Checkmarx

The world runs on code. We secure it.

AppSec Considerations For Modern Application Development

A Comprehensive Guide for Leaders and Practitioners – Part 3

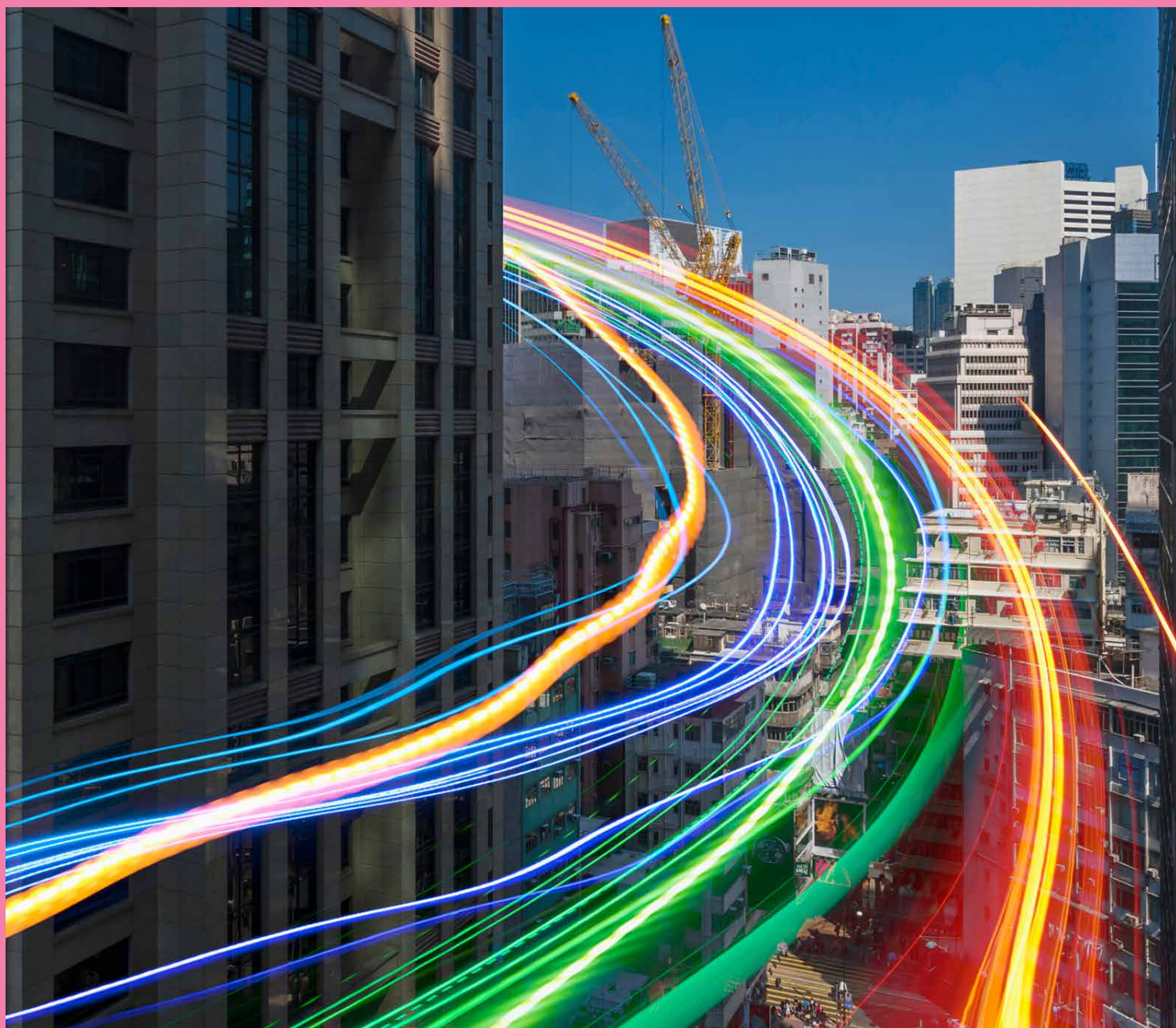


Table of Contents

Introduction

In Part 1 of this e-book series, we looked at the many facets of modern application development (MAD), including benefits and challenges to be expected. In Part 2, we went deeper into the security risks that surface in MAD and how they can impact any organization. If you've read Part 1 and Part 2, you should have a solid idea of what MAD entails, and what security risks to keep in mind.

