The State of Secure Coding Practice: Small Organisations and "Lone, Rogue Coders"

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 a rapidly developing problem. Malware, ransomware and spyware routinely leverage vulnerabilities in software to gain access to systems, escalate privileges and run adversarial code. One approach to solving this issue is to use secure software methods, which attempt to guide organisations in improving their software assurance. However, these methods implicitly assume the presence of substantial resources deployed in a compliance-mandated environment. The distinct and often limited environment in which small organisations, independent teams and lone coders operate is not considered. Advice for software security in small teams is almost absent from the literature, as is a way to measure the levels of secure coding in such teams. In order to address this problem, we must begin by understanding it. As part of the analysis of a large survey on current software security practice, we examined the current software security practices of small and open source organisations, and of lone and non-company developers. We present our results in this paper. We hope that they will facilitate the targeting of security advice to these neglected developer categories.**

***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, measuring security**

I. Introduction

The world's businesses, economies, governments and critical systems are moving online at ever-increasing pace. This global digital transformation comes at a cost, with nations, businesses and individuals more vulnerable than ever before to cybercrime threats such as ransomware, and to cyber-espionage [1].

An understanding of the issues that facilitate online interference is essential. At the root of many cyber attacks are code vulnerabilities. At the root of code vulnerabilities is software code that, due to poor planning or faulty execution, contains errors that allow malicious exploitation. The question arises, why do developers not code securely? The field known as Developer Centred Security (DCS) focuses on reasons why software developers fail to secure their code. Well-explored issues include the poor usability of security tools [2] and APIs [3], inadequate documentation [4] and resources [5], conflicting advice [6], and a lack of knowledge [7], awareness [8], understanding [9] or motivation [10].

There are also well-known constraints in the coding environment such as lack of time [11], security not being a priority [12], and over-speedy deployment [13]. Other issues outside the developer's sphere of influence include lack of clarity on who is responsible for security [14], budget pressure [15], and questionable management decisions, such as choosing to release insecure code for business reasons [10].

Secure software methods in industry attempt to address some of these issues, introducing practices and activities to be undertaken throughout the development lifecycle [16], [17]. Cybersecurity guidelines for critical systems include such practices, although the sections on secure software development can be rather short [1]. Such methods and guidelines tend to implicitly assume the presence of substantial resources deployed in a compliance-mandated environment. But what about software which is not developed in this type of environment? Much of modern software has dependencies on third-party components. These may be created by lone developers, or by small organisations or open source software (OSS) teams having few resources, few developers and no budget. Small, under-resourced teams can have a disproportionate influence on global software security. An obvious recent example is Log4Shell (CVE-2021-44228), a vulnerability in popular open-source Java logging tool Log4j. Disclosed in late 2021, this popular defect was widely weaponised by threat actors. By late 2022 it was reportedly being used by North Korea to obtain initial entry to US energy company networks [18].

The distinct and often limited environment in which isolated developers, open source developers, freelancers and small organisations operate is not clearly understood. It has not been much studied in secure software development literature, and to date there has been little attempt made to separate out secure software development guidance for such teams. This contrasts with the overall area of cybersecurity, where in the UK, for example, the 'Cyber Essentials' program [19] attempts to provide small organisations with the information and guidelines they need to achieve a base level of security.

The existence, importance and unique software development needs of small coder groups have been acknowledged by the creation of the ISO/IEC 29110 Series of '*Systems and*

TABLE I
QUESTIONS ON TWELVE COMMON ACTIVITIES (CAS)

No.	Usable	Question
CAQ1	Yes	Static analysis tools are brought into the code review process to make the review more efficient and consistent.
CAQ2	No	Compliance constraints are translated into software requirements for individual projects and are communicated to the engineering teams.
CAQ3	Yes	QA efforts go beyond functional testing to perform basic adversarial tests and probe simple edge cases and boundary conditions, with no particular attacker skills required.
CAQ4	Yes	QA targets declarative security mechanisms with tests derived from requirements and security features. A test could try to access administrative functionality as an unprivileged user, for example, or verify that a user account becomes locked after some number of failed authentication attempts.
CAQ5	Yes	Penetration test tools are used internally.
CAQ6	Yes	Emergency codebase response can be done. The organisation or team can make quick code and configuration changes when software (e.g., application, API, microservice, infrastructure) is under attack.
CAQ7	Yes	Defects found in operations are entered into established defect management systems and tracked through the fix process.
CAQ8	Yes	Bugs found in operations monitoring are fed back to development, and may change developer behaviour. For example, viewing production logs may reveal a need for increased logging.
CAQ9	Yes	Penetration testing results are fed back to engineering through established defect management or mitigation channels, with development and operations responding via a defect management and release process.
CAQ10	Yes	Host and network security basics are in place across any data centers and networks and remain in place during new releases.
CAQ11	No	Security-aware reviewers identify the security features in an application and its deployment configuration (authentication, access control, use of cryptography, etc.), and then inspect the design and runtime parameters for problems that would cause these features to fail at their purpose or otherwise prove insufficient.
CAQ12	No	External penetration testers are used to identify security problems.

