Studying Secure Coding in the Laboratory: Why, What, Where, How, and Who?

Ita Ryan
ADVANCE Centre for Research Training
School of Computer Science and IT
University College Cork
Cork, Ireland
ita.ryan@cs.ucc.ie

Klaas-Jan Stol
Lero, the SFI Research Centre for Software
School of Computer Science and IT
University College Cork
Cork, Ireland
k.stol@ucc.ie

Utz Roedig
Connect Research Centre
School of Computer Science and IT
University College Cork
Cork, Ireland
u.roedig@cs.ucc.ie

***Abstract*—Software security is an area of growing concern, with over 192,000 known vulnerabilities in public software at the time of writing. Many aids to secure coding exist. Assessing the effectiveness of such aids in a laboratory environment is difficult. There are a number of concerns to address, such as recruitment issues and the level of instrumentation needed to perform an accurate measurement. Based on an extensive literature review of software development aids, we describe recent approaches to running laboratory studies, their characteristics, and their benefits and drawbacks. This paper should be of use to anyone planning to undertake coding studies with software developers.**

***Index Terms*—Software security, secure development tools, secure development processes, secure development, software programmer, software developer, application security, security issue, secure programming, secure application development, secure development lifecycle**

I. Why: Introduction

Amid a proliferation of ransomware and other cybercrime, cyber patriotism, cyber espionage and even cyber warfare, software security is an area of ever-growing concern [1]. There are over 192,000 known vulnerabilities in public software at time of writing in January 2023 [2], with 25,093 added in 2022 alone [3]. Critical systems are highly vulnerable [4]. Software security is in trouble. Software is released without even cursory security precautions [5].

Recent academic work on secure coding has attempted to analyse how developers address security, and identify what motivators and constraints exist. In an extensive secure software development literature review, to be published elsewhere, we queried 23 top software, usability and security venues such as ICSE, CHI and CCS for papers describing secure coding aids. The resulting studies took a wide range of academic approaches to studying secure development, from highly-controlled laboratory tests, through artefact analysis, to ethnographic field studies allowing observations of developers '*in the wild*.' In this paper we look specifically at those studies in which recruited participants were invited to perform a coding-related task, and at the study environment for the task. From the total number of papers in our literature review (over 100 studies since 2016), we identified 16 such papers. This paper examines the study settings of these 16 laboratory papers, focusing particularly on the instrumentation and recruitment options available, what choices were made, and the pros and cons of those decisions. Our analysis should be of use to anyone planning to undertake laboratory studies with software developers. Table I presents an overview of the studies.

To the best of our knowledge, this paper represents the first attempt to describe the choices available when constructing laboratory settings for security-related development studies, and their pros and cons. Huaman et al. evaluated the features of existing secure development lab environments [6]. They used the constraints and requirements identified to create a prototype online lab environment called OLab. In their review of developer centred security papers involving user studies, Tahaei and Vaniea [7] explored characteristics such as how well recruitment and data analysis were described. They assessed adherence to good practices like defining research questions, running pilot studies and obtaining ethics approval.

Naiakshina et al. gave a detailed and informative description of their study design in the first of a series of studies on secure password storage [8]. They included issues that arose during the three pilot runs. Their second study used the same conditions [9], allowing them to extract quantitative information from the combined studies. This second paper discussed methodological considerations such as deception effect, task length, laboratory setting and qualitative v. quantitative study design, giving useful and relevant advice to other researchers.

Acar et al. set up an online, highly-instrumented laboratory environment to compare the security effect of different cryptographic libraries, [10] reusing it to evaluate GitHub users as potential subjects for secure development studies [11]. This environment became known as the '*Developer Observatory*.' Stransky et al. described lessons learned while developing and using it [12].

The next four sections focus on the key questions we seek to address: What (Sec. II), Where (Sec. III), How (Sec. IV), and Who (Sec. V). We conclude with recommendations for researchers who wish to study developers in the context of secure coding in the future.