In Part 3, we'll look at many modern application security (AppSec) considerations leaders and practitioners should be familiar with. Since MAD significantly changes software development practices in general, AppSec practices need to evolve as well. We'll wrap things up with some tips you should consider as you plan and pivot your organization's AppSec initiatives to address this new paradigm.

At Checkmarx, our mission is to improve software security for organizations worldwide by helping them reduce exploitable vulnerabilities. As the world adopts modern application development, we hope this e-book series can help.

A Short Discussion About Risk

Looking at the long list of risks we highlighted in Part 2, you'll see you can manage some risks with better AppSec practices, others with improved application security testing (AST) approaches, and some with enhanced AppSec awareness and training for developers—but there's no silver bullet that will eliminate all software risk. You need a broad set of strategies and solutions to cover all the angles. Let's take a closer look.

Managing Applicative Code Risks in MAD

Of all the code elements in a modern application, the first type that comes to mind is the applicative code: the source code that makes up all the functions the software performs from the "applications" perspective. Applicative code can be developed in-house or borrowed from a third party (e.g., open source). In MAD, however, open source comes with some caveats.

Open Source Code

It's easy to understand why open source is so pervasive. By importing open source libraries, components, and other resources into applications, developers avoid having to reinvent the wheel. They can reuse prewritten code, freeing up more time to code innovative features that don't yet exist.

Open source has its downsides, though. To avoid the security and compliance challenges that often accompany open source, developers and security teams need full visibility into the open source they use in applicative code, including its associated risks. Because of this (and partly driven by recent supply chain attacks and new government legislation), the concept of a software bill of materials (SBOM) is becoming more prominent in the context of open source.

> What an SBOM Is, and Why You Need One

"SBOM" is a term adapted from industrial manufacturing. A "BOM" is a list of raw materials or assembly components, along with the quantities of each, needed to manufacture a complete product. Think of it as a list of ingredients. The BOM serves as a verification document and helps ensure all necessary parts are present, available, and in the proper quantities. If a part is missing or its delivery is delayed, for example, there would be a flag in the BOM showing the issue with that component.

An SBOM, then, usually lists open source packages, libraries, and components found in a certain software application. For example, you may have a node.js project where the node_modules folder contains all the packages required to build and run the application.

If any items in the SBOM turn out to contain vulnerabilities or malicious code, your software pipeline is at risk, and affected items must be updated or replaced. If the application imports libraries from NPM, Maven Central, or any other registry, you can count them as open source parts of your application. The key here is to understand what really goes into an application your organization relies on.

> Why You Need an Accurate Parts List for Your Applications

With complete knowledge of all parts of your applications and what's required to build or compile them, you can mitigate a whole series of issues and risks. For example, if a new open source vulnerability is disclosed, you can check affected versions of the code against your SBOM. If you have matches, you can identify affected systems and take steps to remediate associated risks.

Let's look at another example. Suppose you were to discover that a library or dependency has been removed from a registry for some reason (e.g., license changes, legislative requirements, depreciation, lack of community support, maintainer decision). You wouldn't be able to keep using that component without potentially increasing risk, so you'd have to use a different version or consider a replacement. If you maintained an accurate SBOM, you'd be able to improve risk awareness and effectively mitigate that risk.

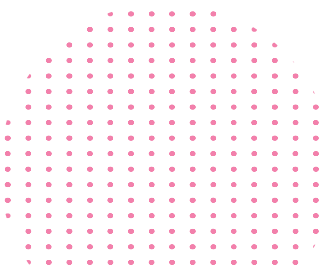
> Mitigating Open Source and Supply Chain Risk

The easiest and most reliable way to mitigate risk in the open source software supply chain is to ensure you get open source from its originator, validating it against the posted hash images. In some cases, this minor effort can catch supply chain attacks before they happen. Getting your software from the source isn't always an option, though, especially since many organizations upload their open source images to Docker Hub or add them to a central repository like Maven. To complicate matters further, you might also have to deal with private in-house registries.

