What Norwegian Developers Want and Need From Security-Directed Program Analysis Tools: A Survey

Elias Brattli Sørensen, Edvard Kristoffer Karlsen, and Jingyue Li

ABSTRACT

Code enforcing access control policies often has high inherent complexity, making it challenging to test using only classical review and testing techniques. To more thoroughly test such code, it is strategic to also use program analysis tools, which often can find subtle, critical bugs going unnoticed to humans. These powerful tools are however rarely used in software consultancy practice, due to factors such as bad usability or unsatisfactory non-functional characteristics. To encourage wider adoption of such tools, more must be learned about how to design them to the preferences of software consultants. Towards this goal, we conducted a survey of Norwegian software consultants. Among our findings is a positive relation between preference for soundness over completeness in tools and preference for annotation-based over automated tools. 51% of the developers surveyed prefer soundness over completeness when detecting access control vulnerabilities, while only 37.5% view completeness as the more important characteristic. Qualitative responses illuminate concerns regarding usability, soundness, completeness, and performance.

CCS CONCEPTS

• Security and privacy → Vulnerability scanners; Access control; • Software and its engineering → Software testing and debugging; Formal software verification.

KEYWORDS

program analysis, static analysis, survey, access control vulnerabilities, consultants

ACM Reference Format:

Elias Brattli Sørensen, Edvard Kristoffer Karlsen, and Jingyue Li. 2020. What Norwegian Developers Want and Need From Security-Directed Program Analysis Tools: A Survey . In Evaluation and Assessment in Software Engineering (EASE 2020), April 15–17, 2020, Trondheim, Norway. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3383219.3383293

1 INTRODUCTION

Access control vulnerabilities (ACVs) regularly result in critical data leaks in software systems. When the application $\mathcal{A}$, used by customers of the Norwegian supermarket chain REMA 1000, was launched, its back-end API allowed unauthorized retrieval of sensitive data [7]. Another example is the Facebook scandal in April 2019 [3], where user data for hundreds of millions of users was exposed on a cloud server. ACVs often occur when code implementing authentication checks or enforcing authorization rules has faulty logic. Such subtle errors can often be found by program analysis tools, which model and reason about the behavior of programs. However, these tools are regrettably seldom used by software consultants, due to factors like bad usability or unsatisfactory non-functional characteristics.

To encourage wider adoption of program analysis tools, more must be learned about developers’ requirements to these assets. This paper presents results from a survey of Norwegian software consultants, aiming to investigate the following research questions:

RQ1 What non-functional requirements do consultants have for program analysis tools for detecting ACVs?

RQ2 How does the background of consultants affect their relative preferences for the opposing tool characteristics soundness versus completeness and automatic versus annotation-based?

Quantitative and qualitative data support the following principal findings for RQ1:

• High soundness is considered more important than high completeness when uncovering ACVs.
• There is a near 50/50 preference distribution between fully automated and annotation-based tools.
• Seamless workflow integration may increase the chance program analysis tools are used.
• Most of the respondents reply false positive rates should not exceed 10%.

Regarding RQ2, hypothesis tests show that neither degree of general experience, experience with security-critical system, nor amount of security-oriented education significantly influence developers’ relative preferences for the opposing tool characteristics soundness versus completeness and automatic versus annotation-based.

In addition, we identify a weak, positive relation between preference for soundness over completeness in tools and preference for annotation-based over automated tools.

The remainder of the text is structured as follows: Section 2 presents related work. Section 3 describes the research design. Section 4 presents quantitative results for RQ1 and RQ2, while Section 5 presents qualitative results. Section 6 discusses the results and threats to validity. Finally, Section 7 concludes and suggests ideas for further work.

