLIC;
*a2 = SENSITIVE;
*a3 = a2;
if (cnd == 0) {
     *a3 = a1; }
if (cnd == 1) {
     *a3 = a2; }
if (cnd == 2) {
     *a3 = a2; }
pp1 = a3;
p2 = *pp1;
if (cnd == 2) {
     b2 = p2; }
if (cnd == 1) {
     b1 = p2; }
if (cnd == 0) {
     b2 = p2; }
print(*b2);
```

```
***** MODEL 8 *****
a1: ->Mem1
a2: ->Mem2
a3: ->Mem3
b1: ->Mem2
b2: ->Mem1, ->Mem2
cond: ?
pp1: ->Mem3
p2: ->Mem1, ->Mem2

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

Does the print statement print SENSITIVE?

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

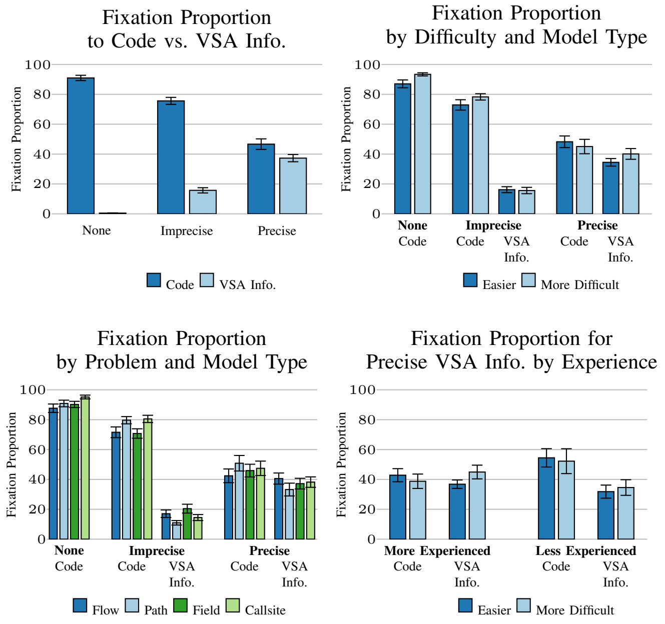


Fig. 12. The average proportion (percentage) of fixations to the Code and VSA Information Regions of Interest (ROIs) for each VSA information type condition, additionally broken down by problem difficulty, by problem type, and, for the precise VSA information condition only, by experience.

0.89) or imprecise VSA information ($t(17) = 0.51$) was available. 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 that impacted the eye tracking data from five participants, only 15 participants (8 experienced participants and 7 less experienced participants) were included in the eye tracking analysis. Human eye movements consist of saccades (fast movements of the eyes from one location to another) and fixations (periods when the eyes are relatively stationary). Virtually all visual information processing takes place during fixations [26], [27]. In our analysis, we used the default algorithm in the EyeWorks software to calculate fixations, with one degree of visual angle (estimated to be 52 pixels based on the distance between the participants' eyes and the computer monitor in our setup) used for the spatial parameter. We divided each stimulus into regions of interest (ROIs) for the eye tracking analysis. At the highest level of analysis, we defined two ROIs: one for the code, on the left side of the screen, and one for the VSA information, on the right side of the screen. We also conducted a more fine-grained analysis in which we subdivided the code and VSA information ROIs into smaller regions. We split the VSA information region into a Variables ROI and a Memory ROI. For the Code ROI, we identified seven possible subdivisions: Start, Memory Initialization, Conditional Source, Conditional Destination, Sensitive, Public, and Print. Not every stimulus had every ROI, and the code ROIs could overlap with one another. Each line of code was included in all relevant ROIs for that line. Figure 11 shows example ROIs for one stimulus. A more detailed explanation of the ROIs is provided in the supplemental materials.

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 [15], [38]. 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.

