Systematically Evaluating Static Analysis-Based Security Testing Tools - The Gaps within Design and Practice

Amit Seal Ami
William & Mary
Williamsburg, Virginia, USA
aami@wm.edu

ABSTRACT

In this work, we demonstrate that while reliance on security-focused program analysis techniques grows, gaps prevent the effective design, implementation, and evaluation of such techniques, particularly for static analysis-based security testing tools (SASTs). We demonstrate that adapting mutation testing techniques specific to the security domain is a practical approach to finding previously unknown, undocumented flaws that compromise the effectiveness of SASTs. Furthermore, practitioners do not consider existing benchmark-based evaluation sufficient, thus relying on subjective factors, such as reputation, when choosing a SAST. Finally, we report that the industry is not ready to handle such flaws because of several factors, including a paradoxical assumption.

CCS CONCEPTS

• **Security and privacy → Software security engineering**.

KEYWORDS

static analysis, SAST, crypto-API misuse detector, mutation testing, mutation-based evaluation, security, software engineering

ACM Reference Format:
Amit Seal Ami. 2023. Systematically Evaluating Static Analysis-Based Security Testing Tools - The Gaps within Design and Practice. In *Proceedings of The 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2023).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

After over a decade of research and development in both academia and industry, security-focused program-analysis techniques are now being used at nearly every stage of software development and maintenance lifecycle, from requirements engineering to fault localization and fixing, *e.g.,* through GitHub CodeScan Initiative[10] for finding vulnerabilities, such as crypto-API misuse and sensitive data-leaks. Furthermore, such techniques have gained worldwide attention because of the recent high-profile attacks and exploit across the public sector *e.g.,* SolarWind[15], triggering responses from both corporate and government entities, such as emphasizing security through the improvement of existing approaches, *e.g.,* Static Application Security Testing (SAST) tools. In essence, the use and dependence on program-analysis techniques will only increase to ensure software security.

The underlying cause of optimistically using and depending on the security-focused analysis techniques, such as crypto-API misuse detectors (in short, crypto-detectors) and sensitive data-leak detectors, is the convenience these techniques offer through automation, support for continuous integration and development (CI/CD), and the promise of detecting vulnerabilities as long as these are within scope without flaws. In addition, the potential of identifying vulnerabilities statically, *i.e.,*, without depending on runtime-analysis in a time-consuming manner, has made the SAST-based tools an attractive choice among the security-focused program-analysis techniques.

However, such optimism is unwarranted, as we have generally relied on manually curated, static benchmarks to gauge the effectiveness of SASTs. This is also because of the lack of a systematic framework that can handle the various patterns of vulnerabilities, *i.e.,* variants. Thus, this doctoral research proposes the systematic evaluation of SASTs, susceptible data-leak detectors, and crypto-API misuse detectors while leveraging the well-founded approach of mutation analysis.

While traditional mutation analysis is used to gauge the effectiveness of existing test cases, this thesis advocates that vulnerabilities can be mutated to represent both the diverse variations of vulnerabilities that are implemented and introduced by developers, either intentionally or unintentionally and the complex usage patterns of relevant security-specific APIs, such as crypto-primitives enabling APIs from language-specific frameworks. We propose that we can systematically evaluate, analyze, and identify flaws in SASTs by introducing mutated vulnerabilities in open source program source code, which, then, is analyzed by a target SAST. This research analyzes both sensitive-data leak detectors and crypto-detectors from industry and academia. We identify previously unknown, unique flaws while gaining insights, such as possible causes and remedies.

In addition to this, this work identifies a key gap in the adoption and use of SASTs in the software industry: the research community does not possess an in-depth understanding of how SASTs are perceived in the industry, *e.g.,*, whether practitioners are aware of the flaws, or limitations, these SASTs may have, and whether such perceptions and beliefs impact the adoption and use of SASTs. Without addressing this critical gap, it is impossible to create SASTs that are effective, *i.e.,* possess fundamental properties that help ensure software security, and aligned *i.e.,* practitioners have an accurate understanding of what SASTs provide, instead of practitioners having an inaccurate expectation of, and from SASTs.

ESEC/FSE 2023, 11 - 17 November, 2023, San Francisco, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Therefore, to explore this gap through a qualitative, interview-based study in this research and report how practitioners with different security and business needs choose SASTs and depend on those. Furthermore, we study the beliefs and expectations of practitioners regarding the limitations and flaws of SASTs, and how they address those flaws of SASTs.

