Report from Dagstuhl Seminar 23181

Empirical Evaluation of Secure Development Processes

Eric Bodden[*1], **Sam Weber**[*2], **and Laurie Williams**[*3]

1 **Universität Paderborn, DE.** eric.bodden@uni-paderborn.de
2 **Carnegie Mellon University – Pittsburgh, US.** smweber@andrew.cmu.edu
3 **North Carolina State University – Raleigh, US.** williams@csc.ncsu.edu

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 23181 "Empirical Evaluation of Secure Development Processes". This was the second seminar on this subject. It brought together researchers and practitioners from the fields of software engineering, IT security and human factors, to discuss challenges and possible solutions with respect to empirically assessing secure engineering activities.

Seminar May 1–5, 2023 – https://www.dagstuhl.de/23181
2012 ACM Subject Classification Software and its engineering → Software creation and management; Security and privacy → Software security engineering
Keywords and phrases Empirical assessment, Secure development lifecycle
Digital Object Identifier 10.4230/DagRep.13.5.1

1 Executive Summary

Eric Bodden (Universität Paderborn, DE)
Sam Weber (Carnegie Mellon University – Pittsburgh, US)
Laurie Williams (North Carolina State University – Raleigh, US)

In the past decades, the cybersecurity community has created many principles and practices for developing secure software. However, this knowledge has generally been assembled by the application of common sense and experience, and while individual measures and techniques are often based on real-world data, broader strategies and processes for creating secure software are usually not subjected to rigorous evaluation. This is a serious shortcoming: common sense can be mistaken and experiences over-generalized. Evaluation techniques are necessary to provide a firm scientific basis that can support progress in the field.

Some such techniques do exist for the later software development stages – implementation and testing. Here one enjoys good automation and the mapping between technique and end-product is relatively clear-cut. It is also in these stages where security teams succeed at least partially in providing software developers with concrete prescriptive guidance. Unfortunately, the earlier developmental stages – requirements elicitation, threat modeling, architecture – are just as critical to the security of the final product, yet pose a much greater experimental challenge because of the gap between the process and the product. Experience has shown only limited success at turning software engineers into security experts, particularly so for these crucial initial stages.

[*] Editor / Organizer

Empirical Evaluation of Secure Development Processes, *Dagstuhl Reports*, Vol. 13, Issue 5, pp. 1–21
Editors: Eric Bodden, Sam Weber, and Laurie Williams
Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Our previous Dagstuhl Seminar 19231 formed a community interested in empirical investigation of secure development practices. This Dagstuhl Seminar now sought to compile a volume merging empirical software engineering and security research to assist the involved communities, including industry and academia, in focusing their research efforts, and to help newcomers to our field find fertile research areas.

The seminar was designed to be highly interactive, with only three introductory presentations on how security researchers, software engineering researchers, and practitioners think about secure software engineering, and which challenges they perceive, particularly with respect to empirical assessment and evidence. Participants then regularly regrouped in altogether one dozen interactive breakout sessions on various topics covering all activities of a prototypical secure development lifecycle, with the intention of eventually gaining the ability to formulate chapters in a to-be-written textbook on the subject.

A special highlight of the seminar was the remote talk by Steve Lipner, former security executive at Microsoft and now executive director of SAFECode, who recapped the most interesting recollections about his introduction of the first secure development lifecycle at Microsoft some 25 years ago, known as the Window Security Push, details about which can be found below.

2 Table of Contents

3 Overview of Talks

3.1 How do Software Engineering researchers see the world?

Eric Bodden (Universität Paderborn, DE)

In this talk I briefly convey my personal experience on how software engineering researchers perceive research on *secure* software engineering. As I will explain, there are a number of related fields such as programming languages and formal verification that also contribute to the goal of securing software. I will contrast software engineering from these fields. Moreover I will discuss the subjects that have received most attention in the field of software engineering and thee publishing culture within the community. This culture is very different from that in IT security, and focuses much more on defenses than attacks, and on processes just as much as tools. This also impacts meta-issues such as the quest for more reproducible studies and sharing of artifacts.

3.2 What practitioners seek from the community

Alex Gantman (Qualcomm Research – San Diego, US)

In this brief talk I will highlight my personal view on what challenges software security practitioners face. This includes: (1) Measuring the outcomes and impact of our work (beyond measuring the effort invested), (2) Scaling to large code bases, large organizations, and complex supply chains, and (3) Lack of robust theoretical foundations for our practices.

3.3 Inside the Windows Security Push: A Twenty-Year Retrospective

Steve Lipner (remotely)

Main reference Steve Lipner, Michael Howard: "Inside the Windows Security Push: A Twenty-Year Retrospective", IEEE Secur. Priv., Vol. 21(2), pp. 24–31, 2023.
URL https://doi.org//10.1109/MSEC.2022.3228098

This talk discusses a follow-up to an article on the Windows security push in the first issue of IEEE Security and Privacy (January 2003). It provides additional detail on the security push and its results, and describes the creation and evolution of the Security Development Lifecycle (SDL) that integrated software security into Microsoft's development process.

3.4 How Security People Think of the World

Sam Weber (Carnegie Mellon University – Pittsburgh, US)

In this talk I briefly share my personal view on how security researchers view the world of secure software engineering, from the early beginnings within the military to the worldwide community we have today. I will explain how we arrived at an attacker mindset and a culture of distrust, and why security is hard to achieve. Lastly, I will highlight some areas that in my opinion require further research, particularly software/system architecture, finding non-generic issues with security tools and certification as well as risk management.