Ultimately, having a better view of the open source your applications depend on will give you a clear view of the vulnerabilities associated with its usage and your overall risk. In MAD, you need a solution that provides an inventory of all open source in use and easily integrates with CI/build servers, artifact servers, and development environments.

Software composition analysis (SCA) can help your organization build and maintain an SBOM. Using an SCA solution designed for MAD, you can scan and compile the list of materials in your applications and gain full visibility into your codebase. You also need to be able to apply a Zero Trust model to open source to reveal potential attackers, locate hidden backdoors, and identify malicious code, with the ultimate goal of identifying malicious packages and mitigating risk. To do this, you need supply chain security solutions that provide behavioral analysis, link analysis, machine learning, and even threat intel about the open source supply chain.

Now that we've covered open source, let's look at proprietary code.

Proprietary Code

Proprietary code is the code your developers write in-house, often created to provide the necessary business logic to operationalize all open source code within an application. Like open source, security vulnerabilities inside your proprietary code can also be exploited. Worse, any security vulnerability not found until late in the software delivery cycle can be disastrous. Using a static application security testing (SAST) solution to scan proprietary code for vulnerabilities is imperative.

Applications developed with this approach are often made up of multiple microservices, a foundational element in MAD. Being able to scan incrementally for vulnerabilities at the source code level, without compiling code into a fully working application, is a necessity in MAD. Microservices are only pieces of the entire application, and their code should be scanned when any changes are made. The challenge is that many SAST solutions require a fully compiled application to run successful scans, and this doesn't work well in MAD. Let's look at why that is.

> Incremental Scans Are a Requirement in MAD

Traditional SAST scans can take hours to complete, and pushback is a given if full scans introduce delays in the pipeline and slow down software on its way into production. As a result, organizations often run fewer SAST scans, or they run full scans on a nightly basis to account for expected delays. These approaches only sidestep the issue, though. The key to faster scans is to introduce a SAST solution with incremental scanning capabilities, especially in light of microservices.

When a SAST solution is automated within source code management (SCM) tools, scans are launched incrementally against the branch of code a developer is working on. Pull requests, push events, merge requests, and so on will trigger incremental SAST scans and produce results when any code changes are made.

If you're trying to retrofit your existing scanning solutions into a MAD initiative, you'll eventually see that SAST solutions that only support full scans cause inevitable delays. To solve this, consider SAST solutions that can incrementally scan at the source code level, and scan as early and often as possible.

API Code

APIs are incredibly important for modern applications. Rather than serving only to import third-party data into an application, APIs are now an essential component of the architecture that makes modern applications work. Most microservices rely on internal APIs to discover and communicate with each other.

If you've taken a chemistry class, you might remember building molecules using ball-and-stick models, where balls represent atoms and sticks represent the bonds between them. In modern applications, the balls are the microservices and the sticks are the APIs. If your microservices can't talk to each other because of an issue with the APIs connecting them, your microservices-based applications stop working.

Other Uses of APIs in MAD

Likewise, you can't effectively deploy a containerized application on an orchestration platform like Kubernetes without using APIs to manage the various moving parts of your cluster: worker nodes, master nodes, pods, sidecars, and so on. It's also hard to deploy applications scalably in the cloud without using APIs to help automate application management, balance traffic, and so on. Although it's possible to manage cloud environments in other ways—via a web interface or CLI, for example—an API-centric approach is usually the most efficient.

Conventional dot-com era use cases for APIs also remain important today. Developers still frequently rely on APIs to integrate applications with external platforms as well as monitor and secure third-party resources. In modern cloud native environments, though, the role of APIs extends far beyond integrations and monitoring.

