Developers Are Neither Enemies Nor Users: They Are Collaborators

Partha Das Chowdhury, Joseph Hallett, Nikhil Patnaik, Mohammad Tahaei and Awais Rashid
Bristol Cyber Security Group
University of Bristol, UK
{partha.daschowdhury, joseph.hallett, nikhil.patnaik, mohammad.tahaei, awais.rashid}@bristol.ac.uk

***Abstract*—Developers struggle to program securely. Prior works have reviewed the methods used to run user-studies with developers, systematized the ancestry of security API usability recommendations, and proposed research agendas to help understand developers' knowledge, attitudes towards security and priorities. In contrast we study the research to date and abstract out categories of challenges, behaviors and interventions from the results of developer-centered studies. We analyze the abstractions and identify five misplaced beliefs or *tropes* about developers embedded in the core design of APIs and tools. These tropes hamper the effectiveness of interventions to help developers program securely. Increased collaboration between developers, security experts and API designers to help developers understand the security assumptions of APIs alongside creating new useful abstractions—derived from such collaborations—will lead to systems with better security.**

***Index Terms*—secure software development, interventions, challenges, beliefs**

I. Introduction

Developers' security expertise depends on varying factors such as the quality of security training they have had and their security beliefs [1]. Securing applications may, at times, be considered an afterthought: a task separate from developing the application itself. When security is considered, developers use off-the-shelf components designed with a range of security policies and assumptions that may not be effective [2]–[4]. Applications are secured using cryptographic APIs, but developers find them hard to use [2], [4], [5]. These technical challenges may push developers into unknowingly adopting potentially dangerous behaviors to deal with the difficulty of secure programming.

Three notable studies in developer-centered security include the works of Patnaik et al. [6], Tahaei et al. [2] and Acar et al. [4]. Patnaik et al. trace 45 years of recommendations from 65 papers to support API developers to improve the usability of their libraries. They find a strong focus on helping developers with constructing and structuring their code with the intention of making the code more usable and easier to understand; but less focus on documentation, writing requirements and code quality assessment. They also find weak levels of empirical validation among papers [6]. Tahaei et al. review 49 Developer-Centered Security (DCS) papers to understand different methodologies used to run the studies and find the emerging research themes in DCS. They find that it is difficult to study developers in a lab environment because of factors that are difficult to replicate such as the long periods of time developers spend working on a code including refactoring, code-reviews, and decision making processes that involve multiple stakeholders. It is also hard to study developers under limited time constraints—as experiments can range from 15 minutes to 3 days but this isn't always enough. Tahaei et al. compare the methodologies used to that of research in Human-Computer Interaction (HCI), and note that they are implemented poorly [2]. Acar et al. argues for a better understanding of the motivations and priorities of developers rather than blaming them for not being mindful of security. They stress the need for developer-centered studies to understand the challenges that developers face when using security APIs, and the resources available to improve the usability of these APIs. They propose taking the developer out of the loop whenever possible while recommending usable APIs in other instances along with secure and usable information sources and tools [4]. Our study complements these prior works. We analyze 72 papers written over 13 years in order to identify overarching challenges faced by developers and their consequent behavior. We study how interventions intersect with the developer behavior in response to the challenges. We create a categorization for the challenges, behaviors and interventions; and answer the following questions:

RQ1. What challenges do developers face and what behaviors do they display in the existing developer-centered research on secure software development?
RQ2. How do the interventions deal with the behaviors of developers and the challenges they face?
RQ3. How can we address the limitations revealed from the analyses of the challenges and interventions?

We find five technical and five behavioral challenges that developers face and four categories of interventions. Many developers address the challenges with behavioral solutions while the interventions come with human-centered, as well as technical solutions. Our analysis reveals *tropes*: misplaced beliefs about how developers behave embedded in the core

This research is supported in part by EPSRC Grant EP/P011799/2 and the National Cyber Security Centre.

design of APIs, tools and secure programming approaches.[1] Such tropes hamper the effectiveness of interventions to support developers in overcoming the technical challenges they face and promoting more secure behaviors, thus adversely affecting secure software development. For example, many APIs assume that *developers are experts* and that they understand security: the challenges we identify suggest that the opposite is true. A correct understanding of the developers' skills is critical for designing appropriate interventions. We posit that *collaboration* between developers, security experts and library developers is key to unravelling these tropes and their adverse impacts on interventions.

II. Method

We reviewed the papers covered in the systematic literature reviews and research agendas related to DCS [2], [4], [8]. We then selected papers that provided an insight into developer challenges, behaviors shown by developers, and interventions suggested. We performed snowballing [9] as well to find and add new papers to our dataset until we reached saturation and did not observe any new papers in our search. This resulted in an initial set of 292 papers, including papers from the bibliographies of our seed papers and a further manual search of the repositories for additional papers. We then read the title and abstract of these papers, and shortlisted papers that explicitly included a study involving at least one developer. Our final set includes 72 papers. To develop our categories of challenges, behaviors and interventions, multiple authors discussed and constructed themes over multiple sessions, based on the contents of the papers.

III. Challenges and Consequent Behaviors

Table I describes the technical challenges developers face when attempting to program securely. These do not cover specific challenges (e.g. *OpenSSL is hard to use*) but rather higher level challenges. In contrast, Table II describes the broad categories of behaviors developers adopt to address these challenges. The split between technical challenges and behaviors comes from how developers try and deal with the difficulty of programming securely. If a developer struggles to understand how to use an API correctly (the *miscommunication* challenge in Table I), they may search Stack Overflow for solutions. The answers to Stack Overflow questions are voted and this may create *herding* behaviors (Table II), regardless of whether the answer they follow is correct or not [42]. The *intangibility* challenge represents a challenge with security as a whole, rather than being specific to any particular development practice. Figure 1 shows which technical issues lead to an instant behavior.

