Software Vulnerabilities as Cognitive Blindspots; assessing the suitability of a dual processing theory of decision making for secure coding

MATTHEW IVORY, Lancaster University, UK
JOHN TOWSE, Lancaster University, UK
MIRIAM STURDEE, St. Andrews University, Scotland
MARK LEVINE, Lancaster University, UK
BASHAR NUSEIBEH, City St George's, University of London, UK

Software vulnerabilities are present in many software systems, putting people who entrust software with their data in harm's way. Many vulnerabilities are avoidable since they are well documented - yet they remain widespread. One explanation for their persistence is they represent software blindspots, problems that are implicit in the mental models of developers and which escape attention (Brun et al., 2023). Our current understanding of how attention and decision making influence specific secure coding behaviours is limited, and so we present a preregistered study to evaluate whether differences in decision making style impact blindspots and the identification of code vulnerabilities. Programmers were given code puzzles to complete, including some that contained vulnerabilities. Participants also competed the cognitive reflection test and measures of rational decision making. We replicate several key predictions from previous blindspot research, map the analysis onto dual-systems research, and describe effect sizes of psychological constructs. We then model data simulations to demonstrate the sampling required for highly powered empirical studies in this domain. We support previous findings that technical or cybersecurity expertise have little impact on the ability to detect vulnerabilities. We argue that dual processing theory helps to interpret security behaviours and the presence of software blindspots.

CCS Concepts: • **Security and privacy → Software security engineering**.

Additional Key Words and Phrases: security, cognitive psychology, dual processing theory, code comprehension, blindspots

ACM Reference Format:
Matthew Ivory, John Towse, Miriam Sturdee, Mark Levine, and Bashar Nuseibeh. 2026. Software Vulnerabilities as Cognitive Blindspots; assessing the suitability of a dual processing theory of decision making for secure coding. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 19 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Software vulnerabilities are a pervasive issue within software engineering, and they can have serious real-world consequences, such as impact on urgent medical care [46] or psychological harm [35]. Many of the vulnerabilities detected in modern software are not new. Well-known vulnerabilities are detected in over three-quarters of software, indicating that despite their notoriety, they remain unhandled despite the tools available to software developers. To

Authors' Contact Information: Matthew Ivory, matthew.ivory@lancaster.ac.uk, Lancaster University, Lancaster, UK; John Towse, Lancaster University, Lancaster, UK; Miriam Sturdee, St. Andrews University, St Andrews, Scotland; Mark Levine, Lancaster University, Lancaster, UK; Bashar Nuseibeh, City St George's, University of London, London, UK.

Manuscript submitted to ACM

address this, we offer support to the hypothesis that vulnerabilities exist as cognitive blindspots in decision making during secure coding.

Secure coding is the practice that ensures software "does not contain known vulnerabilities" [39], and vulnerabilities are defined as unexpected logic flows which create compromising software behaviours, allowing for unintended access to information or functionality [19]. It is observed that technical expertise, experience, and security knowledge are poor indicators of predispositions towards secure coding [4, 32]. Acknowledging that technical expertise has minimal effect on producing secure code, this study focuses on software engineers' cognitive capabilities. We explore the role of human decision making and cognitive psychology to understand individual differences in secure coding predispositions better. By treating software engineers as a diverse population displaying individual cognitive differences, and who must satisfice [45], we can better understand how cognition influences vulnerability detection. One theory is that software vulnerabilities are systematically missed by developers because they occupy 'blindspots' in their attention.

A blindspot is code where the expected behaviour of a function diverges from the intended behaviour, and software vulnerabilities exist because of these blindspots [6]. Oliveira et al. [33] tested the hypothesis with measures of working memory and cognitive processing speed. They asked engineers to solve short Java programming puzzles that contained either a blindspot in the form of insecure API use or no blindspot. They found puzzles without a blindspot were solved significantly more often than puzzles with a blindspot, but neither working memory nor processing speed affected accuracy. Brun et al. [4] replicated this in Python, finding only a small effect of long-term memory. This body of research explored general measures of cognition but did not explore *why* cognitive blindspots exist in security contexts. We propose to address this issue by applying the dual processing theory of decision making, which posits people make decisions using one of two systems, an intuitive default system and an interventionist computational system [11] towards the same tasks.

To this end, we ask the question, **How does dual processing theory account for software developers' ability to detect security blindspots in code?**

We extend previous findings that blindspots contribute to impaired code comprehension [4, 6, 33] by applying a dual processing theory of decision making. Dual processing theory posits individuals make decisions based on one of two distinct systems, a default intuitive system that relies upon heuristics, and a second system that uses all available information to make reasoned choices [12]. In taking the position that decision making processes are not homogeneous and that individual differences are present in general populations [13], we seek to test whether this heterogeneity influences secure coding. Dual processing theory has not been widely applied within software engineering research despite having a proven influence on performance in general judgments [52]. Only one known study has explored dual processing theory in software engineers that found a complex relationship between risk sensitivity and system 2 engagement [19]. Moreover, relevance to software behaviours is unknown, which is addressed in this paper.

This study offers a greater connection between software engineering and the individual differences observed in decision making. We successfully replicate findings that blindspots are difficult to detect, and that detection cannot be explained by technical knowledge or experience. The evidence for dual processing effects is variable, primarily due to a requirement for larger participant samples in future work. Importantly we do not reject dual processing theory's value for explaining secure coding.

The paper is structured as follows. In section 2, we provide essential background content relevant to the theoretical underpinnings of this research. Section 3 details the hypotheses, relevant methodological decisions, measures, and procedures for research transparency. We report our results in section 4, which are discussed in terms of their theoretical support and implications for software engineering practice and research in section 5. To contextualise our study within

the wider software engineering literature, we explore related work in section 6. Section 7 summarises the study's contributions.

2 Background

This paper defines *cognitive blindspots* as units of information that are systematically overlooked, typically through unconscious or intuitive approaches to making judgments, resulting in non-optimal decisions. Blindspots are due to the human disposition to deploy decision making *heuristics* (mental short-cuts). Where heuristics lead to fallacies or erroneous thinking, they create biases. These are systematic, flawed judgments that deviate from ideal performance [10].

Biases are present throughout the whole software development process [38], and their influence can result in non-optimal decisions being made. In the example of tool selection, developers tend to use familiar tools [43], even if they are not appropriate for specific tasks [1]. No single tool can detect all software vulnerabilities meaning that a familiarity bias can result in non-optimal decisions [9].

Human working memory is sharply limited [2], creating the potential for overload. Heuristics help manage the cognitive complexity of the decision space by rendering the problem down to a simplified form [3]. However, because the mental representations of the task are now partial, incomplete, or non-exhaustive [22], important facets can become obscured to the mind's eye, creating cognitive blindspots. From a psychological perspective, this means that vulnerabilities that are known "in principle" to a developer are hidden from view during the software task.

The presence of heuristics, biases, and attentional blindness all point towards the presence of more than one system of cognitive information processing, because we recognise different approaches to decision making, where choices can be made intuitively and unconsciously, or we can exert effort and make more careful, reasoned choices. These differences can be explained by the dual processing theory.

The default-interventionist psychological model of dual processing theory proposes two cognitive systems [12]: the default and *intuitive* system 1 and a more intentional and *rational* system 2 that is only engaged when sufficiently cued [8]. System 1 processing is the default style of decision making, driven by heuristics to reduce complex judgments into simpler cognitive operations [23]. In contrast, system 2 processing is more deliberate but computationally demanding and is typically deployed when individuals seek an optimal solution by using all available information. System 2 is the interventionist mode, and it only overrides System 1 when an individual consciously or unconsciously sees a greater need for accuracy. System 1 is liable to generate simpler, less complete mental models than System 2 [21], and blindspots can reside in the created gaps.