Software Engineering Standards and Guides for Very Small Entities' [20]. However, these guides do not currently have a secure development component, though the addition of security improvements has been proposed [21]. Advice for software security in small teams is largely absent from the literature. An academic study in 2022 was unable to find any formal secure development guidelines for OSS teams or small organisations [22].

In addition, there is currently no established way to measure the levels of secure coding in such environments. While measurement of secure coding by individuals is well understood [23], researchers have not yet converged on a metric to measure secure coding for organisations and teams. In previous work [24], we proposed a potential lightweight metric derived by evaluating organisations' use of the twelve security activities most commonly used in industry, as determined in a study by Weir et al. [25]. We called these the '*common activities*' (CAs). Questions to evaluate the CAs can be seen in Table I. Weir et al.'s work was based on data accumulated by the BSIMM team working with large or very large organisations. Implementing the CAs ideally requires the deployment of large budgets and the existence of multiple teams within an organisation. Thus, it seems unlikely that the Ryan metric would be suitable for small organisations, lone developers or OSS teams.

This intuition has not yet been verified using data. As part of the analysis of a large survey on current software security practice, we examined use of the CAs by small and medium organisations, lone developers, and developers outside organisations. In this paper we present a summary of use of the CAs and other security measures by these developer categories.

Our contribution is that we provide information about the current state of software security practice in small and medium-sized organisations, amongst coders outside organisations such as open source developers and freelancers, and amongst developers who code alone within organisations.

This is a first attempt to assess the levels of secure coding practice for these cohorts. It will provide much-needed information to the software security community. The paper is organised as follows: Section II discusses related work, section III is on method, section IV provides the results, section V discusses the results and section VI concludes the work.

II. RELATED WORK

A. State of Practice

An investigation of app developers' rationale by van der Linden et al. [26] looked at how app developers deal with choices that are not overtly security-related, such as choice of advertising library, finding that app developers need support to make the right choices. Wermke et al. [27] examined security and trust in OSS projects by interviewing 27 open source contributors on their practices. They found that there were few engagements with security challenges. Guidance and policies were ad-hoc except for large projects. Most of the contributors thought of improvements in terms of bug bounties, code-review hours, code upgrades and in general more person hours. The authors recommended that security help for OSS projects take project size and resources into account.

B. Secure Coding Guidelines

It seems that the help and support that these researchers advocate is currently lacking. Research has found that there is a shortage of secure coding guidance online, with significant gaps in coverage in several areas such as program analysis tools and logging [28]. Where guidelines do exist, developers are not always aware of them [29]. This may be somewhat mitigated in large organisations by the availability of secure coding standards such as BSIMM [16] or SAMM [17], but

there is a marked shortage of secure coding guidelines designed explicitly for small organisations and teams.

In a recent literature review, Bender et al. looked at usability, security and privacy development processes for app developers, small and medium companies, and the open source community [22]. They found 15 papers across the range of three development types, but all dealt with usability. They found nothing related to security or privacy.

Although not tailored to small teams and isolated developers, work on security interventions by Weir et al. is of interest because they explicitly considered situations where budgetary and resource support could be low [30]. Their interventions approach should be of benefit in introducing security practices to small organisations and teams. They set out to find the best way to encourage developers to use secure coding practices. From detailed consultation with industry experts, they identified eight interventions of interest. Of these, they considered three to be sufficiently lightweight to introduce in their relatively brief intervention sessions; '*incentivization session*,' '*threat assessment*,' and '*on-the-job training*.' During these sessions they hoped to promote the other five interventions at opportune moments. It is interesting that they did not include automated security analysis as one of their three primary interventions [30]. However, they did identify automated static analysis as an intervention to be encouraged when opportunity arose.

C. Secure Coding Assessment

Assessing individuals' level of secure coding is well understood, and a level of researcher convergence has emerged around the Naiakshina criteria [6], [23], [31]. There have been multiple attempts to define measurement tools for secure coding at a group or organisational level. Since these are frequently derived from industry guidelines such as BSIMM and SAMM, they often include practices that are not available to or practical for small teams [32]. Assal et al. developed a useful approach; in an interview study, they assessed whether participants used techniques to ensure security in each of six software development lifecycle steps from '*design*' to '*post-dev testing*' [15]. The qualitative analysis was work-intensive, requiring the researchers to understand the development process and security steps and to be able to correctly interpret developer responses. The approach would not easily translate to analysis of survey answers at scale, although it was a useful and flexible way to compare teams' approach to secure coding.

We did not find any studies dealing specifically with the assessment of secure coding for small organisations or teams.

