Security Interventions: Pushing Programmers To Become The Solution

Lloyd Hilsgen
hilsg008@umn.edu
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

Abstract

In recent literature, there has been a move away from dealing with security breaches and towards proactively fighting them before they happen. To best address potential security breaches we must address both code vulnerabilities and the developers whose coding practices produce them. We can break down the sources of human error into 3 major groups: knowledge deficits, attention deficits, and intention deficits. To address this human error, we use security interventions. Security interventions are techniques and resources that push the programmer to implement secure code. While most security interventions address knowledge deficits, addressing the other 2 requires convincing programmers of the value of using more secure implementations of what they're writing or have written. This paper describes research on security interventions produced by Rauf et al. [6]. This research concludes that security interventions are more successful when there is less attention on specific code fixes and more on convincing programmers of potential security issues. Developers that are successfully convinced will adjust their coding habits and therefore be more likely to implement secure code.

1 Introduction

When developers write applications people use everyday, it is possible to introduce unexpected vulnerabilities that can be exploited by those with less than pure intentions. The theme of this paper will be identifying and developing strategies to combat the introduction of these vulnerabilities.

One might think code vulnerabilities are purely accidental mistakes that arise from inexperience or ignorance, and once a source of vulnerabilities has been identified, it is quickly removed. However, 83% of data breaches occurred in organizations that have previously dealt with data breaches, with the average cost of each data breach being $4.35 million [1]. When security breaches like these happen, it is easy to focus on the simple code change that would've avoided such a breach. As such, when programmers and suggestions provided by developer tools are trying to prevent such breaches, it is too common to simply step in and improve single lines of code. While code fixes can remove vulnerabilities, avoiding the introduction of security flaws requires a more developer-centric approach that incorporates more context than can be captured by any individual line of code [3].

The human error that leads to code vulnerabilities can be categorized into three main groups [6] that will be explained further in Section 3.

- **Knowledge deficits:** when a developer or group of developers do not know enough information about a vulnerability to be able to identify it.
- **Attention deficits:** when code becomes so complex that it becomes difficult to pay attention to details, thus allowing subtle errors that cause vulnerabilities to be overlooked.
- **Intention deficits:** when an issue is known, but the easier and more simplistic solution is chosen anyways.

To avoid these deficits and produce better code, software developers have started incorporating security interventions. Security interventions are when other programmers, experts or security tools search for and attempt to correct potential security breaches before they happen. These interventions can only be effective if used correctly. As such, developers must examine how these interventions are used and change their habits to maximize the benefits of security interventions. Rather than solving simple mistakes, focusing on adopting the best practices results in better developers, who produce fewer vulnerabilities [3].

These security interventions can be categorized into three main groups.

- **Awareness Interventions:** Resources and solutions designed to improve the coding habits of developers by providing information to developers about potential issues. This can happen while writing code, or asynchronously.
- **Automated Interventions:** Resources and solutions designed to automatically provide corrections for problematic code while the developer is writing it.
- **Interactive Interventions:** Resources and solutions that test code and provide information through the use of APIs. Interactive interventions differ from automated interventions in that they provide a list of potential fixes and further resources through a developer-friendly interface.

The primary focus of this paper is on the security paradigm developed by Rauf et al. [6] to help developers avoid the code practices responsible for code vulnerabilities. In Section 2, I expand upon the 3 groups of security interventions. In Section 3, I will examine the sources of human error that lead to code vulnerabilities. In Section 4, I will explain how Raum et al. used the sources of human error to analyse the 3 groups of security interventions. In Section 5, I will show how existing security interventions can be improved to adapt to developer needs.

2 Security Interventions

In this paper, security interventions are going to be defined as techniques and resources that push the programmer to implement more secure code. These can be anything from code review sessions, to programs that automatically test code for potential security vulnerabilities. While the depth of said interventions varies wildly depending on circumstance, workplace culture, and source, there can still be many crucial parallels between them, as illustrated in Figure 1. Interventions that focus on persuading developers to implement solutions rather than instructing them to do a certain fix, are much more successful [3].