TABLE I
STUDIES ON SECURITY-RELATED DEVELOPMENT IN LABORATORY SETTINGS

ID	Year	Lab used	Research goal	Developer Task	Recruitment	N	Payment	Ref
AL1	2016	Acar Lab	Assess the impact on code security of using differing information sources such as Stack Overflow or books.	Code 4 Android tasks with data storage, HTTPS, ICC, permissions. Different information resources provided for each of 4 conditions.	Students, developer outreach	54	$30 (US), €18 card (DE)	[13]
AD1	2017	Developer Observatory	Examine relationship between cryptographic library design / usability and security of code produced.	Code a short set of tasks using either symmetric- or asymmetric key cryptography. Use one of 5 Python cryptographic libraries.	GitHub	256	No	[10]
N1	2017	Naiakshina Lab	Assess how developers deal with password storage, and whether library support or initial priming make a difference.	Write code to store passwords and authenticate users. (Four treatment groups with varying library support and security instructions.)	Computer Science(CS) Students	20	€100	[8]
GD1	2018	Developer Observatory	Having integrated security advice into a cryptographic API, evaluate the effect on the security of resulting code.	Code symmetric encryption, symmetric key generation and storage tasks. Subjects randomly assigned to PyCrypto control condition or PyCrypto (security warning) patch condition.	GitHub, developer outreach	53	No	[14]
N2	2018	Naiakshina Lab	Evaluate the effect of using deception in developer studies. Compare password storage security between developers who are asked, and are not asked, to code securely.	Same as Naiakshina 2017 (ID N1).	CS Students	20	€100	[9]
N3	2019	None	Assess code security when developers believe that client is a genuine organisation. Hire online freelancers and compare results to previous studies (IDs N1 and N2).	Finish website by coding password storage. For realism, subjects told that previous developer has left. Two different conditions re security priming.	Freelancers online	43	€100 or €200	[15]
Sm1	2019	Smith Lab	Examine strategies developers use to find answers while using the Find Security Bugs (FSB) static analysis tool.	Assess code with the help of FSB, and justify proposed changes. Code contains vulnerabilities such as potential path traversal and predictable random number generation.	North Carolina State University students, ex-students	10	None	[16]
W1	2019	Wijayarathna Lab	Study the JSSE API's usability.	Secure a client-server connection using the JSSE API.	GitHub	11	$15 voucher	[17]
B1	2021	Braz Lab	Investigate whether developers can detect improper input invalidation, and if not, why not.	Do a code review. Code has obvious SQLI or IVQI flaw. Developers not primed, but asked to review for security afterwards.	Developer outreach	146	$5 charity	[18]
F1	2021	Developer Observatory	Does ranking secure answers more highly in Google search impact on security/functionality?	Write small Java programs to solve AES encryption tasks, using the provided instance of Google.	GitHub	218	Draw for $50 vouchers	[19]
GL1	2021	Gorski Lab	Examine the effect of placing security information at different locations in API documentation for non-security APIs.	Solve four web development tasks using the documentation provided.	Students	49	Project partial waiver	[20]
L1	2021	Luo Lab	Investigate whether developers work well with IDE-integrated cloud-based SAST tools.	Fix issues in a prepared Java application, using either the IDE prototype or AWS Console.	AWS mailing lists	32	None	[21]
H1	2021	Online tasks, questions	Evaluate whether writing a specification before writing code affects the security of the code.	Write a snippet of code or pseudocode to store a password. Half of subjects were prompted to write a specification first.	Prolific Academic	138	£5	[22]
B2	2022	Braz Lab	Investigate whether asking developers to focus on security during code review increases the detection of vulnerabilities.	Perform a code review. Four treatment groups include a control, a request for security, and two types of security checklist.	Developer outreach	150	$5 charity	[23]

TABLE I
STUDIES ON SECURITY-RELATED DEVELOPMENT IN LABORATORY SETTINGS (continued)

ID	Year	Lab used	Research goal	Developer Task	Recruitment	N	Payment	Ref
G1	2022	Developer Observatory	Evaluate a security tool called Let's Hash, geared towards helping developers to code securely.	Complete three short authentication-related tasks. Three treatment effects, using normal resources or one of two versions of Let's Hash.	Freelancer.com	179	€40	[24]
Sa1	2022	Enterprise Capture the Flag (CTF)	Assess developers' understanding of website attack and defense.	Launch as many attacks as possible on a designated website, in a CTF contest.	Internal software company challenge	82	None	[25]

II. WHAT: STUDY TOPICS

We found that the laboratory studies in our review could be divided into three topic groups, which we describe here.

A. Status Quo

Papers on the status quo looked at how developers interact with security, other things being equal. Are they any good at detecting security issues during code review, and can we tell why [18]? What (if anything) do they know about attack and defence for websites [25]? What problems do they encounter when they're trying to write code with a security API [17]? Do they even *try* to code securely? What characteristics of five different cryptography APIs help developers to use them securely [10]? What strategies do developers adopt when using well-known static analysis tool FSB [16]?