III. Method

We undertook a large-scale survey of current security practice for software developers. We publicised the survey to our personal contacts and via social media and developer groups, receiving 1,100 responses over three months, of which 962 were valid responses. Ryan et al. [24] describes the composition of the survey questions, pilot tests, ethics approval and recruitment processes in detail. It also gives demographic data for the survey participants. The focus of Ryan et al. was to devise and use a metric for software security for organisations, to attempt to assess software security culture, and to compare the results of the two measures to discover whether a high level of software security practice necessarily entails a good security culture.

Ryan et al. describes in detail the rationale for using the empirically-determined 12 CAs as a lightweight metric to measure software security in coding environments. In this paper we revisit the survey data, this time focusing on developers who code in small and medium organisations, non-organisational developers, freelancers and developers who code alone. We evaluate the CAs in light of their relevance to small and impoverished teams. We posit that nine of the CAs may be accessible. We have indicated these with a 'Yes' in the 'Usable' column in Table I. We examine survey responses and use the results to assess the truth of our conjecture on which of the common security activities may be suitable for isolated developers and resource-constrained developers.

This paper is a first exploration of the secure development work environment of five developer cohorts that have not previously been investigated. The questions asked were derived from data sourced in much larger organisations. Therefore we have presented the data simply. The reader can easily compare the rates of adherence to the 12 most common security activities between the five developer categories. The corresponding adherence for all valid respondents is displayed for information reasons.

In addition, we briefly examine use of security tools by the developer categories of interest. We theorise that one approach to accessing secure coding that is available to isolated developers is leveraging automation of secure coding aids developed elsewhere. Such automation can be applied, for example, to tools for static analysis, code review, configuration, dynamic scanning, monitoring, licensing and testing. There is overhead to setting up such tools, but they provide an analytic consistency not otherwise achievable on a small team.

A. Data

Where participants are quoted in the text, the quotes appear exactly as entered into the survey. Data analysis was done in R. Data was plotted using the viridis colourblind-friendly 'plasma' colour map [1].

IV. Results

As discussed, the CAs are derived from data on large organisations. They are likely to be less useful for secure coding measurement in small teams or open source environments. The resources available in such environments are more constrained. Asked for feedback on the twelve CAs, one survey respondent remarked:

> '*(Some of these don't fit that well for the open source project that I work on)*'.

While it is easy to assume that lone coders will take fewer security measures, this assumption may not make sense to the coders themselves. For example, one of the respondents said:

[1] https://ggplot2.tidyverse.org/reference/scale_viridis.html

TABLE II
CATEGORIES OF DEVELOPERS CONSIDERED IN THIS STUDY

Name	N	Description
Lone Coders in Organisations	64	Codes alone rather than on a team. Works within an organisation with size greater than one.
Coders in small organisations	234	Codes on a team within a small organisation (less than 50 staff members).
Coders in medium-sized organisations	129	Codes on a team within a medium-sized organisation (between 50 and 199 staff members).
Coders not working for organisations	67	Codes in a non-organisational environment, either alone or on a team.
Freelancers	23	Codes in a single-developer organisation, states that they code alone.
All Developers	962	All valid survey participants. Includes developers in large organisations.

'*I suspect, even as a lone, rogue coder, that I generally write more secure code than a lot of companies.*'

In Table I we made an initial assessment of which CAs we thought it would be difficult for coders working alone or in small teams to undertake. However, this assessment was based on the theoretical accessibility of activities, rather than on data. In our analysis we look at the data for several categories of coder, briefly described in Table II. They are as follows:

Lone Coders in Organisations: When asked whether they code alone or in a team, these respondents stated that they code alone. There are 141 valid respondents who stated that they code alone, but in this category we consider only the 64 of them who stated that they work in an organisation with a size greater than one. See Fig. 1. We wanted to see what secure coding practices these isolated coders, working inside organisations, are using. There are certainly some in this cohort who do not feel supported. One participant stated that he coded alone in an organisation with between 10 and 50 developers and a staff of more than 10,000. Asked for comments at the end of the survey, he had this to say:

'*TLDR - we got pwn't, we learned nothing, back to old boy's club and the mentality security doesn't contribute to bottom line of revenue so we get Foxtrot-All for budgets/tools/time/attention/consideration. But of course it's our fault when inevitably kaboom happens.*'

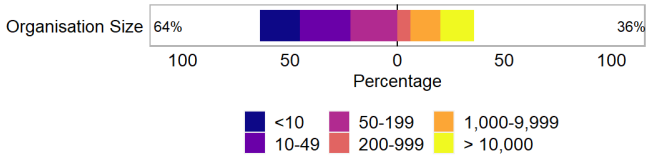

Fig. 1. **Organisation size for the 64 respondents who stated that they code alone within organisations**. This cohort is of interest because they may lack the support available to those working on teams.

Coders in Small Organisations These respondents, when asked whether they code alone or in a team, said that they coded within a team. Asked to estimate the size of the organisation they work in, they specified a value of up to 49 employees, which we consider a small organisation [33].

Coders in Medium-Sized Organisations These respondents stated that they worked on a team in an organisation with between 50 and 199 employees.