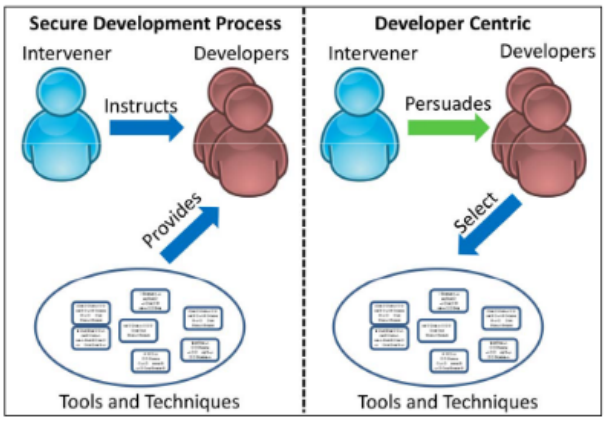

Figure 1. Crucial differences between secure development and developer centric interventions, from [3]

Analysis of existing security resources provides us with 3 major categories of security interventions, detailed in the following 3 subsections.

2.1 Awareness Interventions

Awareness interventions are resources and approaches used by the developer to improve security, designed to bring awareness of new threats. These can be things such as vulnerability databases, checklists, or online articles. Because the majority of programmers source their security information directly from Google [4], these can often be the only resources a developer uses. Like Google itself these resources provide the developer with quick and relatively easy solutions to the most blatant programming errors [6]. Awareness interventions by their nature bring awareness to a specific issue or group of issues.

2.2 Automated Interventions

Automated interventions are programs that run through code to search for and often mitigate known vulnerabilities. They range in their methodology, from those written to directly check for well-known, specific code problems, to Machine Learning algorithms that attempt to break your code. The largest benefit from these systems is that they automatically implement checks for new vulnerabilities, so that developers can avoid the high overhead of trying to keep up with the latest discoveries, while still feeling confident in the security of their work. They also often provide more than the simple code fix. A large number of them will reference awareness interventions as a way to provide further understanding and actually convince a developer of the value of a specific solution [6].

2.3 Interactive Interventions

Interactive interventions are interfaces that provide the developer with a better understanding of the vulnerabilities being addressed by incorporating suggestions for code fixes and information into a far more easily understood package. While the majority of interactive interventions are relatively new, the increased accessibility of such programs makes them much better at convincing the majority of developers to adopt security fixes [6].

Interactive interventions combine the best of the other two types. Their interfaces make them much easier for inexperienced developers, while their automated checks can quickly provide feedback and be continuously improved to be up to date with the latest vulnerabilities.

3 Sources of Security Issues

When writing code, developers can make mistakes which lead to vulnerabilities. By categorizing the sources of these mistakes, we can get a better understanding of how to stop developers from producing insecure code. Rauf et al. categorizes the deficits that developers may have when writing code into the three categories outlined in the following 3 subsections. In this paper we will define internal factors as specific to an individual developer and external factors as organization wide factors. In table 2 we provide examples of the 3 deficits and the differences between internal and external factors.

3.1 Knowledge Deficits

The clearest source of security issues are those arising from a lack of information. The majority of programmers do not

Table 1. A list of common security tools, organized based on the categories outlined in this section. AW= Awareness Interventions, AU= Automated Interventions, IN= Interactive Interventions

Intervention	Type
Open Web Application Security Project	AW
Common Vulnerability and Exposures	AW
Info Pamphlets	AW
Vulnerability Databases	AW
Checklists	AW
Code Review	AW
Application Testing	AU
Automated Theorem Provers	AU
Vulnerability Prediction Tools	AU
Self Written Tests	AU
Snyk	IN
ASIDE	IN
PyCrypto API	IN

write their code completely from scratch: they will take advantage of other resources such as libraries, packages, and executables. However these resources can have vulnerabilities that the developer is unaware of. When a programmer, resource, or expert doesn't know that the implementation being provided is insecure, this is known as a Knowledge Deficit. These deficits can come from a variety of sources.

For example, the most common source of code solutions for most programmers is Google, rather than the specialized tools shown in Table 1 [4]. Even those who use security interventions may be too inexperienced or the solutions themselves may still be vulnerable to attack. For those who are only working with awareness interventions the potential for Knowledge Deficits can be much higher without constant research. For security interventions to be useful to developers, those without experience need to be given the time and resources to deepen their understanding.