When we took problem difficulty level into consideration, 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 once again that when precise VSA information was available, participants had a lower proportion of fixations to to 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 ($F(3, 56) = 2.99$, $p < 0.04$). Post-hoc paired t-tests showed a significantly higher proportion of fixations to the imprecise VSA information for field problems than for callsite ($t(14) = 2.29$, $p < 0.04$) or path problems ($t(14) = 3.49$, $p < 0.01$). They also showed a significantly higher proportion of fixations to the imprecise VSA information for flow problems than for path problems ($t(14) = 2.64$, $p < 0.02$). No other comparisons reached significance (all ts< 1.52, all ps> 0.07).

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 reduction in the 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

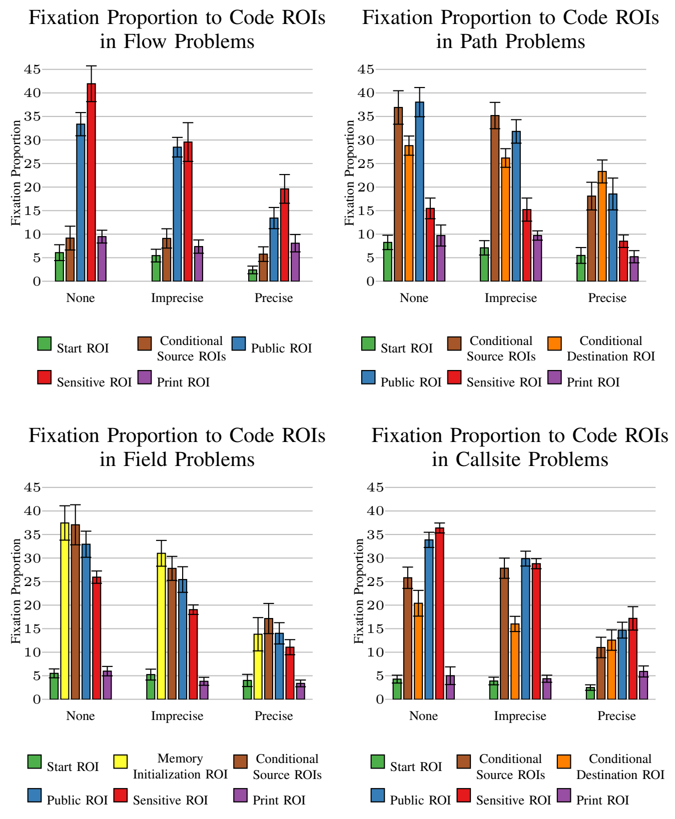
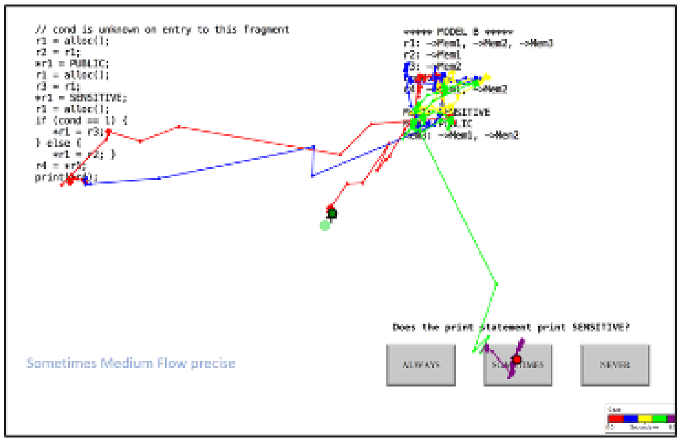


Fig. 13. The average proportion of fixations to the line-by-line ROIs in the Code region for the four different problem types. The callsite problems are shown in (a), the field problems in (b), the flow problems in (c) and the path problems in (d).