B. Impact: Resources

These papers considered the impact of the resources available to developers. What difference does the choice of information resource make when people are coding [13]? Does ranking secure resources higher in Google Search help developers to find more secure answers [19]? Do developers code more securely if a helpful tool is provided at coding time [24]? Does it help if the API pops up useful warnings when the developer makes a security mistake [14]? If the documentation for an API is modified to include security advice, will developers code more securely [20]? If they have access to a cloud-based Static Application Security Testing (SAST) tool in their IDE, will they code more securely or will they be confused by the delays in these long-running tools [21]? Does using a more security-focused library result in more secure code [8], [9]?

C. Impact: Instructions

These papers examined the effect of specifically instructing developers to code securely. Is it better for code security to leave developers to their own devices, or to ask them to code securely [8], [9]? If developers believe that their code will be used in real software, will they be more likely to code securely [15]? Do priming or security checklists help developers to find vulnerabilities during code review [23]? If developers are asked to write a specification before writing code, will the code they write be more secure [22]?

III. WHERE: STUDY SETTINGS

Studies were conducted in laboratory environments with widely differing characteristics, both on site and online. Some environments were used for several different studies. Table II presents these environments and summarises their characteristics. In most papers the study environment was not named by its creators. For convenience, we have assigned names to those study environments based on the lead author's name in the paper where the environment was first used, and the abbreviation *'Lab'*.

Four of our studies took place on site. The benefits of running on-site studies include the ability to prevent interruptions and ensure that any outside help obtained is documented [8]. Complete control of the environment allows for close observation [13], and the use of extensive [8] and/or tailored [20] instrumentation. In some cases, the availability of local expertise makes an on-site study the obvious choice [16].

Nevertheless, on-site laboratory studies have limitations, including issues for distributed research teams and practical recruitment concerns (see Section V). An obvious alternative is to move the laboratory online, using surveys or web pages. Braz et al. developed a website where study participants could undertake a code review, entering their review on a web page [18], [23]. Hallett et al. used online tasks and questions to study the impact that writing a specification has on the security of subsequent pseudocode [22].

These studies on code reviews, specifications and pseudocode did not involve writing and running code. When researching the actual act of coding, moving online implies a dramatic loss of potential instrumentation. This issue was resolved in two different ways by our researchers. Wijayarathna and Arachchilage asked developers to code on their own devices, recording screen and voice [17]. The audio and visual recordings were sent to the research team once the coding tasks were concluded. Developers worked in a familiar environment in which they were comfortable; for example, three different IDEs were used. Access to the recordings of all activities undertaken by the developers during coding solved the issue of having no instrumentation data.

The second approach was taken by Acar et al., who instrumented the online environment itself [10]. This introduced the option of online, scalable coding studies. They deployed Jupyter Notebooks online for Python lab studies, developing a study

TABLE II
THE CHARACTERISTICS OF THE LABORATORIES WE ENCOUNTERED IN OUR REVIEW, INDICATING WHAT WAS TRACKED BY EACH ONE

Lab	Online	Programming environment	Language	Instrumentation	Copy/Paste	Screen recording	Think Aloud	Info	Used by
Acar Lab	No	Android Studio	Java	Browser records websites visited	No	No	Yes	[13]	[13]
Gorski Lab	No	VS Code	Golang	Eye-tracking	Yes	Recorded	Yes	[20]	[20]
Naiakshina Lab	No	Eclipse	Java	Instrumented Ubuntu distribution	Yes	Recorded, snapshot	No	[9]	[8], [9]
Smith Lab	No	Eclipse and FSB	Java	No	From screen	Recorded	Yes	[16]	[16]
Luo Lab	?[1]	VS Code / AWS Console	Java	No	No	Some[2]	Some[2]	[21]	[21]
Braz Lab	Yes	Website	Java	NA	NA	NA	No	[18]	[18], [23]
Developer Observatory[3]	Yes	Jupyter notebooks	Python	Yes	Yes	Snapshot	No	[12]	[10], [14], [19], [24]
Enterprise Capture The Flag	Yes	Website	NA	Different attack attempts recorded	NA	NA	NA	[25]	[25]
Wijayarathna Lab	Yes	Any	Java	No	No	Recorded	Yes	[17]	[17]

[1] It is unclear whether the study was administered online or in person
[2] Three participants from each treatment group were randomly chosen for monitoring
[3] https://developer-observatory.com

framework which was later named '*Developer Observatory*.' The Developer Observatory is flexible, and was also used by Gorski et al. to evaluate API warnings [14], by Fischer et al. to assess the effect of ranking Google results by security [19], and by Geierhaas et al. to assess a password hashing tool [24]. The development and use of Developer Observatory were described in detail in an interesting paper by Stransky et al. [12]. Coordination tasks such as calibrating email invitations and managing virtual machine instances were the most challenging aspects of running the Developer Observatory.