3 Related Work

This study builds upon a previous programme that posits software vulnerabilities are often missed by engineers as the vulnerabilities occupy cognitive blindspots [6]. Support for this idea came from Oliveira et al. [34], who tested the effect of priming engineers to review data for security. Following this, other studies explored cognitive aspects of blindspot detection using Java [33] and Python [4]. Both studies supported the phenomenon, emphasising the underlying hypothesis that software vulnerabilities can be difficult for engineers to detect. Our research extends the paradigm by applying dual processing theory [11] and measures of system 1 and system 2 processing to the hypothesis. The findings also reinforce the message that psychological, not just technical, variables impact coding quality [39]

3.1 Cognition and Software Engineering

Within software security, an increased focus has been placed on the cognition and behaviours of those involved in software creation [14]. Security is perceived as a reactive, event-driven process [26] with little priority over functionality [25]. The reduced perceived importance of security and its temporal association with specific events (as opposed to procedural, proactive workflows) means that it is less likely to be at the forefront of an engineer's mind during decision making, aligning with the dual processing theory.

Using prompts to encourage secure coding is of some benefit. Prompting can be the cue required to engage system 2 processing and override or suppress system 1. Simple interventions of requesting security have demonstrated their power in improving security with a broad range of populations, including students, freelancers, and professional developers [30–32]. Prompting can also work more subtly, such as asking engineers to write design specifications before writing secure code [16].

Biases within software security have been explored previously and help in explaining many common issues [28]. One commonly reported bias is optimism bias, which is found in many forms, such as underestimating time requirements for projects [29], downplaying security threats [27], or reduced concern over software security [19]. Other biases include confirmation bias, where individuals seek information that confirms internal hypotheses rather than disprove them [48]. Confirmation bias is seen as an issue in software testing [41], as engineers may be less motivated or capable of applying non-confirmatory tests on edge cases, where vulnerabilities often lie. Biases result from system 1 processing, and their noted prevalence in software engineering research highlights just how common this processing is.

Research into cognition and psychology surrounding software engineering has explored ideas relevant to understanding security behaviours. We contribute to this existing research by applying a theory of decision making that aligns with work by others [4, 6, 33], highlighting its potential to explain behaviours observed surrounding software vulnerabilities.

We present the first study into secure software development that explicitly uses dual processing theory to interpret the results. Within software engineering, the theory's potential has been discussed [see [36, 37, 40]], but it has not yet been empirically tested.

4 Methodology

This study is motivated to establish the viability of dual processing theory in secure software engineering research. A sample of 37 participants was used to test measures of dual processing, as well as to simulate further data and conduct power analyses to determine suitable effect sizes for future research.

4.1 Hypotheses

The study was preregistered, lodging the design and hypotheses in advance, which can be found here: https://doi.org/10.17605/OSF.IO/CE78G.

To facilitate conceptual organisation, we have reordered and mapped the original hypotheses into three distinguishable groups, each addressing different parts of the main research question, *how does dual processing theory account for software developers' ability to detect security blindspots in code*? The first two hypothesis exists to test the effects found previously by Brun et al. [4] and Oliveira et al. [33] and is captured under its own grouping of whether the previous findings replicate.

1. Python puzzles containing API blindspots will be more difficult to solve than scenarios without blindspots correctly.

6. Developers' perceived ratings of puzzle difficulty, effort, familiarity, and confidence will affect their ability to solve scenarios containing blindspots. This hypothesis replicates the finding from Brun et al. relating to the self-reported measures of puzzle interaction.

The following four hypotheses are grouped as they test different facets of dual processing theory [11] and specifically assess whether differences are seen between individuals who exhibit higher or lower propensity towards rational decision-making. In hypothesis 2, the propensity to engage in rational decision-making can be measured using the cognitive reflection test [13], a measure of cognitive reflection (see section 4.3.1). Hypothesis 3 uses optimism bias as a proxy for a tendency towards system 1 processing. Hypothesis 4 and 5 use scales of rational and intuitive decision making to test the strength of an individual's tendency towards either cognitive system.

2. Developers with higher levels of cognitive reflection will solve scenarios with blindspots more effectively than developers with lower cognitive reflection.

3. Developers demonstrating realistic levels of optimism will solve scenarios with blindspots more effectively than developers with higher levels of optimism.

4. Developers with higher levels of rational decision making will solve scenarios with blindspots more effectively than developers with lower levels of rational decision making.

5. Developers with lower levels of an intuitive decision making style will solve scenarios with blindspots more effectively than developers with higher levels of intuitive decision making.

Finally, the remaining two hypotheses assess another aspect of dual processing, that if vulnerabilities exist as a consequence of decision-making then experience and expertise will have no influence on the ability to detect blindspots. Hypotheses 7 and 8 are based upon previous findings that experience and declarative knowledge do not modulate vulnerability detection, which aligns with the proposed theory that security vulnerabilities occupy a cognitive blindspot when interacting with software code.

7. Developers with more experience and familiarity with programming and Python will show no difference in their ability to solve scenarios with API blindspots.

8. Developers with more cybersecurity knowledge and experience will show no differences in their ability to solve scenarios with API blindspots.

4.2 Participants

A sample of 37 participants provided sufficient statistical power to test Hypothesis 1 and to generate simulated data for estimating the sample sizes needed for future psychological research in software engineering. Participants were recruited from Upwork.com, a freelancing website in late 2023. Adverts invited freelancers to submit proposals for a "Code Review". Five listings were placed (when one stopped receiving proposals, it was archived, and an identical advert uploaded). Participants confirmed they were over 18, had at least one year's experience with Python that was not limited to a university course, had experience with API functions, and asked to describe a recent Python project. The advertising materials and information sheet described the study focus to be measuring general code comprehension rather than security to avoid priming participants to actively look for security issues. The Faculty-level ethics committee approved this project. Participants received $28 as compensation.

In total, 126 proposals were submitted, and 89 were rejected. Rejection criteria were not formalised before data collection to ensure only high-quality participants were selected. Examples of rejected proposals included suspicion

Table 1. Demographic breakdown of participant ethnicity

Ethnicity	Count
Asian	13
Black	1
Mixed two or more ethnic groups	1
Other	2
White	20

of duplicate accounts (with two or more different applicants giving almost identical answers and no prior Upwork experience), use of AI-generated content for screening questions (determined via online checkers or manual inspection), a failure to mention appropriate experience, a lack of English fluency (assessed from responses and further communication), or failure to convince the primary researcher that they fulfilled the criteria properly. In cases where it was unclear whether participants were to be approved, further communication was established to determine whether a proposal would be accepted. Upwork profiles, job history, and linked profiles (e.g., GitHub) were also used to determine eligibility.

The mean age of the 37 participants was 29.11 (SD = 9.55), ranging from 19 to 71. One participant preferred not to report gender, 35 reported male, and one self-described as "freelancer". Participant ethnicity is reported in Table 1, with the vast majority being White or Asian. Two participants had no education beyond high school level, two had some university experience, 18 had an undergraduate degree, 3 had some postgraduate university education, and 12 had postgraduate degrees. The average experience in general programming was 6.05 years (SD = 4.67), and Python experience was less at 4.39 years (SD = 2.46).

4.3 Materials

The study used a quantitative behavioural design in which code puzzles were varied as a within-participants manipulation and individual differences assessed through established measures. Measures of dual processing theory include the cognitive reflection test [13] and the rational and intuitive subscales of the general decision making style scale [42], which are described below. Cognitive reflection has been identified to play a role in explaining risk awareness around security in populations of software developers [19], and the rational decision making subscale has been identified as a predictor of good users' behaviours [15].