Miscommunication describes when API providers do not give adequate clarity of what API to use and their security assumptions, resulting in *confused mental models*. Acar et al. conducted an experiment where 256 python developers were asked to perform a series of cryptographic tasks using 5 Python based cryptographic libraries. They found some APIs are designed for simplicity, reducing the decision space and the chance of it being misused, but simplicity is not enough [11], [13], [59], [60]. One study reports that for 20% of the functionally correct tasks, the developers believed their code was secure, when in fact it was not [11]. Naikashina et al. found that, when storing passwords, while developers might hash or salt them, they still often end up stored insecurely [40], [46], [48]. Patnaik et al. perform a thematic analysis over 2400 StackOverflow posts seeking help with 7 cryptographic libraries. They report 4 usability smells against Green & Smith's 10 principles [5]. The usability smells result due to missing or hidden information [8]. Van der Linden et al. conduct an experiment with developers to find their coding considerations. They report that developers depict security thinking with their own code but not in testing, seeking help from Stack Overflow for incorporation of APIs [43]. The questions surrounding how to use a particular API or respond in cases of bugs have been reported in [8], [11], [61] as well.

> **Take-away.** Collaboration between API providers and developers to help explain and understand APIs and their security assumptions can lead to security as well as functional correctness.

Narrow Scope and miscommunication lead to *shifting responsibility* among developers where the actions of the API provider lead to developers having to make tedious code updates. *Narrow scope* can result in *herding* behavior. Acar et al. report the hardships faced by developers when libraries do not support auxiliary tasks, e.g. secure key exchange [11]. Fahl et al. found that a significant reason behind insecure use of TLS was the insufficient capabilities of the API. Developers improvised to fulfill their requirements and ended up using APIs incorrectly [59]. Balebako et al. studies app developers to understand their security and privacy decisions in their development activities. It was reported that less than one-third of app developers understood that data were being collected by third parties [54], a strongly supported finding [39], [45], [49], [62]. Derr et al. conduct a study with 203 app developers and a concerning conclusion is that API providers contribute to the poor use of updated libraries. Among the reasons cited are overload of updates, confusing version control and conflicts with the preferences of the developers [4], [28].

> **Take-away.** A participatory model of development, where developers and API designers collaborate, will lead to more comprehensible APIs and an improved understanding of their data flow and security requirements.

Hidden Information is a challenge seen across many empirical studies where usability information does not exist or are not accessible. They include safe uses of a particular library or bug fixing capabilities. Balebako et al. found that

[1]The notion of tropes has been previously used to study how insecurities manifest in internet of things environments [7].

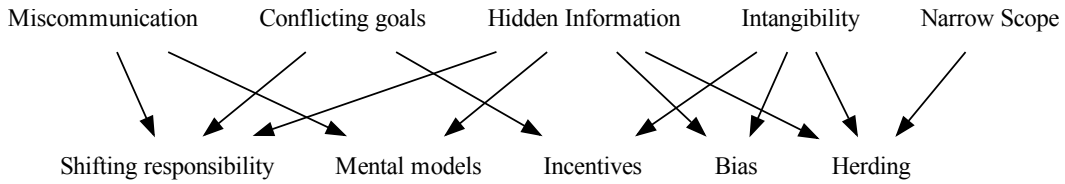

Fig. 1. Relationships between technical challenges (top) developers face and the behaviors (bottom) they adopt to mitigate them.

TABLE I
TECHNICAL CHALLENGES DEVELOPERS FACE, AND THEIR CATEGORIZATION.

Theme	Description	Examples
Miscommunication	Situations where the API developers' beliefs and knowledge isn't communicated to the end-developers.	"Should I Use a particular API for this particular threat model"? • Measure - What works and what doesn't in an overload of libraries • Closed - What are the security assumptions of a particular library? [2], [8], [10]–[20]
Hidden information	Access to the data is not easy; where it either does not exist, or the data is obscured existing in a form that is nearly impossible to comprehend and use.	"How do I Use this" • Safety - What are the safe uses of access points for security and privacy? • Incident Response - Lack of clear documentation on debugging and fixing. [2], [4], [8], [10], [11], [13], [17]–[19], [21]–[27]
Conflicting goals	Conflicting security goals from organizations, security experts and system and library developers create tensions.	"May I Use a particular patch without affecting the existing functionalities"? • Incompatible updates - where a changing API can break developer's code. • Power plays - Lack of communication between Security Experts and Developers [2], [4], [5], [10], [28], [28]–[33]
Intangibility (Metrics)	Security is difficult to quantify, and it is hard for developers to be sure whether their actions are harming or hindering the security of their code.	"Will a particular use of a API make the application secure"? • Perceptions - People put less emphasis on outcomes difficult to quantify • Gains - Developers cannot ascertain benefits of their use of an API [2], [11], [34]–[38]
Narrow scope	APIs are overly focused on a single task.	"How do I integrate non crypto yet security requirements with a particular API"? • Unusable - where APIs do not support auxiliary functions • Limited - APIs are not defined broadly to include non security requirements [5], [22], [25], [28]

developers do not have enough correct information to use security tools [54]. Consequently, developers adopt insecure practices based on their *mental models*, their own *biases* and prior beliefs or display *herding* behaviors where they follow what others do. Van der Linden et al. conduct an observation of developer's use of Stack Overflow with 1,188 participants. They found developers go by surface features of Stack Overflow posts (such as answer length) over correctness [42]. *Hidden information* has a bearing on our characterization of *shifting responsibility*. Braz et al. conduct an observation based study among developers to detect the extent to which they can detect improper input validation. They do highlight that lack of security knowledge is an important reason behind poor detection of vulnerabilities [63].