2 BACKGROUND AND RELATED WORKS

As mentioned, SASTs are gaining attention and focus from both public and private sectors because of the recent focus on security and privacy of software. Static analysis techniques are adapted for finding security vulnerabilities, *e.g.,* [8], resource leaks [4, 5, 7, 20], and crypto-API misuse [13, 18]. There exists several works on evaluating SASTs, *e.g.,* [16, 17]. However, our work is the first of its type that adapts mutation testing technique that is guided by a comprehensive, data-driven taxonomy for finding design and/or implementation flaws in crypto-detectors.

Additionally, researchers have studied the use of static analysis with practitioners, reporting that false-positive is perceived as a significant problem [9, 12]. Finally, researchers have identified that practitioners need security-specific support in the form of APIs and tools, *e.g.,* [11, 14, 19].

3 COMPLETED WORKS

3.1 Systematic Evaluation of Static Analysis based Security Testing Tools

The lack of effective evaluation of SASTs is due to both (a) continued reliance on manually curated benchmarks, which may be incomplete, inaccurate, non-representative of the diverse implementation patterns adopted by developers, and impractical to maintain continuously, and (b) lack of a systematic, evolving framework. The lack of a systematic, evolving framework, in turn, is attributed to domain-specific factors, *e.g.,* for crypto-APIs, a systematic approach needs to (i) consider all existing crypto-APIs, (ii) instantiate realistic misuse-case mutations that are within the scope of crypto-detectors, and (iii) need to be scalable without significant manual intervention.

We address these challenges for the crypto-API domain by constructing the first data-driven taxonomy of crypto-API misuse by analyzing academic and industry sources from the past 20 years and identifying 105 crypto-API misuse cases. Further, to address the second challenge, we analyze the claims made by the target crypto-detectors from industry, academia, and open source community and report that *i.e.,* crypto-detectors offer *strong* security guarantees. Thus, we define a threat model consisting of three types of adversaries: a benign developer who may accidentally misuse, a benign developer who may introduce a vulnerability while trying to address an existing one, and an evasive developer. Furthermore, we designed the crypto-API mutation operators that mutate crypto-API misuse/vulnerabilities, *i.e.,* instantiates misuse variations. Additionally, we design mutation scopes for seeding the mutations, representing realistic crypto-API use and threats. Finally, we implement the **M**utation **A**nalysis for evaluating **S**tatic **C**rypto-API misuse detectors, the MASC framework, that uses the mutation operators, mutation scopes, and static analysis techniques to create mutated, vulnerable software from open source applications to be analyzed by crypto-detectors.

Our research evaluated 9 crypto-detectors from industry, academia, and open source community and found 19 flaws in crypto-detectors. During the responsible vulnerability disclosure process, we report that while crypto-detectors are expected to be evaluated from a security-centric evaluation perspective, *i.e.,* hostile-reviewing because of their strong security guarantees, these are often only designed from a technique-centric perspective, *i.e.,* what static analysis techniques can or can not accomplish. Furthermore, we find that developers of crypto-detectors may have different detection scopes by *design*. For example, whereas some crypto-detectors are designed to detect any crypto-API misuse as long as it is statically analyzable, some crypto-detectors consider additional factors, such as visibility and frequency in the wild. Further details of this work can be found in our proceedings paper [1], whereas a separate contextualization of mutation testing techniques for android-specific SASTs can be found in our earlier journal paper[2].

3.2 Identifying the Gaps: Perspectives of Practitioners regarding SASTs

Discovering that SASTs may have security-compromising flaws, preventing those from detecting vulnerabilities that they claim to and are designed to detect, with different design factors and goals, revealed a key gap that the research community previously was not aware of or did not address appropriately: the research community possesses a limited understanding of how software developers perceive SASTs, their expectations from and beliefs in SASTs regarding vulnerabilities, and how these perceptions and beliefs impact the adoption and use of SASTs in practice.

To address this gap, our research studied the perspectives and beliefs of a diverse group of practitioners through IRB-approved surveys and interviews. This diverse group of practitioners represented different business and security needs, different levels of experience regarding software engineering and security, and different security needs due to (state-required) compliance from around the world. Specifically, we explored the processes of choosing and using SASTs for ensuring security in services and products, their understanding and assumptions regarding the limitations and flaws of SASTs, and how they address, *i.e.,* acknowledge, navigate, or work around those flaws and limitations of SASTs.