4.3.1 Cognitive Reflection Test. The Cognitive Reflection Test (CRT) is a short test designed for measuring cognitive reflection, an individual's tendency to engage in system 2 processing during decision making [13]. The questions offer intuitive but incorrect responses, and only through system 2 engagement is the correct answer identified. An example question is, "A bat and a ball cost $1.10 in total. The bat costs $1.00 more than the ball. How much does the ball cost?". The intuitive and incorrect response of 10 cents is the most common answer [13], instead of the correct answer of five cents, implying that system 1 processing is the default processing style.

We used two cognitive reflection tests, CRT and CRT-2. The CRT was initially developed by Frederick [13], and the CRT-2 by Thomson and Oppenheimer [51]. The second version was designed to reduce the numerical nature of the original test. In the present study, the wording was altered, so attempts to find answers via online searches would be more difficult, but the question design remained unaltered. Participants were also asked whether they had seen the questions before to measure whether they may have been responding based on previous experience; however, this is not a significant concern as research has shown CRT performance is stable over time [49]. Response times were recorded.

4.3.2 OWASP Vulnerability Test. The OWASP Vulnerability Test (OVT) was used as a domain-specific measure of optimism bias for software engineers [19]. This measure consists of two sections, and the first asks participants to estimate the likelihood of the *average developer* producing software containing one of the top five OWASP vulnerabilities (injection flaws, broken authentication, sensitive data exposure, XML external entity flaws, and broken access control). They are presented with five questions, one for each vulnerability, in a randomised order. Following a separation task, implemented to minimise recall of previous answers, participants are asked about the same vulnerabilities but to report the likelihood of *themselves* incorporating these vulnerabilities. This test measures comparative optimism, as previously used in other research [17].

4.3.3 General Decision Making Style Scale. The General Decision Making Style Scale (GDMS) measures five styles of decision making [42]: rational, avoidant, dependent, intuitive, and spontaneous. These subscales assess an individual's decision making approach. The 25-question scale is presented in a randomised matrix with a five-point Likert response (strongly disagree – strongly agree). For the hypotheses, only the rational and intuitive subscales were of interest, as they are theoretically linked to dual processing theory, with the intuitive scale mapping against system 1 processing and higher levels of rational decision making mapping onto system 2.

4.3.4 Puzzles. The Python puzzles were developed by Brun et al. [4]. They consist of 10-21 lines of code with an accompanying scenario description providing context to the puzzle. An example of a scenario and puzzle is shown in Figure 1. Participants read the scenario and code and answered a free-text question about the behaviour once the code was executed. They are then asked a multiple-choice question of expected behaviour given specific inputs. Following this, they were asked to rate their confidence in solving the puzzle from 1-10, the percentage of others they would expect to solve the puzzle (from 1-100), perceived difficulty (from 1-10), familiarity with the functions (from 1-10), and scenario clarity (from 1-10). To aid the deception, they were asked whether any parts of the puzzle were confusing and how, whether there were unfamiliar functions, and to list these. They were asked what resources they used to review the code and, finally, to report the fatigue they experienced in completing the puzzle (from 1-10). Puzzles were scored as being correct or incorrect based on the multiple-choice question.

We presented participants with eight puzzles, six with blindspots and two without blindspots. Five of the six blindspot puzzles focused on input/output vulnerabilities, including networking activity and reading and writing to and from streams, files, and internal memory buffers. The sixth was a string manipulation vulnerability of user input. Historic software vulnerabilities illustrate the various ways that systems have been compromised. Injection attacks, such as the 2008 Heartland Payment Systems SQL injection that revealed how unsanitised inputs can give attackers direct control over backend databases. Buffer overflows, such as the 1988 Morris Worm, demonstrated how exceeding memory boundaries allows arbitrary code execution and has remained a persistent threat across decades of software. File-handling vulnerabilities, including those exploited in the 2017 Apache Struts breach, showed how improper validation of uploaded or parsed files can lead to remote code execution. Finally, weaknesses in cryptographic protocols, most notably the 2014 Heartbleed bug in OpenSSL, highlighted how subtle implementation errors in SSL/TLS can expose sensitive data and undermine trust in secure communications.

Puzzles were presented in two blocks, allowing participants to take a break in between. Each block had three blindspot puzzles and one non-blindspot puzzle and were balanced in terms of cyclomatic complexity. Table 2 shows the puzzle order, type of vulnerability and cyclomatic complexity. The complete set of puzzles used, their order, and associated questions can be found in the online repository here: https://osf.io/2r4zx/overview?view_only=

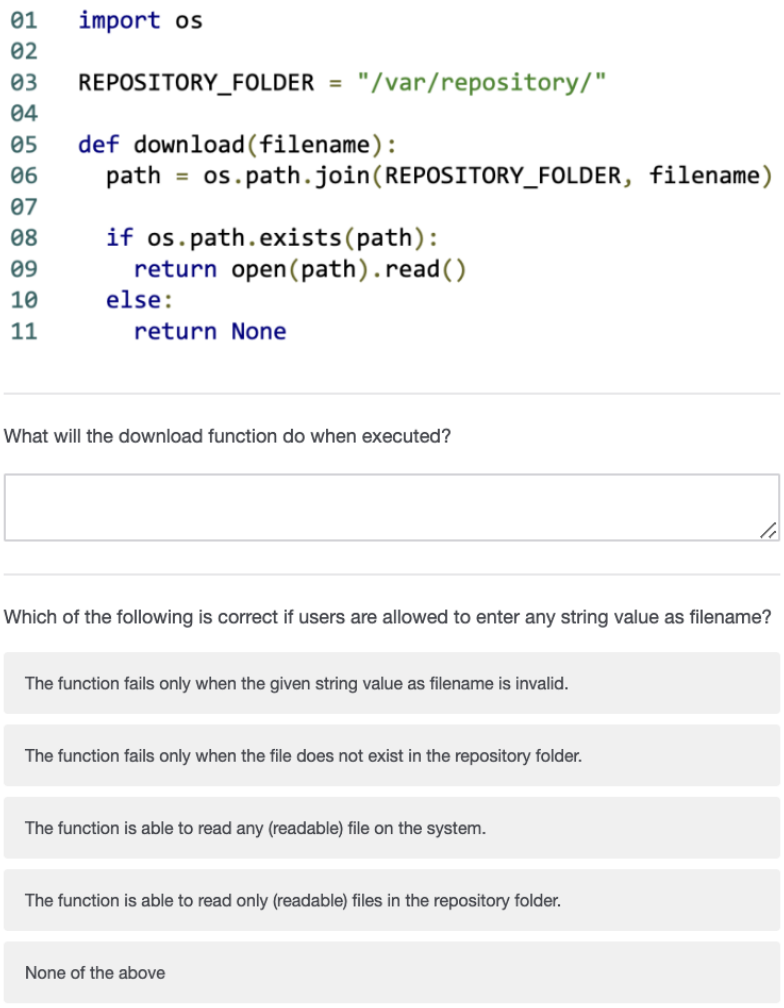

Fig. 1. An example of a puzzle containing a blindspot. All puzzles were formatted similarly, with the context given first, followed by an image of the code, and then asked to describe the code's behaviour. Each puzzle was prefaced with the scenario as context. The context would explain the general setup and use of the code section and provide examples, if necessary, of its use case. It also noted that readers should assume all necessary permissions are given for execution.

28e405da65254829ba11f0cbddfc14ef. This contains the code along with descriptions of the blindspots behaviour in the code.

4.4 Procedure

Following participant approval participants were sent an Upwork contract and an attached information sheet. Participants were not told of the specific goal of examining blindspots in puzzles until data collection was complete.