Take-away. Closer collaboration between API providers and developer may help the both find and fix bugs, and ease the *bias, herding* and *shifting responsibility* behaviors, though encouraging open source collaboration is an ongoing problem [64].

Intangibility comes from a lack of visible benefit of security—problematic if security is not a functional requirement [4]. This leads to *herding*, *bias* and developers are unable to perceive the *incentives* of security. The pattern of *herding* as well as *bias* in adoption of security tools is reported in [32]. App developers tend to adhere to social groups and bring in their privacy beliefs into their development process [27], [53]. Braz et al. notes that security is not a primary goal [63]. The concerns on incentives, learning support and usability are found across developer-centered studies [2], [35]. Sometimes implementing security and privacy is not in a developer's interest: developers might choose libraries for financial reasons [45], [62], or for usability reasons [29], [36], [55].

TABLE II
BEHAVIORS DEVELOPERS EXHIBIT WHEN TRYING TO PROGRAM SECURELY AND THEIR CATEGORIZATION.

Theme	Description	Examples
Confused mental models	It is difficult for developers to process the complexities of security concepts and/or libraries	"Should we hash and salt even when we use TLS"? • Mental models - Developers understanding of the inner workings of the API are often inconsistent with the complex crypto concepts • Adversarial Thinking - Developers are not trained in real world failures and adversarial models; thus not driven by objective security considerations. [2], [4], [19], [24], [35], [39]–[50]
Shifting Responsibility	Cost of actions of one entity is borne by another entity	"How do I update without affecting functionalities"? • API providers - Negative impact of poor or missing documentation, cumbersome library update processes. • Security experts - Insecure end products due to lack of communication between security experts and developers. [2], [8], [10]–[12], [28], [51]
Bias	Use of information sources based on subjective considerations rather objective security considerations	"How to distinguish a correct answer from a wrong answer on Stack Overflow"? • Correspondence bias - Completeness of answers or explained code snippets and other surface features appeal more to developers than accuracy. • Prior Beliefs - Developers exhibit a tendency to bring their own privacy and security beliefs into their development process. [2], [4], [10], [27], [28], [42], [43], [45], [46], [49], [52], [53]
Herding	Following the group behavior	"How many other developers are using this particular solution"? • Social Groups - Use of social circles for various activities like testing or choosing security parameters that others in the social group use. • Network Effect - Choosing solutions based on the number of their users. Group behavior is also observed in case of assurance mechanisms [30]–[32], [43], [54]
Incentives	Misalignment of the returns with the efforts of expected behavior	"Will I be paid for the extra work to add security"? • Financial - Security prevents mass market developers from doing things that might earn more revenue. • Organizational Goals - Software development methods rush for functionality. Developers need regular motivation and organization push [2]–[4], [28], [30], [33], [36], [40], [55]–[58]

Take-away. The API providers are ideally placed to collaborate with developers and link the security benefits and pitfalls of how their APIs are used.

Conflicting goals signifies concentration of security knowledge within experts (*power play*) [29], [30], [33] leading to *shifting responsibility* and *incentives*. Derr et al. and Vaniea et al. find that security is not a priority when picking a library by app developers [28], [51]. Off-the-shelf components are built with different security policies and expectations from the developers' commitments and expectations from the system. Georgiev et al. identified man-in-the-middle (MITM) vulnerabilities in various applications and SDK. The study finds that these MITM vulnerabilities were due to poorly designed cryptographic APIs [20].

Take-away. Consideration of revenue is a rational and legitimate behavior—a reasonable approach might be to explain to developers the reason behind the restrictions, and work with them rather than blaming them.

IV. INTERVENTIONS AND TROPES THAT HAMPER THEIR EFFECTIVENESS

Table III describes four classes of interventions proposed to help developers produce secure code. This includes research on developer-centered usable security *developers as users*; as well as various technical *platform security* initiatives such as patching and sandboxing techniques, and *assurance mechanisms* such as testing, code review, and training. Some interventions try to find human solutions to human problems, e.g., workshops, incentivization sessions and on-the-job training. Others aim to find technical solutions to human problems.

Several studies [15], [70], [74], [75] report assumptions about developers' expertise and training. The issues caused by a lack of, or outdated training is exacerbated when developers are not provided with well-documented APIs and tools [11], [23], [76]. A related issue is of abstractions [34], [37]: they are used to make the APIs more usable, but high-abstraction levels can also lead to restricted flexibility [26]. The incentives for inducing developers to spend the effort to program securely are also not aligned with expected returns from doing so [28], [48]. The challenges result in bias and group behavior [42], [52]. The improvisations to which developers resort are aided by online sources which can be insecure [43], [52], [59]. Adhering to social groups and choosing answers based on

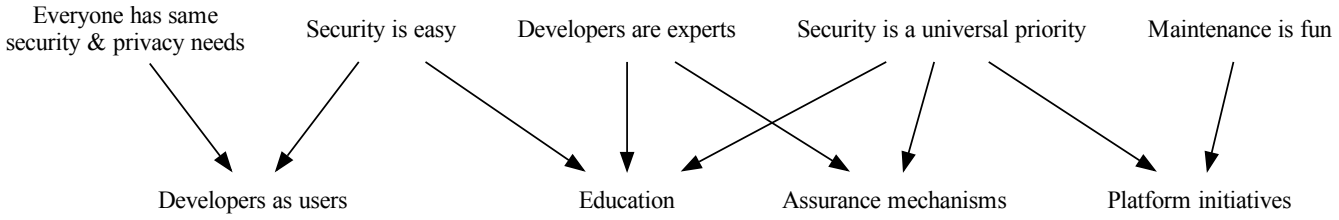

Fig. 2. The tropes we identify (top) and which interventions (bottom) they adversely impact.

