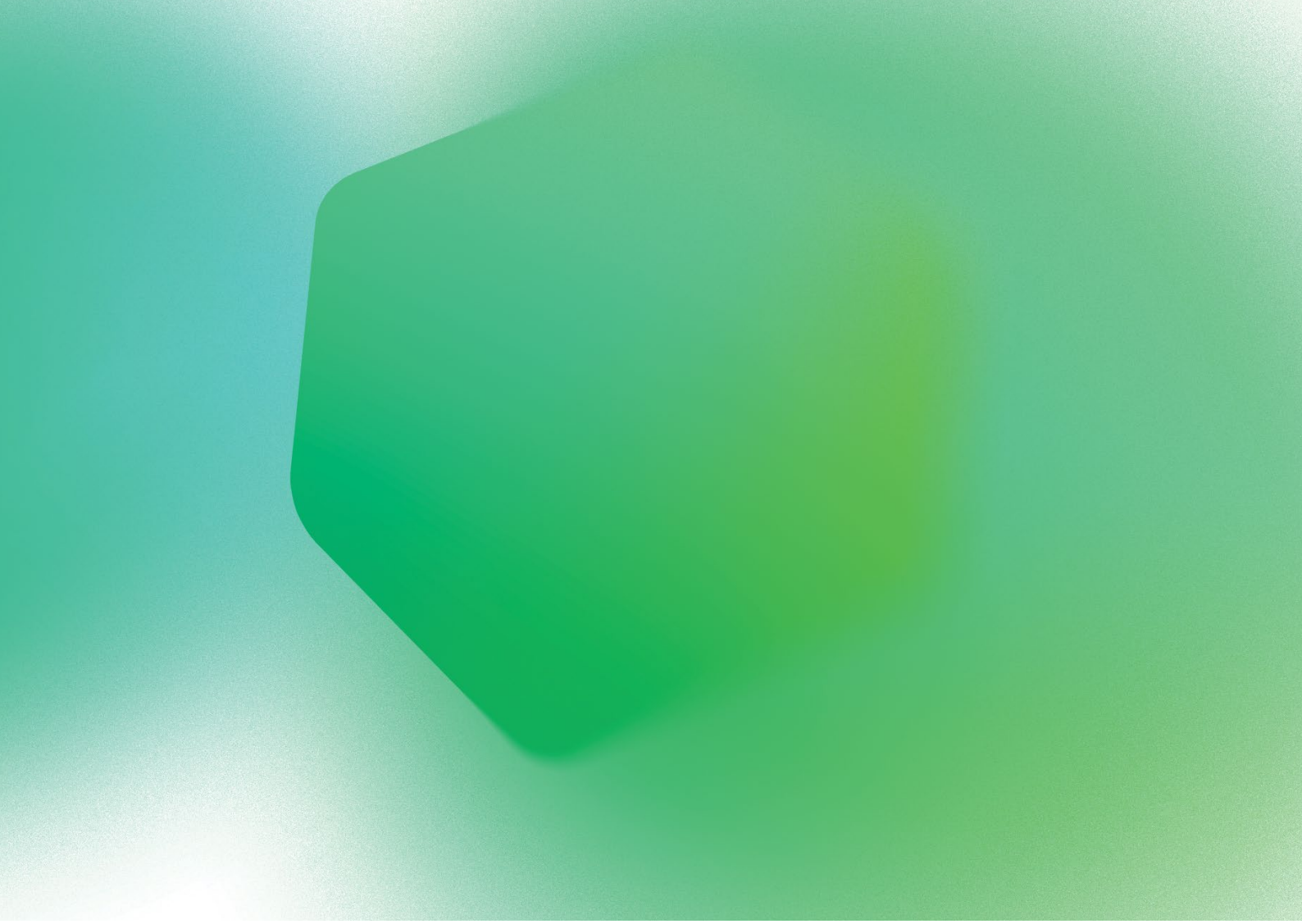

DevOps – an unsuspecting target for the world's most sophisticated cybercriminals

kaspersky

Learn more on kaspersky.com
#bringonthefuture

A dynamic approach that changes the world one build at a time. Automatically.

DevOps
DevOps stands for Development Operations. It is an approach to software development that focuses on Continuous Integration (CI) and Continuous Delivery (CD) – automated pipelines that help organizations improve business-impacting KPIs like time-to-market, product development speed, agility and so on.

There are organizations around that, thanks to DevOps, have switched from annual or semi-annual to daily or even hourly software releases. DevOps techniques like 'everything is code' mean that source code written and committed by a developer before lunch can actually be available in production by the end of the same day.

Not just the product, but the development and testing environments, are often built on the fly from components sourced from internal and external repositories and orchestrated by automation servers. DevOps write instructions as code and, when executed, this code guides the assembly of development pipelines and entity sourcing, and the build, test and release of the final product into production.