4 Working groups

4.1 Breakout Group "Humans in Empirical Evaluation of Secure Development Processes"

Yasemin Acar (George Washington University, DC, US), Robert Biddle (Carleton University – Ottawa, CA), and Sascha Fahl (Leibniz Universität Hannover, DE)

This breakout session was motivated by much work in the "Usable Security" research community over the past 20 years (e.g. Lipford and Garfinkel [1]), where the emphasis has been on end-user human factors relating to security, and increasing interest in research on human factors relating to software developers. Our goal was to consider issues specific to human factors relating to developers and topics specific to development of secure software.

We began with an introduction to some key aspects of the area, inviting discussion. Our group consisted of 10 researchers, 4 of whom have been actively engaged in the usable security community for many years. That community had coalesced around such viewpoints articulated by Zurko and Simon as early as 1997 in "user-centered security" [3], and Adams and Sasse in their 1999 paper "User are Not the Enemy" [2], and took positive approaches on better understanding end-users and their contexts, and proposing then evaluating better designs to help users be secure. For software developers, we want to eschew regarding developers as stupid or lazy, and focusing on how we can make secure development easier.

We discussed classic usable security questions such as

- How can we get these people to do secure thing?
- Why are they not doing the secure thing?
- How can we make "doing the secure thing" easier for people to do?
- How do their minds work? What are they (not) worried about?
- How do they use technology (insecurely)?

and discussed that these are generally also researched in software engineering research.

We discussed similarities in methodology across security and software engineering research, and decided to both delve deeper into cognitive frameworks and seminal papers with generally accepted methodology from both fields.

References

1 Garfinkel, Simson, and Heather Richter Lipford. Usable security: History, themes, and challenges. Morgan & Claypool Publishers, 2014.
2 Adams, Anne, and Martina Angela Sasse. "Users are not the enemy." Communications of the ACM 42.12 (1999): 40-46.
3 Zurko, Mary Ellen, and Richard T. Simon. "User-centered security." Proceedings of the 1996 workshop on New security paradigms. 1996.

4.2 Breakout Session "Software Supply Chains"

Eric Bodden (Universität Paderborn, DE) and Laurie Williams (North Carolina State University – Raleigh, US)

During our discussion session, we explored various aspects of third-party libraries in software supply chains:

We began by addressing the Log4J issue and the SolarWinds attack, emphasizing the distinction between unintentional programmer errors and deliberate malicious injections. The topic of supply chain attacks was central, with debates on whether known vulnerabilities should be part of these discussions and where to draw the line regarding live downloading or code inclusion. We also touched on challenges, such as defining security in the supply chain, prioritizing vulnerabilities, and handling undocumented behavior in third-party packages. The effectiveness of SBOM was discussed, along with concerns about human factors, like developers making decisions under time constraints and ensuring security in open-source contributions. We explored the role of liability and the importance of interdisciplinary collaboration with legal experts. Self-attestation and the challenges of assessing component behavior were also raised. Cyber-physical impacts of software in supply chains were considered, along with potential risks associated with third-party external attestation. We delved into how to measure supply chain security and identify suitable metrics. Economic considerations were also part of the conversation, including the ongoing cost implications of using open-source software versus in-house development. Our discussion uncovered opportunities related to trust boundaries, assured open-source components, and the security of ecosystems not originally designed for supply chain security. We debated the influence of social proof on library selection and the need for developers to comprehend library features. Additionally, we discussed selective loading of library code and the importance of secure dependency tools. We examined risks associated with solo developers and small teams, including their motivations and interpersonal dynamics. The distinction between bugs and vulnerabilities in software supply chains was clarified. Lastly, there was a suggestion to conduct a case study on a software supply chain and explore interdisciplinary research with legal experts. In summary, our discussion was comprehensive, covering a wide range of topics related to third-party libraries in software supply chains, including challenges, opportunities, human factors, security concerns, and economic considerations. This conversation underscored the intricate nature of supply chain security in the software industry.

4.3 Breakout Session "Security Metrics"

Daniela Soares Cruzes (NTNU – Trondheim, NO) and Akond Rahman (Auburn University, US)

Main reference Patrick Morrison, David Moye, Rahul Pandita, Laurie A. Williams: "Mapping the field of software life cycle security metrics", Inf. Softw. Technol., Vol. 102, pp. 146–159, 2018.
URL https://doi.org//10.1016/j.infsof.2018.05.011
Main reference Marcus Pendleton, Richard Garcia-Lebron, Jin-Hee Cho, Shouhuai Xu: "A Survey on Systems Security Metrics", ACM Comput. Surv., Vol. 49(4), pp. 62:1–62:35, 2017.
URL https://doi.org//10.1145/3005714

The goal of our breakout session was to understand the challenges and opportunities for security metrics in the context of empirical evaluation of secure software processes. We started the discussion by discussing existing reviews of security metrics [1, 2]. During this breakout group we discussed a wide range of topics related to challenges and opportunities for future work related to security metrics. Some of the highlights of the breakout session are: (i) the usefulness of metrics is dependent on context and stakeholder preferences, (ii) a lack of a metric suite that provides a holistic overview of the systems, and (iii) we derived a set of research questions, which included questions related with incentives and evaluation measures. We believe that this breakout session provides the groundwork for synthesizing the challenges, opportunities, and open research questions for the secure software development community. Examples of open research questions that came out of this session include but are not limited to: (i) how do we evaluate metrics that do not have a ground truth?; (ii) how to determine the usefulness of metrics?; (iii) how do we measure risk for interface designs?; (iv) what approaches can we use to evaluate metrics?; and (v) how much evidence is required to demonstrate initial viability of a security metric?