TABLE III
INTERVENTIONS TO HELP DEVELOPERS PROGRAM SECURELY.

Theme	Description	Examples
Developers as users	Developers as users of APIs and code, as much as non-developers are users of their end-products. Developers need considerations for usability just like everyone else.	"How to empower developers without knowledge of security"? • Abstraction - Include non security components, safe defaults. • Interactions - Incentivization sessions, Security reminders, assist developers through compile time alerts, IDEs. [4], [5], [14], [21], [24], [29], [31]–[33], [57], [65]–[69]
Platform initiatives	Developers build code on top of platforms and within organizations. The developers and are not wholly responsible for a systems security and code can be constrained and updated externally.	"How to propagate updates to the apps"? • Sandboxing - Google application security program, Gradle, iOS app wrapping • Patching - Minimal updates without additional functionalities pushed by an API or system provider [10], [70], [71]
Assurance mechanisms	The findings of observation based studies on organized assurance mechanisms.	"How to ease the cognitive load of developers?" • Testing - Intervention mechanisms based testing the developers' outputs, for example: *Red team* exercises, penetration testing; program analysis, and fuzzing • Organizational culture - secure programming training and incentivization, code and architecture review, organizational focus on security. [14], [29]–[33], [57], [58], [66]–[69], [72]
Education	Not all developers are security experts and API providers should not assume they know how developers will use their code.	"What is the security maturity of developers"? • Experience - Developers with knowledge performed better with the interventions [11], [15], [16], [18], [26], [30]–[32], [40], [43], [57], [70], [73]–[75]

surface features are examples of finding human solutions to technical problems without solving the rooted beliefs or tropes in the larger environment.

A. Tropes

Tropes represent misplaced beliefs about developers and programming that adversely affect the secure use of APIs and the adoption of mitigating interventions (Figure 2).

The trope that *developers are experts* comes from studies where developers are expected to understand details about how to use an API without guidance from the API provider themselves [8], [11], e.g., the PyCrypto library requires developers to understand how certificate stores work [11].

The trope that *maintenance is fun* comes from studies that *mistakenly* assume that developers actively seek out and enjoy dealing with breaking changes in libraries. Derr et al. showed that the libraries in many apps could be trivially updated [28], yet were not. Vaniea and Rashidi found that API providers routinely misunderstood how hard it was for developers to adopt changes to their APIs [51].

The *security is easy* trope arises from a belief that developers are aware of the extent to which their code is insecure. Two studies found that developers are overconfident in the security of their code [29], [41]. Oltrogge et al. found that app generators routinely generated insecure apps [1]. Despite several studies highlighting that developers need more training in security [29], [42], a trope that secure programming is an easy task for developers still remains.

Similarly, there appears to be another trope that assumes that *security is a universal priority* and security is desired in applications for its own sake; while prior work highlighted that developers do not always value security as a feature [43], [77], [78]. The work of Naiakshina et al. showed that developers would only implement password storage securely if explicitly asked to [40]. This suggests that there is a mismatch between what security experts think developers value and what they do in practice.

Even if security were easy and a universal priority, it is a mistaken trope to assume that *everyone has the same security needs*. Sane defaults for security APIs [5] assume

that everyone needs a similar default level of security, and that their use of the API is identical. Different applications have different threat models and it is a mistake to assume that there are universal security choices.

B. Effect of Tropes on Interventions

Security is a universal priority inhibits [30], [32], [79] the *assurance mechanisms*, *education* and *platform initiatives*. The role of *assurance mechanisms* have been discussed as effective for secure software development. *Testing* has been identified to aid in secure development [67] while other work highlights the importance *organizational culture* through incentivization sessions, on the job training and other lightweight interventions [29], [30], [57]. An experiment with over eighty developers across eight organizations reports that interventions can be successful without security specialists but needs organizational push [80]. Another study observes that smaller organizations need adequate support for secure coding [54]. When we evaluate these interventions from two perspective of security is not a universal priority [2], [4] and it contributes to poor use of updated libraries [51], we see the adverse affect on the *platform initiatives*. Developers on their own may not make use of the *assurance mechanisms*, *education* and *platform initiatives* unless externally nudged by regular organizational motivations [3], [30], [31], [80].

Take-away. A shared ownership of the security and functionalities of the systems is the way forward even if the underlying reasons for achieving that is different for each of the stakeholders. This can make systems development as much about diplomacy as it is engineering [81].

Developers are experts risks the appropriation of the benefits of *assurance mechanisms* [16], [30] as well as *education*. Solo developers would need to understand and adopt interventions like, penetration testing, threat modeling; skills which need extensive training even within organized institutional software development teams [30]. Braz et al. empirically observes the interactions between the developer and improper inputs. Developers with adequate experience and support are able to detect improper inputs with proper priming [63]. The importance of priming has also been observed in [29], [41]. The empirical studies report that developers with *education* performs better be it for secure uses of APIs or to detect insecure uses through the interventions [15], [48], [73].