Supply chain attack
A cyberattack where the attackers access a software or development environment component, directly or indirectly, and use it to deliver their payload into the target's infrastructure.

Inevitably, the automated building of software that's then distributed by software vendors straight into major corporates worldwide creates the sort of attack vector that cybercriminals dream about. And the fact that so many components of both the product and the software development environments are fetched from the internet on the fly means that developers too become low hanging fruit. The result – a proliferation of supply chain attacks.

Supply chain attacks

There's no shortage of news about the latest supply chain attacks. It can be extremely difficult, even for us security professionals, to be one hundred percent certain about what's happened in most attacks of this type. But we can make some pretty good guesses, learn a great deal, and apply this knowledge in addressing the corporate risk posed by criminal infiltration of the software supply chain.

Here are a handful of interesting examples of supply chain attacks, in no particular order:

- In August 2017, some APT actors outfitted software created by NetSarang with malicious modules. According to investigators, the attackers may have compromised the software build servers.
- In 2018, cybercriminals infected the Piriform application build server, after which CCleaner program builds with clean source code were weaponized during compilation.
- In 2019, our experts discovered the ShadowHammer APT campaign, during which malefactors embedded a backdoor into software products from several companies. According to the results of the investigation, the attackers either had access to the source code or introduced malicious code at the compilation stage.
- In April 2020, IT news websites reported that RubyGems, the official channel for distributing libraries for the Ruby programming language, had been poisoned. An attacker, used the tactic known as 'typosquatting', uploaded fake packages containing a malicious script, so that all programmers using the code in their projects unwittingly infected users' computers with malware that changed their cryptocurrency wallet addresses.
- Typosquatting, generally considered the most common tactic for cyberpoisoning, has also been deployed in attacks through the Python Package Index and in uploading fake images to Docker Hub.
- In the Copay cryptocurrency wallet incident, the attackers used a library whose repository was hosted on GitHub. Its creator had lost interest and given away the administrator rights, compromising the popular library, which many developers used in their products.
- Sometimes, cybercriminals are able to use the account of a legitimate developer without the latter's knowledge, and substitute real packages for fake ones. That happened in the case of ESLint, whose libraries were hosted in the NPM (Node Package Manager) online database.

The question that I believe we should all keep asking ourselves is – what if my organization becomes another target? Would I be able to mitigate this risk?

No easy answers

Many such attacks can be prevented by the deployment of security to development infrastructure servers, the routine security vetting of containers and anti-malware testing of the production artifacts.

But here's the problem. Traditional security products tend to lack integration interfaces – so their deployment creates bottlenecks in the automation process. The result is a lot of time-wasting manual intervention due to fragmented automation, overcomplicated processes, and limited visibility – none of which goes down well in DevOps environments.

"Divide and conquer" – the attackers' mantra

So there are fundamental differences between the operational goals of the parties involved in maintaining and operating in the development environment – IT, infrastructure, InfoSec and DevOps. And these differences, and the security stalemates that can result, play into the hands of attackers.

If you're a DevOps Engineer, you may well have spent time coming up with good reasons why you don't want a particular security product deployed onto your systems. You may even have found yourself complaining to business units hungry for your output that an InfoSec guy is setting out to 'Negatively Impact Your Time-to-Market'. Or if you're an InfoSec guy reeling from your first brush with DevOps, you probably know about having to swallow your pride, crawl back to your office and lick your wounds after finding yourself labelled 'A Negative Impact On The Business'. The corporate battle-lines have been drawn.

But if history teaches us anything, it's that 'United we stand and divided we fall'. And unity can be achieved if everyone stays locked onto a single strategic goal – delivering a safe product to partners and customers in time. Working together, DevOps and InfoSec can ensure that software productivity continues forging ahead exponentially, while running rings around invading cybercriminals.

Any structure is only as good as its foundations

Let's start with this agreed premise - that some form of security foundation is essential.

It's imperative that you deploy run-time protection to each and every server and workstation. If it has an operating system and it's connected to a network, it needs protection. Developers' workstations, build servers, containerization hosts – the lot.

So now we need to think about performance. There's a whole set of technologies that are very respectful to performance but which can make a huge difference in terms of systems security. There's memory protection, exploit prevention, vulnerability assessment, network threat protection and so on and so forth.

These are just a few examples. The point is – no single technology is a silver bullet, but the right combination can be used to deliver the right DevOps security/performance balance. A multi-layered approach will take us closest to where we want to be.

There's also significant potential for security systems automation. A solution with a reasonable level of integration with other platforms will leverage that integration for workload discovery, status check and closing security gaps – all fully automated and on the fly. This level of automation can be super-crucial in cloud infrastructure auto-scaling, where the fast dynamics of the environment impair visibility and hinder control, for example, or with non-persistent or semi-persistent virtual machines, where new instances are spun on demand. Remember – if you can't see it, you can't protect it.