Not included in this review, because as yet no published study uses it, is new online lab prototype OLab [6]. Its creators are not ready to release their work generally, but invite contact from interested researchers.

IV. How: Instrumentation

The studies in our review used varying degrees of instrumentation depending on the study location and requirements. Gorski et al.'s study was concerned with the placement of security information in documentation, with different placements in different treatments. They used an on-site lab setting. This allowed them to use eye tracking as well as screen and audio recordings to closely monitor participants' activities [20].

Naiakshina et al. [8] evaluated password storage code against a list of known password storage security requirements. Submissions were marked based on whether they followed best practices such as hashing, salting and iterating the passwords. This pragmatic approach allowed the researchers to assess and compare the effects on password security of varying libraries and instructions. The Naiakshina lab at the University of Bonn used an instrumented Ubuntu Linux distribution with code-specific tracking features [9]. Every compiled snippet was stored to a history folder. The authors had access to web history, both video recordings and snapshots of the desktop, and time-stamped records of clipboard use. In the first of their series of papers, Naiakshina et al. discussed the value of using instrumentation to obtain data that was not strictly necessary for the study results [8]. They found that it increased the researchers' ability to interpret their findings.

The screen and voice recordings requested by Wijayarathna and Arachchilage were a rich source of data [17], enabling painstaking but complete reconstructions of the study participants' work. However, there were only 11 developers in the study. This approach was not scalable due to a limitation also encountered by Naiakshina et al. [8] when evaluating think aloud: it was very labour intensive.

The Developer Observatory researchers instrumented the Jupyter notebooks to record details of the coding process, thus replicating a laboratory environment. They stored the state of the code each time the '*Run*' button was clicked, and also stored

run results including stack traces. As well as task runs and completions, they took snapshots of the screen. Paste events were captured and it was possible to prompt the subject to enter the source URL from which code was copied. The Developer Observatory does not have instrumented browsing. It does not record the screen or do audio recordings, which precludes the use of think aloud unless recordings are arranged separately by the researcher. It does, however, allow researchers to configure the observatory by using JavaScript events as triggers. This makes it a flexible option for future studies [12].

A. Copy and Paste

Copy and paste behaviour is key to the understanding of some research questions. Acar et al. [13] did a detailed study on use of information sources during coding, and found that use of Stack Overflow (SO) resulted in the least secure code. Naiakshina et al.'s series of studies used a security metric to compare the security of code written by different treatment groups with different characteristics. Copy and paste behaviour was not relevant to the metric. However, by qualitative analysis of the coding process, including copy and paste, Naiakshina et al. were able to contribute to the ongoing debate about the impact of copy and paste on code security [9]. They demonstrated that while use of copy and paste did result in some insecure code, it was also an essential component in all the secure code they observed. None of their subjects produced secure code without using copy and paste. Copy and paste events had a statistically significant positive effect on code security. In a later study with freelancers which took place outside the lab environment, Naiakshina et al. [15] did not have the ability to monitor copy and paste events. However, when analysing the code received they identified 16 instances where code was obviously copied and pasted from the Internet. Since they could not say for certain what the situation for other instances was they could not analyse this further, but a perusal of Table 2 in their paper indicates that, again, copy and paste may have been beneficial for security. (This loss of vital data when working outside the lab environment, though inevitable for the freelancer study, shows the importance of instrumenting the study environment where possible.)

Naiakshina's finding that copy and paste is important for secure coding was interpreted by some as contradicting the findings of Acar et al. [7] on the subject of copy and paste. However, Acar et al.'s results were specific to copy and pasting *from SO*. In our larger literature review, papers that found that copying and pasting from SO can result in insecure code did not also assess whether it can result in secure code [26], [27]. Apart from some sympathetic analysis by Lopez et al. from the '*Motivating Jenny*' group [28], this has led to a perception that leveraging resources such as SO via copy and paste has a negative impact on developer security. Naiakshina et al. were able to show that this is not always the case. Their detailed qualitative analysis, to which we direct the reader [9], encourages a nuanced view of the value of copy and paste, and by extension of SO and associated web forums.

From what we have observed in our study, copy and paste records are used in two ways by researchers:

1) As an automatically observed event, instances of copy and paste during code generation can be easily included in quantitative statistical analysis [10].

2) During qualitative analysis, a record of copy and paste and all other events is essential to understanding [17].

