Developer Essentials: Top Five Interventions to Support Secure Software Development

March 2017

Summary Report

Charles Weir (Security Lancaster, Lancaster University, UK)

Prof. Awais Rashid (Security Lancaster, Lancaster University, UK)

Prof. James Noble (ECS, Victoria University, Wellington, NZ)

Executive Summary

Cyber security is a big and increasing problem. Almost every week we hear of a new exploit or security breach that leads to major concerns about our digital infrastructure. Software systems are at the very heart of this digital infrastructure. Therefore, while there may be many commercial, social and practical factors that contribute, it is certain that the decisions of software development teams must have a significant impact on the vulnerability of those systems.

In this research we explored ways in which outside actors – such as management, coaches, security teams, industry bodies, and government agencies – may positively influence the security of the software created by development teams, while keeping the development competitive and practically viable. This means that the costs of such 'interventions' need to be acceptable relative to the risks that they address.

We interviewed 14 specialists in introducing software security to development teams. Based on a rigorous analysis of their responses, we were surprised to find that three of the most cost effective and scalable interventions are 'cultural interventions' – ones that work to influence the working of development teams, rather than the artefacts they produce:

1. Developing a 'threat model' and using that model to achieve commercially negotiated, risk based, agreement how threats are to be addressed;
2. A motivational workshop engaging the team with the genuine security problems as they affect their specific projects, while making it clear how they are to address those problems; and
3. Continuing 'nudges' to the developers to remind them of the importance of security.

The other two low-cost and effective interventions relate to the code produced.

4. The use of source code analysis tools; and
5. The informed choice of components based on their security quality.

We therefore suggest that providing guidelines, technical support and mentoring in each of these five interventions will have a significant effect on improving the security quality of code developed in future.

1 Introduction

Cyber security is a major issue. With a major exploit being described in the media pretty much every week, and increasing liability for companies who are affected, it is important to understand what can be done to improve the situation.

While there are many aspects to an organisation's security and privacy, the quality of the software developed by programmers has a large impact on whether or not cyber-attacks are effective.

Accordingly this research project asked the question:

> *What interventions can change the environment for members of a development team to achieve good security, considering motivational factors, choice of tools, supporting processes and potential blockers, culture, awareness, training and skills?*

We addressed this question by interviewing fourteen experts in the field, and analysing their responses. This report presents our initial findings; we believe there will be further and deeper conclusions from more detailed analysis.

Section 2 describes the research we did and outlines previous work by others. Section 3 describes some of our findings, and section 4 adds conclusions based on these findings.

2 The Research

We conducted face-to-face interviews with fourteen professionals in software security:

Table 1: The Interviews

CC	Organisation size	Organisation type	Est SCM	Typical Role
UK	Medium	Outsourced software developer and consultant	High	CEO
UK	Solo	Security consultant	Low–Med	Consultant
UK	Large	Security and military supplier	High	Team leader
UK	Large	Research organisation	Medium	Research and support
US	Large	Operating System Supplier	High	Security team leader
UK	Large	Security and military supplier	High	Security expert
UK	Medium	Software security tool supplier	Medium	CEO
UK	Large	Telecommunications provider	Medium	Security expert
UK	Solo	Security consultant	High	Consultant
DE	Large	Software package supplier	High	Security expert
UK	Medium	Software security service supplier	Low-Med	Training and consultancy
UK	Medium	Telecoms service provider	High	Security expert
UK	Medium	Telecoms service provider	High	Team lead
DE	Large	Research organisation	Low-High	Research and consultant

All the interviewees were involved in influencing software development teams as at least part of their job. We chose them opportunistically, from contacts in the software engineering world, and devised questions to emphasise their successes and expectations rather than problems and setbacks. To analyse the interviews, we used Constructivist Grounded Theory [3], which establishes intellectual rigour with line by line textual analysis.

Table 1 summarises the interviews, with an indication of country, organisation type and size, the main day-to-day role of the interviewee(s), and a subjective estimate of the 'secure software capability maturity' [6] of the associated software teams.

We consulted the interviewees as experts, rather than analysed them as subjects, using questions to draw out what they themselves had found most effective, and what they had seen to be most effective in other teams.

Table 2: Related Work

Directly related to this work are two recent surveys of industry specialists. Such et al.[16] investigated the economics of well-known software security assurance techniques, concluding that public review and tool-based static analysis were the most cost-effective. Black et al.[2] investigated technical approaches, providing a good overview of the subject but coming to no conclusions.

Many academic and commercial teams have produced static analysis tools; we found little proof of their effectiveness, except for a limited trial [24] that found that users still needed to be motivated to fix the errors.