Table 2. The presentation order and attributes of the Python puzzles presented to participants. The puzzle ID relates to the original ID given in Brun et al. (2023) and cyclomatic scores are a measure of code complexity with higher scores indicating more difficult code to comprehend or modify. Puzzles 3 and 7 have no vulnerability and are used as stimuli to test differences between vulnerable and non-vulnerable code.

Puzzle	ID	Vulnerability Type	Blindspot Type	Cyclomatic Score	Number of Lines
1	P02	Injection	Validation Missing	5	10
2	P09	Injection	Validation Missing	3	11
3	PX06	Injection	–	2	21
4	P21	Overflow	Function Misuse	2	16
5	P30	File	TOCTTOU	3	10
6	P36	File	Validation Missing	4	7
7	PX15	SSL	–	2	5
8	P31	File	Missing Verification	3	7

Participants were sent an online Qualtrics link and could only participate if they provided full informed consent. Participants completed the eight Python puzzles. They then completed demographic questions of age, gender, ethnicity, employment, and education. They were then asked to report general and Python-specific programming experience.

After this, the cognitive measures were presented in the following order: first OVT, CRT, GDMS, second OVT. The two OVT sections were split to reduce the likelihood of participants simply recalling and repeating their initial answers. All questions within each measure were presented in a randomised order. Following this, cybersecurity questions were asked, reporting how much of their knowledge was self-taught or formally taught and the frequency with which they were required to employ it. They were then debriefed, which explained the study's true purpose, followed by explanations of the puzzle vulnerabilities to ensure participants were aware of these blindspots. This concluded the study.

4.5 Analysis

Analysis was implemented through R 4.2.2, and the data and analysis scripts can be found here: https://osf.io/2r4zx/overview?view_only=28e405da65254829ba11f0cbddfc14ef.

Mixed-effect ordinal logistic regression (MOLR) and general linear regression models were used. Where independent variables varied across puzzles within participants (e.g., confidence of solving puzzles), MOLR models with a dependent variable of puzzle accuracy and random effect of puzzle were used to control to account for the nested data structure. Where independent variables did not vary, linear regression models were used with a dependent variable of total puzzle accuracy. The analytic models reported here match those employed by Brun et al. [4]. As with the analysis by Brun and colleagues, Bayes factors were used to assess non-significant terms in models to determine whether the absence of an effect is due to data insensitivity (a value between .33 and 1) or a preference to accept the null hypothesis (a value under .33). The GDMS rational and intuitive scores were calculated by summing question responses (strongly disagree = 1; strongly agree = 5) to obtain a single score for each subscale between 5 and 25 [47].

Following this, we conducted power analyses by simulating new data to understand the minimum number of participants for future confirmatory work and further explore the cognitive measures that are significant predictors with larger samples. Data from participants were used to simulate 740 new observations by adding random noise (±1 SD) to independent variables, truncated to variable limits. Dependent variables (accuracy) were unchanged to preserve observed patterns. Statistical power was estimated from 1,000 model iterations using random subsets of simulated

Table 3. Percentage of correctly solved answers compared to the sample from Frederick (2005)

Score	Frederick (2005)	Present Study
0	33%	0%
1	28%	5.41%
2	23%	16.21%
3	17%	78.38%

data ranging from 10 to 740 participants. From this, power was calculated as the percentage of models that provided significant terms at the .05 threshold. H_1 was excluded, as 37 participants provided sufficient power per preregistration. Power calculations were conducted using a High-End Computing Cluster due to their computationally intensive nature. These calculations were performed using R 3.6.0. Scripts, data, and associated files for reproducing these calculations are found at https://osf.io/2r4zx/overview?view_only=28e405da65254829ba11f0cbddfc14ef.

5 Results

The main hypotheses were tested using the data collected from the 37 participants, and then all the hypotheses bar H_1 were subject to power analyses. H_1 was predetermined to be suitably powered using the data provided by Brun et al. [4].

5.1 H_1: The Effect of Blindspot on Accuracy

To address the hypothesis that Python puzzles with an API blindspot will be more challenging to solve than those without, a MOLR model was used, regressing the presence of blindspots against accuracy with a random effect of participant. The data did not support the full model, so the random effect was removed, resulting in the simpler model, which was supported by the Bayesian Inference Criterion (BIC). The revised model had a BIC value of 368.12 compared to the original 373.81, indicating a preference for this simpler model.

The revised model was found to be significant, with the blindspot presence having a coefficient β of -1.26, $p < .001$, giving an odds-ratio of .28, indicating that if a puzzle possesses a blindspot, the odds a participant solves the puzzle incorrectly is 3.53 times more likely than solving it correctly. Participants in the present study were more than twice as likely to solve puzzles without blindspots than reported in Brun et al. [4].

5.2 H_2: The Effect of Cognitive Reflection

Comparing the descriptive profile of CRT scores with the original findings by Frederick [13] in Table 3, our responses are very different to the original findings, with a large majority successfully answering all questions. It is unclear why participants scored so highly on measures that do not typically elicit high performance, but it may be due to the use of generative AI language models providing answers. Testing this potential cause with ChatGPT found that six of the seven questions were correctly solved. As such, our findings should be treated cautiously as they may not reflect the intended measure.

To address H_2, that higher levels of cognitive reflection will associate with solving scenarios with blindspots more effectively, a linear regression using both CRT and CRT-2 was deployed. Model examination suggested removing the CRT term, leaving only CRT-2 as a predictor. BIC values indicated that the model using only CRT-2 was preferable. The model using solely CRT-2 was non-significant, $p = .077$. The Bayes factor for the model using both CRT variables compared to the null is .21, indicating a preference for the null hypothesis rather than data insensitivity, but for the

model with just CRT-2, the Bayes factor was .93, suggesting that data insensitivity is far more likely than a preference for the null hypothesis.

5.3 H_3: The Effect of Optimism and Confidence

To test the effect seen in Ivory et al. [19], that engineers are likely to see themselves as less likely to be susceptible to including vulnerabilities in their work, the mean average of the OVT task was 125.84 (SD = 17.88), with a minimum score of 95 and a maximum of 155. A score of 100 would indicate individuals scoring themselves as equally likely as the general engineering population, but the higher average score indicates that the optimistic self-belief is persistent across samples. A one-sample t-test confirms that the OVT scores significantly differ from an average score of 100, $t(36) = 8.79$, $p < .001$, with a large effect size $d = 1.441$.

We hypothesised that developers who demonstrate realistic levels of optimism will solve scenarios with blindspots more effectively than developers with higher levels of optimism, acting as a measure of system 1 processing. MOLR for accuracy scores with OVT, self-confidence, confidence in others, and random effects yielded just a significant effect of self-confidence. This indicated a less complex model was appropriate using just self-confidence and random effects, as the difference between the two models was not significant, $p = .792$.

The model with only self-confidence was significant, with a coefficient of .21, $p = .014$, equivalent to an odds ratio increase of 1.23, meaning that the odds of correctly solving a puzzle increase by 1.23 times for each unit increase in confidence. This finding indicates that self-confidence is an indicator of correctly solving puzzles containing blindspots. Testing the non-significant terms in the model, a Bayes factor of .01 was reported indicating that the null hypothesis is more likely and that optimism is not predictive of puzzle accuracy.

5.4 H_4: The Effect of Rational Decision Making

H_4 hypothesised that developers with higher GDMS rational scores would solve scenarios with blindspots more effectively. This was tested using linear regression, and the model was not significant, $F(1, 32) = 1.54$, $p = .224$, indicating that rational decision making does not affect the detection of vulnerabilities. The Bayes factor was .38, indicating data insensitivity.

5.5 H_5: The Effect of the Intuitive Style of Decision Making

H_5 expected that lower GDMS intuitive scores would associate with solving puzzles with blindspots more accurately. Linear regression did not support this, $F(1, 32) = .13$, $p = .726$. The Bayes factor was .18, indicating a preference towards the null and that H_5 cannot be supported likely due to no effect existing.