Developers are not all the same: in particular solo developers have different levels of expertise and mental models [11], [41] than those who work in large corporate teams. The expectation that solo *developers are experts* and fully understand the need for penetration testing, threat modeling, and identify correct security parameters may be misplaced; no matter what *education* or *assurance mechanisms* are available to them.

Take-away. These interventions can only be effective if they are aligned according to the security abilities of the developers. This requires collaboration.

Furthermore, *Security is easy* limits the approach of treating *developers as users* and makes *education* less effective. Not all developers are equally able and have differing levels of experience [63], [69]. Braz et al. reports the developers with experience and knowledge detected vulnerabilities better [63], a result echoed in case of empirical observation of IDE based interventions reported in [3]. *Security is* not *easy* [1] and this cannot be remedied by just attempting to educate developers on how to do it right [82]. Instead they need to be able to understand the design choices underlying an API, rather than just be shown the right way of using it.

Take-away. If API providers help explain the APIs to the developers that will remove several obstacles, including those reported in [10], [52].

Maintenance is fun adversely affects *platform initiatives*. *Platform initiatives* can include mechanisms to help ensure that applications use updated libraries; but these mechanisms can lead to complex version control systems for updates, and interference with app functionalities by the API providers. Developers must overcome a steep learning curve to properly patch their libraries. Research in [28] looks into the adoption of library updates and argues for mediated *patching* of core functionalities to prevent conflict of interests between developers and API providers. These *platform initiatives* can make *maintenance* not *fun* for developers.

Take-away. Library developers have it tough [83]. Updating and potentially breaking APIs can be problematic. API providers and developers need to collaborate in the update process rather than developers sticking with outdated libraries or API providers breaking developers code.

Everyone has same security needs adversely affects the intervention *developers as users*. Research in [4], [21] make a case for learning from the advances in HCI and treat *developers as users*. Green et al. makes 10 specific recommendations including moving the level of *abstraction* to include non security components and having safe defaults [5]. Acar et al. makes the case for safe defaults [11]. Minderman et al. argues along similar lines of [4], [5] where the authors propose putting a wrapper around the APIs [61]. These interventions will ineffect assume a uniform threat model, but threat models are not universal. The assumption *everyone has same security needs* made by the intervention *developers as users* is misplaced.

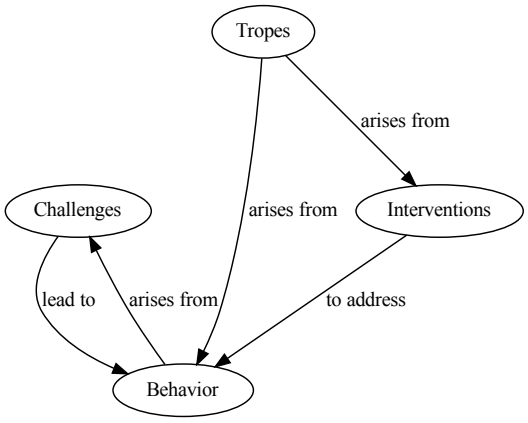

Fig. 3. Relationships between challenges, behaviors, interventions, and tropes.

Take-away. Increasing the level of *abstraction* can lead to an API becoming focused on a single threat model. We recommend threat assessments incorporating the concerns of both the API provider and the developers who use them to ensure the threat model is as true to life as possible.

V. Discussion and Conclusion

The challenges lead to revealed behavior and the interventions were designed to address the challenges and behavior (Figure 3). However, the tropes about developers lead to the failure of the interventions to adequately address all the developers' challenges and behaviors. This means that tropes about *what we think developers need* may confuse API designers and security experts, and lead to interventions which address the challenges we *think* developers have, rather than the ones they actually face.

How then can we encourage collaboration between API providers, security experts and developers? Communication can play a key role as to how knowledge is transferred to practitioners [84]. Perhaps approaches such as of Computer-Supported Collaborative Work can be adapted for secure software development and establish effective communication channels. Whilst forums can be an asynchronous platform for developers, they sometimes lead to bias and herding behaviors [42]; perhaps by combining them with tools such as *CryptoGuard* [85] to detect misuses of cryptographic APIs can help spot when advice is awry and mitigate these behaviors.

Close collaboration between all three parties may offer a solution and align the assumptions with reality. If API designers and security experts keep the developers out of the loop by not requiring them to understand security, then developers will have less flexibility. Developers need help understanding which APIs to use, when to use them and how to use them safely. They need to understand how to debug the security aspects of their own applications. API designers and security experts need to help developers have this autonomy by helping developers understand security details (e.g. modes, algorithms, iterations in cryptography) and by offering them matching incentives in terms of usability [3], [30], [86]. Broadly, it will build an understanding of the abilities, expectations, goals and preferences of the developers and API providers. The understanding will help developers say clearly what they want the system to do (functionalities) in the terms of the API designers, and understand the things the system should not do (security) in their own terms.

The API providers are also developers. Their contexts, priorities and perceptions of security will influence what is possible from a library. Realistic tropes about developers will enable the realization of effective interventions sitting at the intersection of developers and API providers. The effective interventions can be technical, human or a combination of both. Collaboration will lead to a more dynamic approach to securing systems; the usability problems would be easy to identify and can be continually addressed.[2] By collaborating with developers, and not treating them as enemies nor users, we can create secure software that works for all.

References

[1] M. Oltrogge, E. Derr, C. Stransky, Y. Acar, S. Fahl, C. Rossow, G. Pellegrino, S. Bugiel, and M. Backes, "The Rise of the Citizen Developer: Assessing the Security Impact of Online App Generators," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018.

