DevSecOps: Is security achievable in the frantic landscape of modern software development?

Introduction

The world of DevOps is not, at a glance, one of caution. Quite the opposite, the philosophies of DevOps strive to get as far away as possible from slow and bulky traditional methods of development where you spend more time deliberationg if and how you should do something than you do building it. But in the chase for speed and agility DevOps adopters may have turned reckless, and might have lost sight of where they came from.

A survey performed in 2018 by Tripwire on 311 IT security professionals at companies with more than 100 employees showed 60 percent of companies had experienced a container security incident during the past year. As containers are a core technology of the DevOps practise, this suggests that switching to the aforementioned development method may have introduced new vulnerabilities in the software currently in development. Has DevOps turned developers reckless? And is security impossible to achieve with the current raging tempo of software development?

History

We will first take a look at the history and predecessors of DevOps to gain a deeper understanding of what is it and where it comes from.

DevOps was not concieved from a vacuum. It is an amalgamation of ideas and practises with a past ranging back decades, depending on how generous you are when determining what constitutes a contribution. To start investigating the roots of DevOps, let us examine what it strives to get away from: the waterfall model.

The waterfall model of production (it can and has been used to produce a wide variety of products, including software) represents the rigid method a project leader might intuitively adopt if they sought to rigorously plan a project from start to finish. It splits the development process into stages where each stage depends on the former being finished to progress. Is is easily understood and managed, but lacks flexibility, and any errors commited in the early planning phases will not become apparent until far later in the review and operations phases.

An early method for moving away from this type of production was the Lean method pioneered by Toyota all throught the later half of the nineteen hundreds. This method focuses on satisfying customer needs, and removing everything in the production process that does not produce value to the customer. It is also known for making quality a distributed responsibility, assuming the worker building each part of the product is most knowledgeable about that part of the production process, and thus should perform oversight over it. It is famously exemplified by Toyota installing a shut down chord at each work station along the production line in its factory, giving every employee on the line the power to completely halt production if they noticed a quality issue.

A later, more software centric approach is the Agile development method which seeks to minimise the lenght of the cycle of building, delivering the product to the customer, and adjusting to feedback. An agile team should constantly iterate on its work by delivering and receiving feedback on incremental changes througout the development process. Agile aims to bring the developers as close as possible to the customer and improve their ability to conform to the customer's needs.

If agile aims to bring developers closer to customers, DevOps wants do to the same with maintainers and developers. In the current landscape of software development, often products arent shipped to customers as much as services are continuously constructed and maintained. With DevOps practises we take the upkeep of the software into consideration in our development plans, and fucus on managing deployment of new software so that it can be done efficiently, incrementally, and without disrupting operations. DevOps when used to develop live services becomes sort of an extreme version of the agile mindset where the product is always available to the end user, and developers are constantly adapting to requirements and user feedback.

What is DevSecOps?

It is not obvoius where security comes into play in the DevOps process. The constantly changing nature of devops software and reliance on automated testing to assure that the product works don't seem to give a lot of room for deeper security considerations. To alleviate this problem DevSecOps was born.

DevSecOps builds upon the DevOps model by striving to inject security measures into the production pipeline. Like Continuous Integration and Continuous integration are used to ensure the softwares functionality, DevSecOps insterts security checks into the full extent of the deployment process. It is important that security is not only considered in one step of the development, but in each one of them. Traditional security measures are not fast

and adaptible enough to conform to the pace of modern software development, so we need to integrate security into the new development processes.

DevSecOps is sometimes referred to as "shifting left", which implies shifting security measures to earliyer stages of the deployment. If we imagine the deployment pipeline as a set of stages from left to right, starting with commiting to a source repository, followed building, CI, staging, and so on, shifting left means to move security audits from the end of this lineup into the earlier stages. This can actually be a benefit to efficiency, even disregarding security concerns. Imagine a SQL injection vulnerability is accidentally commited into a project. If this is not discovered until late in the deployment pipeline, the developer may well have moved on onto another task when the alarm bells go off. It will now take some time and effort for them to find their place in the older code to be able to remedy the fault, and their current work is impacted by being interrupted! Any shortening in the time between commiting and receiving a security warning in these cases will help developers work more efficiently. Lets investigate some examples of ways of doing this presented in the talk DevSecOps: What, Why and How by Anant Shrivastava.