By applying reflexive thematic analysis with inductive coding [6], we captured both the latent and semantic meaning of participants' perceptions and contexts, such as limitations of security resources, assurances of SASTs, organizational priorities, and product nature. Our research identified that while practitioners care about and prioritize security, a SAST's objective effectiveness in detecting vulnerabilities is not (or barely) considered by practitioners when it comes to choosing. Hence, there exists a gap between the need of practitioners from SASTs (security guarantees) and the criteria for selecting SASTs (subjective, often ad-hoc processes). Moreover, the gap is caused by two significant factors: (i) practitioners believe that SASTs just work and detect everything within scope, and (ii) they do not consider existing, benchmark-based evaluation of SASTs sufficient, as they consider benchmarks to be simplistic, or biased. As previously discussed, the MASC framework proposed by this research addresses this particular issue through a systematic evaluation framework. Furthermore, we report that while the program analysis community specifically focuses on reducing false positives, practitioners want SASTs to be able to find critical vulnerabilities before anything else, even at the cost of a high number of false positives. Finally, our research revealed a critical paradox in the assumptions made by practitioners. In short, practitioners rely on SASTs to cover the gaps in manual analysis while also believing that their manual analysis techniques can cover the gaps introduced through (unknown) flaws in SASTs. A detailed version of our work, with additional findings, research directions, and insights, is available in our (to be published) proceedings paper [3].

4 CONTRIBUTION TO KNOWLEDGE

It is necessary to evaluate, identify, and address the flaws of security-focused automated program analysis techniques, such as SASTs. This research contextualizes mutation testing techniques in the domain of SASTs and evaluates data leak detectors and crypto-detectors to discover previously unknown, undocumented flaws and identify the gaps that contribute significantly to creating those flaws. Thus, it contributes by helping improve the state-of-the-art SASTs. Further, we argue that our work is beneficial for the software engineering community *in general*, as it helps improve SASTs by identifying flaws, which would be used to improve the security and privacy aspect of the software. Furthermore, we study the perspectives, beliefs, assumptions, and understandings of diverse practitioners regarding SASTs, through which we identify several gaps that the research community needs to address, such as prioritizing the reduction of false negatives for security, the need for better, systematic evaluation techniques suitable to their contexts and the lack of trust towards benchmark based evaluations, and that practitioners are not prepared to handle the flaws of SASTs.

Overall, this research significantly changes and shapes views towards both the evaluation and design of SASTs, and the design priorities of SASTs, a critical component for developing and maintaining software security that continues to gain attention from both public and private sectors due to the focus on security and privacy of software at present era.

ACKNOWLEDGMENTS

The author is advised by Dr. Adwait Nadkarni and Dr. Denys Poshyvanyk at the Computer Science Department of William & Mary. This work is supported in part by NSF-1815336, NSF-1815186, NSF CNS-2132281, CNS-2132285 and CCF-1955853 grants, and a CoVA CCI Dissertation Fellowship. Any opinions, findings, and conclusions expressed herein are the author's and do not necessarily reflect those of the sponsors.

REFERENCES

[1] Amit Seal Ami, Nathan Cooper, Kaushal Kafle, Kevin Moran, Denys Poshyvanyk, and Adwait Nadkarni. 2022. Why Crypto-detectors Fail: A Systematic Evaluation of Cryptographic Misuse Detection Techniques. In *2022 IEEE Symposium on Security and Privacy (S&P)*. IEEE Computer Society, San Francisco, CA, USA, 397–414.

[2] Amit Seal Ami, Kaushal Kafle, Kevin Moran, Adwait Nadkarni, and Denys Poshyvanyk. 2021. Systematic Mutation-Based Evaluation of the Soundness of Security-Focused Android Static Analysis Techniques. *ACM Transactions on Privacy and Security* 24, 3 (Feb. 2021), 15:1–15:37. https://doi.org/10.1145/3439802

[3] Amit Seal Ami, Kevin Moran, Denys Poshyvanyk, and Adwait Nadkarni. 2024. "False negative - that one is going to kill you" - Understanding Industry Perspectives of Static Analysis based Security Testing. In *Proceedings of the 2024 IEEE Symposium on Security and Privacy (S&P)*. IEEE Computer Society, San Francisco, CA, USA. to be published in.

[4] Steven Arzt and Eric Bodden. 2016. StubDroid: Automatic Inference of Precise Data-Flow Summaries for the Android Framework. In *International Conference for Software Engineering (ICSE)*.