There has been some work on how to create this motivation and encourage tool use, generally based on Rogers' 'Diffusion of Innovations' [15]. Several surveys [10,22,23] identified the importance of developers seeing colleagues using such tools successfully, something that most tools do not facilitate.

Prior to 2010, the main way of getting teams to improve software security was the 'Secure Development Life-Cycle', a prescriptive set of instructions to the development team [21]. However these proved unpopular with developers [5,7,14], and has been replaced by 'Security Capability Maturity Models' (SCMMs) [9,11], to allow management influence based on a variety of measurements. By stipulating targets rather than formal routines, SCMMs allow development teams to find their own approaches to security.

Addressing the problem of helping such teams, a recent survey of app developers found they mostly learned security through web search and from peers; it also highlights the poor quality of many web resources [19].

Research on the effects of external consultancy [12,18] suggests that a single time-limited involvement is generally ineffective in the longer term. Others have investigated the effects of developers' professional interactions, identifying benefits from improving relationships with security practitioners [1,20], and encouraging challenging communication [19]. Meanwhile, several works [4,8,18] stress the importance of security as a business goal, driven from board level.

3 Exploring Interventions

We identified eight main kinds of intervention in use by our experts:

Incentivisation Workshop	Most of the interviewees discussed a form of presentation or workshop to help motivate the developers themselves to understand and prevent security problems. Some did this regularly, as part of the induction process for new employees; others made it a one-to-one for each developer. Bigger companies do as much as a two-day security sensitisation course for every programmer. External consultants and security specialists described using an incentivisation workshop to establish their credibility. Often this is based on a penetration test of the software being developed. Few of the experts suggested merely 'scaring the developers into security'; instead the consensus was to shock them, but leave them knowing how to solve the problem. Almost all stressed the importance of personalising the workshop to cover the specific threats for a given project and the reasons why they matter.
Threat Modelling	Many of the interviewees discussed the importance of some form of Threat Modelling with the programming team: of analysing the likely attackers, threats and commercial impact of attacks for the systems under development. One interviewee also pointed out the importance of recognising a hierarchy of attacks, and of addressing simple attacks before more complex ones.
Component Choice	Several experts change programming teams' use of plug-ins and frameworks. There are two aspects to this. First, using an insecure plug-in automatically makes the developed system insecure, regardless of the quality of the code developed by the team. So a 'low hanging fruit' for development is to use only plug-ins that are well written and securely implemented. This is non-trivial, given the wide range of plug-ins available. For some environments there are web sites with security reviews of plug-ins; cross referencing – preferably automatically – with these sites is a powerful security technique. Where such sites are unavailable, or for new plug-ins, there is a value to code reviews of the plug-in. This does however have a significant cost to development teams since it costs effort, however much automation may be involved, and restricts the plug-ins that that they can use. Second, since plug-ins are widely shared, any weakness in a plug-in becomes known to attackers, and therefore it is important to keep plug-ins upgraded to the latest versions.

Developer Training	Other than the Incentivisation Workshop, there was less emphasis than we expected on developer training as an intervention. Most use less formal training, in the form of drip feed workshops discussing the latest forms of attack, or new security techniques. Some included penetration and attacking techniques amongst the things taught to developers. However a more common form of training seems to be mentoring, typically by including programmers more expert in security, or by having security experts work closely with the team.
Static Analysis	Many of the interviewees discussed static analysis tools. Few saw them as the most important intervention, other than for security experts evaluating large bodies of code and for compliance checking. Most saw them as valuable in automating the removal of certain classes of security bugs, and as part of a larger security story for developers. However, several did not use such tools at all. Some pointed out that even standard compilers, used properly, can help considerably. When used for external reviews and audits, the tools tended to be used without extensive configuration. Some experts stressed the importance of configuring and even writing one's own tools to suit non-standard projects. In terms of actual use by developers, several interviewees stressed the value of integration with the development environments. Interestingly, however, these were all the people who are involved in the creation of such tools! It seems intuitive that this would be a good idea, but we found no user evidence from this research to confirm its value.
Penetration Testing	Many of the interviewees stressed the importance of penetration testing. Several stressed the importance of a tight integration between the penetration testers and the development team. This can be mean embedding pen testers within the teams, having developers seconded briefly to pen testing team, or having discussion workshops. A tight integration means both that the penetration testing can be as effective as possible, and that the developers learn from the penetration testers. Unfortunately penetration testing requires significant expertise; this expertise is in a relatively short supply, and is correspondingly expensive. That also makes a programme of widespread penetration testing very hard to scale. Indeed some experts felt penetration testing to be inappropriate, or usually not very helpful.
Code Review	Another frequently-recommended technique was code review. Many of our interviewees were working with relatively expert teams on software security, and these had a consistent story about code review being one of the most important factors in their secure delivery.