3.2 Attention Deficits

Building larger and more complex systems will inevitably lead to the introduction of more bugs, which can potentially be exploited by malicious third parties. With any kind of complex task the most difficult problems to avoid are ones where the difficulty involved with programming causes short lapses in judgement, known as attention deficits. Practices that force developers to multitask can make these lapses far more common.

We know that lapses in judgement regarding socially engineered breaches such as phishing emails are more common during multitasking [7]. Multitasking while programming has the same effect, it causes what would be the knowledge and intent necessary to program securely to falter. Keeping a developer engaged with and an active component of the production of secure code can push them towards the better solutions.

3.3 Intention Deficits

Currently, the majority of developers consider software security to be "not their problem" [3]. Because the majority of attitudes that developers have towards security source from the organizations that they work in [3], if an organization does not prioritize security, neither will its developers. While all organizations have barriers to producing secure code such as time and money requirements, customer priority, and the extra effort required, those that do not care about security will often unintentionally create more. Organizations that do not care about security will often place security in the hands of a small portion of their staff and encourage those not in

Table 2. A list of several common factors in security breaches. I: Internal Factors, E: External Factors, KD= Knowledge Deficit, AD= Attention Deficit, ID= Intention Deficit

No.	Internal/External Factor	Deficit
I1	Misconceptions	KD
I2	Use of Outdated Information	KD
I3	False Assumptions/ Inferences	KD
I4	Misplaced Trust on Frameworks/APIs	KD
I5	Lack of Domain Knowledge	KD
I6	Lack of Experience	KD
I7	Lack of Knowledge of Tools/Vulnerability	KD
I8	Not Identifying Security Blind Spots	AD
I9	Not Handling Cognitive Load	AD
I10	Developer's Insecure Habits	AD
I11	Loss of Focus on Security	AD/ID
I12	Requires Too Much Effort	ID
I13	Disregarding Usefulness of Secure Practices	ID
I14	Perceived Lack of Own Security Knowledge	ID
I15	Attitude of "Someone Else's Responsibility"	ID
I16	Attitude of "No One Will Notice/Care"	ID
E1	Inadequate Information To Be Found	KD
E2	Lack of Information Sharing Among Teams	KD
E3	Task Complexity	AD
E4	Poor Division of Labor	AD
E5	Absence of Expectation of Secure Coding	AD/ID
E6	Limited Resources	AD/ID
E7	Lack of Security Culture	ID
E8	Lack of Prioritization of Security Features	ID
E9	Usability Issues With Security Tools	KD/ID

that group to rush through solutions. Customers and managers who do not value security will accept vulnerabilities as a fact of life and encourage others to do the same.

While the systemic use of security tools decreases the chances of code vulnerabilities [6], workplaces that use security tools can still create cultures that do not prioritize security. When security is seen as a hindrance on the development process, such as when it feels as though security tools are being "forced onto" the organization by higher ups, developers will ignore the suggestions provided by these security tools. If an organization wants to produce secure code, developers need to be convinced that security is valuable, such as in Figure 1

4 Drawbacks of Current Security Interventions

4.1 Awareness Interventions

By their very nature, awareness interventions are designed to bring attention to very specific issues. They assume that by even engaging with these tools a developer has already been convinced of the value of secure code and the specific solutions provided. Code reviews, for example, are generally as effective as workplace culture allows them to be [2]. If the developers already place security as a priority, so they don't have intention deficits, and are able to comprehend the code being reviewed, so they don't have attention deficits, code reviews can be incredibly effective at avoiding vulnerabilities.

This means that while awareness interventions can be great at avoiding knowledge deficits, these interventions provide little for the majority of developers lacking in either attention or intention. This can worsen the existing problem if the developer already does not have confidence in their own security knowledge, as their lack of understanding and the complexity of the solution can serve as a barrier to entry. Developers who lack confidence may also use these interventions as evidence that security should be left to experts. Not only does this lack of understanding make it harder to believe in the value of implementing a certain fix, but it also convinces developers that they need some kind of training to produce any secure code.