4.4 Breakout Session "Architecture & Design"

Joanna Cecilia da Silva Santos (University of Notre Dame, US)

Software architecture design plays a crucial role in ensuring that security requirements are addressed. It allows for the identification and mitigation of potential security risks early in the development lifecycle. Given this importance, the Architecture & Design working group discussed the disjoint between software engineering and software security community, the need for mapping studies between the well established practices in software architecture and software security. Moreover, the group deliberated about secure software design and architecture (including architecture design principles and practices), and the challenges of implementing secure design principles. The key challenges identified were

1. Architectural models may not always be available: how to analyze software architectures regarding security?
2. If models are available, how to conduct these analyses in a scalable and automated fashion?
3. Software architecture descriptions may have different formats. How to analyze such a heterogeneous set of architecture descriptions?

4. The discussions about (secure) design decisions tend to be more qualitative than quantitative. How to measure these decisions? That is, how to evaluate the effectiveness of software designs for security?
5. How to tame with architectural drift?
6. How do you integrate design practices into the software development process of an organization?
7. Secure coding tend to be small low-level practices that are not tied to higher-level secure design decisions. How to connect secure coding practices to these design decisions?
8. How to integrate commercial-off-the-shelf (COTS) products? How to re-evaluate security properties after integrating other components?

Given these challenges, the group discussed the idea of creating a body of knowledge to guide empirical evaluation. The envisioned body of knowledge would include:

- Anti-patterns, which would focus on what not to do;
- Attack surfaces;
- Architectural styles and patterns along with their security implications;
- A minimum set of secure design decisions captured in a checklist.

This body of knowledge can start with anecdotal stories about (in)security by design that leads to (severe) vulnerabilities, which would be used to highlight the importance of constructing software systems that are secure by design.

The group proposed several research ideas, such as an evaluation experiment to assess the effectiveness of design decisions, observational studies to evaluate design changes in response to security incidents, and a study of mental checklists used by practitioners.

4.5 Breakout Session "Adversariality"

Olga Gadyatskaya (Leiden University, NL), Robert Biddle (Carleton University – Ottawa, CA), Haipeng Cai (Washington State University – Pullman, US), Sven Peldszus (Ruhr-Universität Bochum, DE), Sam Weber (Carnegie Mellon University – Pittsburgh, US), and Charles Weir (Lancaster University, GB)

Adversaries are a core part of the security process. Yet, we often struggle to understand and to predict them. In this session we discussed what we know about adversaries and what could help us to protect our systems from unknown miscreants.

One of the key discussion points was that adversariality is an inherent part of nature that drives evolution. Parasites exist in all ecosystems, and no barriers to stop them work perfectly. At the same time, resilience of ecosystems to parasites is ensured by diversity. Yet, in software engineering monocultures flourish because it is easier to develop and maintain them, and it is very often that one system dominates a whole market. For example, Chrome is the leading browser today, and, moreover, all popular browsers but Firefox are based on WebKit. We have seen how big the cost of a single vulnerability can be in a monoculture with the Heartbleed bug in the OpenSSL library.

So the solution seems to be in diversity. Competition ensures that our ecosystems are more robust. In many areas alternatives do exist: operating systems, programming languages, network protocol stacks, cryptography libraries, machine architectures, and others.

At the same time, at the level of individual organizations monocultures are appreciated as they reduce maintenance and monitoring costs. Ensuring diversity also comes with its own challenges, not least the scale and depth of supply chains, where software diversity might be evident at one level, but depend on the same components at deeper levels. Thus, we need to come up with new methods to ensure diversity cost-effectively and assess the security protections it affords to organizations.

We have also discussed that the security game is about costs. Defensive mechanisms make costs higher for the attacker. For example, slowing down the password check routine helps tremendously in preventing bruteforcing and other kinds of attacks. Economic models are important for understanding how adversaries operate and how to disrupt them. We observed that other disciplines, such as criminology, study attackers as well. Joining forces can be a way forward.

4.6 Breakout Session "SDL Practices and Budget"

Olga Gadyatskaya (Leiden University, NL), Robert Biddle (Carleton University – Ottawa, CA), Eric Bodden (Universität Paderborn, DE), Daniela Soares Cruzes (NTNU – Trondheim, NO), Alex Gantman (Qualcomm Research – San Diego, US), Alessandra Gorla (IMDEA Software Institute – Madrid, ES), Henrik Plate (Endor Labs – Palo Alto, US), and Sam Weber (Carnegie Mellon University – Pittsburgh, US)

In this breakout session we discussed cost-effectiveness of individual practices in secure development lifecycle (SDL), how it can be defined and improved.

One of the points discussed was measurability: some practices yield results that are inherently more measurable than others. For example, it is easier to measure the number of vulnerabilities found in fuzzing compared to measuring aggregated outcomes of code review. Cost is another aspect that is inherently hard to measure: it is difficult to estimate the cost of using a library that might need to be patched later.

Organizations might be able to make decisions about cost-effectiveness if they know their return on security investment and can predict their security and business risks. Yet, this is very challenging, especially for the early phases of SDL. One method to understand cost-effectiveness of early SDL phases is to review completed projects and analyze what could have been done differently. However, this does not allow to fully grasp the situation, as SDL processes usually address what has already been encountered by that organization, but not yet-unknown challenges. Similarly, there is a lack of understanding of the cost-effectiveness of such activities as awareness and training for developers.

Moving forward, it would be interesting to conduct a large industry survey on distribution of investments over different SDL phases. We hypothesized that for many organizations most of the effort is spent on implementation and testing phases because all developers are involved there, while all other phases would have much smaller effort allocated to them. It would be interesting to see how this distribution changes with increasing maturity of organizations, and whether we can prove that spending more effort and budget in the early phases will lead to decrease of effort required for the later phases, especially the maintenance phase. Organizations are willing to invest money in security, but they need to know how to spend it better.

