Create a Certification Framework for Secure Development Practices

Positioning paper in response to the call for position papers on Standards and Guidelines to Enhance Software Supply Chain Security, targeting area 2, “Initial list of secure software development lifecycle standards, best practices, and other guidelines acceptable for the development of software for purchase by the federal government.”. Addressing point 4(e) (ix) “attesting to conformity with secure software development practices”, out of the executive order.

By, Matias Madou, Ph.D., CTO Secure Code Warrior, mm@scw.io, +32 495 25 49 78, Brian Chess, Ph.D. bchess@vantuyl.com and Pieter De Cremer, Ph.D candidate, pdecremer@securecodewarrior.com

Security automation tools (SAST, DAST, SCA) have made it easier to identify software vulnerabilities, but this has had little impact on the prevalence of vulnerabilities in almost all types of software [1,2,3]. These tools are typically deployed too late in the software development life cycle, slow down agility and do not provide specific guidance to remediate these vulnerabilities. To turn the tide, fundamental changes must be made in software development practices. The ability to detect vulnerabilities is not enough; we need fewer vulnerabilities to be created. The vast majority (90%) of vulnerabilities are caused by problems in the code through insecure coding patterns that have been known for years [3,4,5]. The most effective way to prevent vulnerabilities is hence to train, assess, and certify developers on secure coding, including these patterns. To produce a measurable change in developers' abilities, training must be more relevant, efficient, usable. In this paper we argue the best path toward this goal is to put more emphasis on building out a certification program for software developers.

With the use of security tools, the security team has become quite competent at finding problems in the code. Once these (potential) vulnerabilities are discovered, it is up to the development team to fix them. In order to help them manage this task, the security team pushes discovered vulnerabilities into a bug tracking system. They even organise the vulnerabilities by category and prioritize them by severity of impact. To actually understand each issue, and to fix them in a consistent way, however, developers are often on their own. On average a company that is security aware hires only 1 security expert for each 75-200 developers [5]. This expert simply can not assist each of those developers.

It is evident that security is no longer just the task of a secure software assessor. Every software developer producing code should be responsible for doing this securely from the start. Many secure development methodologies recommend providing developers with targeted role-specific training [8,9,10,11]. And in the industry this is widely practiced, as we can observe from software maturity models [6,7]. We do not expect software developers to become security experts, but they should know basic secure coding hygiene.

The acquired skills should be relevant to the software developer’s work. If the context in which they learn is similar to the context in which they need to use their acquired skills, this will improve recollection [12]. This phenomenon is one of the reasons why pilots do not learn to fly a plane through slide presentations but by using flight simulators with specific types of airliners [12]. Similarly, developers should not learn secure coding through books or slide presentations but through working in actual code in specific environments (IoT, Web, API, Mobile, ICS). Each developer should receive training in the same programming language and framework they are using daily in order to understand syntax specific secure and insecure coding patterns. While many security concepts are generally applicable, the actual solutions to problems are often programming language framework specific, and these solutions are exactly what developers should be taught [13]. A software developer working on a web application in Java/Spring, and a developer working with older programming languages including COBOL should not receive the same training. The former should be taught secure coding by means of Java/Spring code for web applications, the latter should be trained through COBOL code examples resembling their legacy code bases.

The National Initiative for Cybersecurity Education (NICE) Framework provides a reference for educators and organisations to develop a role-specific training program that covers the knowledge, skills, and abilities required for each member of the workforce [10]. This training should be periodically reviewed and updated as needed [10]. Everyone inside of the organization involved in the Secure Development Life Cycle (SDLC)

should be prepared to perform their roles and responsibilities, including the software developers themselves (NICE Workrole SP-DEV-01). To achieve this they need role-specific skills and training to prevent the introduction of insecurities and to fix detected problems in a scalable way.

Our position is that software developers have to become upskilled in writing secure code through training to demonstrate verifiable knowledge and skills (NICE K&S) [10]. We need to ensure that every role and responsibility throughout the SDLC is performed by a workforce with highly relevant skills or competencies [9]. These required skills and competencies are clearly not the same for every software developer, as security threats vary a lot over different programming languages and software types [14,15,16]. This competency should be recorded by a certification body that provides a reference for educators and a test to assess the developers [10].

The required training should be relevant to the developer's work. Based on code examples in the right language and framework, and in specific code environments (IoT, Web, API, Mobile, ICS). The training should also be efficient in achieving the developer's needs. Developers should be provided with exercises that help them acquire the right skills that are immediately applicable. These skills are recognizing insecure code patterns and (re)writing their code so that it is secure. Exploiting vulnerabilities is not their main learning objective, and hence training should be tailored to that goal. This can be done through defensive exercises, not by recycling training designed for penetration testers and other security experts. The appropriate training should ultimately lead to a measurable skill acquisition.

Many metrics exist to rate the security and quality of code, or the technical debt of a project. However, there is much less attention for measuring the capabilities of the software developers who are producing this code. It is key in managing a security-aware workforce that this acquired K&S level should be verified through standardized certification.

We highly encourage NIST to build out a standardized framework for certification. This way all types of software developers can acquire more targeted and highly relevant certification. The NICE Framework can be used as a reference to build this certification program. For example, to acquire the certification for input validation (NICE Skil S0019) each software developer (NICE Workrole SP-DEV-01) should be able to demonstrate this skill in the language and framework they are using daily (e.g. Java, Spring). By building this certification framework NIST can provide clear guidelines for the development of software for purchase by the federal government. These guidelines should include that software should be produced with conformity to secure software development practices, including appropriate certification of all developers producing code.

[1] Trustwave. *Global Security Report 2018*
[2] Black, P. E. *A software assurance reference dataset: Thousands of programs with known bugs.* Journal of research of the NIST, 2018.
[3] U.S. Department of Homeland Security. *Infosheet Software Assurance*
[4] The Open Web Application Security Project® (OWASP)
[5] Common Vulnerabilities and Exposures (CWE)
[6] McGraw,G. et al. *Building Security in Maturity Model (BSIMM) 9, 10, 11*
[7] Pravir C et. al. *The Open Software Assurance Maturity Model*
[8] Potter, B. *Microsoft SDL threat modelling tool.* Network Security, 2009
[9] McGraw, G. *Software security touchpoint: Architectural risk analysis*. Addison Wesley Software Security Series, 2009
[10] NIST Special Publication, the Workforce Framework for Cybersecurity (NICE Framework): original and revision
[11] Dodson et al. (2020) Mitigating the Risk of Software Vulnerabilities by Adopting a Secure Software Development Framework (SSDF).(Cybersecurity White Paper, NIST)
[12] Dirksen, J. *Design for how people learn*. New Riders, 2015.
[13] Japan Smartphone Security Association, Android Application Secure Design/Secure Coding Guidebook, 2016
[14] Kurilova, Darya et. al. *Wyvern: Impacting software security via programming language.* Evaluation and Usability of Programming Languages and Tools, 2014.
[15] Seixas, N. et. al. *Looking at web security vulnerabilities from the programming language perspective: a field study*. International Symposium on Software Reliability Engineering, 2009.
[16] Delaitre, M. *Sate V report: Ten years of static analysis tool expositions*