A simple, but prudent measure is installing pre-commit hooks onto developers machines that scan commited code for easily identified security leaks. We can use regexes to find compromising data such as database keys and block the commit from proceeding. This is already autimatically done by some version control system providers (GitHub will identify and warn you of commited security tokens) but it is more efficient and reliable not to depend on such functionalities. Another example of developer-level measures are IDE plugins that generate warnings for easily detectible security issues, like the use of == instead of === in JavaScript, for example. These types of measures are easily circumventable by the developer and will not protect against malicious behaviour from within, but they will assist in preventing accidental damage.

An interesting problem with assuring software security today is that developers don't write that much of their programs themselves. Modern software development depends on usage of publically available modules to operate at the pace it does. To manually ensure the security of each and every import in a large project quickly becomes unfeasible, and therefore another suggestion for DevSecOps is to use Software Composition Analysis. This is a system for analysing the components of you software and identifying any known vulnerabilities, deprecations or outdated versions of imported code.

The next step is SAST, Static Analysis Security Testing. This is a term for systems that statically analyse your code (and any imported code) to identify possible security issues within the structure of the program. This is more sophisticated than the regex based

approaches described earlier, but we still only look at the code and not the running program.

To investigate the program in a deployed environment we turn to a method named DAST, or Dynamic Analysis Security Testing. Now we scan the application from the outside as it is running, the same approach that would be used by a hypothetical attacker. An interesting feature of DAST software is that it does not need to be language-specific since it only interacts with your application through its public API. DAST and SAST can be used in conjunction, for example one can validate the results of the other.

There is another aspect ot DevSecOps apart from the technical, automated strategies described above: culture. DevSecOps aims to implement security as a culture where everyone is responsible for the security of the final product. In his talk Shrivastava mentions that the focus of a dedicated security team within a DevSecOps organisation is to eliminate the need for a dedicated security team. This is, again important from an efficiency standpoint. If software is developed with security as a concern throughout the process, by people who are all educated in relevant security theory, security can be exercised as a part of rapid development, as opposed to acting as an obstacle to it.

During the history summary earlier in this essay you may have noticed DevOps seems to be quite closely based on the Agile methodology, and not as directly influenced by Lean philosofies. But while DevSecOps might just seem like a new, shiny step in the development forward, I think it can be seen as a return to the priciples that helped build Toyotas decades ago. A distribution of responsibility regarding quality amongst the production staff, trust that the employees know their areas best and will report problems they find are important principles both in DevSecOps today and the car factories of 1950's Japan.

The State of DevOps Security today

The State of DevOps Report is an annual servey investigating the world of DevOps performed by the increasingly interestingly named companies puppet, circleci, and splunk, all actors in DevOps-enableing technology. The 2019 survey gathers data from 2949 respondents who are "technical professionals with a working knowledge of their IT operations and software delivery process", and for 2019 it focuses solely on the situation regarding security in DevOps.

It is immediately apparent that this document does not refer to the integration of security

measures into DevOps as DevSecOps. The reasoning behind this is presented on page 26, where it is stated that:

> While we appreciate the emphasis on Security that the term DevSecOps brings, we're sticking to just DevOps. We believe security is an integral part of both the Dev and Ops domains. But if using the term DevSecOps helps drive heightened awareness for the importance of building security into all aspects of software delivery, we're all for it.

While the report is not expressly focused on DevSecOps, it is clearly the de facto topic.

The report states that most companies still have not been able to integrate security into their delivery pipeline, and it poses a hindrance to implementing DevOps fully. But they also find that in companies with a wide and successful adoption of DevOps, security not only works but thrives.

One of the author's main points is that doing DevOps well enables you to do security well. They argue this on grounds that we have encountered before: a high level of adoption of DevOps practises drive security practises as well, through automation and monitoring as well as culture and sharing. The processes that ensure software is reliable are easily adaptable to improve security as well (as we have seen in earlier examples), and a culture of sharing and cooperation helps build a shared responsibility for security and an exchange of security knowledge.

Conclusion

So, DevOps may not be about being cautious, but caution and sluggishness are not neccesarily requirements of security. A possible avenue could instead be to use modern development techniques to embrace security, making it more of a priority while at the same time helping efficiency. A way to drive this development could be a change in attitude: from considering security as an accessory or "extra precaution" to instead see it as an integral part of what constitutes quality. Because we use the same methods to achieve security now as physical production lines used to ensure quality before us. And while the people building cars were not as worried about directed attacks towards the car's functionality (car break-ins is probably the closest analogy) the pursuit of quality was to withstand the attacks of the car's operating environment, from bumpy roads to rain and snow. And with the global and constant cyber attacks we now live with, enduring, ruthless battle has become the normal environment. It is imperative that we code with it in mind.