5.6 H_6: The Effect of Puzzle Attributes

H_6 tested the non-directional statement that developers' perceived ratings of puzzle difficulty, effort, familiarity, and confidence would affect their ability to solve scenarios containing blindspots. The MOLR model was found to be non-significant with no significant terms. The Bayes factor reported was .002, indicating a strong preference for the null hypothesis, suggesting that perceived ratings did not influence puzzle-solving ability.

5.7 H_7: The Effect of Expertise and Experience

H_7 stated that experience and programming ability would not affect puzzle solving, based upon previous findings, and the hypothesis sought to replicate the finding by Brun et al. [4] and Oliveira et al. [33]. Linear regression with total

puzzles solved as the dependent variable, with programming experience, Python experience, and general programming familiarity as predictors, was not significant, $F(3,30) = 1.83$, $p = .163$. The Bayes factor was .09, suggesting that no effect was more likely than data insensitivity. This supports our hypothesis and confirms findings by Brun et al. [4].

5.8 H_8: The Effect of Security Knowledge

H_8 hypothesised that cybersecurity knowledge and experience would have no effect on blindspot detection. Linear regression using cybersecurity experience and level of formal cybersecurity knowledge as predictors did not yield a significant effect, $F(5,28) = .41$, $p = .840$, and a Bayes factor of < .01 indicates a strong chance of no effect being present, supporting our hypothesis.

5.9 Simulated Samples for Power Analysis

Power calculations were derived from model simulations. Table 4 shows the required sample sizes needed to achieve a statistical power of .80 and the expected effect (the average coefficient within the regression model) achieved with appropriate power. Measures where no effect was detected, even in large samples included: hypothesis 3, where the effect for OVT and confidence tends towards zero, as was also seen for perceived effort in hypothesis 6. For hypothesis 8, the sample size is predicted to be over twice as large as the simulated sample, assuming a linear relationship between sample size and power. For all other measures used, they achieved statistical power within 740 participants, with seven of the remaining ten measures achieving power in 203 participants or less.

H_2: The large sample associated with the CRT suggests any effect is small and unlikely to have any significant impact, whereas CRT-2 is likely to be more effective as a measure, with a .37 increase in accuracy for every correct response to CRT-2.

H_3: A unit increase in perceived self-confidence was associated with a coefficient of .16.

H_4 and H_5: For the GDMS scale, a unit increase in rational thinking results in a .06 increase in overall accuracy, and no effect was detected using intuitive thinking.

H_6: For perceived difficulty, a unit increase in rating was associated with a coefficient of -.04, resulting in a minor decrease in accuracy. For familiarity with functions, a .07 coefficient was reported, relating to an increase in accuracy, and for self-confidence in a model with the terms of difficulty and familiarity, a coefficient of .13 was reported.

H_7: For Python experience, a small coefficient of .10 is seen, suggesting that for a unit increase in reported Python experience, overall accuracy increases by .10, and for technical proficiency, a -.12 decrease is seen in overall accuracy.

H_8: For cybersecurity experience, an effect of -.05 was detected, indicating a minimal decrease in overall accuracy for each unit increase in experience.

6 Discussion

We successfully replicated the finding that blindspot puzzles were challenging. This is true even for those that cause well-known vulnerabilities, such as code injection or missing input validation. The effect we obtained was statistically reliable and had a larger effect size than in the Brun et al. [4] paper from which the materials were derived. We also replicated the conclusion that technical expertise does not predict vulnerability detection. It also extended that study in two important respects - the potential mediating factors of psychological variables and the modelling of effect and sampling requirements for relevant research studies.

Table 4. Potential effect sizes and samples reported from the simulation and power analysis. * = this duplicated measure is the result of it being included in a second model from hypothesis 3, ** = the values for cybersecurity are based upon linear regression predictions and assume a linear relationship between power, coefficient, and sample size.

Hypothesis	Measure	Coefficient Size	Required Sample
2	CRT	-.18	698
2	CRT-2	.37	93
3	OVT	–	–
3	Confidence-others	–	–
3	Confidence-self	.16	63
4	Rational	.06	193
5	Intuitive	–	–
6	Difficulty	-.04	694
6	Effort	–	–
6	Familiarity	.07	186
6	Confidence-self*	.13	88
7	Experience	.10	144
7	Technical Proficiency	-.12	203
8	Cybersecurity Experience**	-.05	1517

6.1 Theoretical Support

We support the hypothesis proposed by Cappos et al. [6] by reporting on measures of dual processing systems in decision making. The power analysis indicates that dual processing theory may be associated with vulnerability detection. We saw no association for optimism bias susceptibility. For non-cognitive measures, self-confidence in solving puzzles was associated with higher accuracy, as was familiarity with code functions. The self-perceived difficulty was negatively associated. In contrast to Brun et al. [4], we saw a small positive effect for Python experience and a small negative effect for general proficiency.

The successful replication of Brun et al. [4] and Oliveira et al. [33] is valuable, but it generates further unanswered questions, specifically "What is it about blindspots that render software engineers less capable of identifying them?". From the cognitive science perspective, blindspots can be explained through decision making styles, and the implications that intuitive versus rational approaches influence the mental representation of the software problem. An intuitive approach (system 1 thinking) increases the likelihood that the coding problem is not fully represented in their mental model, thereby creating the blindspot itself. More analytic and algorithmic thinking is expected to identify the puzzle issues and form a more complete mental model that allows them to work toward a solution. Encouraging developers to create mental representations of software security can help them to draw inferences around its success [20].

We find this explanatory framework very appealing and plausible. However, the evidence from the current data is clearly somewhat mixed. The prediction from this above argument is that participants with higher CRT scores would solve more blindspot puzzles (by virtue of deploying more system 2 thinking). The data did not support this unequivocally. However, this may be due, at least in part, to the assessment of CRT performance, which may have been confounded by AI use.

This study's primary theoretical outcome is its support for the proposed paradigm of software vulnerabilities occupying blindspots in our cognition [6]. By applying the dual processing theory [11], we used system 1 and system 2 processing measures to explore the potential effects these have on detecting software vulnerabilities. When interpreting

results via dual processing theory, it is of little surprise that factors of technical expertise or cybersecurity had little impact. One's tendency to engage system 2 processing is a separate cognitive process to declarative knowledge, and the cue required to suppress system 1 is not linked to general intelligence [50]. The finding that higher levels of rational decision making styles tend towards increased detection, and higher levels of intuitive decision making tend towards decreased detection aligns with dual processing theory, highlighting that developers who tend towards more rational, system 2 processing styles of decision making are more likely to spot vulnerabilities.

We used two methods for assessing the data beyond the statistical modelling of the collected data: Bayes factors, and data simulation. We used Bayes factors to explore the likelihood of null results being a result of data insensitivity or that no effect is present. We also used simulated data to identify effects collected from a larger sample. These methods offer a greater insight into the data, mitigate the limitation of using a small sample size, and provide an enhanced understanding of the reported non-significant results. Agreement between the two methods was found for the two measures that have an effect: CRT-2 and rational decision making. Intuitive decision making has a weak Bayes factor of .27, and the evidence points towards a potentially small effect existing. For measures of function familiarity and self-confidence, the sample sizes identified via the power analysis (186 for function familiarity and 88 for self-confidence) suggest these are also worth exploring in further research with larger samples.

The OVT was used to measure domain-specific optimism bias and to collect information on the likelihood or strength of biases used during system 1 processing. Whilst an effect for overly optimistic views was seen for the sample in general, it did not associate with detecting blindspots in any significant pattern. This may be because, despite being domain-specific for software engineers, it is not specific enough when considering Python code. It may also be that whilst it is a measure of optimism bias, it does not adequately capture system 1 processing.