1 For purpose of this study, we consider the important non-functional characteristics workflow integration point, computational efficiency, and soundness and completeness.
2 RELATED WORK

Christakis and Bird conducted a survey targeting developers in Microsoft [4]. They investigated developers’ requirements to static analyzers and compiled a ranked list of barriers against their use. Among their primary findings were:

- Developers want the opportunity to customize analyzers.
- Programmatic annotations are most preferable, before rules given in global configuration files and annotations coded in comments.
- 90% of developers accept 5% false positives, 50% of developers accept 15% false positives, and only 24% of developers accept more than 20% false positives.
- There is a 50/50 distribution in preferences for soundness versus completeness.

Thomas et al. [27] conducted a study investigating the implications of interactive code annotations within the IDE. Their main findings were that it is easy to write annotations for access control logic, but hard to find causes of vulnerabilities. Even non-security people were able to describe access control policies with reasonable effort.

Sadowski et al. [22] worked with a Google project called Ticorder. Their main findings include:

- Low false alarm rate is important.
- It is important to allow customization at project level, and not only at user level.
- Analysis tools should not only find, but also fix bugs. Tools that automatically apply fixes reduce the need for context switches.
- Program analysis tools should be shardable to ensure that analyses can run at large scale.

Tripp et al. [28] present a tool called ALETHEIA. Their main idea is to apply statistical learning to user-tailor warning output. The tool learns from feedback on a smaller set of warnings. They confirm the well-known finding that developers are very bothered by an excess of false positives.

Tymchuk et al. [29] interviewed experienced developers to understand how they were influenced by an IDE tool providing just-in-time feedback for good coding practices. Usefulness of analyzers in different situations was assessed, and they gathered feedback about the behavior of the tool. They found that the main negative issues of static analyzers are false positives, unclear explanations, annoying user experience, and annoying rules.

Li et al. [14] performed an experimental validation of various open-source IDE plugins that detect security vulnerabilities. They investigated vulnerability class scopes, quality of detection, and user-friendliness of tool warnings. They found a mismatch between the claimed and actual coverage of the tools, as well as unexpectedly high false positive rates. Several tools had limited information in their output, with drawbacks such as imprecise or lacking explanations of vulnerabilities. Another issue was missing opportunities to direct a tool; some tools are only able to scan full code bases, and not smaller units.

3 RESEARCH DESIGN

To direct the design of the study, we conducted a review [26] of relevant research in program analysis tools for detecting access control vulnerabilities [2, 5, 6, 8–11, 13, 15–18, 20, 21, 24, 25, 31–33].

Research guidelines suggested by Kitchenam and Pfleeger [12] and by Oates [19] were also considered in the design process. Christakis and Bird’s survey of Microsoft developers [4] was a particularly important influence. The main purpose of the questionnaire was to explore developers’ relative preferences between opposing tool characteristics, and their thoughts on various challenges with and requirements of tools. The questionnaire was distributed to approximately 750 consultants, from seven consultancy firms. Each of the invitees received a reminder a few days after the initial invitation, and the questionnaire was open for a week.

After reviewing relevant literature and similar surveys, six statistical hypotheses, listed in Table 1, were selected. While hypotheses 1–5 lay wholly within the scope of RQ2, hypothesis 6, which does not concern a background-specific relation, does not.

Two details concerning the formulation of the six hypotheses should be clarified: First, many hypotheses, and survey questions, concern the relative preference between soundness and completeness, on the underlying assumption that increased performance with regard to one attribute necessitates a decrease in performance with the other attribute. Thus, when the phrase “prefers soundness over completeness” is used, it means only that one is willing to sacrifice some degree of completeness for increased soundness, not that one would not ideally want both. Second, for brevity, the qualifier “software consultants” is generally omitted from the hypotheses, and the shorthand “tools” is used to mean specifically “program analysis tools for detecting access control vulnerabilities”.