An example of the first approach can be found in Acar et al.'s study of the security of code generated by developers using five different cryptographic APIs [10]. The study environment automatically logged copy and paste events. In their subsequent analysis the researchers used '*copy-paste*' to indicate whether a participant pasted code during a task. They found a statistically significant '*copy-paste*' correlation for functionally correct tasks but not for secure tasks. Only functionally correct tasks were analysed for security. Since the absence of a correlation would indicate that copy and paste was used for both secure and insecure code, their finding was in line with Naiakshina's finding that copy and paste is used not only in insecure but also in secure code. The second approach is exemplified by Wijayarathna and Arachchilage [17], who noted copy and paste events during their detailed analysis of the screen recordings for their 11 participants. Thus, for example, they observed participants triggering exceptions by copying code with preset (and therefore incorrect) passwords, and code with 'hostname' as the server address. Although labour intensive, their in-depth analysis provided deep understanding of the effect of copy and paste at a qualitative level.

Noting that developers are always likely to copy and paste when coding, Acar et al. [13] suggested multiple improvements to the coding environment, including improving the security of sites such as SO and creating tools and resources that proffer simple and correct copy and paste options. The latter approach was used by Geierhaas et al. in the Let's Hash tool [24]. The researchers understood that developers using the tool would tend to use copy and paste, and therefore made secure code snippets easily available via the tool.

B. Think Aloud

Asking participants to verbalise their thoughts during a study is a way to capture useful qualitative data. Tahaei and Vaniea discussed the risk that interrupting participants, asking them to think aloud, can affect study results [7]. In the studies we analysed it seemed that this risk was not realised. Developers were asked to think aloud at the beginning of the study, but did not appear to be interrupted mid-task.

Wijayarathna and Arachchilage's subjects were asked to think aloud while developing [17]. They were also asked to complete a tailored cognitive dimensions-based questionnaire at the end of the study. In a comparison of the results from think aloud and the questionnaire, the authors felt that they got more useful results from the questionnaire. However, clarity and richness was added to these results by the think aloud video in which participants discussed their thinking as they undertook tasks. It took two to three times the duration of each recording to analyse it, but with only 11 participants and an average

time to task completion of about an hour, it was practical to review the audio and video. In their analysis of Developer Observatory, Stranksy et al. noted that they had not been able to ask participants to think aloud [12]. The Wijayarathna and Arachchilage approach shows that this can be done in online studies by asking participants for screen and voice recordings, at the cost of adding complexity to participant instructions.

Wijayarathna and Arachchilage found that participants continued to think aloud throughout the programming tasks. By contrast, Naiakshina et al. [8] found in the pilot phase of their study that when asked to think aloud their participants tended not to talk much. When they did talk it was usually about routine activities such as search. As a result of this observation, Naiakshina et al. removed the think aloud element from their study. The low level of think aloud they observed may reflect a difference in task type, the relatively lengthy duration of up to eight hours, or the unfamiliarity of the lab environment.

Studying ten developers' use of a static analysis tool in the lab, Smith et al. used think aloud with screen recordings [16]. Although think aloud worked well in the other three categories, they noted that developers' reflections on their own understanding and relationship to the task were sparsely reflected in the think aloud recordings. They posited that these reflections may take place internally.

Because think aloud is very labour-intensive, it would normally be impractical in a larger study. Luo et al., assessing the viability of introducing IDE support for cloud-based static analysis, had a useful solution to this [21]. They randomly selected three participants from each of their four conditions and asked them to record their screens and think aloud. This approach is a good way of obtaining a manageable amount of qualitative data to add richness and context to study results.

C. Screen Recordings

Screen recordings can be used for detailed qualitative analysis, as is illustrated above by their use in both think aloud and copy paste analysis. Analysis of screen recordings is slow, taking 2-3 times the duration of the recordings [29].

If the laboratory environment is on site, screen recordings should be taken. They allow for the elimination of any doubt as to how a solution was achieved. However, if a study takes place online, the overhead of screen recordings is higher. The researcher must ask the participant to record the screen. The participant must be able to record it and remember to do so, and must also send the recording to the researcher. These extra activities require a high level of commitment compared to use, for example, of Developer Observatory where instrumentation happens behind the scenes. There are privacy implications and potential technical issues. Subjects may be resistant to recording their screen, or may find it difficult to do, and these issues may limit the number of participants in a study. Therefore, the researcher should give careful consideration to the need for screen recordings in online studies. An instrumented online laboratory environment such as Developer Observatory may provide sufficient detail without them, depending on the study subject.

D. Deception

Is deception necessary in secure development studies? The series of Naiakshina papers showed that it is. In these studies the deception element involved telling subjects during recruitment that the study concerned API use, without mentioning security. Half of the subjects were not asked to code securely when the study began either, and most of these subjects did not make any attempt to secure passwords. The other half were asked to code securely when they began the study. Most of this group did try to store passwords securely, and often succeeded.