	Some concentrated on code review by security experts, whether as a form of technology transfer, or simply as an external viewpoint. Others used code review by fellow programmers, with varying levels of intensity. Still others used a pair programming or buddy programming approach to provide a continuous review of the code as it was developed.
Continuous Reminder	A particular problem highlighted by some experts is that while the initial motivational talk may be effective, its impact is relatively short-term. After a few weeks or months the development team will revert to their previous insecure development approaches. To counter this, interviewees mentioned a variety of approaches, all of which could be summed up as kinds of 'nudge' [17] – small reminders of the importance of security issues. Examples include a security competition, positive feedback when a team achieves a secure product, using public security disasters in the news as lessons, and drip feed reminders in the development environment. Several also recommended having one of the developers in a team become a security specialist, not so much for their expertise as to provide a continuous reminder.

4 Conclusions

Our particular aim was to identify interventions that can work effectively with a wide range of development environments.

We can identify two aspects of effectiveness. First is the financial cost; whether financial costs are acceptable will depend largely on the corporate environment. Second is the effect of team discipline. We might define team discipline as the likelihood that an initiative started by the team (such as code reviews, or test-first development) is still being carried out six months later. As some of the research found [13,18], for security initiatives this probability is often quite low.

We know that teams can create secure software if they are both highly disciplined and very well-funded [2]; if interventions are to have any significant impact they should also work in other environments. Specifically they will need to work with teams that lack significant funding for security, and they will need to work with teams that lack the very high discipline found in many of the most successful secure developers.

Figure 1: The Cost of Interventions

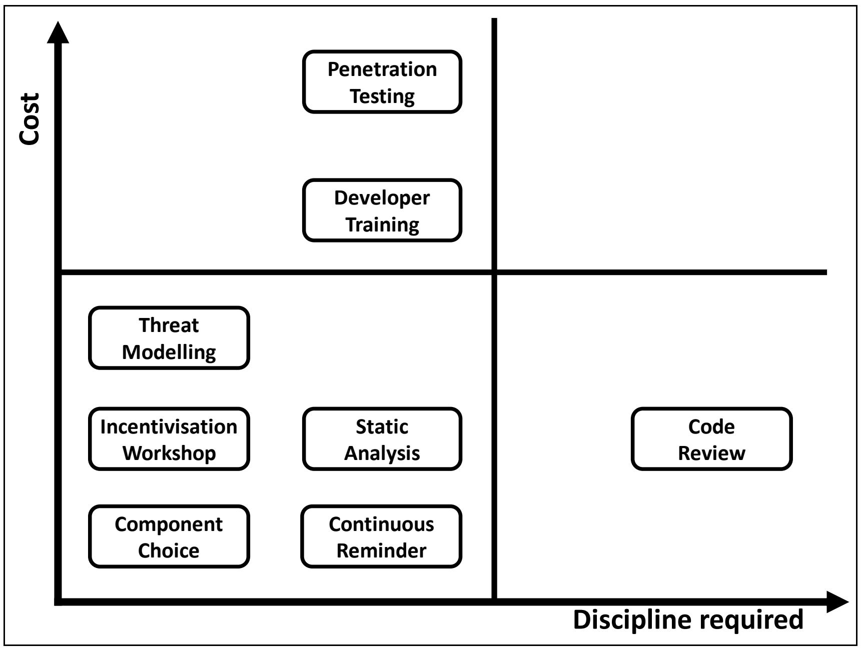

Figure 1 shows how the interventions identified by our experts fit into these criteria. The horizontal axis positions each intervention in terms of discipline required; the vertical axis in terms of financial cost [16].

Given that these are all interventions that the experts consider highly effective, there will be most impact from promoting the interventions that programming teams are most likely to adopt. We propose that these are the interventions that require a minimum of both discipline and financial cost: those in the bottom left-hand quadrant. We believe the other three to be important too; however these five represent ‘quick wins’ that will provide significant benefit.

Of these five interventions, three – Threat Modelling, Incentivisation Workshop, and Continuous Reminder – are ‘cultural interventions’, changes to the ways the developer teams work. The remaining two – Static Analysis and Component Choice – have low-cost options for most environments (though their costs can range to very high).

We therefore propose that to give maximum impact over a wide range of development teams, the best approach will be to provide guidance on how to achieve those five interventions.