4.7 Breakout Session "Threat Modeling"

Kevin Hermann (Ruhr-Universität Bochum, DE)

The second discussion on threat modeling delved into the concept of a threat model product line, which involves developing distinct threat models for shipping a product to different companies or countries. It emphasized the importance of traceability and variability in threat models to identify areas for improvement and assess associated costs.

Threat Model Product Lines

Threat models vary based on the specific context, country, and product variants. Chipsets and cloud-based platforms are shipped or offered in multiple countries which may have different threats. Applying a change for one customer to mitigate a threat in one context can lead to disabling functionalities for other customers. Therefore, challenges arise by addressing vulnerabilities in one variant without disrupting functionality in others. Building threat models during the product stage and incorporating them into incident response are valuable, given that vulnerabilities can still emerge. Propagating changes across different variants is a complex task, often requiring modifications to existing threat models.

Challenges

Evaluating this approach is a major challenge, as no metrics to assess the effectiveness of threat models have been established, yet. However, considering factors beyond the STRIDE model, such as attacker resources and utilizing attack tree models are useful to estimate risks and costs in variant development. As an example, if an IoT device which has no value suddenly enters the White House, its value increases and therefore the potential for attacks. Additionally, the selection of relevant factors for creating effective threat models is difficult, as modelling irrelevant factors can lead to overestimation.

Research Directions

Creating a tool to derive multiple threat models from a single model, potentially through the use of STRIDE tools, for which Data Flow Diagrams (DFDs) are required, could be the first step to present the idea of threat model product lines. However, difficulties arise on validating the DFDs created for a threat model. Comparing graphical models is hard, as it often requires human interpretation. Instead, the use of natural languages seems promising, as they are simple to compare.

Conclusion

In conclusion, the breakout session provided valuable insights into the complexities of threat modeling in different contexts, challenges associated with merging and propagating models, the necessity of multiple threat models for the same product, and the importance of credible and validated threat models for effective security measures. Finally, first ideas for research directions for threat model product lines were discussed.

4.8 Breakout Session "More Synergy between Software Engineering and Security"

Ranindya Paramitha (University of Trento, IT)

Joint work of Yasemin Acar, Evan Austin, Haoipeng Cai, Sascha Fahl, Ben Hermann, David Lo, Alena Naiakshina, Ranindya Paramitha, Henrik Plate, Dominik Wermke, Laurie Williams

Software Engineering and Security research are two different yet intersecting worlds whose intersections have been discussed through decades [1]. Having the two communities together sitting in the same room brought some interesting questions: (1) How are Software Engineering and Security research similar and different in general? (2) Is it possible/ how to borrow methodologies from each other? (3) How to do something impactful with this synergy?

Security paper is famous for being related to something "scary". There are several "kinds" of security research: (1) Attacks: the type of security research that focuses on the discovery of the "bad", eg. [2] in CCS'22. The focus is to show the attacks' interesting impacts against many targets or small but important targets. (2) Defenses: this kind of research tends to be harder on getting the paper published. The reason for this is that finding holes in defense is considered easier than criticizing attacks. (3) Security measurement: including manual analysis, human factor/ usable security. One interesting challenge is to understand how to increase the cost of attack: how to make it expensive enough to attack so people do not attack a system, which can be economically or psychologically. (4) Tooling (to support attacks/ defenses). In general, security research focuses more on finding new attacks (the discovery of the "bad", security fatalism) and generalizing them.

On the other hand, Software Engineering research focuses more on the fundamental issues, and not directly finding something: applying methodologies to improve practices in software engineering. Back in 2000, Software Engineering focused more on (1) process modeling (laying out a process, eg. agile software model [3], SDLC) with less validation (no validation/ toy problems were common). These days there is more research on (2) empirical analysis: observation studies and generating theories from it, eg. finding the pattern of how people collaborate in the software ecosystem. This includes (3) experiments with a negative result, as the community believes that interesting questions and well-designed experiments are still valuable even with a negative result, eg. the Replications and Negative Results (RENE) track in ICPC'22 collocated with ICSE'22. There is also (4) human factors research, which uses methods such as systematized user surveys to practitioners (eg. [4]) in order to understand what and how developers think in software engineering, eg. why they work in one way and not the other. On these latest years, there has been an emerging track called (5) Registered Report track (eg. in ESEM'22), which allows researchers to submit 6 pages experiment design, get peer-reviewed, present it, and then have a maximum 1-year period to conduct the experiment and submit the full paper to a journal with continuity acceptance.

Software Engineering and Security have intersections in several ways, eg. the concept of "bugs" in Software Engineering is similar to "vulnerabilities" in Security. "Novelty" in Software Engineering research is valued like "attack" in Security. Nowadays, papers from one community can also be accepted in others, eg. tooling papers that can find a class of vulnerability for a lot of packages can be accepted in both Software Engineering and Security conferences. The possible synergy or "bridge" to get the best of both worlds is to have research with great fundamental methodology (Software Engineering) but with a big "splash" impact (Security). However, there is still a need for the systematization of knowledge in the intersection between Software Engineering and Security, both from different papers but also

other artifacts (ie. gray literature). This can bring scientific contribution to finding the gap, which area research has been done, which area where more research is still needed, and even which research area is not promising.

References

1 Mouratidis, H., & Giorgini, P. (Eds.). (2006). Integrating Security and Software Engineering: Advances and Future Visions: Advances and Future Visions.