conditional destination is critical to determining whether or 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. Not only did the VSA information reduce 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



```
// cond is unknown on entry to this fragment
r1 = alloc();
r2 = r1;
*r1 = PUBLIC;
r1 = alloc();
r3 = r1;
*r1 = SENSITIVE;
r1 = alloc();
if (cond == 1) {
   *r1 = r3;
} else {
   *r1 = r2; }
r4 = *r1;
print(*r4);
```

```
***** MODEL B *****
r1: ->Mem1, ->Mem2, ->Mem3
r2: ->Mem1
r3: ->Mem2
r4: ->Mem1, ->Mem2
Mem2: SENSITIVE
Mem1: PUBLIC
Mem3: ->Mem1, ->Mem2
```

Does the print statement print SENSITIVE?

Fig. 14. A plot of the gaze data for one participant on one trial. The green circle represents 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.

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 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 fixating on 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 s ($SD = 11.28s$) for the Print ROI. 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 s ($SD = 6.41s$), slightly longer for the VSA Information ROI at 9.49 s ($SD = 7.18$), and longest for the Print ROI at 19.27 s ($SD = 17.20$). This indicates that participants often started reading the code from the top and glanced back and forth between the code and the VSA information as they worked their way through the code to the bottom. However, the overall averages obscure the fact that different participants used different strategies. While seven participants followed the pattern that was observed when averaging across all participants, eight participants did not. A group of four participants consistently fixated on the VSA Information ROI first, then fixated on the Print ROI, and fixated on the Start ROI later. Once again, all four of those participants were in the highly experienced group. Another two of the highly experienced participants tended to fixate on the Print ROI first, followed by the Start ROI, with their first fixation on the VSA Information ROI coming later, after they had worked their way backwards through the code. Finally, two participants, both of whom were among the least experienced, had unique patterns. One of these participants tended to fixate on the VSA Information ROI first, followed by the Start ROI and then the Print ROI. The other tended to fixate on the Print ROI first, then the VSA Information ROI, followed by the Start ROI.

The average times to first fixation in our three regions of interest was 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 s ($SD = 2.97$), somewhat longer for the Start ROI at 9.83 s ($SD = 9.32s$), and longest for the Print ROI at 15.14s ($SD = 17.12s$). A total of 11 participants (six highly experienced and five less experienced) consistently fixated on the VSA Information ROI first. Seven of those 11, 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 this 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 [37], so it is likely that the additional time spent on imprecise VSA information 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

[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 [19], or because their analyses are time-limited and they can identify and respond to classification errors later as needed. Exploring this would be interesting future work.

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, 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 was nearly eliminated as well.

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 response times when there was no VSA information or imprecise VSA information, but providing a 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 the 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.

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. The availability of precise memory VSA information changed the ways in which participants worked through the problems and led to several of the less experienced participants adopting 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 in which situations accuracy is important within the realistic, long time scales required by common reverse engineering tasks today.

Acknowledgments

The authors would like to thank Breannan Howell, David Hannasch, Denis Bueno, our editors, and the RAMSeS team and collaborators for their great ideas and support. The authors would also like to thank the reverse engineers who made time to be our subjects for this study.

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.

References

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

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

[3] ——, "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

[4] 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.

[5] S. Becker, C. Wiesen, N. Albartus, N. Rummel, and C. Paar, "An exploratory study of hardware reverse engineering — technical and cognitive processes," in *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*. USENIX Association, Aug. 2020, pp. 285–300. [Online]. Available: https://www.usenix.org/conference/soups2020/presentation/becker

[6] 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.

[7] 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.

[8] 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

[9] 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.

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

[11] 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

[12] 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.

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

[14] 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

[15] 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

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

[17] 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.

[18] 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.

[19] ——, "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.

[20] 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

[21] 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.

[22] 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.

[23] 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

[24] 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.

[25] 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

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

[27] ——, "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

[28] 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.

[29] 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.

[30] 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.

[31] 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

[32] 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.

[33] 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

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

[35] 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

[36] 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

[37] 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

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

[39] 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.

[40] 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.