4.2 Automated Interventions

Because automated interventions search through code and provide solutions to known errors, they are automatically fighting errors regardless of the type of deficit responsible for the error. However, it is crucial that the programmer using automated interventions uses them responsibly. Developers who use a single automated intervention as the only proof that code is secure will introduce vulnerabilities into their code if they try to brute force a solution. On the same note, an over reliance on these tools without a deeper understanding of the vulnerabilities being caught can contribute to an existing knowledge or intention deficit. Because the majority of these systems are designed as a "black box" where code goes in and vulnerabilities come out [6], if a developer does not understand the tool being provided, there's a strong incentive to ignore the tools findings [6]. These black box style systems can also convince developers that they do not have the skills necessary to produce secure code. If the resource does not provide adequate information about the potential for a vulnerability of being exploited, developers can make poor decisions whether to implement solutions to the vulnerabilities produced as they lack the knowledge necessary to know what to prioritize [8].

4.3 Interactive Interventions

By combining the best of the previous two types of interventions, interactive interventions are much better at pushing developers to see themselves as a part of the push for more secure code. They provide both solutions to and the resources for insecure implementations while remaining user friendly. This helps developers focus on improving their habits rather than the specific code fixes. Focusing on the developer has been shown to be far more effective in the short and long term at producing secure code [3]. The increased accessibility, especially given that some of these interventions such as Snyk can be installed as a simple VSCode plugin, makes interactive interventions easier to implement and better at getting developers to use them.

Table 3. The ability for each group of interventions to address deficits [6]. KD: Knowledge Deficits, AD: Attention Deficits, ID: Intention Deficits

	Awareness	**Automated**	**Interactive**
KD	✔	✔	✔
AD	?	✔	✔
ID	×	?	✔

4.4 Comparing These Interventions

To summarize the previous 3 subsections, I will use Table 3.

Awareness interventions, by providing awareness to a potential code vulnerability are designed to combat knowledge deficits. However, depending on the quality of the awareness intervention, it may increase or decrease a developer's ability to notice the vulnerability it's trying to combat. For example, an awareness intervention could be filled with jargon or poorly designed. This would make it so a developer may know that the vulnerability exists, but unlikely that the developer would be able to identify it. Also dependant on the quality of an intervention is the general ability for it to combat intention deficits. The majority of awareness interventions are designed to be viewed by those with some reason to care about security in the first place [6]. As such,

these awareness interventions would not be able to combat any intention deficit.

Because automated interventions are able to run through the code itself, they are able to combat all three deficits. However, developers may not have the intent necessary to implement the more secure versions that are provided by these interventions. Depending upon the quality of an automated intervention, it may provide little information on how to solve any vulnerabilities it detects. If a developer with an intention deficit was given this sparse information, it would likely cause them to ignore the responses given. Automated interventions may also be difficult to use, or in some way act as a barrier to the development process. These barriers to a security tool would turn the security tool, and therefore secure coding practices, into an enemy that developers would much rather avoid.

Interactive interventions are similar to automated interventions, as such they are also able to combat all three deficits. However, unlike automated interventions, interactive interventions provide further resources for any of the vulnerabilities it finds, work with the developer, and focus on being developer friendly. This developer centric focus means that it is far less likely that a developer with some intention deficit would ignore the solutions provided. These interventions also acknowledge that there are instances when a vulnerability is unlikely to be exploited, and therefore could be ignored to safe time and resources. By scoring any vulnerability on the chance that it will be exploited, interactive interventions allow developers to implement securely even in organizations that do not prioritize security.

5 Adapting Security Interventions To Developer Needs

As shown above, security interventions can have a varying ability at addressing knowledge and attention deficits, however they often miss the human aspect of how the solutions provided actually gets implemented. These tools often assume that the developer has an infinite amount of time, energy, and resources. No matter how useful an intervention can be, it can not be assumed that the developer will change their attitudes towards security as a whole.

5.1 Changing the attitude towards security

To combat intention deficits, we must change the culture surrounding development. While some of this can be done simply by convincing people to use ***any*** kind of intervention [5], adopting more secure practices must be a more active process. As was discussed in Section 4.4, developers with intention deficits may attempt to ignore any security intervention implemented. If incorporated into an organization in the wrong way, security interventions can bring far less improvement than expected. Therefore, improving security on an organizational level requires also changing security culture. Security must be something that the majority of individual developers feel invested in. Developers must feel that it is valuable to improve their coding practices and improve organization wide security culture [3]. This also means combating the "blame game." If a culture is to take security seriously, all involved are an essential piece in writing secure code.