'Don't dis the basics'

Don't just dismiss a performance-friendly technology because it sounds 'old school'. Take signature-based protection – a technique barely recognizable in 2020 from what it was 25 years ago. Back then, we talked about things like MD5 hashes of files: today's signatures are capable of detecting groups of threats, finding similarities in polymorphic malware samples and spotting injected malicious code in benign files. The technology is so advanced and optimized that it easily handles known variants of malware with a very little overhead. If you don't like thinking about something as basic as "known malware" protection, think how embarrassing it would be to build the most advanced cyber-defenses, only to find you've left the main gates wide open for any 10-year old malware to waltz in.

Automated pipelines need security, too

First and foremost, your Endpoint Protection solution needs to be fully effective in pre-filtering incidents, before EDR comes into play. The earlier in the attack kill chain the vast majority of threats can be identified and countered automatically, the less the overall impact on resources. Most security incidents can be seen off right away by a good EPP solution, leaving your EDR solution and your security staff free to focus on the more advanced, and therefore the more dangerous, threats. We know we keep saying this, but – make sure your EPP solution is pulling its weight.

Let's look now at the building blocks we can use to build security into automated pipelines. What are the requirements? It's pretty simple – we need a service that will perform the necessary function and which can be accessed through some kind of interface. We need to understand how that interface works, how to interact with it, and what it can tell us. In other words, this interface needs to be well documented. For easy integration into our pipeline, we need to be able to access the service through scripts, so we can define what we need using code. We want to approach the security step just as we approach all other steps - creating an environment, creating a project, building it, testing it and ultimately deploying it.

The aim is to shift, or rather extend, the focus of security far beyond source code analysis and unit testing. These of course are a foundation – if your product is software, I'm pretty sure you'll want your product to be safe and not prone to breaking (i.e. not vulnerable). If your product is any more complex than 'Hello World!' and relies on third party code, you want to establish operational security as well – management of configurations and patching, users and privileges, monitoring events and collecting and analyzing logs. But if you're striving to achieve a continuous development flow through integration, hopefully delivery and ideally deployment, you'll have to adopt principles and processes like 'Everything Secure', 'JIT security', 'Fail Often Fail Fast' and 'Automated Security Testing' – all kinds of AST! And also of course security testing orchestration.

- **Everything Secure** means that whatever you pull or build needs to be sanitized – we're all very familiar with this concept right now! Think of it as like grocery shopping – now you need to wash your fruit and vegetables before you consume them.

- **Just-In-Time security** – in a dynamic environment you have to have a security service that you can throw stuff at whenever needed and get a near-instant response. If it doesn't support scripting – sorry, you're out of luck – this is not the security solution for you.

- **Fail Often Fail Fast** is the Agile practice that aims to minimize the impact of a security incident. The sooner a vulnerability is discovered, the easier it is to fix and the lower the cost of the incident. Our goal is to drive this to zero. This is especially beneficial when you integrate commits several times a day.

- Different kinds of **Automated Security Testing** – static, dynamic, behavioral, interactive. Testing of source code, compiler setting verification, binary testing – that all, with minimal impact, help fail fast so you can detect a problem early, correct it and continue with the release, minimizing the cost of the incident.

- Of course, with so many moving parts, **orchestration** proves critical, offering visibility, control and streamlining of the overall process. This cries out for a Centralized Security Management Console.

You can have your cake and eat it

What if DevOps can have their 'Security as Code' service to sanitize, patch, verify, test and so on, while InfoSec can have their policies, reports and dashboards?

Plot twist – what if DevOps want those reports and dashboards, too?

There are DevOps benefits to be had from a centralized security management console. Remember, your security solution can see a threat whether you've scripted a certain security check or not, because having a solid foundation in place means that ALL servers are protected no matter where – in your own datacenter or in one or more public clouds Your solution should also be able to see that threat in a larger context – that of your entire infrastructure. It will see exploit attempts, fileless attacks, memory attacks, network exploits and so on. And you can leverage that information to script back into your pipeline, in order to take action. You can define what events will fire that script up and how you integrate this safety element into your process.
It's a win-win for everybody. Except cybercriminals.

Security is an integral part of development

To ensure the security of the development environment and of the product itself, security needs to be incorporated into development operations. That means that security services need to be available whenever needed (on demand, or Just-in-Time) and they need to be automatable (to provide automation interfaces). This way, DevOps engineers will be able to weave security into their instruction code and the automation server will be able to perform those steps exactly when needed.

Or even better – don't think of security as "steps". It's a really more of a mindset. Those who wield it, prosper. Those who don't yet – will have to play a catch up game.

Kaspersky Hybrid Cloud Security: kaspersky.com/hybrid
Security for DevOps: kaspersky.com/devops
Security for AWS: kaspersky.com/aws

www.kaspersky.com

kaspersky BRING ON THE FUTURE