[5] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2013. FlowDroid: Precise Context, Flow, Field, Object-Sensitive and Lifecycle-Aware Taint Analysis for Android Apps. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI '14*. ACM Press, Edinburgh, United Kingdom, 259–269. https://doi.org/10.1145/2594291.2594299

[6] V. Braun and V. Clarke. 2021. *Thematic Analysis: A Practical Guide*. SAGE Publications. https://books.google.com/books?id=eMArEAAAQBAJ

[7] S. Calzavara, I. Grishchenko, and M. Maffei. 2016. HornDroid: Practical and Sound Static Analysis of Android Applications by SMT Solving. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*. 47–62. https://doi.org/10.1109/EuroSP.2016.16

[8] B. Chess and G. McGraw. 2004. Static Analysis for Security. *IEEE Security Privacy* 2, 6 (Nov. 2004), 76–79. https://doi.org/10.1109/MSP.2004.111

[9] 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*. ACM, Singapore Singapore, 332–343. https://doi.org/10.1145/2970276.2970347

[10] Github 2020. About code scanning - GitHub Docs. https://docs.github.com/en/free-pro-team@latest/github/finding-security-vulnerabilities-and-errors-in-your-code/about-code-scanning.

[11] Peter Leo Gorski, Luigi Lo Iacono, Dominik Wermke, Christian Stransky, Sebastian Möller, Yasemin Acar, and Sascha Fahl. 2018. Developers Deserve Security Warnings, Too: On the Effect of Integrated Security Advice on Cryptographic API Misuse. In *Fourteenth Symposium on Usable Privacy and Security, SOUPS 2018, Baltimore, MD, USA, August 12-14, 2018*, Mary Ellen Zurko and Heather Richter Lipford (Eds.). USENIX Association, 265–281.

[12] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why Don't Software Developers Use Static Analysis Tools to Find Bugs?. In *35th International Conference on Software Engineering (ICSE)* (San Francisco, CA, USA, 2013-05). IEEE, 672–681. https://doi.org/10.1109/ICSE.2013.6606613

[13] Stefan Krüger, Sarah Nadi, Michael Reif, Karim Ali, Mira Mezini, Eric Bodden, Florian Göpfert, Felix Günther, Christian Weinert, Daniel Demmler, and Ram Kamath. 2017. CogniCrypt: Supporting Developers in Using Cryptography. In *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2017)*. IEEE Press, Piscataway, NJ, USA, 931–936.

[14] Sarah Nadi, Stefan Krüger, Mira Mezini, and Eric Bodden. 2016. Jumping Through Hoops: Why Do Java Developers Struggle with Cryptography APIs?. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*. ACM, New York, NY, USA, 935–946. https://doi.org/10.1145/2884781.2884790

[15] U. S. Government Accountability Office. [n. d.]. SolarWinds Cyberattack Demands Significant Federal and Private-Sector Response (Infographic) | U.S. GAO. https://www.gao.gov/blog/solarwinds-cyberattack-demands-significant-federal-and-private-sector-response-infographic.

[16] Felix Pauck, Eric Bodden, and Heike Wehrheim. 2018. Do Android Taint Analysis Tools Keep Their Promises?. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018)*. ACM, New York, NY, USA, 331–341. https://doi.org/10.1145/3236024.3236029

[17] Lina Qiu, Yingying Wang, and Julia Rubin. 2018. Analyzing the Analyzers: FlowDroid/IccTA, AmanDroid, and DroidSafe. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis - ISSTA 2018*. ACM Press, Amsterdam, Netherlands, 176–186. https://doi.org/10.1145/3213846.3213873

[18] Sazzadur Rahaman, Ya Xiao, Sharmin Afrose, Fahad Shaon, Ke Tian, Miles Frantz, Murat Kantarcioglu, and Danfeng (Daphne) Yao. 2019. CryptoGuard: High Precision Detection of Cryptographic Vulnerabilities in Massive-sized Java Projects. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security - CCS '19*. ACM Press, London, United Kingdom, 2455–2472. https://doi.org/10.1145/3319535.3345659

[19] Justin Smith, Brittany Johnson, Emerson Murphy-Hill, Bill Chu, and Heather Richter Lipford. 2015. Questions Developers Ask While Diagnosing Potential Security Vulnerabilities with Static Analysis. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, Bergamo Italy, 248–259. https://doi.org/10.1145/2786805.2786812

[20] Fengguo Wei, Sankardas Roy, Xinming Ou, and Robby. 2018. Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps. *ACM Transactions on Privacy and Security* 21, 3 (April 2018), 1–32. https://doi.org/10.1145/3183575