Coders not Working for Organisations Some developers chose 'Not applicable' for both organisation size and number of developers in the organisation. These developers may be open source developers, students or independent developers. Although they do not have organisational support, and some code alone, many may have support from teams and communities.

Freelancers These developers chose an organisation size of 1 and stated that they worked alone. We deduce that they work as freelancers. They may be the most isolated of all the categories, with no organisational, institutional or open source support.

All Developers Finally, in order to give a point of comparison for the data, we included the CA percentages for the category comprising all 962 valid survey participants.

A. *Usage of CAs by Developer Category*

In this section we examine the usage of the twelve CAs by each of the five categories of coder. The CAs are collected into five groups, each containing two or three related activities. These are now discussed.

1) Penetration Testing: When considering the twelve CAs and their accessibility to under-resourced coders, we thought that internal penetration testing could be a practical option, since it does not necessarily require additional staff or resources and can theoretically be conducted by a coder on their own code. We were interested to see that actually, there was more external than internal penetration testing for the developer categories we examined. None of the small number of freelance coders in the study undertook any internal penetration testing at all. See Fig. 2.

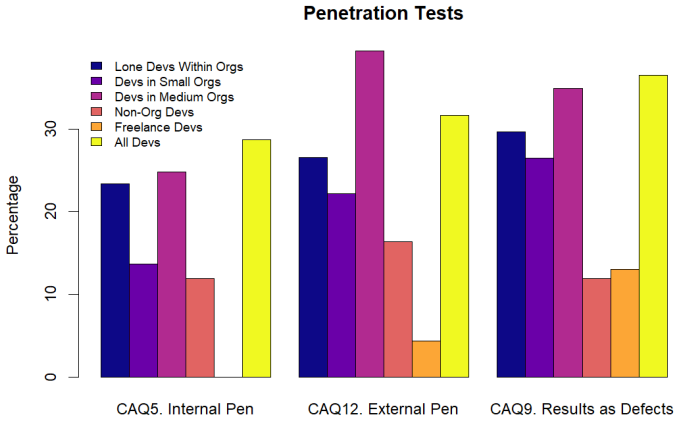

Fig. 2. **Results for the three penetration testing CAs for all five developer categories and the whole sample.** None of the freelance coders who responded to the survey did any internal penetration testing.

2) Review: Two of the CAs deal with review. The first is incorporating static analysis into code review. This is the most commonly used activity from the BSIMM data [25]. The second is having security aware reviewers inspect the features of the software and determine if there are any possible issues in the deployment configuration or design that would cause security problems. While both would seem to be natural undertakings for a security-aware environment, the first assumes that code reviews are taking place. For isolated developers who code alone within organisations, code reviews will necessarily be problematic. The second activity, having security-aware reviewers look over the code and deployment, could be difficult to provide in any resource-constrained environment. The take-up of these activities can be seen in Fig. 3.

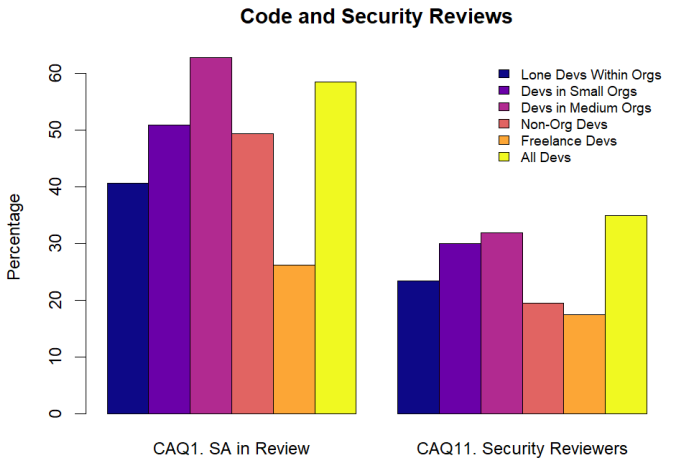

Fig. 3. **Results for the two review CAs for all five developer categories and the whole sample.** Freelancers are least likely to use review activities, while developers in medium-sized organisations use them the most.

There are nuances to all of these activities, as illustrated by one of our respondents who commented:

> '*Security-aware reviewers will review code and/or designs when there is an obvious security angle in new work, but there is no formal review process for the security content of changes that aren't obviously related to security.*'

This speaks to the shortage of resources in most environments, as all code changes should ideally be reviewed for security. Defects can be chained by attackers to provide access in ways that are not immediately obvious to the average developer.

Among the categories of developer we consider, freelancers have least access to both review activities. Developers in medium-sized organisations use both review activities the most. Access to security reviewers is higher in '*all developers*' than in any of the five categories.