5 References

[1] Ashenden, D. and Lawrence, D. Security Dialogues : Building Better Relationships. *IEEE Security & Privacy Magazine*, June (2016).

[2] Black, P.E., Badger, L., Guttman, B., and Fong, E. *Dramatically Reducing Software Vulnerabilities: Report to the White House Office of Science and Technology Policy*. Gaithersburg, MD, 2016.

[3] Charmaz, K. *Constructing Grounded Theory*. Sage, London, 2014.

[4] Dybå, T. An Empirical Investigation of the Key Factors for Success in Software Process Improvement. *IEEE Transactions on Software Engineering 31*, 5 (2005), 410–424.

[5] Hardgrave, B., Davis, F., and Riemenschneider, C. Investigating Determinants of Software Developers' Intentions to Follow Methodologies. *Journal of Management Information Systems 20*, 1 (2003), 123–151.

[6] ISO/IEC. ISO/IEC 21827:2008 - Systems Security Engineering - Capability Maturity Model. *2008*, (2008), 144.

[7] Lavallee, M. and Robillard, P.N. The Impacts of Software Process Improvement on Developers: A Systematic Review. *34th International Conference on Software Engineering, ICSE 2012*, (2012), 113–122.

[8] McGraw, G. Four Software Security Findings. *Computer 49*, 1 (2016), 84–87.

[9] McGraw, G., Migues, S., and West, J. Building Security In Maturity Model (BSIMM7). 2016. https://go.bsimm.com/hubfs/BSIMM/BSIMM7.pdf.

[10] Murphy-Hill, E., Lee, D.Y., Murphy, G.C., and McGrenere, J. How Do Users Discover New Tools in Software Development and Beyond? *Computer Supported Cooperative Work (CSCW) 24*, 5 (2015), 389–422.

[11] OWASP. Software Assurance Maturity Model Project. https://www.owasp.org/index.php/OWASP_SAMM_Project.

[12] Poller, A., Kocksch, L., Kinder-Kurlanda, K., and Epp, F.A. First-Time Security Audits as a Turning Point? *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems - CHI EA '16*, (2016), 1288–1294.

[13] Poller, A., Kocksch, L., Türpe, S., Epp, F.A., and Kinder-Kurlanda, K. Can Security Become a Routine? A Study of Organizational Change in an Agile Software Development Group. *Proc. CSCW'17*, (2017).

[14] Riemenschneider, C.K., Hardgrave, B.C., and Davis, F.D. Explaining Software Developer Acceptance of Methodologies: A Comparison of Five Theoretical Models. *IEEE Transactions on Software Engineering 28*, 12 (2002), 1135–1145.

[15] Rogers, E.M. *Diffusion of Innovations*. Simon and Schuster, 2010.

[16] Such, J.M., Gouglidis, A., Knowles, W., Misra, G., and Rashid, A. Information Assurance Techniques: Perceived Cost Effectiveness. *Computers and Security 60*, (2016), 117–133.

[17] Thaler, R.H. and Sunstein, C.R. *Nudge : Improving Decisions about Health, Wealth and Happiness*. Penguin Books, 2009.

[18] Türpe, S., Kocksch, L., Poller, A., Türpe, S., Kocksch, L., and Poller, A. Penetration Tests a Turning Point in Security Practices? Organizational Challenges and Implications in a Software Development Team. *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, USENIX Association (2016).

[19] Weir, C. How to Improve the Security Skills of Mobile App Developers: An Analysis of Expert Knowledge. 2017. http://eprints.lancs.ac.uk/84664/1/2017weirmbr.pdf.

[20] Werlinger, R., Hawkey, K., Botta, D., and Beznosov, K. Security Practitioners in Context: Their Activities and Interactions with Other Stakeholders within Organizations. *International Journal of Human Computer Studies 67*, 7 (2009), 584–606.

[21] De Win, B., Scandariato, R., Buyens, K., Grégoire, J., and Joosen, W. On the Secure Software Development Process: CLASP, SDL and Touchpoints Compared. *Information and Software Technology 51*, 7 (2009), 1152–1171.

[22] Witschey, J., Xiao, S., and Murphy-Hill, E. Technical and Personal Factors Influencing Developers' Adoption of Security Tools. *Proceedings of the 2014 ACM Workshop on Security Information Workers - SIW '14*, (2014), 23–26.

[23] Xiao, S., Witschey, J., and Murphy-Hill, E. Social Influences on Secure Development Tool Adoption: Why Security Tools Spread. *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW*, (2014), 1095–1106.

[24] Xie, J., Lipford, H.R., and Chu, B.B.-T. Evaluating Interactive Support for Secure Programming. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2012), 2707–2716.