Many of the students in the first two studies by Naiakshina et al. [8], [9] stated that they would have coded passwords securely '*in real life*', but did not because the code was only written for a study and would not be used in a real environment. Naiakshina et al. tested this statement in a third study undertaken with freelancers [15]. The researchers exercised an extra level of deception, posing as a real company and asking the freelancers to help finish the authentication code for their website. The results were similar to those of the previous two studies; the freelancers did not tend to implement secure solutions unless prompted. In a later replication study, Danilova et al. established that the added element of deception, hiding the fact that the task was for an academic study, was unnecessary, at least in this case [30]. In the replication study freelancers were told that the work was for an academic project. This study obtained similar effect sizes and other results to Naiakshina et al. [15].

Braz et al. studied the effect of security priming on detection of vulnerabilities during code review. They exercised deception during recruitment, not stating that security was the focus of the study [23]. Security was not mentioned at any time to their control group. Their findings echoed Naiakshina et al.'s; participants asked to focus on security were eight times more likely to detect vulnerabilities than the control group.

There are ethical implications to deceiving study participants. It should only be done when absolutely necessary [9]. The results from Naiakshina et al. and Braz et al. show that deception is genuinely important for obtaining useful results in secure development studies. Care should be taken to ensure that study participants are not negatively affected by the deception.

V. Who: Recruitment Choices

A. Using Students for Developer Studies

It is difficult to recruit developers for laboratory studies. Professional developers are normally well paid, making it hard to provide commensurate payment. There may be few suitable candidates in the study locale [17]. Academics are frequently forced to rely either on students or on developers with goodwill, who are self-selecting. Industry researchers, for example at Microsoft Research, can invite colleagues more easily.

CS students are the subjects of many laboratory studies, but whether they are an appropriate proxy for professional developers remains a topic of debate [31]. An online study with GitHub users found no difference in the functionality or security of code based on participants' self-reported status as students or professional developers; however, the authors noted

that few participants identified as being students exclusively [11]. Tahaei and Vaniea concluded in 2019 that the literature was not yet decisive on this question [7].

Intuitively, there are two main reasons why CS students may not share important characteristics of professional developers. One is education level. Only 62% of professional software developers in SO's 2022 survey stated that they learned to code at a school, university, or college [32]. The other is that students usually lack real-world coding experience.

Acar et al. recruited both students and professional developers to study the impact of information sources during development. They found that while developers were slightly more likely to produce a functional solution, developer and student groups produced equally (in)secure code [13].

This finding was echoed by Naiakshina et al. [15] and Danilova et al. [30], who found that non-primed freelancers were almost as bad as non-primed students [8] in their level of secure coding, which was very low. A study with professional developers recruited in Germany [33] showed more secure development but a similar response to treatment effects, with security priming having a large effect on both groups. Naiakshina et al. concluded that although their code is less secure in absolute terms, students can reasonably be used instead of professional developers for studies comparing treatment effects. Gorski et al.'s use of a physical laboratory to compare different ways of embedding security information in documentation for non-security APIs [20], using students from their own university, is such a study.

Apart from cost and availability, there can be other specific reasons to use students. Smith et al., looking at developers' strategies while using a static analysis tool, wished to ensure that their study participants were already familiar with their university's iTrust software [16]. They added vulnerabilities to iTrust, and recruited only students or ex-students who were familiar with the iTrust codebase.

B. Online Studies

As discussed, on-site laboratory studies are limited by geography and the proximity of competent and willing study subjects. Finding enough participants to reach the required statistical power is difficult.

Recruitment can also be hard for online studies. Luo et al. [21] and Sahin et al. [25] ran studies within organisations, recruiting internally. Four online studies in our review recruited by extracting email addresses from GitHub [10], [14], [17], [19]. Scraping GitHub facilitates the targeting of developers who use a specific language or security feature, but GitHub users receive many academic recruitment emails and may consider them spam [34]. Most authors of our online studies also used extensive developer outreach; contacting personal networks, messaging online forums and posting on social media. This is a slow process requiring considerable imagination and hard work, and often results in disappointingly low numbers [35].

Two of our review papers recruited via paid online services. Using payment entails a well-documented risk that the recruits will be profit-driven and may not be genuine developers [36]. Studies requiring participants to write code [24] or pseudo-code [22] should be able to detect unqualified applicants. Danilova et al. suggested and tested coding questions that can be used for filtering purposes, especially useful for paid surveys where imposters are difficult to detect [36].

VI. Conclusion

The 16 papers in our review illustrate a range of approaches to running security lab studies with software developers. We discussed reasons to run labs on site or online, approaches to instrumentation and recruitment, and potential for reuse. Based on our analysis, we have some advice for researchers.