APIs Require Modern Security Approaches

Since modern apps rely heavily on APIs to communicate between distributed and loosely coupled applications and services, API security is high on the list of AppSec considerations. To ensure APIs don't become a liability, organizations need to prioritize API discovery, accountability, management, and security on their list of risk mitigation techniques.

API security risks are fundamentally different from web application risks, so you need purpose-built solutions that can identify internal and external APIs from an API contract, determine secure or insecure APIs, discover shadow APIs, and spot missing or weak authentication and authorization. You also need a way to visualize east-west traffic and service flow to show API relationships between services, and a means of better understanding risk associated with connected APIs. It's easy to see why API security is becoming top of mind for MAD initiatives.

Next, let's look at container code.

Managing Container Code Risks in MAD

There are many reasons to use containers: They're more agile and resource-efficient than virtual machines (VMs), more flexible and secure than applications running directly on the OS, and easy to orchestrate at massive scale using platforms like Kubernetes.

However, containers can also increase security risks. The benefits usually outweigh the risks, but it's important to assess and remediate the security problems containers can introduce. To reduce container risks, you need to know where images came from, what's in them, and what issues may occur when they run.

In addition, you need solutions able to provide registry scanning that supports the Open Container Initiative (OCI), Amazon ECR, private registries, and OpenShift. Container security scanning solutions should also provide exploitable path information, remediation guidance, and integration into infrastructure as code (IaC) testing solutions. Let's look at that topic next.

Managing Infrastructure as Code Risks in MAD

Developers and DevOps teams use IaC tools to describe common infrastructure components in a configuration language, which then serves as a blueprint to provision infrastructure services on demand. IaC gives Devs and Ops teams better control of the change process and helps make deploying changes more efficient and consistent. However, trying to implement IaC in real life often comes with risks.

How Devs Should Detect and Mitigate the Most Common IaC Risks

Most of the common risks (as we covered in Part 2 of this e-book series) can be mitigated by adhering to some common security best practices. Here are a few recommendations:

> Spend some time with the tools

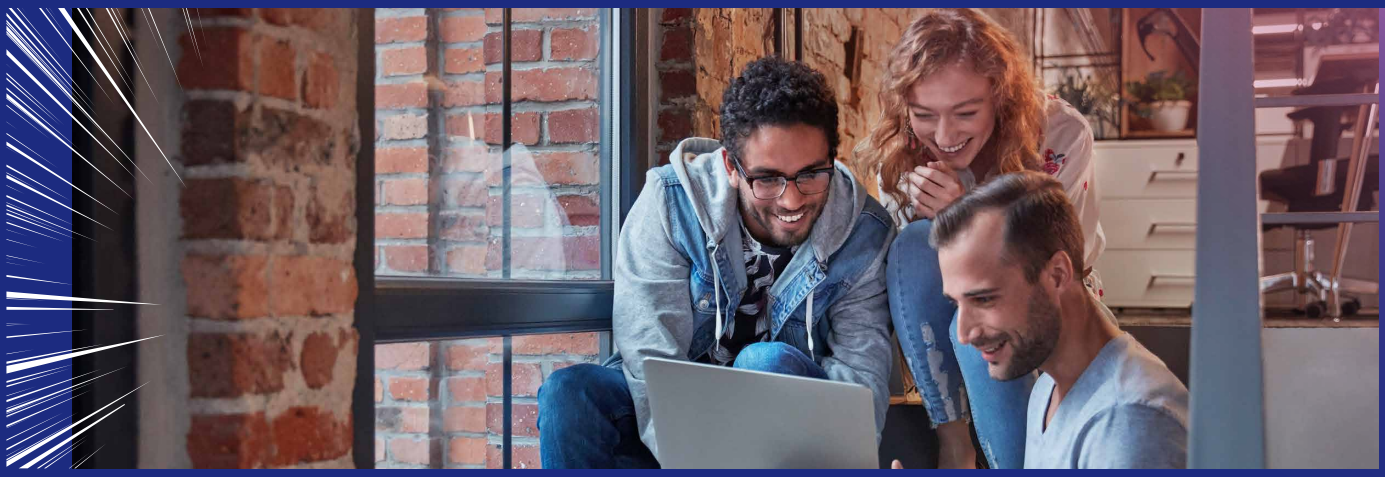
Try to fully understand how each IaC tool works, including its quirks, open issues, and best practices. Participate in meetups and subscribe to events so you can learn from other industry experts. This will give you an advantage when trying to figure out how to perform tasks, simple and advanced alike, without compromising your security posture.