Inferential statistical analysis involving ordinal data is a contested, methodologically challenging issue [1, 23, 30]. It is especially difficult to assess when classical parametric statistical tests are applicable, and how to safely prepare ordinal data for use with such tests. To err on the side of caution, we opted to use only non-parametric tests in the analysis. In particular, Kendall’s Tau rank correlation coefficient was used for the majority of hypotheses, as it is a natural choice for investigating relationships involving ordinal variables representing preferences. The downside of using non-parametric tests is that they generally have lower statistical power than parametric ones; non-parametric tests are more likely to result in type II errors, where one fails to reject a false null hypothesis. For purpose of our study, we regard it preferable to err on the side of rejecting a hypothesized relation, rather than erroneously concluding one exists when that is not the case.

3.1 Semi-structured interviews

Three respondents were invited to semi-structured, follow-up interviews after participating in the questionnaire. The focus of these interviews was to get insight into how these respondents interpreted the questions and gave their responses, to discover potential weaknesses in the survey design and enhance understanding of the survey data and results. The qualitative responses were analyzed semantically, though not with any formal coding framework.

4 RESULTS

Of the approximately 750 consultants invited to participate in the questionnaire, 80 persons responded. Among these, 87% primarily write and maintain (production) code, 5% work as testers, 4%
Table 1: Statistical Hypotheses

| Null hypotheses                                                                 | Alternative hypotheses                                                                 |
|---------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| H1 Consultants working with security-critical systems do not tend to have higher | H1 Consultants working with security-critical systems tend to have higher relative preference for annotation-based versus automated tools than consultants who do not work with such systems. |
| relative preference for annotation-based versus automated tools than consultants who do not work with such systems. |
| H2 There is not a positive relation between level of experience and preference for completeness over soundness in tools. | H2 There is a positive relation between level of experience and preference for completeness over soundness in tools. |
| H3 There is not a positive relation between level of experience working with security-critical systems and preference for soundness over completeness in tools. | H3 There is a positive relation between level of experience working with security-critical systems and preference for soundness over completeness in tools. |
| H4 There is not a positive relation between amount of security-oriented education and preference for soundness over completeness in tools. | H4 There is a positive relation between amount of security-oriented education and preference for soundness over completeness in tools. |
| H5 There is not a positive relation between amount of security-focused education and preference for annotation-based over automated tools. | H5 There is a positive relation between amount of security-focused education and preference for annotation-based over automated tools. |
| H6 There is not a positive relation between preference for soundness over completeness in tools and preference for annotation-based over automated tools. | H6 There is a positive relation between preference for soundness over completeness in tools and preference for annotation-based over automated tools. |

Figure 1: Distribution of relative preference for soundness versus completeness

Figure 2: Joint distribution of relative preference for soundness vs. completeness and accepted rate of false positives

work as system administrators, and the remaining 4% work with management.

4.1 RQ1: Non-functional characteristics

The survey explored where in the development cycle consultants would prefer to use a program analysis tool, i.e. their preferred workflow integration point. “Direct integration in an IDE” was the most popular option, before “integration in a Continuous Integration/Continuous Delivery (CI/CD) pipeline” and “integration in the (local) build process”. Further, the consultants were asked to indicate their relative preference between the conflicting attributes soundness and completeness. The ordinal data illustrated in Figure 1 show a total of 51% preferring to find as many critical errors as possible (soundness), while only 37.5% of the consultants viewed having fewer false positives as the more important attribute when detecting data leaks.

4.2 RQ2: Background-related effects

Hypothesis 1: Consultants working with security-critical systems tend to have higher relative preference for annotation-based versus automated tools than consultants who do not work with such systems.

Among the respondents, 49 persons work with security-critical systems, while 19 persons do not work with such systems. To test
Hypothesis 2: There is a positive relation between level of experience and preference for completeness over soundness in tools.

Hypothesis 3: There is a positive relation between level of experience working with security-critical systems and preference for soundness over completeness in tools.

Hypothesis 4: There is a positive relation between amount of security-oriented education and preference for soundness over completeness in tools.

Hypothesis 5: There is a positive relation between amount of security-focused education and preference for annotation-based over automated tools.

Hypothesis 6: There is a positive relation between preference for soundness over completeness in tools and preference for annotation-based over automated tools.