2 Mingrui Ai, Kaiping Xue, Bo Luo, Lutong Chen, Nenghai Yu, Qibin Sun, and Feng Wu. 2022. Blacktooth: Breaking through the Defense of Bluetooth in Silence. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22). Association for Computing Machinery, New York, NY, USA, 55–68. https://doi.org/10.1145/3548606.3560668

3 Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J. & Thomas, D. (2001). Manifesto for Agile Software Development Manifesto for Agile Software Development.

4 Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J.& Thomas, D. (2001). Manifesto for Agile Software Development Manifesto for Agile Software Development.

4.9 Breakout Session "Code and Design Reviews"

Sven Peldszus (Ruhr-Universität Bochum, DE)

The code and design review breakout group focused on the discussion of how to evaluate tools and human factors in group and design reviews. Reviews are an essential part of security evaluations during the development of software systems. Following the principle of security by design, security has to be considered already at design time. The planned design must be reviewed concerning the security of the system as well as its implementation later. Despite such reviews can be supported by tooling, the reviewer has an essential role.

Effectiveness of static code analysis

In practice, there is a significant difference in what different reviewers look for in code reviews and their performance. The job of security experts is to find the one critical bug. In contrast to this, the general developer is likely to accept reviews that consist of more than 70% true positive findings.

While it is essential to find as many security bugs as possible, a significant practical barrier is false positive findings that hinder addressing security effectively. False positive findings have to be resolved by humans that come with their individual preferences and might be more effective in handling specific kinds of false positive findings. To optimize the outcomes of code reviews, it is important to tailor these closer to the target audience of the review. To this end, it could be beneficial to learn what are the usual most favorite false positive findings of different stakeholders.

A general approach for optimizing tool-assisted reviews could be to learn from current false positives to mitigate them in the future. Such learning must be prepared by intensive data mining. However, to optimize the reviews, we must identify the false positive findings to

avoid in the future, we must rank all the identified false positives according to some criteria that have yet to be identified. One possibility could be a rating of how critical a specific kind of finding is to fix. The intuition is that developers will start to work on the most critical finding kinds and benefit there the most from fewer false positives. Another possibility could be to consider manual downranks of developers.

For all of these metrics, one practical barrier is to get hands on the necessary data, which is usually not publicly available. One would have to massively collect practical feedback from developers that label the findings of review tools. Since it is not feasible to request a single developer to label all findings of each analysis tool, one could divide the work among multiple parties.

This huge effort in labeling, which is a huge barrier to academic studies to improve the analysis tools, is also a huge barrier in practice when a new tool should be deployed. After the deployment, the outlined task has to be performed to identify the findings of the new analyzer that must be fixed. Here, the application of the new analyzer on only newly written code can help to reduce the number of findings. Developers can focus on the new code they are working on and improve it while reducing the risk of the outcomes of the new analysis tool being ignored entirely. Still, there will be less precise results than expected, since many details have to be trained, including optimizing the configuration of the new tool and also the developers themselves.

Despite these outlined challenges in getting hands with the number of findings and false positives, incremental scans and synchronizing scans across branches are huge barriers. In particular, a false positive identified and labeled in one branch should be also labeled on the other branches on which it is present as a false positive finding.

While these individual observations and ideas sound reasonable, it remains to gather data in an empirical analysis to precisely identify how large the effort of revies for developers is. Thereby, we have to be careful about what exactly to measure. Among others, there are findings in published works that show that developers struggle with configuring checks, which increases their effort in the end. Also, the findings themselves will not change just by tweaking their distribution. Another question is whether we can transfer our own experiences from applying security analysis tools on open-source projects to the integration of the tools in a large toolchain. In the second scenario, one has to consider multiple stakeholders participating in the review. The security team can help in configuring the analysis tools but others have to run them.

When it comes to the effectiveness of a security review, experiences from the industry have shown that success metrics are essential. Since having a non-exploitable system is usually the major goal, one could consider rating findings according to their exploitability. While fixing a buffer overflow might address a true positive finding, this does not necessarily improve the system's security. However, not having an exploit does not prove the effectiveness of static analysis since we might just not be aware of an exploit, yet. Also, it has been shown that there is a correlation between the pure number of issues in a file and the likelihood of a vulnerability, not taking any additional metrics such as exploitability or criticality into account. Still, even such simple metrics as the plain number of findings might not be applicable due to changes. Therefore, a purely empirical approach by counting the number of findings is probably not the right one.

Comparison of static analysis tools

Even assuming a success metric, the issue of insufficient resources for fixing all issues remains and it is questionable whether it makes sense to deploy tools that can find more or other kinds of vulnerabilities when even we are practically not able to fix all the existing critical vulnerabilities. When analysis tools are applied in classes, students report many findings and question the usefulness of the tool for identifying real issues.

Since the pure deployment of additional analysis tools is not the solution, we have to be more selective in which analysis tools we deploy for what purpose. We need means to properly compare analysis tools to decide on which ones to deploy. To this end, we have to focus more on an analytical comparison of what the tools do for a qualitative comparison of different analysis tools.

Such a comparison of security analysis tools could be based on databases of labeled findings and the rules of the tools themselves. One issue in this regard is the availability of such information which is usually only partly publicly accessible and if so only for a single tool. Comparing rulesets among tools and estimating the overlap between tools is a yet unsolved challenge. Comparable to mutation testing, an assessment of static analysis tools could be realized by artificially introducing bugs. Whether vulnerable samples generated by ChatGPT are suitable as a baseline is currently unclear. In the end, we still rely on people building a limited number of ground truths. Recent work mainly focuses on a quantitative comparison of how many bugs in a known data set can be detected by which tool.