Many of the effects we identify are relatively small, indicating that solely employing engineers based on our findings would not solve security issues. The effects are more subtle and are indicative of the complexity of software engineering. Reducing these complex behaviours into easy-to-capture aspects of cognition would not be appropriate for determining who is assigned security tasks. However, they are an appropriate step forward in understanding how to support software engineers in identifying vulnerabilities.

6.2 Implications

The primary outcome of this research is that it confirms that further work using this paradigm and dual processing theory would benefit the software engineering community. Linking the theory to the paradigm with preliminary results also provides opportunities to explore targeted interventions for improving system 2 engagement when interacting with software code. Prompting for security has been shown to have a positive effect [16], but these changes are often short-lasting [53].

The indication that decision making styles may affect blindspot detection (as seen through the data simulation and power analysis) has implications for those involved in writing or reviewing code. The notion that increased rationality and decreased decision making are associated with greater detection of blindspots suggests that interventions should account for these differences. The findings *do not* suggest that those who report lower rational decision making styles are ill-suited for security tasks, as dual processing theory highlights a difference in the cue strength that causes system 2 to intervene during the decision making process.

Research from domains where decision making is critical, such as medical and clinical diagnostics, has previously explored potential interventions for enhancing rationality. These interventions include cognitive forcing strategies [7], which promote the development of internal models through learning metacognitive processes, recognising biases and

where they may occur, before applying mental or physical checks to ensure developers are accounting for these biases. For software engineering, an example of a cognitive forcing strategy would be to provide materials and workshops to teach engineers about decision making theories and heuristics and biases. Providing context for specific biases, such as confirmation bias during testing stages [41], and the importance of using metacognitive awareness of planning non-confirmatory tests to identify edge cases and vulnerabilities more easily can increase awareness over decision making approaches. Cognitive forcing strategies are naturally individual and account for individual differences in default decision making styles. Those typically more intuitive may favour physical checks such as checklists that encourage rational thinking. In contrast, more nuanced prompts may suit those predisposed to more rational styles.

Diverse perspectives may also be beneficial in engaging more reflective thinking styles. A catalyst for engaging system 2 processing can be peer communication [44], as it allows for greater exploration of potential viewpoints and reduces potential biases. Aligning with different social identities, such as those shared by software users, can enhance feelings of responsibility [18], which can also result in decisions being taken that account for others [24]. By acknowledging these views, developers may look at software code differently, allowing them to identify software vulnerabilities that would otherwise be missed.

6.3 Limitations

6.3.1 External Validity. The extent to which these findings generalize beyond the specific context of this study may be limited by the sample, which consisted of freelance software engineers which may not represent the broader software engineering community as a whole. The sample was collected online using Upwork, an online freelancing platform criticised for potentially including inexperienced or low-quality participants [30]. So, to ensure data quality, we used multiple resources to verify participants' experience as programmers, and they were asked screening questions about this experience and prompted for further details if initial responses were not satisfactory. Their freelance profiles were also checked for previous work and experience; GitHub or other linked public sites were checked or requested if missing, along with any other resources available to check suitability. Finally, applicants suspected of responding using AI-generated content were declined, along with those who could only provide information limited to academic settings, as student participants were not used. As a result, only 29.37% of applicants were accepted.

The study framing, intentionally, did not mention the security focus of our project, since doing so would have explicitly drawn attention to, or primed, security of the code. A potential critique of our outcomes is that participants did not suitably consider the code outside of the explicit constraints of the instructions. This would affect the ecological validity of the findings, if responses are not considered in the broader environment that code would normally exist in. However, we note that many real-world software challenges require the engineer to go beyond a clients' to-do list, and our instructions were merely silent as to the origin of any code issues. This would affect the ecological validity of the findings, if responses are not considered in the broader environment that code would normally exist in. The design of the puzzle responses was multiple-choice which ensured that there is only one accurate answer to each puzzle. If the blindspot was missed, then there was another inaccurate answer that would be appealing towards the participant, whereas when the blindspot was identified then the correct answer would become more obvious. The study design also aligns with dual processing theory, in that participants could not have been made aware of the security focus, as this would prime them towards looking for security issues.

6.3.2 Internal Validity. Although the study design aimed to minimise confounding factors, certain threats remain. Specifically, the use of LLMs, could have affected performance. This work was conducted at a very early stage of public

exposure to large language models (LLMs). At the time of data collection, awareness, proficiency, and established norms surrounding LLM use in software engineering were still nascent. As such, no guidelines had been planned or presented to participants requesting for answers to not be artificially-generated. This may have impacted the quality of the CRT responses. It is unclear why a test that typically experiences floor effects did not present any zero-score responses, with over 78% answering all questions correctly. Participants may have used external resources like ChatGPT to provide answers. In hindsight, we note the survey flow did not state that participants could not use resources at this point in the survey. If this was the case and participants were not being tested on their reflective abilities, no genuine conclusions can be drawn. As a result, the CRT data may be contaminated and not provide valuable findings. Future work should ensure participants know they cannot use resources to answer questions designed to measure cognition. The simulation found a negative association for CRT and a positive association for CRT-2. These findings are incongruent as they are intended to measure the same cognitive dimension. As a result, it is not easy to draw meaningful conclusions from the findings without being too selective as to the interpretation. If AI was used to answer the questions, then any interpretation is fundamentally flawed. As such, we decline to provide any strong discussion over these findings.

6.3.3 Construct Validity. The operationalisation of the key constructs may also impose limitations. Our measures of dual processing theory were based on the cognitive reflection test as well as the GDMS which may not fully capture the underlying theory. Similarly, measurement instruments (e.g., surveys, coding schemes, behavioural metrics) may be subject to limited sensitivity, particularly in the sense that participants completed a small number of code puzzles that may not reflect their real-world tasks. Although we attempted to align the constructs with prior work, such as Brun et al. [4], alternative operationalisations could yield different results.

6.3.4 Reliability and Objectivity. One of this study's main limitations is the sample size of 37 participants. With a modest sample size not powered for all hypotheses, the results should be treated cautiously, and this is recognised and handled in two ways. The first is through the conservative discussion of the results and their implications, focusing more on the support the findings provide for the "vulnerabilities as blindspots" paradigm [6]. By deliberately keeping the discussion from translating findings to practical application at this stage, we recognise the limits of the research. The second is the inclusion of the simulation of 740 novel observations. With the simulated data, statistical power was calculated for future research, and combined with the provision of all research materials, this provides a clear path for confirmatory studies.

To reduce researcher bias, we preregistered the study design and analysis. Nonetheless, certain decisions such as the selection of presented puzzles may reflect subjective judgement. Future research could present a much larger array of puzzles across multiple participants. Making all materials, code, and data available supports transparency and allows independent verification.

6.4 Future Work

The obvious and initial development would be to carry out a suitably powered confirmatory study. For most cognitive measures collected, a minimum sample size of 203 participants would be required to achieve a statistical power level of .8 (giving an 80% chance of detecting a true effect if it exists). A sample of around 100 would be sufficient to capture the stronger effects measured.

Other developments that warrant further exploration are to remove measures that likely hold no predictive value and explore other measures in their place, such as the Need for Cognition scale [5], a measure of an individual's tendency to engage in cognition during decision making.

Additional opportunities exist to test the effectiveness of psychological interventions for reducing software engineers' blindspots when producing or interacting with software code. Previous research has shown the benefit of short-term interventions such as nudging [31], but longer-term interventions should be explored or developed to promote the persistence of security behaviours, such as cognitive forcing.