5 QUALITATIVE RESPONSES

To get further insights for answering RQ1, the questionnaire contained open-ended questions asking the consultants for their thoughts on where to fit a program analysis tool into their workflow, how they would like vulnerabilities reported, which other vulnerabilities they find important, and what challenges they see with using program analysis tools for security. The follow-up interviews also gave valuable insights.
5.1 Workflow integration point

After giving their opinion on their preferred workflow integration point, each participant was asked to elaborate further. Several interesting responses provided potentially valuable insights. One consultant discussed possibilities of integrating program analysis in the implementation step in the development cycle: “In the day to day basis I would find it natural that program analysis is executed in the build process, for example in Jenkins”.

An experienced consultant provided other perspectives in their written response: “I think this is difficult to answer. For me it is natural that such access control is something that is tested in integration test and is defined in code. It should be part of an active development of an API, and one should construct it 100% restrictively, to then open the security following needs. For me this is a part of the craft of doing software development. If we should have some architecture that ‘automagically’ understands business rules, then it is nice to have it running in the production systems, either as firewall or other rule-based systems. But the rules must still be written?”

The last response indicates a natural skepticism, and points to the complexity of most software systems, which makes it hard to trust that a tool can handle such complexity. The response also suggests a lower need for pedagogical tooling for more senior developers. Several consultants liked the idea of using program analysis tools during code review, or as part of a CI/CD pipeline.

5.2 Vulnerability report formats

After rating various vulnerability report methods and formats, respondents were asked for their own suggestions. One developer would like the opportunity to have access control vulnerabilities trigger compilation errors, so the code can not execute until the issue is fixed. Several respondents pointed out that they would like the output of program analysis in logs. That way they may configure dashboard-based, mail-based or other types of reporting on their own.

An important aspect of having the tool output during build or in CI/CD, was that the build or pipeline must break if there is a vulnerability. Otherwise the vulnerability report could easily drown among other log warnings. One respondent wanted the output to result in warnings in the local build process, but to result in errors when code is processed in a CI pipeline. It was pointed out how program analysis could, and should, be used in harmony with other protection methods: “Whatever can be detected automatically should be detected as early as possible, then via either IDE, build or CI. Meanwhile, I would believe that some things are detectable only via a larger penetration test that is carried out by experts, who then typically would write a report from their test.”

Others worried about the time load tools could carry with them: “One would not like things to take a long time, so the IDE is preferable. However, not if it heavily burdens the performance of the IDE, then it is better to put analysis later. So the answer to the questions depends on how high a load the tool puts on each step.” This respondent also highlighted the importance of analyzing during run-time in addition to static analysis. This motivation for several analysis modes may come from that attacks vectors change over time.

5.3 Challenges with program analysis

In the end of the questionnaire the respondents were asked to describe any challenges they could see with using program analysis to detect access control vulnerabilities.

The following are some of the challenges with program analysis tools that the developers in consulting mentioned:

- Properly detecting complex patterns and contexts is challenging.
- Developers may turn off an analysis if it takes too long.
- Results of analysis may appear in a hidden place, somewhere the developer must actively seek.
- The tool may be too generic for the domain.
- Result credibility is weakened with too many false alarms.
- There is lacking trust that a tool will be able to detect errors, due to the complexity in software development.
- Poor performance in soundness or completeness is challenging.
- There are probably situations in which non-standard program behavior is misunderstood by the tool.
- When the tool usability is too bad developers will not use it.

5.4 Follow-up interviews

Three of the consultants were taken in to follow-up interviews, in which they got to review their questionnaire responses and provide thoughts about the topics in question.