[2] M. Tahaei and K. Vaniea, "A Survey on Developer-Centred Security," in *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2019.

[3] J. Xie, H. Lipford, and B.-T. Chu, "Evaluating Interactive Support for Secure Programming," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2012.

[4] Y. Acar, S. Fahl, and M. L. Mazurek, "You are Not Your Developer, Either: A Research Agenda for Usable Security and Privacy Research Beyond End Users," in *2016 IEEE Cybersecurity Development (SecDev)*, 2016.

[5] M. Green and M. Smith, "Developers are Not the Enemy!: The Need for Usable Security APIs," *IEEE Security Privacy*, vol. 14, 2016.

[6] N. Patnaik, A. C. Dwyer, J. Hallett, and A. Rashid, "Don't forget your classics: Systematizing 45 years of Ancestry for Security API Usability Recommendations," 2021.

[7] A. Michalec, D. Van Der Linden, S. Milyaeva, and A. Rashid, "Industry Responses to the European Directive on Security of Network and Information Systems (NIS): Understanding policy implementation practices across critical infrastructures," in *Proceedings of the Sixteenth Symposium on Usable Privacy and Security*, Aug. 2020.

[8] N. Patnaik, J. Hallett, and A. Rashid, "Usability Smells: An Analysis of Developers' Struggle With Crypto Libraries," in *Proceedings of the Fifteenth Symposium on Usable Privacy and Security*, Aug. 2019.

[9] C. Wohlin, "Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, 2014.

[10] S. Nadi, S. Krüger, M. Mezini, and E. Bodden, ""Jumping Through Hoops": Why do Java Developers Struggle with Cryptography APIs?" in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, 2016.

[11] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky, "Comparing the Usability of Cryptographic APIs," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017.

[12] N. Meng, S. Nagy, D. Yao, W. Zhuang, and G. Arango-Argoty, "Secure Coding Practices in Java: Challenges and Vulnerabilities," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, 2018.

[2] David Deutsch's book "The Beginning of Infinity" in the chapter "Unsustainable" reflects on the spirit of collaboration about science.

[13] J. Smith, B. Johnson, E. Murphy-Hill, B. Chu, and H. R. Lipford, "Questions Developers Ask While Diagnosing Potential Security Vulnerabilities with Static Analysis," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015.

[14] D. C. Nguyen, D. Wermke, Y. Acar, M. Backes, C. Weir, and S. Fahl, "A Stitch in Time: Supporting Android Developers in Writing Secure Code," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[15] J. Zhu, J. Xie, H. R. Lipford, and B. Chu, "Supporting secure programming in web applications through interactive static analysis," *Journal of Advanced Research*, vol. 5, 2014.

[16] H. Assal, S. Chiasson, and R. Biddle, "Cesar: Visual representation of source code vulnerabilities," in *2016 IEEE Symposium on Visualization for Cyber Security (VizSec)*, 2016.

[17] S. Fahl, M. Harbach, H. Perl, M. Koetter, and M. Smith, "Rethinking SSL Development in an Appified World," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, 2013.

[18] M. Oltrogge, Y. Acar, S. Dechand, M. Smith, and S. Fahl, "To Pin or Not to Pin—App Developers Bullet Proof Their TLS Connections," in *24th USENIX Security Symposium (USENIX Security 15)*, Aug. 2015.

[19] J. Smith, B. Johnson, E. Murphy-Hill, B. Chu, and H. R. Lipford, "How Developers Diagnose Potential Security Vulnerabilities with a Static Analysis Tool," *IEEE Transactions on Software Engineering*, vol. 45, 2019.

[20] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov, "The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, 2012.

[21] A. Rashid, *Developer-Centred Security*, 2019.

[22] G. Wurster and P. C. van Oorschot, "The Developer is the Enemy," in *Proceedings of the 2008 New Security Paradigms Workshop*, 2008.

[23] D. S. Oliveira, T. Lin, M. S. Rahman, R. Akefirad, D. Ellis, E. Perez, R. Bobhate, L. A. DeLong, J. Cappos, Y. Brun, and N. C. Ebner, "API Blindspots: Why Experienced Developers Write Vulnerable Code," in *Proceedings of the USENIX Symposium on Usable Privacy and Security (SOUPS)*, August 2018.

[24] K. Yskout, R. Scandariato, and W. Joosen, "Does Organizing Security Patterns Focus Architectural Choices?" in *Proceedings of the 34th International Conference on Software Engineering*, 2012.

[25] M. Christakis and C. Bird, "What Developers Want and Need from Program Analysis: An Empirical Study," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016.

[26] L. Lo Iacono and P. L. Gorski, "I Do and I Understand. Not Yet True for Security APIs. So Sad," in *European Workshop on Usable Security*, 4 2017.

[27] A. Senarath and N. A. G. Arachchilage, "Why Developers Cannot Embed Privacy into Software Systems? An Empirical Investigation," in *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, 2018.

[28] E. Derr, S. Bugiel, S. Fahl, Y. Acar, and M. Backes, "Keep Me Updated: An Empirical Study of Third-Party Library Updatability on Android," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[29] T. W. Thomas, M. Tabassum, B. Chu, and H. Lipford, "Security During Application Development: An Application Security Expert Perspective," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018.

[30] C. Weir, I. Becker, J. Noble, L. Blair, A. Sasse, and A. Rashid, "Interventions for Software Security: Creating a Lightweight Program of Assurance Techniques for Developers," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2019.