Fixing vulnerabilities

The detection of possible vulnerabilities and their rating is only one of the steps in effectively securing a system. After deciding on what are the concrete vulnerabilities that threaten a system, these have to be fixed. Here, plenty of research has been done in the direction of automated fix generation. While these fixe generators are effective in generating fixes, they are yet not used in practice. In the end, the generated fixes can be seen as a blueprint for fixing a detected issue but manual checking of the generated fixes is non-neglectable. In practice, simple automated dependency updates are often rejected by developers which raises the question of how good such tooling has to get.

As in the reviews themselves human peculiarities seem to play an essential also in fixing identified vulnerabilities. Developers might not be satisfied by just accepting proposed fixes while their fix was compatible with the proposed one. Further, even when a vulnerability is fixed by a developer in one team, most likely it will not be fixed in other teams working on the same code in another branch. Often organizational overhead but also lack of communication prevent the effective propagation of security fixes.

Alike to this manual problem in fixing multiple versions of a system, also static analysis works only on a single version but the fix is needed on all versions. While developers have this often in their minds at least for the variants for which they are responsible, tools currently entirely lack such features.

Besides identical duplicates of one bug, we also have to consider additional instances of bugs in similar locations. In the end, humans tend to do the same mistake more than once. Currently, we rely on them to remember these additional locations after one instance has been detected.

One of the challenges in finding such additional locations of bugs is the significant impact of context information that makes a bug probably only so some variants or branches. Therefore, it is essential to check bug-fixing code before it is pushed to other branches.

Design reviews

While static code analyzers have concrete instances on which they can work, design reviews are more challenging. While effective formal methods exist, they are often only feasible for some systems due to their huge overhead. One fundamental problem is usually the lack of design specifications such as models on which a design review can be performed.

While the recovery of models to use in a design review is in principle possible, the main question is how does such a review process look like. In the end, static analysis is well-integrated into today's development processes, while this is not the case for model-based design reviews. While it is favorable to also have such integration for design reviews, there is the danger of the process becoming more important than the product itself.

While static code checks immediately work on the concrete artifacts, for design reviews we have to identify a suitable degree of abstraction. Models can range here from a single very detailed model that is close to the source code to an abstract component diagram with data flows comparable to data flow diagrams in threat modeling.

Depending on the degree of abstraction, different security issues might be identified but most likely not detailed security vulnerabilities such as those identified by static code analyzers. Still, given suitable traceability between the models and code event the low-level static code analysis can benefit from the integration of design models into the process. Among others, design models contain information about elements that are not part of the code but with which it interacts. This information can be leveraged to tailor static code analysis such as taint analysis based on the planned interaction with external entities.

Altogether, design models allow us to systematically structure systems, divide them according to security concerns by creating insulation capsules around security-critical parts, and plan concrete security protections according to the division. Due to possibly non-trivial constraints such structurings and analysis have to be supported by tools. We have to provide architects with guidelines on how to design a secure system. In this regard, anti-pattern catalogs and associations between design patterns and suitable security patterns could help. However, the extraction of such patterns is still open work. Here, pattern mining from repositories could be suitable.

To get more benefit out of such design reviews than an initial plan, their integration into the development process is necessary. Whenever there is any change, we have to find means to reflect it as well in the code as in the design models. However, this integration would allow us also to provide developers with easy-to-comprehend information about changes such as explicitly showing how dependencies among components change when a specific call is added to the implementation. On the downside, an attacker might use exactly this benefit of easily accessible information about security measures to plan an attack.

To be practicable and applicable, we need easy and cheap processes that allow us to build such models incrementally. Here, in particular, scalability is a major concern. While we have had such security-by-design approaches already for a relatively long time, it is unclear what exact improvements we need for bringing them into practice. Nevertheless, examples such as formal methods demonstrate that this is possible.

Summary of tool-assisted security reviews

In security reviews one has to take the perspective of an attacker which is a special case compared to other domains such as safety. Attackers are actively looking for opportunities instead of things happening accidentally. To this end, as a reviewer, you have to always keep all possibilities in mind, while usually lacking the necessary context knowledge. The main task of tooling for design reviews and static code analysis is to help in identifying and presenting unknown dangers.

When considering entirely manual code reviews, reviewers usually tend to report the more obvious findings and the completeness is usually questionable. Still, practices such as pair programming and reviewing the commits of others have been proven to be effective. Here, tool support can help in facilitating these practices, e.g., by suggesting reviewers based on the touched artifacts.

In contrast to manual reviews, tools are often more complete but usually at the cost of precision. Targeting this issue, the question is which low-level findings should be shown to developers. Among others, tools should not only provide huge lists of low-level findings but automatically derive suggestions of suitable security patterns based on the identified dangers. For example, the static analysis identifies what a component is doing and suggests patterns corresponding to that. But even just highlighting the use of critical APIs could be beneficial.

The ultimate goal of the deep integration of tools in the planning and review process is to allow the development of security mechanisms that would be infeasible to plan only manually. For example, we could target more fine-grained rights management. Starting on a high abstraction at the system level, this should be systematically pushed into the applications. This would allow us to make sure in the application code that some parts only have certain permissions and would allow for more control over third-party libraries. The main problem could be getting developers to use the more complex structures that probably result from this. Here, tooling can make such rights management or other security measures easier to include.

Open research problems

We conclude with a summary of open research problems that we identified in the discussion above.

Despite static analysis tools already being widely deployed, their configuration is still an open problem that hinders their effective use. We have to identify simpler ways of configuring static analysis tools. Therefore, it is essential to judge the quality of the results of a tool with a specific configuration. We need to identify suitable measures of security to support such a comparison. But we not only have to be able to compare configurations of an individual tool, but we need means to systematically compare different analysis tools to allow effective tool selection.

