⊕ **POSITION PAPER**

PROMOTE A PAVED PATH SECURE DEVELOPMENT METHODOLOGY

Authored by: Matias Madou, Ph.D., Brian Chess, Ph.D., and Pieter De Cremer, Ph.D candidate

Promote a Paved Path Secure Development Methodology

Positioning paper in response to the call for position papers on Standards and Guidelines to Enhance Software Supply Chain Security, targeting area 4, "Initial minimum requirements for testing software source code". Addressing point 4(e) (iv) "employing automated tools, or comparable processes, that check for known and potential vulnerabilities and remediate them, which shall operate regularly, or at a minimum prior to product, version, or update release;", 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 (SDLC), slow down agility and do not provide specific guidance to remediate these vulnerabilities. To turn the tide, fundamental changes must be made in 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 problem is no longer finding vulnerabilities, but producing code that is secure from the start, as well as remediating problems in a scalable way. In this paper we argue the best path toward this goal is to work with a *paved path methodology.* In this methodology the developer is guided along a known secure path, provided with real-time guidance when writing code.

The automated Continuous Integration and Continuous Delivery (CICD) pipeline, introduced with the DevOps movement, allows for automatically running security tools. This makes sure that the tools and their feedback are more accessible to the developer. However, for bigger projects such tools easily need several hours to complete their analyses because even moderately complex applications contain complex data flows [8,12]. Such long scanning speeds are not ideal for DevOps pipelines, where tight feedback loops are crucial. Almost all developers (96%) report that the biggest inhibitor to productivity is this disconnect between development and security workflows [6].

To solve this, the security team tunes the tools so that they run more lightweight analyses [9]. One common way to achieve this is disabling certain rules. More lightweight analyses do not hinder the developers' productivity and also allow for fast feedback of the analysis results. Of course, the full scan should still be run regularly, for example on a daily basis. This is already an improvement over regularly pushing security problems into a bug tracking system, but it is still too disconnected from the development workflow. There is still only 1 expert to help up to 200 developers, so the problem of preventing and fixing security bugs at scale remains. The security team acknowledges this, as they rank creating developer-friendly workflows as their top priority, even ahead of protecting the production software itself [6].

Our position is the use of such a developer-friendly workflow, named *the paved path methodology*. In this methodology the security team should not force security testing on developers, but instead gradually build a *paved path* for developers to follow. This paved path should be different for each project and heavily depend on the technology stack for that project. Together developers and security experts should build standards and patterns that lay out the paved path. For example, they can decide together how key management should be handled by deciding on the library and the tools needed, or even by creating a new (wrapper) library. Developers will then follow that path, as it is the easiest way for them to implement a feature that needs key management.

Developers are known to dislike and often disable security tools during development [6]. They frequently perceive it as one of the biggest inhibitors of productivity. Tools supporting the paved path methodology are in the first place designed as developer tools, security should come as an indirect result. This can be achieved with tools that guide the developer along the paved path.

Guiding the developer along the paved path should boost productivity while lowering the developer's cognitive burden. If the security experts have done a good job laying out this paved path, the resulting code

will be secure. When developers are focused on the functionality of their code and using a library for this purpose, security is usually orthogonal to that purpose [6,7,8,9]. A good tool should then warn a developer when he strays from the paved path and guide him back without hurting productivity [10].

We highly encourage NIST to include the use of a tool fitting the developer-friendly paved path methodology in the minimum requirements for testing software source code. This tool does not simply check for vulnerabilities, but helps prevent and remediate them. The tool requires only lightweight analyses, so it is able to be operated continuously and provide feedback in real-time from the start of the SDLC.

The tool should be relevant to the developer's work. To create the paved path, the security and development teams create guidelines that specify the preferred solution for security critical features. These guidelines can be based on existing coding guidelines [11], but need to be made project specific. Enforcing them requires easy customization of the rules that are verified by the tool.

The tool should be efficient in achieving the developer's needs. When using traditional security tools, poor quality of feedback and slow scanning speed are big inhibitors of developer productivity [8]. These inhibitors stem from the complexity of program analysis. Tools that verify if the developer is following the paved path can use much more lightweight analyses, resulting in real-time feedback. If this feedback is delayed for several hours or days, as is often the case with traditional security tools [9, 10], the developer has already moved on to their next task. To process the feedback and remediate any insecurities, they then need to go back and understand the code, requiring a larger investment in time and resources [12].

This tool should be usable. Since the enforced paved path is the desired outcome, it is possible for the tool to provide very targeted remediation feedback. The developer does not need to research any security problems by themselves. Instead the fix is obvious, and can even be automated by the tool. This can be done by reusing existing IDE features, as shown in Figure 1. These features are commonly used for marking generic best coding practices. Avoiding this research makes sure the developer does not need to make context switches but can stay focused on what really matters to him: the functionality of the code.
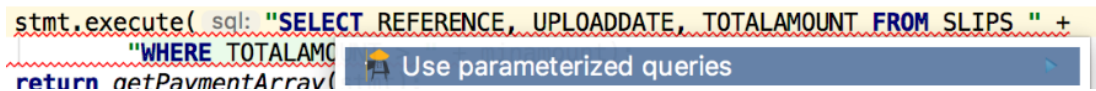

Figure 1: Automated remediation by reusing existing IDE features

The tool should be measurable. In classic security tools false positives are from the perspective of the code: incorrect markings of code that are not actually insecure [12]. However, to the developer, any marking of code that they do not intend to fix (insecure or not) is a false positive, inhibiting productivity. A false positive from the perspective of the developer is called an *effective false positive* [10,13]. Keeping the effective false positive rate low is of great importance to gain, and keep, the developer's trust.

[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] Xie, J. et. al. *Why do programmers make security errors?* IEEE Symposium, 2011
[7] Dirksen, J. *Design for how people learn*. New Riders, 2015.
[8] ShiftLeft. *Developer Productivity & Security Survey*
[9] Kern, C. *Securing the tangled web*. Queue, 2014
[10] De Cremer, P. et. al. *Sensei: Enforcing secure coding guidelines in the integrated development environment*. Software Practice and Experience, 2020
[11] Code Conventions for the Java TM Programming Language
[12] Delaitre, M. *Sate V report: Ten years of static analysis tool expositions*
[13] Meyer, A. N. et. al. *Software developers' perceptions of productivity*. International Symposium on Foundations of Software Engineering, 2014
[14] Sadowski, C, et al. *Tricorder: Building a program analysis ecosystem.* IEEE International Conference on Software Engineering, 2015