> Establish common engineering processes and best practices

Practice peer code reviews, CI/CD checks, linting, and verification. This can reduce the number of common accidents and mistakes that could happen due to oversight.

> Use purpose-built security tools

Look for purpose-built IaC security solutions that help you establish a secure and efficient IaC pipeline. The benefit of these solutions is that you can configure them to match your organization's security policies due to their extensibility and cloud provider coverage.



Managing Developer AppSec Awareness and Training in MAD

As organizations continue to move toward shorter development cycles, more frequent releases, and increasingly complex application architectures, it's become more important than ever to consider application security at all stages of the development life cycle as well as make secure coding skills a top priority.

This means development teams' mindsets need to evolve, and those building your software need to take greater responsibility for the security of the end product. Instead of relying on an extensive post-development testing phase to root out security shortfalls and bolt on temporary solutions, teams must take a shift-left approach by adding security skills training right when developers need it most: while they're developing code.

Effective AppSec awareness and training programs should harness the advantages of modern technology. Much like an engaging mobile app can influence its users' behavior, the foundation for efficient secure coding practices can be rooted in gaming principles and technology-driven traits that keep users engaged in the long term.

Gamification—the application of game design elements and principles in non-game contexts—has widely recognized benefits, yet most AppSec awareness and training solutions don't take advantage of it. Gamified elements can be injected into multiple parts of the program, from simulated attacker vs. defender scenarios to unlocking "achievements."

An effective shift-left approach to AppSec in MAD requires developers to become more aware of security risks in the design and development phases, on top of employing secure coding practices while developing code. That's sound in theory, but changing the culture like this will require buy-in from the development team itself. One way your organization can make this happen is by empowering select developers to act as security champions.

The Role of a Security Champion

In MAD, security champions are motivated developers interested in continuously exploring and adopting best practices for coding securely. These developers must be willing to take an influential position in their team and the wider organization.

Security champions act as resources for the security team, facilitating a better understanding of the processes the development team follows. They also need to be willing to assist fellow devs with security-related questions, educate them on secure coding practices, and help ensure their applications are being built with security in mind. At a higher level, security champions can be involved in setting your organization's standards and policies for coding securely in addition to bringing greater awareness of security concerns to the organization.

> The Benefits of Being a Security Champion

For developers who fill this crucial role of mentor and evangelist, the responsibilities come with career-building benefits. Like any specialized proficiency, it can earn recognition from leadership, which can go a long way toward opening doors for advancement. By positioning themselves as experts on secure development practices, your security champions make themselves invaluable to the organization.

Alongside those loftier career benefits, mentoring other developers can be just as rewarding. Through leading by example and serving as resources for security-related concerns, security champions can earn their peers' respect while helping instill a mindset of shared responsibility for security at the developer level. This can facilitate positive cultural change, enabling the development of more secure end products while helping your organization move toward a less siloed work environment in which security and development teams work better together.

> The Challenge of Culture Change in MAD

No one can argue the importance of AppSec, but security champions may face pushback from fellow devs who feel that engaging these security concerns at the design and development phases will slow them down. These devs might see the upfront cost and overhead of integrating security into their process and feel that it will strain the team's ability to deliver changes efficiently. What they're missing, though, is that this culture change will lead to significant savings at the end of the development cycle.

Making security a shared responsibility across domains and considering it at the earliest points in the development process lowers risk—both the inherent risk of introducing security vulnerabilities in production and the likelihood of threats to the delivery schedule. In other words, continuously evaluating and addressing security issues throughout development makes it far less likely that you'll discover significant security vulnerabilities (the kind that can upend a release schedule) as critical delivery deadlines approach.