[31] S. Xiao, J. Witschey, and E. Murphy-Hill, "Social Influences on Secure Development Tool Adoption: Why Security Tools Spread," in *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, 2014.

[32] J. Witschey, O. Zielinska, A. Welk, E. Murphy-Hill, C. Mayhorn, and T. Zimmermann, "Quantifying Developers' Adoption of Security Tools," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015.

[33] A. Poller, L. Kocksch, S. Türpe, F. A. Epp, and K. Kinder-Kurlanda, "Can Security Become a Routine? A Study of Organizational Change in an Agile Software Development Group," in *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, 2017.

[34] D. Harborth, S. Pape, and K. Rannenberg, "Explaining the Technology Use Behavior of Privacy-Enhancing Technologies: The Case of Tor and Jondonym," *Proceedings on Privacy Enhancing Technologies*, vol. 2020, 2020.

[35] D. Oliveira, M. Rosenthal, N. Morin, K.-C. Yeh, J. Cappos, and Y. Zhuang, "It's the Psychology Stupid: How Heuristics Explain Software Vulnerabilities and How Priming Can Illuminate Developer's Blind Spots," in *Proceedings of the 30th Annual Computer Security Applications Conference*, 2014.

[36] M. Hilton, N. Nelson, T. Tunnell, D. Marinov, and D. Dig, "Trade-Offs in Continuous Integration: Assurance, Security, and Flexibility," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017.

[37] R. Anderson, C. Barton, R. Böhme, R. Clayton, M. J. G. van Eeten, M. Levi, T. Moore, and S. Savage, *Measuring the Cost of Cybercrime*, 2013.

[38] A. Acquisti, L. Brandimarte, and G. Loewenstein, "Secrets and Likes: The Drive for Privacy and the Difficulty of Achieving it in the Digital Age," *Journal of Consumer Psychology*, 2021.

[39] K. R. Daniel Votipka, "Understanding security mistakes developers make: Qualitative analysis from Build It, Break It, Fix It," *USENIX Security Symposium*, 2020.

[40] A. Naiakshina, A. Danilova, E. Gerlitz, E. von Zezschwitz, and M. Smith, *"If You Want, I Can Store the Encrypted Password": A Password-Storage Field Study with Freelance Developers*, 2019.

[41] J. Hallett, N. Patnaik, B. Shreeve, and A. Rashid, ""Do this! Do that!, And nothing will happen" Do specifications lead to securely stored passwords?" in *International Conference on Software Engineering*, Jan. 2021.

[42] D. Van Der Linden, E. Williams, J. Hallett, and A. Rashid, "The impact of surface features on choice of (in)secure answers by Stackoverflow readers," *IEEE Transactions on Software Engineering*, vol. 0, Apr. 2020.

[43] D. Van Der Linden, P. Anthonysamy, B. Nuseibeh, T. Tun, M. Petre, M. Levine, J. Towse, and A. Rashid, "Schrödinger's Security: Opening the Box on App Developers' Security Rationale," in *ICSE '20: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, Jun. 2020.

[44] A. Edmundson, B. Holtkamp, E. Rivera, M. Finifter, A. Mettler, and D. Wagner, "An Empirical Study on the Effectiveness of Security Code Review," in *Proceedings of the 5th International Conference on Engineering Secure Software and Systems*, 2013.

[45] S. Jain and J. Lindqvist, "Should I Protect You? Understanding Developers' Behavior to Privacy-Preserving APIs," in *NDSS Symposium 2014*, 01 2014.

[46] A. Naiakshina, A. Danilova, C. Tiefenau, M. Herzog, S. Dechand, and M. Smith, "Why Do Developers Get Password Storage Wrong? A Qualitative Usability Study," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[47] Y. Acar, C. Stransky, D. Wermke, M. L. Mazurek, and S. Fahl, "Security Developer Studies with Github Users: Exploring a Convenience Sample," in *Proceedings of the Thirteenth USENIX Conference on Usable Privacy and Security*, 2017.

[48] A. Naiakshina, A. Danilova, C. Tiefenau, and M. Smith, "Deception Task Design in Developer Password Studies: Exploring a Student Sample," in *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, Aug. 2018.

[49] I. Hadar, T. Hasson, O. Ayalon, E. Toch, M. Birnhack, S. Sherman, and A. Balissa, "Privacy by Designers: Software Developers' Privacy Mindset," in *Proceedings of the 40th International Conference on Software Engineering*, 2018.

[50] Y. Sawaya, M. Sharif, N. Christin, A. Kubota, A. Nakarai, and A. Yamada, "Self-Confidence Trumps Knowledge: A Cross-Cultural Study of Security Behavior," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2017.

[51] K. Vaniea and Y. Rashidi, "Tales of Software Updates: The Process of Updating Software," in *CHI '16*, 2016.

[52] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, "You Get Where You're Looking for: The Impact of Information Sources on Code Security," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016.

[53] A. R. Senarath and N. A. G. Arachchilage, "Understanding user privacy expectations: A software developer's perspective," *Telematics and Informatics*, vol. 35, 2018.

[54] R. Balebako, A. Marsh, J. Lin, J. Hong, and L. F. Cranor, "The Privacy and Security Behaviors of Smartphone App Developers," in *Workshop on Usable Security*, 2014.

[55] H. Assal and S. Chiasson, "Security in the Software Development Lifecycle," in *Proceedings of the Fourteenth USENIX Conference on Usable Privacy and Security*, 2018.

[56] D. Ashenden and G. Ollis, "Putting the Sec in DevSecOps: Using Social Practice Theory to Improve Secure Software Development," in *New Security Paradigms Workshop 2020*, 2020.