We need a deeper investigation of the effectiveness of security measures. To suggest suitable security measures, we have to know if specific measures that have been realized prevented the vulnerabilities for which they have been planned. Related to this it should also be checked what are the cost of specific measures and what is relation to their benefits. Additional measures should be mined from repositories and relations to other principles such as design patterns should be extracted.

Since we aim at integrating design models and design reviews into the development process, we have to identify suitable levels of abstraction to do so. The question is what impact do different views on security have on planning and checking secure designs? This integration and effectiveness could be increased further by creating relations with other security artifacts such as CVEs.

4.10 Breakout Session "Longitudinal Studies"

Akond Rahman (Auburn University, US)

A longitudinal study is a research study that employs repeated and continuous measures to follow entities over prolonged periods of times. As part of one of the breakout sessions participants discussed if longitudinal studies could aid in empirical evaluation of secure development processes. The discussion started with a controversial argument from one of the participants that longitudinal studies are often convoluted with mining software repositories where some researchers frame empirical studies as longitudinal studies for 'marketing purposes'. With the proper definition of longitudinal studies in context participants discussed teased out multiple challenges in conducting longitudinal studies, which included a lack of motivation amongst academics, the time required to publish results, availability of data, and availability infrastructure and management resources.

While the participants acknowledged the challenges inherent to conducting a longitudinal study, they also agreed on the value this research study can bring to empirical evaluation of secure development processes. The participants laid out the following research questions that can be answered in the context of secure software development using longitudinal studies:

- How does security practices change over time for organizations and across organizations?
- How do industry practitioners use static analyzers?
- What areas in secure software development can benefit from execution of longitudinal studies?
- What are the long-term impacts of using generative artificial intelligence (AI) on secure coding practices?

In all, the session triggered great interest amongst participants, many of whom are now collaborating in conducting a longitudinal study in the domain of secure generative artificial intelligence (AI).

4.11 Breakout Session "Secure Generative AI"

Akond Rahman (Auburn University, US)

Generative artificial intelligence (AI) is the discipline of using unsupervised or semi-supervised machine learning techniques to generate human artifacts, such as software source code, movie reviews, and book summaries. In recent times, the most popular generative AI technique in the context of software engineering is use of large language models (LLMs), such as ChatGPT to automate software engineering tasks. As part of this session, participants discussed their experiences is using ChatGPT for software engineering research. Participants mentioned the use of technique called prompt engineering to use LLMs for generating source code.

All participants agreed that while generative AI helps in automating software engineering tasks, there are some shortcomings. One participant discussed one of their recent paper [1], where they found LLM-generated code to include compilation concerns (e.g., 90% of code not compiling), security smells [2], which provide evidence to the perception of "garbage in,

garbage out". The participants further stated that 90% of the datasets that LLMs use to train have quality concerns. All of these shortcomings further provided insights on what could be possible research directions for securing generative AI.

The open research questions that were discussed are:

- How can LLMs help in writing secure code?
- How should we engineer prompts to generate secure code?
- What strategies should we use to improve the quality of LLMs without re-training?

References

1 M. L. Siddiq, S. H. Majumder, M. R. Mim, S. Jajodia and J. C. S. Santos, "An Empirical Study of Code Smells in Transformer-based Code Generation Techniques," 2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM), Limassol, Cyprus, 2022, pp. 71-82, doi: 10.1109/SCAM55253.2022.00014.

2 A. Rahman, C. Parnin and L. Williams, "The Seven Sins: Security Smells in Infrastructure as Code Scripts," 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), Montreal, QC, Canada, 2019, pp. 164-175, doi: 10.1109/ICSE.2019.00033.

4.12 Breakout Session "Tensions between Industry and Academic Objectives"

Dominik Wermke (CISPA – Saarbrücken, DE) and Henrik Plate (Endor Labs – Palo Alto, US)

4.12.1 Introduction

- What: A discussion breakout session with academics from software engineering, security, and related fields, as well as people from industry about the (research-related) tensions and opportunities between their objectives.
- Why: Being research fields generally close to industry, both Software Engineering and Security often rely on direct interaction with and feedback from industry to push and adapt research ideas further.
- Outcomes: A number of entities and approaches were identified as impactful opportunities for "bridging the gap" between industry and academia, namely institutions like Fraunhofer, conferences like SecDev, and events like developer summits and industry workshops.

4.12.2 Industry Problems

The discussion was initiated with a question from the academic side: What problems does industry face, why don't they share (them with researchers)? To which industry responded that they do share their problems, but academia considers most of them to be not interesting problems. Academia brought up that they are restricted in problem / topic selection by a number of factors, namely that they need to be publishable (and fit specific venue requirements) and enable a career both for junior researchers and involved graduate students. Based on this response, the discussion moved on to if industry problems without solutions are still interesting to academia. As an example, the initially low adoption of test generation in industry because of flaky tests was brought up, a problem which was then picked up by

academics. There was agreement that beautiful / interesting problems from industry can be picked up by academia and that both industry and academia consider their respective problems to be interesting.

It was then discussed whether the discussion framing is part of the problem, namely that industry asks themselves what academia can do for them vs. academia asks how they can solve industry problems. Industry pointed out that the academia framing in reality appears to them to be more along the lines of "show me your problems so I can solve the interesting ones". It was proposed that academics might not accept research (problems from industry) because they might not fit their mental model. Based on that, the general question on whether the academic side is open to expanding what they work on was posed. Boundaries imposed by existing publishing venues were identified as a potentially limiting factor, highlighting opportunities to develop better-suited venues targeting industry problems.