Why and What? These are interlinked. Consider the motivation of your study. Have you thought through your design carefully? Would it improve the study to add an additional treatment? How would that affect the study environment?

Where? The research may be concerned with a specific piece of hardware. This is quite likely when dealing with critical infrastructure. In this case it is probably best to use an on-site laboratory. Additionally, it is difficult to assess details like eye movement unless the research is on site. If you do not have these constraints, conducting the study online will improve the reach and allow you to recruit more participants.

How? Do you need to understand how developers think or feel about a tool? Use think aloud. If you have a large number of participants, use think aloud for a random subset of them. Are you interested in how developers complete a piece of code? Monitor browser history and copy-and-paste.

Who? If the research is designed to test the effectiveness of a new technique or tool, you should not recruit students unless you have a control group. However, if the research measures relative treatment effects, students can be used. If participants are paid they should be screened to ensure that they have development experience.

What Else? Laboratories are necessarily contrived settings, and cannot emulate realistic software development environments [37]. However, they allow the researcher to isolate and study specific easily-controlled phenomena. In our reading we have frequently observed the assertion that time constraints contribute to the neglect of software security by developers. Proof that this is true can be glimpsed in the side-effects of pilots by Naiakshina [8] and Huaman [6]. We would like to see the security effect of restricting available time isolated, and studied in a controlled environment. We are also keen to see further clarification on the security effects of copy and paste. Replication studies are needed in many areas; for example, the use of coding resources [13]. We echo Tahaei and Vaniea's [7] surprise at the paucity of studies comparing the security effects of different programming languages.

Conclusion Considerable research has been done on constructing secure software development studies. New researchers do not need to reinvent the wheel. If working on site, the instrumented version of Ubuntu used by the Naiakshina Lab may be available on application to the research team. If working online, consider using the Developer Observatory or the new OLab tool. We recommend reading the papers in our review.

Identify researchers in your area of interest, and reach out to them for advice.

Acknowledgment

We thank the anonymous reviewers for their feedback. This publication was financially supported by Science Foundation Ireland under Grant numbers 18/CRT/6222, 13/RC/2077_P2, 13/RC/2094_P2, and 15/SIRG/3293. For the purpose of Open Access, the authors have applied a CC-BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

References

[1] I. Ryan, U. Roedig, and K.-J. Stol, "Insecure software on a fragmenting Internet," in *2022 Cyber Research Conference - Ireland (Cyber-RCI)*, 2022, pp. 1–9.

[2] Mitre, "CVE Program Mission ," https://www.cve.org/, 2022, [Online; accessed 21 December 2022].

[3] J. Gamblin, "2022 CVE Data Review ," https://jerrygamblin.com/2023/01/01/2022-cve-data-review/, 2023, [Online; accessed 9 January 2023].

[4] D. Jones, "High risk, critical vulnerabilities found in 25% of all software applications and systems," https://www.cybersecuritydive.com/news/high-risk-critical-vulnerabilities-software/636592/, 2022, [Online; accessed 19 January 2023].

[5] L. Vaas, "BillQuick Billing App Rigged to Inflict Ransomware," https://vulners.com/threatpost/THREATPOST:94E54481AD472743701D499DC7677008, [Online; accessed 21 March 2023].

[6] N. Huaman, A. Krause, D. Wermke, J. H. Klemmer, C. Stransky, Y. Acar, and S. Fahl, "If you Can't get them to the lab: Evaluating a virtual study environment with security information workers," 2022, p. 313–330.

[7] M. Tahaei and K. Vaniea, "A survey on developer-centred security," in *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2019, pp. 129–138.

[8] 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*. ACM, 2017, p. 311–328.

[9] 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*. USENIX Association, 2018, pp. 297–313.

[10] 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)*, May 2017, p. 154–171.

[11] Y. Acar, C. Stransky, D. Wermke, M. L. Mazurek, and S. Fahl, "Security developer studies with GitHub users: Exploring a convenience sample," in *Thirteenth Symposium on Usable Privacy and Security*, 2017, p. 81–95.

[12] C. Stransky, Y. Acar, D. C. Nguyen, D. Wermke, E. M. Redmiles, D. Kim, M. Backes, S. Garfinkel, M. L. Mazurek, and S. Fahl, "Lessons learned from using an online platform to conduct large-scale, online controlled security experiments with software developers," in *CSET '17 (USENIX Workshop on Cyber Security Experimentation and Test)*, 2017.

[13] 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)*. IEEE, May 2016, p. 289–305.