3) Quality Assurance: The presence of a Quality Assurance (QA) team does not always ensure that security testing will take place. Two of the CAs specifically focus on this. Number three advocates that the QA team should conduct adversarial testing, checking edge and boundary cases. Number four suggests that the QA team attempt to attack the security procedures that are in place. For example, they could attempt to access systems when not logged in, or access admin systems when not logged in as an administrator. No hacking skills are required to conduct either set of tests. The incidence of these quality assurance activities within the five developer categories we consider, and the whole sample, can be found in Fig. 4.

Non-organisational developers are least likely to engage in these activities. It can be argued that they rarely have access to QA teams. Developers usually do some basic probing on their own account, however, and could conduct these quality assurance activities themselves.

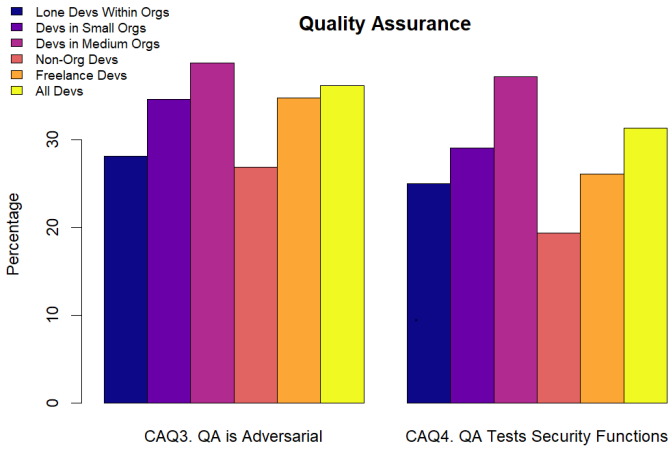

Fig. 4. **Results for the two quality assurance CAs for all five developer categories and the whole sample.** Non-organisational developers are least likely to engage in these activities. Developers in medium-sized organisations are most likely to engage in them, beating the '*all developers*' sample in both cases.

4) Operations Feedback: Two of the CAs deal with feedback from the operations team. CAQ7 mandates that the operations team should add defects found to the standard defect-tracking system, while CAQ8 concerns giving operations feedback to developers. Since we are dealing with lone developers and those who work in small organisations we would expect that there would be a lot of informal as well as formal feedback. In some cases the coders and the operations team could be one and the same person. See Fig. 5 for the results for these questions. As expected, there is a high occurrence of both of these activities in the developer categories we consider. It is interesting that the incidence is low in the non-organisational cohort. This may indicate that there is no formal defect tracking process, or no operations environment, or it may be that these developers simply do not undertake these activities. Predictably, the incidence is highest in medium-sized organisations.

5) Security Processes: Finally, we look at three questions that are concerned with security process. CAQ2 suggests that compliance constraints should be conveyed to software engineering teams as requirements. CAQ10 states that network security basics should be kept in place while new code is being deployed. A team must be able to respond to an attack with swift code and configuration changes to satisfy CAQ6.

We can observe from Fig. 6 that there is little conversion of compliance constraints to requirements outside the organisa-

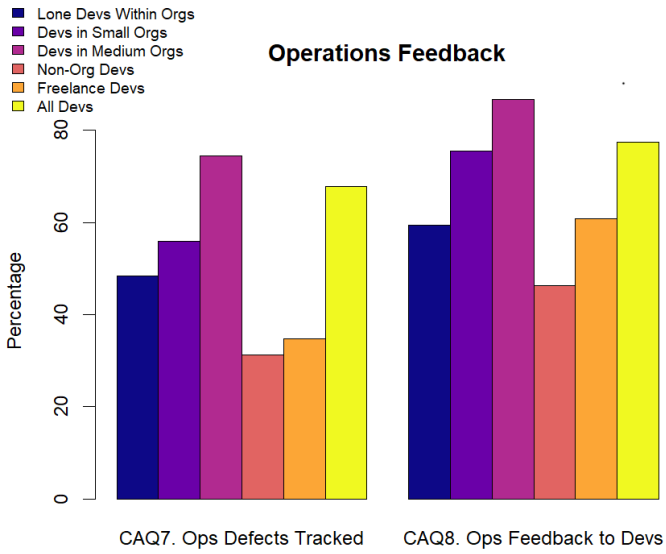

Fig. 5. **Results for the two operations CAs for all five developer categories and the whole sample.** Non-organisational coders are least likely to engage in these activities. Developers in medium-sized organisations are most likely to engage in them, outdoing the '*all developers*' sample.

