**Application Security Program: Volume & Velocity**

**TL;DR:** How to keep things real and build an healthy relationship with Development and Business.

### .....

Unrealistic security requirements show an essential disconnect between the security team and their counterparts, development and business. Neither development nor business will be inclined to be subjected to any draconian security policies.

We need to consider two factors: one is the volume of issue-related changes, and the other is the expected velocity to implement those changes.

* **VOLUME: How many issues** are sent to the development teams?
* **VELOCITY: How much time** do we have to fix these security issues?

To have the development on-board for the Appsec journey, both factors must be addressed.

If we look at the problem from a development perspective, the developer team is always under pressure to deliver code in time and with a high quality level. The security issues get piled up on top of everything else that the developer team is already doing, which ultimately is: to grow and sustain the business. For example: asking them to fix every single issue present in the code-base, regardless of when these issues were introduced for the first time, is just an unrealistic and arbitrary expectation that is bound to cause the entire initiative to fail.

At the same time, from a business perspective, to delay a delivery in production could end in serious losses in terms of missed revenue or sometimes even in terms of penalties. It is strategic to keep this potential risk in mind before asking them to fix EVERYTHING before going live.

Dropping off a report on a developer's desk with thousands of static analysis warnings isn't going to get buy in from the team. Try taking an incremental, low-friction approach. [Cit.: "Agile Application Security Book"]

To minimize the impact of VOLUME on VELOCITY we need to simplify the problem, and since it is a matter of size, the solution is pretty easy: split the volume into **manageable sized chunks**.

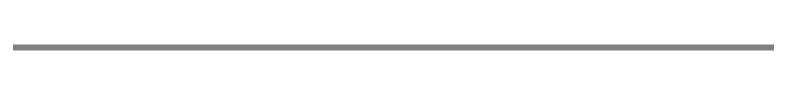
Most of the times, if not all the times, when evaluating legacy implementations for the first time the number of potential issues is very high. As much as we don't like it, most of the times applications are not developed and maintained keeping obsolescence in mind, so depending on the age of the code, the engineer’s programming style, and the paradigms used, applying static analysis to existing code can range from difficult to nearly impossible if a disciplined approach isn’t followed.

To deal with this challenge, first we need to be able to correctly discriminate between what is NEW and what is OLD, then focus on what is directly achievable by the developer team:

* First address the new issues applying a "NO **NEW**SECURITY ISSUES" policy to stop accumulating more security debt.
* Then define a different policy and remediation road-map for backlog issues composing the security debt. A classic approach is defining a grace period differentiated by issue severity.

An arbitrary example of incremental approach could be:

* Fix 80% of backlog CRITICAL issues in the next 3-6 months
* Fix 20% of backlog CRITICAL issues and 80% of backlog HIGH issues in the next 6-12 months
* Fix 20% of backlog HIGH issues and 80% of backlog MEDIUM issues in the next 12-18 months



Disclaimer: the views and opinion present on this post are those of the author and do not necessarily reflect the official policy or position of any other agency, organization, employer or company.