7 Conclusions

We report on a study that assessed the viability of using the dual processing theory of decision making [11] to explain why software vulnerabilities are often missed during code review. This research advances the paradigm that software vulnerabilities often exist as cognitive blindspots [6]. System 1 and 2 processing measures were used alongside Python puzzles containing blindspots.

Our findings suggest cognitive reflection and rational decision making are linked to better performance, whereas intuitive decision making is negatively associated. We support previous findings that technical experience and expertise do not affect blindspot detection. The results are discussed for the support they offer the paradigm and potential ways the findings could be utilised in practical settings once validated with confirmatory research. This study provides the foundations for further work in providing software engineers with psychology-based interventions that are not restricted to programming languages, environments, or IDEs but are grounded in their cognitive competencies.

References

[1] Vaibhav Anu, Kazi Zakia Sultana, and Bharath K. Samanthula. 2020. A Human Error Based Approach to Understanding Programmer-Induced Software Vulnerabilities. In *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 49–54. doi:10.1109/ISSREW51248.2020.00036

[2] Alan D. Baddeley. 1986. *Working Memory.* Clarendon Press/Oxford University Press, New York, NY, US. xi, 289 pages.

[3] Denise R. Beike and Steven J. Sherman. 1994. Social Inference; Inductions, Deductions, and Analogies. In *Handbook of Social Cognition* (2nd ed. ed.), Robert S. Wyer and Thomas K. Srull (Eds.). L. Erlbaum Associates, Hillsdale, N.J.

[4] Yuriy Brun, Tian Lin, Jessie Elise Somerville, Elisha M. Myers, and Natalie C. Ebner. 2023. Blindspots in Python and Java APIs Result in Vulnerable Code. *ACM Transactions on Software Engineering and Methodology* 32, 76 (April 2023), 1–31. Issue 3. doi:10.1145/3571850

[5] John T. Cacioppo and Richard E. Petty. 1982. The Need for Cognition. *Journal of Personality and Social Psychology* 42, 1 (1982), 116–131. doi:10.1037/0022-3514.42.1.116

[6] Justin Cappos, Yanyan Zhuang, Daniela Seabra Oliveira, Marissa Rosenthal, and Kuo-Chuan Yeh. 2014. Vulnerabilities as Blind Spots in Developer's Heuristic-Based Decision-Making Processes. In *Proceedings of the 2014 Workshop on New Security Paradigms Workshop - NSPW '14*. ACM Press, Victoria, British Columbia, Canada, 53–62. doi:10.1145/2683467.2683472

[7] Pat Croskerry. 2003. Cognitive Forcing Strategies in Clinical Decisionmaking. *Annals of Emergency Medicine* 41, 1 (Jan. 2003), 110–120. doi:10.1067/mem.2003.22

[8] Kaja Damnjanović, Vera Novković, Irena Pavlović, Sandra Ilić, and Slobodan Pantelić. 2019. A Cue for Rational Reasoning: Introducing a Reference Point in Cognitive Reflection Tasks. *Europe's Journal of Psychology* 15, 1 (Feb. 2019), 25–40. doi:10.5964/ejop.v15i1.1701

[9] Sarah Elder, Nusrat Zahan, Rui Shu, Monica Metro, Valeri Kozarev, Tim Menzies, and Laurie Williams. 2022. Do I Really Need All This Work to Find Vulnerabilities? *Empirical Software Engineering* 27, 6 (Aug. 2022), 154. doi:10.1007/s10664-022-10179-6

[10] Jonathan St. B. T. Evans. 1984. Heuristic and Analytic Processes in Reasoning. *British Journal of Psychology* 75, 4 (1984), 451–468. doi:10.1111/j.2044-8295.1984.tb01915.x

[11] Jonathan St. B. T. Evans. 2003. In Two Minds: Dual-Process Accounts of Reasoning. *Trends in Cognitive Sciences* 7, 10 (Oct. 2003), 454–459. doi:10.1016/j.tics.2003.08.012

[12] Jonathan St. B. T. Evans and Keith E. Stanovich. 2013. Dual-Process Theories of Higher Cognition: Advancing the Debate. *Perspectives on Psychological Science* 8, 3 (May 2013), 223–241. doi:10.1177/1745691612460685

[13] Shane Frederick. 2005. Cognitive Reflection and Decision Making. *Journal of Economic perspectives* 19, 4 (2005), 25–42. doi:10.1257/089533005775196732

[14] Steven Furnell. 2021. The Cybersecurity Workforce and Skills. *Computers & Security* 100 (Jan. 2021), 102080. doi:10.1016/j.cose.2020.102080

[15] Margaret Gratian, Sruthi Bandi, Michel Cukier, Josiah Dykstra, and Amy Ginther. 2018. Correlating Human Traits and Cyber Security Behavior Intentions. *Computers & Security* 73 (March 2018), 345–358. doi:10.1016/j.cose.2017.11.015

[16] Joseph Hallett, Nikhil Patnaik, Benjamin Shreeve, and Awais Rashid. 2021. "Do This! Do That!, And Nothing Will Happen" Do Specifications Lead to Securely Stored Passwords?. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 486–498. doi:10.1109/ICSE43902.2021.00053

[17] Jerome D. Hoover and Alice F. Healy. 2019. The Bat-and-Ball Problem: Stronger Evidence in Support of a Conscious Error Process. *Decision* 6 (2019), 369–380. doi:10.1037/dec0000107

[18] Matthew Ivory, Miriam Sturdee, John Towse, Mark Levine, and Bashar Nuseibeh. 2023 (in review). Can You Hear the ROAR of Software Security? How Responsibility, Optimism And Risk Shape Developers' Security Perceptions. *Empirical Software Engineering* (2023 (in review)). doi:10.31234/osf.io/pexvz

[19] Matthew Ivory, John Towse, Miriam Sturdee, Mark Levine, and Bashar Nuseibeh. 2023. Recognizing the Known Unknowns; the Interaction Between Reflective Thinking and Optimism for Uncertainty Among Software Developer's Security Perceptions. *Technology, Mind, and Behavior* 4, 4 (Dec. 2023), 319–334. doi:10.1037/tmb0000122

[20] Philip Nicholas Johnson-Laird. 1983. *Mental Models: Towards a Cognitive Science of Language, Inference, and Consciousness*. Harvard University Press.

[21] Philip N. Johnson-Laird. 2010. Mental Models and Human Reasoning. *Proceedings of the National Academy of Sciences* 107, 43 (Oct. 2010), 18243–18250. doi:10.1073/pnas.1012933107

[22] Daniel Kahneman and Shane Frederick. 2002. Representativeness Revisited: Attribute Substitution in Intuitive Judgment. In *Heuristics and Biases* (1 ed.), Thomas Gilovich, Dale Griffin, and Daniel Kahneman (Eds.). Cambridge University Press, 49–81. doi:10.1017/CBO9780511808098.004

[23] Daniel Kahneman, Paul Slovic, and Amos Tversky (Eds.). 1974. *Judgment under Uncertainty: Heuristics and Biases* (1st ed.). Cambridge University Press, Cambridge, United Kingdom.

[24] Roderick M. Kramer, Pamela Pommerenke, and Elizabeth Newton. 1993. The Social Context of Negotiation: Effects of Social Identity and Interpersonal Accountability on Negotiator Decision Making. *Journal of Conflict Resolution* 37, 4 (Dec. 1993), 633–654. doi:10.1177/0022002793037004003

[25] Tamara Lopez, Helen Sharp, Thein Tun, Arosha Bandara, Mark Levine, and Bashar Nuseibeh. 2019. Talking About Security with Professional Developers. In *2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER IP)*. IEEE, 34–40. doi:10.1109/CESSER-IP.2019.00014

[26] Tamara Lopez, Helen Sharp, Thein Tun, Arosha K. Bandara, Mark Levine, and Bashar Nuseibeh. 2019. "Hopefully We Are Mostly Secure": Views on Secure Code in Professional Practice. In *12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, Montreal, QB, Canada, 61–68. doi:10.1109/chase.2019.00023

[27] André Loske, Thomas Widjaja, and Peter Buxmann. 2013. Cloud Computing Providers' Unrealistic Optimism Regarding IT Security Risks: A Threat to Users?. In *ICIS 2013 Proceedings*. ICIS, Milano, Italy.