That's quite a few AppSec considerations to keep in mind. Let's briefly look at AST in the context of MAD.

Application Security Testing Challenges in MAD

MAD techniques and tooling give developers a huge amount of flexibility and control over their own destinies. Yet even with all these advancements, the same core problems applications have always faced (like security) are becoming even more critical to address. Since microservices architecture has become one of the most widely used development models, the number of attack surfaces in MAD has increased exponentially. In addition, many of these services are containerized, and they rely heavily on open source projects for their foundational pieces, as does the IaC that deploys them.

With all these new attack vectors, the risks have never been greater. They begin with longer supply chains that fall outside the development team's sphere of control. These supply chains include a seemingly endless list of open source libraries and frameworks, and they're used extensively in possibly thousands of APIs to underpin critical web and mobile applications.

This is where concepts like AppSec, DevSecOps, and SecOps come into the mix. While distinct, the teams all share the same goal: identify and reduce risks across their organization's application portfolio and supporting infrastructure. These teams leverage MAD processes to introduce modern AST solutions into their SCMs and CI/CD pipelines. Simply put, modern application development needs modern security solutions, and that means modern AST.

AST solutions aren't just being integrated into pipelines along with the shift-left mentality; they're also being placed as close as possible to the developers. The earlier in your application's life cycle you can detect a bug, defect, or vulnerability, the faster you can fix it. Ideally, your product will have been fully scanned multiple times and in multiple ways by the time it's ready for the final sign-off before production.

To wrap things up, let's review some tips you should consider around AST in MAD.

Seven AST Tips to Consider in MAD

When you're looking for AST solutions that fit well into your MAD initiatives, you need to be sure you're enabling digital transformation, not hindering it. A modern AST solution must:

Be built for cloud development and fully understand modern tech stacks, architectures, processes, and vulnerabilities.

Provide one-click cloud-based scans for a single stakeholder using one process, from one platform, with no installations and no scanning servers required.

Deliver automated scans with various scan engines and correlate results across your codebase, providing complete and accurate results through a platform-like approach.

Allow flexible deployments, with identical capacities whether used on-premises, in the cloud, or in hybrid environments.

Be capable of integrating with developer workflows, automating scans across the SDLC, and correlating results to pinpoint risks using unified dashboards and broad reporting capabilities.

Include SAST, SCA, container security, IaC security, API security, supply chain security, an orchestration layer, a correlation layer, advanced reporting, and developer training.

Come with standard support, options for premium support, unlimited scans, concurrent scans, incremental scans, customization, plugins, and add-ons.

Conclusion

We've worked to fill this three-part series with nearly everything you need to know to approach modern application development with confidence, skill, and know-how. We hope you'll share it with other leaders, AppSec professionals, and developers in your organization and elsewhere.

One thing's for sure: With the enormous migration to newer software development and deployment approaches, AppSec is a moving target. From an evolution to almost 100% cloud native, tons of microservices, and vast numbers of APIs to the tremendous consumption of open source, containers galore, and IaC in use everywhere possible, MAD isn't going anywhere but up.

About Checkmarx

Checkmarx is constantly pushing the boundaries of Application Security Testing to make security seamless and simple for the world's developers while giving CISOs the confidence and control they need. As the AppSec testing leader, we provide the industry's most comprehensive solutions, giving development and security teams unparalleled accuracy, coverage, visibility, and guidance to reduce risk across all components of modern software – including proprietary code, open source, APIs, and Infrastructure as Code. Over 1,600 customers, including half of the Fortune 50, trust our security technology, expert research, and global services to securely optimize development, at both speed and scale. For more information, visit our website, check out our blog, or follow us on LinkedIn, Twitter, YouTube, and Facebook.

Checkmarx

Checkmarx at a Glance

1,675+
Customers in 70 countries

750
Employees in 25 countries

45%
of the Fortune 50 are customers

30+
Languages & frameworks

500k+
KICS downloads in 2021

The world runs on code. We secure it.