4.12.3 Applicable Results

Following the statement "Always go after the practical problems", the discussion turned to how to actually turn research results into applicable results in industry and what or who is required to bridge that gap. It was pointed out that there are organizations with the specific goal of bridging that gap (e.g., Frauenhofer in Germany). It was also discussed that bridging the gap likely involves efforts from all involved parties: Industry needs to understand the current state of research and be willing to apply the published and applicable findings. Researchers need to have a mind about what is practical and be willing to receive industry feedback on the applicability of their findings.

4.12.4 Innovation

The next discussion point was based on a slide from Alex Stamos' 2019 USENIX talk about "Tackling the Trust and Safety Crisis" (a pyramid putting what is actually talked about at USENIX at the top vs. other, more impactful areas of InfoSec for industry below it). Based on that slide, the point was brought up that a lot of the innovation in the InfoSec area actually comes from industry and that academia runs after industry. This situation posed the question on whether academia might have been trapped in the valley of industry impact, resulting in pushing researchers away from more fundamental research approaches. As an example, more foundational theories in other areas were brought up, like in the fundamental theories of databases, compilers, and operation systems.

Developer Summits were brought up for a way to bring industry developers together to discuss hard problems. It was postulated that these summits work (for both industry and present researchers) because industry people love to talk to industry people about all the cool stuff they have been doing, and researchers are present soaking it up. A question was posed on how to publish findings from these summits, with the discussion leaning more towards opinion pieces or initial exploratoritive approaches to identify industry's problems. As a research idea for validating perception, "100 questions from/for industry" was introduced. As challenges for hosting developer summits the discussion included: who to contact and how to bring smaller companies together (that can't send someone to summits. It was mentioned that including developers from smaller companies also presents an opportunity, because if smaller companies can send someone, they probably have greater oversight and decision power over the tech in the company.

Another discussed challenge was that people love to talk about things they plan on implementing in the future, not the experiences they actually made, with a suggested solution for research being to carefully listen and have the technical background to discern these

answers. After the break, the discussion continued about Workshop / Discussion events ($\sim$ 2 h, Chatham house rules), with patterns for success of these events being identified as networking, critical mass, and no shortcuts. A challenge was brought up in that it is not always possible to directly trace the impact of conversations you had at these events and not every idea actually being able to be traced back.

4.12.5 Why Do Research Groups Fail?

The next discussion point was around the question of why industry research groups / departments / teams fail. The discussion centered around if there actually has to be a good probability of failure in research, that even the failed cases have to be spun as successes to publish in academia, and the potential cultural split for research departments vs. other teams because they focus on long term problems.

Based on this potential cultural split, the next discussion point focused on the differences in objectives for industry and academia, namely that "research" in academia and industry (might) not mean the same thing. Points included the need to differentiate between short term solutions and long term solutions, with industry required to provide value immediately and academia being more oriented towards long term. Another point was how to define a good goal or bad goal, with research trying to address real problems that are relevant for the industry, having at least a minimum impact.

4.12.6 Recap

The final discussion point focused on recapping both breakout sessions, with tension points between industry and academia including: Academia only grabbing the problems they think are interesting and then leaving, with industry internships having the potential to better bridge this gap. Industry problems are uninteresting for academia because they often are just constraints for business reasons with obvious solutions and research can not help with that. Messy/complex industry systems in general, with the problem that if the better / correct solution doesn't lead to better outcomes, is it really better? The breakout closed with a recap of the potential of places for academia and industry collaboration such as (industry) conferences, Fraunhofer, and developer workshops.

Participants

- Yasemin Acar
George Washington University, Washington, DC, US
- Evan Austin
NRL – Washington, US
- Alexandre Bartel
University of Umeå, SE
- Thorsten Berger
Ruhr-Universität Bochum, DE
- Robert Biddle
Carleton University – Ottawa, CA
- Eric Bodden
Universität Paderborn, DE
- Haipeng Cai
Washington State University – Pullman, US
- Michael Coblenz
University of California – San Diego, US
- Daniela Soares Cruzes
NTNU – Trondheim, NO
- Joanna Cecilia da Silva Santos
University of Notre Dame, US
- Sascha Fahl
Leibniz Universität Hannover, DE
- Olga Gadyatskaya
Leiden University, NL
- Matthias Galster
University of Canterbury – Christchurch, NZ
- Alex Gantman
Qualcomm Research – San Diego, US
- Alessandra Gorla
IMDEA Software Institute – Madrid, ES
- Ben Hermann
TU Dortmund, DE
- Kevin Hermann
Ruhr-Universität Bochum, DE
- Johannes Kinder
LMU München, DE
- Jacques Klein
University of Luxembourg, LU
- Piergiorgio Ladisa
SAP Labs France – Mougins, FR
- David Lo
SMU – Singapore, SG
- Tamara Lopez
The Open University – Milton Keynes, GB
- Fabio Massacci
VU University Amsterdam, NL
- Tim Menzies
North Carolina State University – Raleigh, US
- Mehdi Mirakhorli
Rochester Institute of Technology, US
- Alena Naiakshina
Ruhr-Universität Bochum, DE
- Ranindya Paramitha
University of Trento, IT
- Liliana Pasquale
University College Dublin, IE
- Sven Peldszus
Ruhr-Universität Bochum, DE
- Henrik Plate
Endor Labs – Palo Alto, US
- Akond Rahman
Auburn University, US
- Awais Rashid
University of Bristol, GB
- Brad Reaves
North Carolina State University – Raleigh, US
- Heather Richter Lipford
University of North Carolina – Charlotte, US
- Daniel Votipka
Tufts University – Medford, US
- Sam Weber
Carnegie Mellon University – Pittsburgh, US
- Charles Weir
Lancaster University, GB
- Dominik Wermke
CISPA – Saarbrücken, DE
- Laurie Williams
North Carolina State University – Raleigh, US