The following overall insights and opinions came from these three interviews:

- High soundness and completeness is more important than where in the workflow a tool is used.
- A tool that learns from code practices and version control history may be valuable.
- The idea that the tool has a faster in-editor mode, and a slower mode that runs later is acceptable.
- The false positive rate may be higher if vulnerabilities are presented in an orderly, ranked manner.
- License fees are a possible barrier from usage of program analysis tools.
- A configurable tool sounds intriguing. However, the configuration must be easily understandable, and the defaults must be sensible.
- One of the respondents thinks a tool should focus on finding and ranking more intricate vulnerabilities.

6 DISCUSSION

6.1 Comparison with related work

The data illustrated in Figure 1 shows that nearly 1.4 times as many chose soundness, suggesting significant difference. 31% of the respondents found soundness to be “most important” (a score of 5). Including the scores of “most important” and “slightly more important” as weights, would give a preference ratio of nearly 1.5 in the favor of soundness, which further solidifies the overall preference. This study uses a narrower range for false positive acceptance rates than what was used by Christakis and Bird [4], illustrated in Figure 2. There is a misalignment between what is considered few false positives by participants, and what is considered few in research [4]. This apparent cognitive dissonance may explain the contrast that comes from the majority also viewing soundness as...
the most important factor. Interestingly, as indicated by the follow-up interviews, developers may prefer completeness in the early stage of development, and allow soundness in the later stages together with other thorough testing. The most preferred workflow integration point, embedding inside an IDE, aligns with the findings of Christakis and Bird. However as indicated by interviewees, the preference workflow integration as well as other responses depend on where in the development process the project is. The preference for using annotations had a near 50/50 distribution, as shown in Figure 3. An IDE-integrated tool should be fast, while a CI-based tool could possibly be allowed to scan code all night. Worry about license fees is one barrier against adapting tools in consulting, as well as lack of trust in the performance of open-source tools. These participants’ worries are also confirmed by recent research [14].

Neither degree of general experience, experience with security-critical system, nor amount of security-oriented education significantly influence the relative preferences for the opposing tool characteristics soundness versus completeness and automatic versus annotation-based of developers. The results of the hypothesis tests do not suggest any obvious, new guidelines for tailoring an analysis tool to the preferences of background-specific subsets of developers. However, hypothesis test 6 suggests that there exists a subset of developers who are positively inclined towards tools and more willing to make sacrifices to utilize their strengths in the development process. Still, most developers will avoid using a tool that has bad usability or is lacking in non-functional characteristics. Therefore, designers of program analysis tools should adapt to the process of software developers in order to provide proper value to the development, a point that confirms ideas from related works [4, 22, 28, 29].

Several solutions in the state of the art of program analysis for access control analysis do not have a clear usability perspective. Even the ones claiming to use “interactive communication” as a usability factor in their solution [33], have been found to come with major drawbacks regarding usability and other non-functional characteristics [14]. The worry about false positives is ever apparent, and it is unclear what should be a realistic false positive rate, though the preference towards soundness when mitigating data leaks suggests some acceptance. The developers do not want a strictly automatic or strictly annotation-based tool, but prefer to use something adaptable.

6.2 Threats to validity

The internal validity of the practitioner survey is threatened by biased and imprecise questions that still persisted after trials of testing. Another internal limitation is different understanding of terms. A term properly defined before the question may be missed or skipped by the respondent. Qualitative responses were translated from Norwegian, which carries the risks of semantics getting lost in translation.

The external validity is mainly threatened by sampling bias. A few consulting firms that were accessible through contacts of researchers were selected, and among them most consultants were invited. No probabilistic sampling was done, and the survey relies on self-selection. The mass of the respondents may still be large enough so that results can generalize to other consulting firms in Norway. The questionnaire was sent out to around 750 consultants, among which 400 were invited by e-mail, while the remaining 350 were invited by channel posts in work place chat services. Use of chat service rather than email imposes a greater risk of several potential participants never being properly exposed to the invitation, as the message quickly drowns. The response rate of 10.5% may threaten the generalizability of the study, but given that the 80 respondents come from seven different firms with various business areas, the sample may be an acceptable representation of the Norwegian IT consulting industry. The validity of comparison to related work is also threatened by differences in development culture, so it is hard to draw conclusions reaching outside of Norway for this sample. Additionally, each industry may have different software development life cycles, standards and environments, which means that the preferences of these consultants may not apply for developers with slight differences regarding these factors.

