API Blindspots: Why Experienced Developers Write Vulnerable Code

Muhammad Sajidur Rahman, Tian Lin, Rad Akefirad^, Donovan Ellis, Eliany Perez, Lois Anne Delong‡, Justin Cappos‡, Yuriy Brun*, Natalie Ebner, Daniela Oliveira

University of Florida New York University‡ University of Massachusetts Amherst* Auto1 Inc.^

Background

1

- New instances of existing, well studied vulnerabilities, such as SQL injections and buffer overflows, are frequently reported in vulnerability databases
  - 61% web apps contain at least one vulnerability listed in OWASP Top 10 vulnerability categories *(Vereacode Software Securtiy Report 2016)*
  - 66% of vulnerabilities represented flawed programming practices recommended to avoid by secure programming guidelines *(*CWE/SANS top 25 most dangerous software errors, 2011.)
- Developers often 'blindly' trust and use programming language APIs as if they are outsourcing security implications to the API itself.
- **API security blindspot :** A misconception, misunderstanding, or oversight on the part of the developer when using an API function, which leads to a violation of the recommended API usage protocol with possible introduction of security vulnerabilities.

(Oliveira et. al. 2014, Cappos et. al. 2014)

Questions

2

1. Is there a difference in developer's accuracy to solve programming puzzles with API blindspots compared to non-blindspot puzzles?
2. Which API usage contexts are particularly susceptible to API blindspot?
3. Does cyclomatic complexity have an effect on API blindspot?
4. Does developer's technical expertise help him detect API blindspots?
5. Do developer's perception, personality and cognitive ability have an effect on API blindspot detection?

Methods

3a

Sample:

- n = 109 Java Developers (age range: *21-52* years, *80.7%* male)
- *64.2%* participants were professional developers, rest were senior/graduate students in Computer Science and Engineering.

Study Instrument:

- **Code review task:** 24 programming puzzles, two-third had API blindspots, one-third had no blindspots in API usage. All puzzles were functional and error free.
- **Professional experience assessment:** A self reported assessment on 17 programming concepts and technologies.
- **Personality assessment:** Big Five Inventory questionnaire.
- **Cognitive assessment:** NIH Oral Symbol Digit Test and Brief Test of Adult Cognition by Telephone.

Study Procedure:

- Participants were asked to solve a set of six puzzles which were counterbalanced by blindspot and non-blindspot APIs, types of API usage contexts and cyclomatic complexity.
- Personality assessment was done on a 5-point Likert scale.
- A JavaScript plugin recorded audio responses during cognitive assessment.
- Upon completion, participants were presented with the solutions and explanation.

3b

```
1  // OMITTED: Import whatever is needed
2  public final class SystemUtils {
3    public static boolean setDate (String date)
4        throws Exception {
5      return run("DATE " + date);
6    }
7
8    private static boolean run (String cmd)
9        throws Exception {
10     Process process = Runtime.getRuntime().
       exec("CMD /C " + cmd);
11     int exit = process.waitFor();
12
13     if (exit == 0)
14       return true;
15     else
16       return false;
17   }
18 }
```

➢ Example of a blindspot puzzle targeting a Java Runtime API usage.

➢ **Susceptibility to API Blindspot**: Line 10, `Runtime.exec()` method if input sanitization is not done properly.

Data Analysis

4

- **Hypothesis 1. Developer's accuracy in API Blindspot contexts**
  Multi-level logistic regression
  Significant API Blindspot effect ($B = -.81$, $z = -4.54$, $p < .001$, *odds ratio*$= .44$)
  Significant Blindspot x API category interaction ($\chi^2 = 24.8$, $p < .001$)
  Significant Blindspot x Cyclomatic complexity interaction ($\chi^2 = 30.1$, $p < .001$)
- **Hypothesis 2. Developer perception to detect API Blindspots**
  Multi-level logistic regression
  No significant effect found
- **Hypothesis 3. Cognitive functioning to detect API Blindspots**
  Ordinal logistic regression
  No significant effect found
- **Hypothesis 4. Technical expertise to detect API Blindspots**
  Ordinal logistic regression
  No significant effect found
- **Hypothesis 5. Personality traits to detect API Blindspots**
  Ordinal logistic regression
  Openness and extraversion found to be significant ($p < .05$)

Results

5

1. Our results confirmed H1 that **developers were less likely to correctly solve puzzles with a blindspot compared to puzzles without a blindspot**. This finding suggests that developers experienced security blindspots while using certain API functions.

2. This effect was more pronounced for puzzles with **I/O-related API functions** and when the programming scenario was more complex (i.e., **high cyclomatic complexity**).

3. Our data **did not support that developers' perceptions** of puzzle clarity, confidence, difficulty, and familiarity **was associated with their ability to detect blindspots**. Our results also did not support that developers' level of cognitive functioning predicted their ability to detect blindspots.

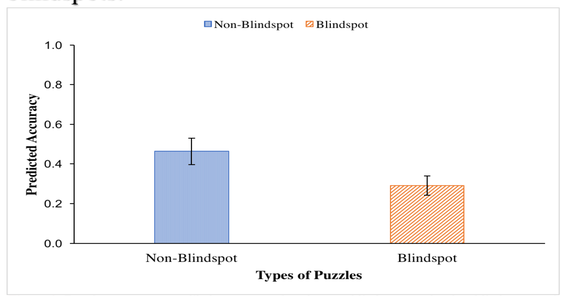

Figure 1: Developers were more likely to accurately solve non-blindspot puzzles than blindspot puzzles. Error bars represent 95% confidence intervals.

4. Our data also **did not support that professional and technical experience was associated with developers' ability to detect blindspots**.

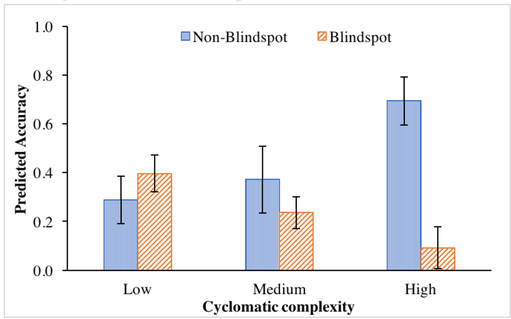

Figure 2: Interaction effect of cyclomatic complexity of the puzzles on accuracy. X-axis shows the three levels of complexity: low(≤ 2), medium(3–4) and high (> 4). Y-axis shows accuracy (predicted probability of correctly solving a puzzle). Error bars represent 95% confidence intervals.

5. Our results partially supported that **more openness and higher extraversion as personality traits in developers were associated with higher likelihood to detect blindspots.**

Discussion

6

- Our data supports the notion that blindspots in API functions lead to the introduction of vulnerabilities in software, even for experienced developers.
- Given these findings, API designers should consider addressing developers' misconceptions and flawed assumptions when working with APIs to increase code security.
- Software Security training and tools **should not come as a "one-size-fits-all"**, but consider developer's decision making process and possible blindspots.
- **Future Directions:** Explore and rank more variants of API blindspots in code repositories and develop detection and recommender tool for developers to write more secure code.

Acknowledgements: Research supported by the National Science Foundation under grant 1513572.