[14] P. L. Gorski, L. L. Iacono, D. Wermke, C. Stransky, S. Moeller, Y. Acar, and S. Fahl, "Developers deserve security warnings, too: On the effect of integrated security advice on cryptographic API misuse," in *Fourteenth Symposium on Usable Privacy and Security*, 2018, p. 265–281.

[15] 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," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, 2019.

[16] 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 Trans Softw Engineer*, vol. 45, no. 9, p. 877–897, 2019.

[17] C. Wijayarathna and N. A. G. Arachchilage, "Why Johnny can't develop a secure application? A usability analysis of Java Secure Socket Extension API," *Computers and Security*, vol. 80, p. 54–73, 2019.

[18] L. Braz, E. Fregnan, G. Çalikli, and A. Bacchelli, "Why don't developers detect improper input validation?': DROP TABLE Papers," in *Proceedings of the 43rd International Conference on Software Engineering*. Madrid, Spain: IEEE Press, 2021, p. 499–511.

[19] F. Fischer, Y. Stachelscheid, and J. Grossklags, "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, 2021, p. 3070–3084.

[20] P. L. Gorski, S. Moller, S. Wiefling, and L. Lo Iacono, ""I just looked for the solution!" - On integrating security-relevant information in non-security API documentation to support secure coding practices," *IEEE Transactions on Software Engineering*, 2021.

[21] L. Luo, M. Schäf, D. Sanchez, and E. Bodden, "IDE support for cloud-based static analyses," in *Proc. 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2021, p. 1178–1189.

[22] 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 *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, p. 486–498.

[23] L. Braz, C. Aeberhard, G. Çalikli, and A. Bacchelli, "Less is more: Supporting developers in vulnerability detection during code review," in *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. ACM, 2022, p. 1317–1329.

[24] L. Geierhaas, A.-M. Ortloff, M. Smith, and A. Naiakshina, "Let's hash: Helping developers with password security," in *Proceedings of the Eighteenth Symposium on Usable Privacy and Security*, 2022, p. 503–522.

[25] M. Sahin, T. Ünlü, C. Hébert, L. A. Shepherd, N. Coull, and C. M. Lean, "Measuring developers' web security awareness from attack and defense perspectives," in *IEEE Security and Privacy Workshops*, 2022, p. 31–43.

[26] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl, "Stack Overflow considered harmful? The impact of copy paste on Android application security," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, p. 121–136.

[27] W. Bai, O. Akgul, and M. L. Mazurek, "A qualitative investigation of insecure code propagation from online forums," in *2019 IEEE Cybersecurity Development (SecDev)*. IEEE, 2019, p. 34–48.

[28] T. Lopez, T. Tun, A. Bandara, M. Levine, B. Nuseibeh, and H. Sharp, "An anatomy of security conversations in Stack Overflow," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Society*. IEEE Press, 2019, p. 31–40.

[29] C. Wijayarathna, N. A. G. Arachchilage, and J. Slay, *A Generic Cognitive Dimensions Questionnaire to Evaluate the Usability of Security APIs*. Cham: Springer International Publishing, 2017, vol. 10292, p. 160–173.

[30] A. Danilova, A. Naiakshina, J. Deuter, and M. Smith, "Replication: On the ecological validity of online security developer studies: Exploring deception in a password-storage study with freelancers," 2020.

[31] D. Falessi, N. Juristo, C. Wohlin, B. Turhan, J. Münch, A. Jedlitschka, and M. Oivo, "Empirical software engineering experts on the use of students and professionals in experiments," *Empirical Software Engineering*, vol. 23, no. 1, pp. 452–489, 2018.

[32] SO, "Stack Overflow Developer Survey - Education," https://survey.stackoverflow.com/2022/#developer-profile-education, 2022, [Online; accessed 21 December 2022].

[33] A. Naiakshina, A. Danilova, E. Gerlitz, and M. Smith, "On conducting security developer studies with CS students: Examining a password-storage study with CS students, freelancers, and company developers," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, 2020, p. 1–13.

[34] S. Baltes and S. Diehl, "Worse than spam: Issues in sampling software developers," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. Association for Computing Machinery, 2016.

[35] N. Patnaik, J. Hallett, M. Tahaei, and A. Rashid, "If you build it, will they come? Developer recruitment for security studies," *ROPES - ICSE 202*, 2022.

[36] A. Danilova, A. Naiakshina, S. Horstmann, and M. Smith, "Do you really code? Designing and evaluating screening questions for online surveys with programmers," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, p. 537–548.

[37] K.-J. Stol and B. Fitzgerald, "The ABC of software engineering research," *ACM Transactions on Software Engineering and Methodology*, vol. 27, no. 3, 2018.