7 CONCLUSION AND FURTHER WORK

This paper surveys and analyses the preferences of Norwegian software consultants in program analysis tools for detecting access control vulnerabilities. 80 IT consultants from seven Norwegian consulting firms were surveyed for their opinions, with embedded long text answers and follow-up interviews.

We find that high soundness is considered more important than high completeness when uncovering ACVs, and observe a near 50/50 preference distribution between fully automated and annotation-based tools. Of the developers surveyed, 51% prefer soundness over completeness when detecting ACVs, and only 37.5% consider completeness the more important characteristic.

The quantitative analysis shows that neither degree of general experience, experience with security-critical system, nor amount of security-oriented education significantly influence developers’ relative preferences for the opposing tool characteristics soundness versus completeness and automatic versus annotation-based. However, the survey data suggests there exists a group of developers who are more positively inclined towards program analysis tools and more willing to make sacrifices to utilize their strengths in the development process.

The preferences regarding opposing characteristics explored in this paper may be determined by additional context-dependent factors, like project life cycle stage and kind of vulnerability. Hence, an interesting avenue for future work is to delve deeper into the various contexts to explore subtle influences over preference of tool usage. There may also exist other relations like the one explored by hypothesis 6, which could be explored. Finally, it would be interesting to look deeper into the statistical nature of the opinions in a larger-scale study with probabilistic sampling, potentially expanding to a wider population.

REFERENCES

[1] Phillip A Bishop and Robert L Herron. 2015. Use and misuse of the likert item responses and other ordinal measures. International journal of exercise science 8, 3 (2015), 297.
[2] Ivan Bocic and Tevfik Bultan. 2016. Finding access control bugs in web applications with cancheck. In ASE 2016 - Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. Association for Computing Machinery, Inc, 155–166. https://doi.org/10.1145/2970276.2970350
[3] CBC News 2019. Hundreds of millions of Facebook user records were exposed on Amazon cloud server. https://www.cbsnews.com/news/millions-facebook-
user-records-exposed-amazon-cloud-server/. Accessed on 19.12.2019.

Maria Christakis and Christian Bird. 2016. What Developers Want and Need from Program Analysis: An Empirical Study. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE 2016). ACM, New York, NY, USA, 332–343. https://doi.org/10.1145/2970267.2970347

Michael Dalton, Christos Kozyrakis, and Nickolai Zeldovich. 2009. Nemesis: Preventing Authentication & Access Control Vulnerabilities in Web Applications. Technical Report. http://www.usenix.org/events/sec09/full_papers/dalton/paper.pdf http://full.handle.net/1721.1/62182http://creativecommons.org/licenses/by-nc-sa/3.0/

G Deepa, P Santhi Thilagam, Amit Praseed, and Alwyn R Pau. 2018. DeLogic: A black-box approach for detecting logic vulnerabilities in web applications. Journal of Network and Computer Applications 109 (2018), 89–109.

digi.no [n.d.]. Kundedata lå åpent tilgjengelig i Rema 1000s Æ-app. – Jeg kunne lastet ned hele kundebasen. https://www.digi.no/artikler/kundedata-la-apot-tilgjengelig-i-rema-1000s-ae-app-jeg-kunne-lastet-ned-heel-kundebasen375923. Accessed on 19.12.2019.

Donia El Kateb, Yehia ElRakaiby, Tejeddine Mouelhi, and Yves Le Traon. 2013. Access control enforcement testing. In Proceedings of the 6th International Workshop on Automation of Software Test. IEEE Press, 64–70.

Zhejun Fang, Yueqing Zhang, Ying Kong, and Qixiu Liu. 2014. Static detection of logic vulnerabilities in Java web applications. Security and Communication Networks 7, 3 (2014), 519–531.