[28] Rahul Mohanani, Iflaah Salman, Burak Turhan, Pilar Rodríguez, and Paul Ralph. 2020. Cognitive Biases in Software Engineering: A Systematic Mapping Study. *IEEE Transactions on Software Engineering* 46, 12 (Dec. 2020), 1318–1339. doi:10.1109/TSE.2018.2877759

[29] Kjetil Mølokken and Magne Jørgensen. 2005. Expert Estimation of Web-Development Projects: Are Software Professionals in Technical Roles More Optimistic Than Those in Non-Technical Roles? *Empirical Software Engineering* 10, 1 (Jan. 2005), 7–30. doi:10.1023/B:EMSE.0000048321.46871.2e

[30] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, and Matthew Smith. 2020. On Conducting Security Developer Studies with CS Students: Examining a Password-Storage Study with CS Students, Freelancers, and Company Developers. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. doi:10.1145/3313831.3376791

[31] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, Emanuel von Zezschwitz, and Matthew Smith. 2019. "If You Want, I Can Store the Encrypted Password": A Password-Storage Field Study with Freelance Developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–12. doi:10.1145/3290605.3300370

[32] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Marco Herzog, Sergej Dechand, and Matthew Smith. 2017. Why Do Developers Get Password Storage Wrong? A Qualitative Usability Study. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. Association for Computing Machinery, New York, NY, USA, 311–328. doi:10.1145/3133956.3134082

[33] Daniela Seabra Oliveira, Tian Lin, Muhammad Sajidur Rahman, Rad Akefirad, Donovan Ellis, Eliany Perez, Rahul Bobhate, Lois A. DeLong, Justin Cappos, and Yuriy Brun. 2018. {API} Blindspots: Why Experienced Developers Write Vulnerable Code. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. USENIX, 315–328.

[34] Daniela Seabra Oliveira, Marissa Rosenthal, Nicole Morin, Kuo-Chuan Yeh, Justin Cappos, and Yanyan Zhuang. 2014. It's the Psychology Stupid. In *Proceedings of the 30th Annual Computer Security Applications Conference*. Association for Computing Machinery, New Orleans, LA, USA, 296–305. doi:10.1145/2664243.2664254

[35] Alexa Palassis, Craig P. Speelman, and Julie Ann Pooley. 2021. An Exploration of the Psychological Impact of Hacking Victimization. *SAGE Open* 11, 4 (Oct. 2021), 21582440211061556. doi:10.1177/21582440211061556

[36] Marian Petre. 2022. Exploring Cognitive Bias 'in the Wild': Technical Perspective. *Commun. ACM* 65, 4 (April 2022), 114–114. doi:10.1145/3517215

[37] Carianne Pretorius, Maryam Razavian, Katrin Eling, and Fred Langerak. 2018. Towards a Dual Processing Perspective of Software Architecture Decision Making. In *2018 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 48–51. doi:10.1109/ICSA-C.2018.00021

[38] Paul Ralph. 2013. Possible Core Theories for Software Engineering. In *2013 2nd SEMAT Workshop on a General Theory of Software Engineering (GTSE)*. IEEE, 35–38. doi:10.1109/GTSE.2013.6613868

[39] Irum Rauf, Marian Petre, Thein Tun, Tamara Lopez, Paul Lunn, Dirk Van Der Linden, John Towse, Helen Sharp, Mark Levine, Awais Rashid, and Bashar Nuseibeh. 2021. The Case for Adaptive Security Interventions. *ACM Transactions on Software Engineering and Methodology* 31, 1 (Sept. 2021),

9:1–9:52. doi:10.1145/3471930

[40] Anthony V. Robins. 2022. Dual Process Theories: Computing Cognition in Context. *ACM Transactions on Computing Education* 22, 4 (Sept. 2022), 41:1–41:31. doi:10.1145/3487055

[41] Iflaah Salman, Burak Turhan, and Sira Vegas. 2019. A Controlled Experiment on Time Pressure and Confirmation Bias in Functional Software Testing. *Empirical Software Engineering* 24, 4 (Aug. 2019), 1727–1761. doi:10.1007/s10664-018-9668-8

[42] Susanne G. Scott and Reginald A. Bruce. 1995. Decision-Making Style: The Development and Assessment of a New Measure. *Educational and Psychological Measurement* 55, 5 (Oct. 1995), 818–831. doi:10.1177/0013164495055005017

[43] Agnia Sergeyuk, Sergey Titov, Yaroslav Golubev, and Timofey Bryksin. 2023. Overcoming the Mental Set Effect in Programming Problem Solving. In *PPIG'23*. Psychology of Programming Interest Group, 22–36.

[44] Ben Shreeve, Catarina Gralha, Awais Rashid, João Araujo, and Miguel Goulão. 2022. Making Sense of the Unknown: How Managers Make Cyber Security Decisions. *ACM Transactions on Software Engineering and Methodology* 32, 83 (Aug. 2022), 1–33. Issue 4. doi:10.1145/3548682

[45] H. A. Simon. 1956. Rational Choice and the Structure of the Environment. *Psychological Review* 63, 2 (1956), 129–138. doi:10.1037/h0042769

[46] William Smart. 2018. *Lessons Learned Review of the WannaCry Ransomware Cyber Attack*. Technical Report. National Health Service, London, United Kingdom. 1–42 pages.

[47] David Spicer and Eugene Sadler-Smith. 2005. An Examination of the General Decision Making Style Questionnaire in Two UK Samples. *Journal of Managerial Psychology - J MANAG PSYCHOL* 20 (March 2005), 137–149. doi:10.1108/02683940510579777

[48] Webb Stacy and Jean MacMillan. 1995. Cognitive Bias in Software Engineering. *Commun. ACM* 38, 6 (1995), 57–63. doi:10.1145/203241.203256

[49] Michael Stagnaro, Gordon Pennycook, and David G Rand. 2018. Performance on the Cognitive Reflection Test Is Stable across Time. *Stagnaro, MN, Pennycook, G., & Rand, DG (2018) Performance on the Cognitive Reflection Test is stable across time. Judgment and Decision Making* 13 (2018), 260–267.

[50] Keith E. Stanovich and Richard F. West. 2014. The Assessment of Rational Thinking: IQ ≠ RQ. *Teaching of Psychology* 41, 3 (July 2014), 265–271. doi:10.1177/0098628314537988

[51] Keela S Thomson and Daniel M Oppenheimer. 2016. Investigating an Alternate Form of the Cognitive Reflection Test. *Judgment and Decision making* 11, 1 (2016), 99.

[52] Maggie E. Toplak, Richard F. West, and Keith E. Stanovich. 2011. The Cognitive Reflection Test as a Predictor of Performance on Heuristics-and-Biases Tasks. *Mem Cognit* 39, 7 (Oct. 2011), 1275–89. pubmed:21541821 doi:10.3758/s13421-011-0104-1

[53] Merije Van Rookhuijzen, Emely De Vet, and Marieke A. Adriaanse. 2021. The Effects of Nudges: One-Shot Only? Exploring the Temporal Spillover Effects of a Default Nudge. *Frontiers in Psychology* 12, 683262 (2021), 1–12.