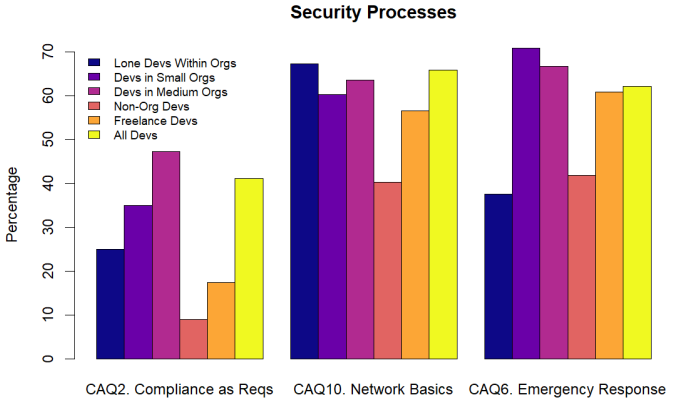

Fig. 6. **Results for the three security processes CAs for all five developer categories and the whole sample.** Freelance and non-organisational developers engage least with compliance. All categories fare well for network basics, particularly lone developers within organisations. Developers in small organisations are most likely to express confidence in their emergency codebase response.

tional environment. The non-organisational cohort lags behind the other cohorts on almost every measure, but it is particularly far back on this one. Freelance compliance is also low. This suggests that the type of work that is done outside organisations is not typically subject to compliance. With the likelihood of an increased compliance and legislative overhead for all developers in the short to medium-term future [34], [35], this may spell trouble ahead for open source and other non-organisational coders.

Lone developers working within organisations, who lag behind on other questions, shine when it comes to ensuring host and network security basics are in place. This may indicate a degree of self-reliance around their deployment environment which would be a natural consequence of working without a team of peers. Lone developers are likely to work on smaller projects where they are involved in deployment, support and all other aspects of the system.

The ability to be able to respond quickly to attack is the only area in which the non-organisational cohort does not lag behind the others. It is slightly ahead of the lone developers on this metric, which may indicate that lone developers inside organisations lack the autonomy to respond quickly to external attack, requiring authorisation to change production code. Coders such as open source teams or app developers are likely to be able to ship without such constraints, though the difference is small. This question gave developers from small organisations their only chance to shine, topping the numbers at 70.9% adherence. Developers within small organisations may be nimble, with short command chains, and able to get needed authorisations quickly. Small organisations have fewer than 50 employees. Their software may be fundamental to the organisation's well-being and could even be its only product.

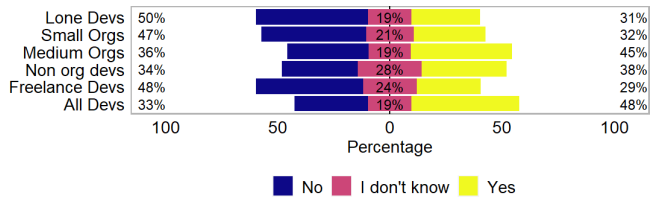

Fig. 7. **Are there security tools available in your environment?** Answers from all six developer categories. We can see that the percentage '*Yes*' answers for the '*all developers*' sample is higher than for any of the developer categories we examine. Tool use in the individual categories we consider is relatively low, particularly amongst freelance and lone developers.

6) Security Tools: In an extensive literature review of software security aids, in which we considered accessibility for under-resourced or isolated developers, we concluded that automation and security tools hold high promise for developers who do not have much security support. Therefore, we explored the usage of security tools in the five developer categories. The results can be seen in Fig. 7.

We were disappointed to observe that developer categories who are most likely to need the support of automation and security tools used them least, with freelancers at 29%, lone developers at 31% and developers in small organisations at 32%. Developers outside organisations had access to security tools at a rate of 38%. There was a noticeable difference between developers in small organisations at 32% and those in medium-sized organisations at 45%. The number for respondents as a whole was 48%, suggesting that large organisations get the most benefit from security tools.

We would like to see much higher numbers here. Only 29% of freelancers use security tools. There is a long way to go.

V. Discussion

A. Threats to Validity

As with most surveys, our respondents are self-selecting and the results may therefore suffer from self-selection bias or

nonresponse bias. We attempted to mitigate this by recruiting organically, by keeping the survey open for three months to collect a large sample, and by publicising the survey through multiple different venues. Nevertheless, our respondents may not accurately reflect the general developer population. However, we believe that the relative CA compliance between the different developer categories we examine is likely to reflect the broader population.

Survey responses sometimes suffer from social desirability bias, as respondents try to anticipate expected answers. To counter this effect, we emphasised that no security expertise was needed to participate.

Our sample size for freelancers is low at 23 participants. We do not claim that these results are generalisable, but we find them thought-provoking and believe that they may stimulate discussion and further research.

B. Discussion

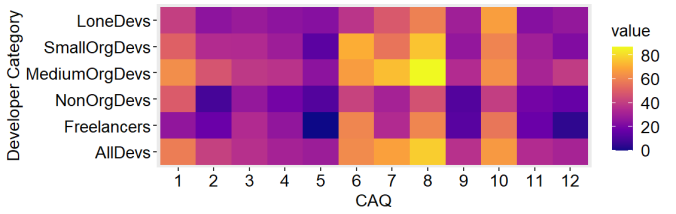

Fig. 8. **Heat map of CA answers.** This heat map allows us to easily see the most-used common activities in the five developer categories.