Jossefin Homaei and Hamid Reza Shabaziari. 2019. Athena: A framework to automatically generate security test oracle via extracting policies from source code and intended software behaviour. Information and Software Technology 107 (2019), 112–124.

Muhammad Noman Khalid, Humera Farooq, Muhammad Iqbal, Muhammad Talha Alam, and Kamran Rasheed. 2019. Predicting Web vulnerabilities in Web Applications Based on Machine Learning. In Intelligent Technologies and Applications, Imran Sarwar Baij, Fairouz Karameddine, and Anna Costa (Eds.). Springer Singapore, Singapore, 473–484.

Barbara A. Kitchenham and Shari L. Pfleeger. 2008. Personal Opinion Surveys. Springer London, London, 63–92. https://doi.org/10.1007/978-1-84800-044-5_3

Jorrit Kromme, Arjen Hommersom, and Harald Vranken. 2018. Discovering software vulnerabilities using data-flow analysis and machine learning. In ACM International Conference Proceeding Series. Association for Computing Machinery. https://doi.org/10.1145/3230833.3230856

Jingyue Li, Sindre Beba, and Magnus Melseth Karlsen. 2019. Evaluation of Open-Source IDE Plugins for Detecting Security Vulnerabilities. In Proceedings of the Evaluation and Assessment on Software Engineering (EASE ’19). ACM, New York, NY, USA, 200–209. https://doi.org/10.1145/3319008.3319011

Anders Møller and Mathias Schwarz. 2014. Automated Detection of Client-State Manipulation Vulnerabilities. ACM Transactions on Software Engineering and Methodology 23, 4 (9 2014), 1–30. https://doi.org/10.1145/2531921

Malíheh Monshizadeh, Prasad Naldurg, and V. N. Venkatakrishnan. 2014. MACE: Detecting Privilege Escalation Vulnerabilities in Web Applications. In ACM Conference on Computer and Communications Security.

Divya Muthukumarman, Dan O’Keeffe, Christian Priebe, David Evers, Brian Shand, and Peter Pietzuch. 2016. FlowWatcher: Defending against data disclosure vulnerabilities in web applications. In Proceedings of the 22nd ACM Conference on Computer and Communications Security. ACM, 603–615.

Joseph P. Near and Daniel Jackson. 2016. Finding security bugs in web applications using a catalog of access control patterns. In Proceedings - International Conference on Software Engineering, Vol. 14–22 May 2016. IEEE Computer Society, 947–958. https://doi.org/10.1109/ICSE.2016.7484836

Briony J. Oates. 2005. Researching information systems and computing. Sage.

Abdul Razzaq, Khalid Latif, H. Farooq Ahmad, Ali Hur, Zahid Anwar, and Petter Charles Bloodsworth. 2014. Semantic security against web application attacks. Information Sciences 254 (1 2014), 19–38. https://doi.org/10.1016/j.ins.2013.08.007

Rebecca Russell, Louis Kim, Lei Hamilton, Tomo Lazovich, Jacob Haer, Onur Ondemir, Paul Ellingwood, and Marc McConley. 2018. Automated vulnerability detection in source code using deep representation learning. In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA). IEEE, 757–762.

Caitlin Sadowski, Jeffrey Van Gogh, Ciera Jaspan, Emma Soderberg, and Collin Winter. 2015. Tricorder: Building a program analysis ecosystem. In Proceedings of the 37th International Conference on Software Engineering. ACM, New York, NY, USA, 297–308.

Sooel Son and Vitaly Shmatikov. 2011. SAFERPHP: Finding semantic vulnerabilities in PHP applications. In Proceedings of the ACM SIGPLAN 6th Workshop on Programming Languages and Analysis for Security. ACM.

Fanqiu Sun, Liang Xu, and Zhendong Su. 2011. Static Detection of Access Control Vulnerabilities in Web Applications. In ISENEIX Security Symposium, Vol. 64.