5.2 Convince the programmer

Security is an active process, done by all developers involved. By pushing individual developers to not just be forced into secure programming, but to see themselves as someone who writes secure code, the typical barrier to entry associated with security can be heavily reduced [3]. Additionally, providing solutions to vulnerabilities is not enough. The developer implementing such changes needs to understand what they're implementing and why. Moving towards interventions that provide the developer with more information on how said vulnerabilities work and the level of threat they pose if left untouched allows developers to make better security decisions.

5.3 Provide Resources

Even though all security interventions provide some improvement to the security of code, the majority of these tools do not provide enough information about the code fixes being provided [6]. If they do provide further explanation, it is done in a way that does not adequately fulfill a developers need to understand a code fix before implementing it. Even if this means something as simple as linking to a Wikipedia page, inclusions of further reading encourage a different relationship with security knowledge [3]. Higher quality resources will provide an understanding that being a secure programmer is an achievable goal and will provide some portion of the path towards that goal.

6 Conclusion

In conclusion, to combat knowledge, attention, and intention deficits developers need to move towards security interventions that provide more than simple code fixes. Security interventions need to be improved in 3 main ways: focusing on pushing developers towards embracing security, providing the resources to come to deeper understandings, and convincing developers that the solutions they provide are worth implementing and learning about. Security however, is about more than just the tools programmers use, it also requires addressing workplace culture. Developers need to see themselves as a part of security. More developers need to get on board with improving their programming habits so that they will produce better code overall. Rauf et al. and Jordan et al. make a case that to achieve these goals those who create security interventions need to adapt interventions to developer needs. These researchers also make the

case that developers and organizations need to adopt interventions that better address the potential deficits leading to code vulnerabilities.

References

[1] [n. d.]. Cost of a data breach 2022. https://www.ibm.com/reports/data-breach

[2] [n. d.]. OWASP Code Review Guide | OWASP Foundation. https://owasp.org/www-project-code-review-guide/

[3] Charles, Charles Weir, Ingolf Becker, James Noble, Lynne Blair, M. Angela Sasse, and Awais Rashid. 2019. Interventions for software security: creating a lightweight program of assurance techniques for developers. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '19)*. IEEE Press, Montreal, Quebec, Canada, 41–50. https://doi.org/10.1109/ICSE-SEIP.2019.00013

[4] Felix Fischer, Yannick Stachelscheid, and Jens Grossklags. 2021. The Effect of Google Search on Software Security: Unobtrusive Security Interventions via Content Re-ranking. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. ACM, Virtual Event Republic of Korea, 3070–3084. https://doi.org/10.1145/3460120.3484763

[5] Tiffany Brooke Jordan, Brittany Johnson, Jim Witschey, and Emerson Murphy-Hill. 2014. Designing Interventions to Persuade Software Developers to Adopt Security Tools. In *Proceedings of the 2014 ACM Workshop on Security Information Workers (SIW '14)*. Association for Computing Machinery, New York, NY, USA, 35–38. https://doi.org/10.1145/2663887.2663900

[6] Irum Rauf, Marian Petre, Thein Tun, Tamara Lopez, Paul Lunn, Dirk Van Der Linden, John Towse, Helen Sharp, Mark Levine, Awais Rashid, and Bashar Nuseibeh. 2022. The Case for Adaptive Security Interventions. *ACM Transactions on Software Engineering and Methodology* 31, 1 (Jan. 2022), 1–52. https://doi.org/10.1145/3471930

[7] Craig Williams, Helen M. Hodgetts, Candice Morey, Bill Macken, Dylan M. Jones, Qiyuan Zhang, and Phillip L. Morgan. 2020. Human Error in Information Security: Exploring the Role of Interruptions and Multitasking in Action Slips. In *HCI International 2020 - Posters*, Constantine Stephanidis and Margherita Antona (Eds.). Vol. 1226. Springer International Publishing, Cham, 622–629. https://doi.org/10.1007/978-3-030-50732-9_80

[8] Jim Witschey, Shundan Xiao, and Emerson Murphy-Hill. 2014. Technical and Personal Factors Influencing Developers' Adoption of Security Tools. In *Proceedings of the 2014 ACM Workshop on Security Information Workers (SIW '14)*. Association for Computing Machinery, New York, NY, USA, 23–26. https://doi.org/10.1145/2663887.2663898