When investigating use of the twelve CAs in our target developer categories, we were prepared to see little to no adherence to the activities. It was encouraging to see that these security practices are present in at least some instances. We can use Fig. 8, which shows a heat map of common activities use, to evaluate our initial intuition as to the three CAs that would be least used. The levels of compliance constraints translated to requirements (CAQ2) reached 47.3% in medium-sized enterprises, but were low at 8.96% for non-organisational developers. The highest level of security review (CAQ11) was 31.8% for medium-sized organisations. The lowest, 19.4% for non-organisational environments, was higher than we expected given the likely budgetary constraints. External penetration testing (CAQ12), used by 39.5% of medium organisations but only 4.35% of freelancers, performed badly but not as badly as internal penetration testing (CAQ5), used by 24.8% of medium organisations but not used at all by our small sample of freelance developers. We speculate that internal penetration testing is too time-consuming and may take too much knowledge for developers to undertake it themselves. Although it is notoriously difficult to persuade management to invest in external penetration testing, several of our participants mentioned bug bounty firms in their responses. For example, one of our respondents from a medium-sized enterprise said:

> '*HackerOne is a service our company subscribes to for bug bounties/external penetration testing and it provides a lot of value for us.*'

As to the other activities, CAQ3 and CAQ4 are relatively little used, which is disappointing since these concern QA work which could be done by developers themselves. Our developers are strongest on CAQs 6, 7, 8 and 10 which concern deployment speed and resilience, developer and operations coordination, and network security basics. These activities can be undertaken for quality assurance and general cybersecurity reasons, without a software security focus. Therefore their presence does not imply a software security mindset in the working environment. In general, the level of practice of the CAs is higher than expected. We were disappointed by the level of security tool use, and would like to see this rise for developers who may not receive much other support.

VI. Conclusion

Security vulnerabilities in software may expose its users to multiple types of cyberattack, including ransomware, cyber terrorism and cyber espionage. This is a particularly acute concern for critical systems. Yet many contemporary software systems depend on components developed externally, often by small organisations or open source teams. We wished to examine aspects of software security practice in five developer categories that are under-represented in the literature. These are lone coders within organisations, coders within small organisations, coders within medium organisations, non-organisational coders, and freelancers (see Table II).

To do this, we analysed a dataset of 962 valid respondents to a survey on the state of software security practice. The survey asked about use of the 12 most common software security activities (CAs) (see Table I). Many of these CAs can be undertaken even by lone developers — for example, ensuring that host and network security basics are in place. However, we posited that some of them would be too expensive or time consuming for under-resourced developers to undertake.

We extracted the data on adherence to the twelve CAs for the five developer categories of interest. The results were encouraging, with some level of adherence to all of the activities within each developer category. However, there is a large degree of variation between developer categories for many of the activities. We found that most of the activities are undertaken most frequently by medium-sized organisations, the largest developer category we studied. This ties in with our understanding that the activities can be resource-intensive.

We also explored security tool use. Intuitively, automated tool use should provide most benefit to under-resourced developers. Surprisingly, we found that security tool use is quite low in our developer cohorts. This may be because developer sources of advice have gaps when it comes to discussing the benefits of security tools [28].

Based on our results, we advocate greater targeting of small and under-resourced software development communities with software security advice. Secure software development standards tailored to these categories of developer are needed. We also suggest that the benefits of security tool use should be promoted to under-resourced developers whenever possible. The accessibility, usability and price of security tools and other

security aids should be considered by the industry in a holistic way. Isolated and under-resourced developers must be able to find and deploy them.

Acknowledgment

We thank the anonymous reviewers for their helpful comments and input. 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.

References

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

[2] J. Jancar, M. Fourné, D. D. A. Braga, M. Sabt, P. Schwabe, G. Barthe, P.-A. Fouque, and Y. Acar, ""They're not that hard to mitigate": What cryptographic library developers think about timing attacks," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, p. 632–649.

[3] 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.

[4] S. Nadi, S. Krüger, M. Mezini, and E. Bodden, "Jumping through hoops: why do Java developers struggle with cryptography APIs?" in *Proceedings of the 38th International Conference on Software Engineering - ICSE '16*. ACM Press, 2016, p. 935–946.

[5] 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, 2016, p. 289–305.

[6] 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.

[7] 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.

[8] M. Rahman, N. Imtiaz, M.-A. Storey, and L. Williams, "Why secret detection tools are not enough: It's not just about false positives - an industrial case study," *Empirical Software Engineering*, vol. 27, 2022.

[9] M. Tahaei, K. Vaniea, K. K. Beznosov, 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*. ACM, May 2021, p. 1–17.

[10] 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.

[11] I. A. Tondel, M. G. Jaatun, and D. S. Cruzes, "IT security is from Mars, software security is from Venus," *IEEE Security Privacy*, vol. 18, 2020.