[57] J. M. Haney, M. Theofanos, Y. Acar, and S. S. Prettyman, ""We make it a big deal in the company": Security Mindsets in Organizations that Develop Cryptographic Products," in *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, Aug. 2018.

[58] A. A. U. Rahman and L. Williams, "Software Security in DevOps: Synthesizing Practitioners' Perceptions and Practices," in *2016 IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED)*, 2016.

[59] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why Eve and Mallory Love Android: An Analysis of Android SSL (in)Security," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, 2012.

[60] M. P. Robillard and R. Deline, "A Field Study of API Learning Obstacles," *Empirical Softw. Engg.*, vol. 16, Dec. 2011.

[61] K. Mindermann, P. Keck, and S. Wagner, "How Usable Are Rust Cryptography APIs?" in *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2018.

[62] M. Tahaei, A. Frik, and K. Vaniea, "Deciding on Personalized Ads: Nudging Developers About User Privacy," in *Seventeenth Symposium on Usable Privacy and Security (SOUPS '21)*, 2021.

[63] L. Braz, E. Fregnan, G. Çalikli, and A. Bacchelli, "Why Don't Developers Detect Improper Input Validation? '; DROP TABLE Papers; –," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021.

[64] D. M. German, "The GNOME project: a case study of open source, global software development," *Software Process: Improvement and Practice*, vol. 8, 2003.

[65] N. Ayewah, W. Pugh, D. Hovemeyer, J. D. Morgenthaler, and J. Penix, "Using Static Analysis to Find Bugs," *IEEE Software*, vol. 25, 2008.

[66] C. Weir, A. Rashid, and J. Noble, "How to Improve the Security Skills of Mobile App Developers? Comparing and Contrasting Expert Views," in *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, Jun. 2016.

[67] J. Such, A. Gouglidis, W. Knowles, G. Misra, and A. Rashid, "Information assurance techniques: Perceived cost effectiveness," *Computers and Security*, vol. 60, Jul. 2016.

[68] I. A. Tondel, M. G. Jaatun, and P. H. Meland, "Security Requirements for the Rest of Us: A Survey," *IEEE Software*, vol. 25, 2008.

[69] J. Zhu, H. R. Lipford, and B. Chu, "Interactive Support for Secure Programming Education," in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, 2013.

[70] M. Whitney, H. Lipford-Richter, B. Chu, and J. Zhu, "Embedding Secure Coding Instruction into the IDE: A Field Study in an Advanced CS Course," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 2015.

[71] A. Z. Baset and T. Denning, "IDE Plugins for Detecting Input-Validation Vulnerabilities," in *2017 IEEE Security and Privacy Workshops (SPW)*, 2017.

[72] N. Ayewah and W. Pugh, "A Report on a Survey and Study of Static Analysis Users," in *Proceedings of the 2008 Workshop on Defects in Large Software Systems*, 2008.

[73] M. Tabassum, S. Watson, and H. Richter Lipford, "Comparing Educational Approaches to Secure programming: Tool vs. (TA)," in *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, Jul. 2017.

[74] T. Thomas, B. Chu, H. Lipford, J. Smith, and E. Murphy-Hill, "A study of interactive code annotation for access control vulnerabilities," in *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2015.

[75] T. W. Thomas, H. Lipford, B. Chu, J. Smith, and E. Murphy-Hill, "What Questions Remain? An Examination of How Developers Understand an Interactive Static Analysis Tool," in *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, Jun. 2016.

[76] J. Smith, L. N. Q. Do, and E. R. Murphy-Hill, "Why Can't Johnny Fix Vulnerabilities: A Usability Evaluation of Static Analysis Tools for Security," in *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*, Aug. 2020.

[77] D. van der Linden, I. Hadar, M. Edwards, and A. Rashid, "Data, data, everywhere: quantifying software developers' privacy attitudes," 09 2019.

[78] I. Rauf, D. van der Linden, M. Levine, J. Towse, B. Nuseibeh, and A. Rashid, "Security but Not for Security's Sake: The Impact of Social Considerations on App Developers' Choices," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020.

[79] L. Williams, G. McGraw, and S. Migues, "Engineering Security Vulnerability Prevention, Detection, and Response," *IEEE Software*, vol. 35, sep 2018.

[80] C. Weir, I. Becker, and L. Blair, "A Passion for Security: Intervening to Help Software Developers," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, may 2021.

[81] B. Christianson, "Living In An Impossible World," *Philosophy and Technology*, vol. 26, January 2013.

[82] M. Tahaei, A. Jenkins, K. Vaniea, and M. K. Wolters, ""I Don't Know Too Much About It": On the Security Mindsets of Computer Science Students," in *Socio-Technical Aspects in Security and Trust*, 2021.

[83] G. Kroah-Hartman, "Non-technical security best-practices for open source projects," June 2021.

[84] A.-S. Claeys and W. T. Coombs, "Organizational Crisis Communication: Suboptimal Crisis Response Selection Decisions and Behavioral Economics," *Communication Theory*, vol. 30, 03 2019.

[85] S. Rahaman, N. Meng, and D. Yao, "Tutorial: Principles and Practices of Secure Crypto Coding in Java," in *2018 IEEE Cybersecurity Development (SecDev)*, 2018.

[86] M. Tahaei, K. Vaniea, B. Konstantin, and M. K. Wolters, "Security Notifications in Static Analysis Tools: Developers' Attitudes, Comprehension, and Ability to Act on Them," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021.