**Application Security Program: It's all about balance!**

TL;DR: How to keep application security real without going broke.

**.....**

Normally organizations/companies don’t perceive security as a market differentiator, and while lately I have observed a slow change in their perspective, it is evident that any application security program, as it happens with many things in life, to be considered successful, must be affordable.

We have known for some time now that the cost of remediation in production outweighs the cost of remediation in development by a factor of 15, and that the same remediation after the test phase costs 2.5 times more ([IBM System Science Institute](https://www.researchgate.net/figure/IBM-System-Science-Institute-Relative-Cost-of-Fixing-Defects_fig1_255965523)).

Let's imagine a potential scenario where we have an application that's being developed, and we think there is a bug (whether it is a security bug or not is not important for the time being). We can decide our confidence level is not sufficient to treat it as a real bug, so we leave the bug alone and we wait until the test phase to verify if it's real or not.

During the test phase we have no supporting evidence to improve our confidence level, so we forget about it and we send the code in production.

A few months later a user stumbles upon the bug and, being a nice person, they report it back. Now the application (being in production) is managed by the maintenance team (aka Business As Usual or BAU team).

Someone in the BAU team will spend time identifying where the problem is inside the code (they didn't write the application in the first place), then more time to see if a hot-fix can be implemented or if the problem requires an escalation.

If we are lucky it's a hot-fix, and the change will be scheduled and put together with a bunch of other HF, implemented, tested, pushed to the dev repository, and deployed in production (mind… this is the happy path).

OR if we are not so lucky, it will require escalation, where the bug will be passed to someone who will review it and hopefully understand what it is about and what to do, then will define a solution and file it for implementation for either the currently developed version (if there is one) or for future version (good luck).

This is a story about a generic bug, now let's think about the same story for a security bug. Scary eh?

Let's not think about the scary part, but just focus on the number of actions. Those actions are the overhead costs for issues identified in production.

Now if we don't "shift left", we know that the issues found during the test phase (or in production) will require some code change. These changes will require to go through several controls. Without even touching the potential losses of an extended time to market (or the extended risk exposure if in production), the cost related to these controls will be sustained twice for the same feature.

In reaction to this, we see that these secondary controls are often skipped to keep the cost of change as low as possible. The “procedural tool” to achieve that saving is the proverbial technical debt. We see these concerns entering the risk register and ultimately having no real evidence that the fix has been implemented correctly (or at all).

So, at its essence, the organizations (the PMs) trade $$ with unknown risk.

It's like betting: sometimes you win, sometimes you lose (risk appetite).

Finding issues in development doesn't come for free as well, but these costs are mainly sustained upfront and they are very predictable.

If we really had to make it even clearer, security doesn't MAKE MONEY, at best it SAVES it. It goes that it is best if security costs less than it saves.

A good strategy to achieve this goal is to maximize savings, identifying and fixing software security defects as early as possible in the development pipeline, when the application is being written (enter the "shift left").

We need to intercept the issues when they are introduced, and we need to minimize the time to remediation. This will positively affect the costs associated with the issue remediation, which will effectively happen at development time rather than after production.

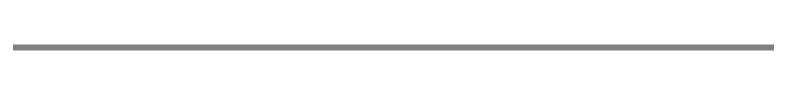
The shift left is a **necessary** condition, but it is **not a sufficient** condition for success. Any solution that can deliver such capability must be able to sustain CI/CD release speed.

The need for Automation is not driven only by the existing skill shortages in the security job market. It's a necessary condition for cost efficiency at scale. At the same time Automation is great because we don't want to waste highly skilled human intellectual capital doing menial tasks that a machine can do for us!

Cost efficiency doesn’t come just from automation and prevention. It comes from sensible expectations and realistic strategies for rolling out an application security program like I discussed in my previous post about volume and velocity.

If the system is absolutely accurate and capable of incredibly high throughput, but costs as much as the overall development cost, unless you have very great reason for doing that, there is a good chance that you will see your service canned at the first budget review, and probably even earlier than that.

It's all about (dynamic) balance.



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.