[12] H. Palombo, A. Z. Tabari, D. Lende, J. Ligatti, and X. Ou, "An ethnographic understanding of software (in)security and a co-creation model to improve secure software development," in *Sixteenth Symposium on Usable Privacy and Security*. USENIX Association, 2020.

[13] A. A. Ur Rahman and L. Williams, "Software security in DevOps: Synthesizing practitioners' perceptions and practices," in *Proceedings of the International Workshop on Continuous Software Evolution and Delivery*. ACM, 2016, p. 70–76.

[14] I. Tøndel, D. Cruzes, M. Jaatun, and G. Sindre, "Influencing the security prioritisation of an Agile software development project," *Computers and Security*, vol. 118, 2022.

[15] H. Assal and S. Chiasson, "Security in the software development lifecycle," in *Proceedings of the Fourteenth USENIX Conference on Usable Privacy and Security*. USA: USENIX Association, 2018, p. 281–296.

[16] M. W. Sammy Migues, John Steven, "BSIMM," https://www.bsimm.com/, 2020, [Online; accessed 17 December 2020].

[17] OWASP, "SAMM," https://www.opensamm.org/, 2021, [Online; accessed 13-April-2021].

[18] W. McCurdy, "Lazarus hackers are using Log4j to hack US energy companies," https://www.techradar.com/news/lazarus-hackers-are-using-log4j-to-hack-us-energy-companies, 2023, [Online; accessed 16 March 2023].

[19] N. C. S. Centre, "Cyber Essentials," https://www.ncsc.gov.uk/cyberessentials/overview, 2023, [Online; accessed 27 January 2023].

[20] ISO, "ISO/IEC 29110 Series," https://committee.iso.org/sites/jtc1sc7/home/projects/flagship-standards/isoiec-29110-series.html, 2023, [Online; accessed 16 March 2023].

[21] J. Mejía, M. Muñoz, P. Maciel-Gallegos, and Y. Quiñonez, "Proposal to integrate security practices into the ISO/IEC 29110 standard to develop mobile apps," in *New Perspectives in Software Engineering*. Springer International Publishing, 2022, p. 29–40.

[22] T. Bender, R. Huesmann, and A. Heinemann, "Software development processes for ADs, SMCs and OSCs supporting usability, security, and privacy goals - an overview," in *The 16th International Conference on Availability, Reliability and Security*. ACM, 2021, p. 1–6.

[23] 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.

[24] I. Ryan, U. Roedig, and K.-J. Stol, "Measuring secure coding practice and culture: A finger pointing at the moon is not the moon," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, May 2023.

[25] C. Weir, S. Migues, M. Ware, and L. Williams, "Infiltrating security into development: exploring the world's largest software security study," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2021, p. 1326–1336.

[26] D. van der Linden, P. Anthonysamy, B. Nuseibeh, T. T. Tun, M. Petre, M. Levine, J. Towse, and A. Rashid, "Schrödinger's security: Opening the box on app developers' security rationale," in *Proceedings of the 42nd International Conference on Software Engineering (ICSE)*, 2020.

[27] D. Wermke, N. Wöhler, J. H. Klemmer, M. Fourné, Y. Acar, and S. Fahl, "Committed to trust: A qualitative study on security & trust in open source software projects," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, p. 1880–1896.

[28] Y. Acar, C. Stransky, D. Wermke, C. Weir, M. L. Mazurek, and S. Fahl, "Developers need support, too: A survey of security advice for software developers," in *2017 IEEE Cybersecurity Development (SecDev)*. IEEE, Sep 2017, p. 22–26.

[29] T. Espinha Gasiba, I. Andrei-Cristian, U. Lechner, and M. Pinto-Albuquerque, "Raising security awareness of cloud deployments using infrastructure as code through cybersecurity challenges," in *The 16th International Conference on Availability, Reliability and Security*. ACM, 2021, p. 1–8.

[30] C. Weir, I. Becker, J. Noble, L. Blair, M. A. Sasse, and A. Rashid, "Interventions for long-term software security: Creating a lightweight program of assurance techniques for developers," *Software - Practice and Experience*, vol. 50, no. 3, p. 275–298, 2020.

[31] 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.

[32] P. Morrison, "A security practices evaluation framework," in *Proceedings of the 37th International Conference on Software Engineering - Volume 2*. IEEE Press, 2015, p. 935–938.

[33] Enterprise Ireland, "SME Definition," https://www.enterprise-ireland.com/en/About-Us/Our-Clients/SME-Definition.html, 2023, [Online; accessed 30 January 2023].

[34] European Commission, "New EU cybersecurity rules ensure more secure hardware and software," https://digital-strategy.ec.europa.eu/en/news/new-eu-cybersecurity-rules-ensure-more-secure-hardware-and-software-products, 2023, [Online; accessed 30 January 2023].

[35] N. L. Review, "Federal government outlines new security and attestation requirements for software," https://www.natlawreview.com/article/federal-government-outlines-new-security-and-attestation-requirements-software, 2023, [Online; accessed 30 January 2023].