Checkmarx

The world runs on code. We secure it.

AppSec-Überlegungen zu Modern Application Development

Leitfaden für Führungskräfte und Experten aus der Praxis – Teil 3

Inhaltsverzeichnis

Einleitung

In Teil 1 dieser Reihe haben wir uns mit den Facetten von Modern Application Development (MAD) befasst, einschließlich der zu erwartenden Vorteile und Herausforderungen. In Teil 2 haben wir uns mit den Sicherheitsrisiken befasst, die bei MAD auftreten, und wie sie sich auf jedes Unternehmen auswirken können. Wenn Sie Teil 1 und 2 gelesen haben, sollten Sie eine solide Vorstellung davon haben, was MAD mit sich bringt und welche Sicherheitsrisiken Sie im Auge behalten sollten.

In Teil 3 werden wir uns mit Überlegungen zu moderner Application Security (AppSec) befassen, mit denen Führungskräfte und Experten aus der Praxis vertraut sein sollten. Da MAD die Softwareentwicklung nachhaltig prägt, müssen sich auch die AppSec-Methoden weiterentwickeln. Abschließend geben wir Ihnen einige Tipps, die Sie bei der Planung und Ausrichtung der AppSec-Initiativen Ihres Unternehmens auf dieses neue Paradigma berücksichtigen sollten.

Checkmarx hat es sich zum Ziel gesetzt, Software Security für Unternehmen weltweit zu verbessern, indem wir sie dabei unterstützen, ausnutzbare Schwachstellen zu reduzieren. Wir hoffen, dass diese eBook-Reihe bei der Einführung von Modern Application Development dazu beitragen kann.

Eine kurze Diskussion zum Thema Risiko

Wenn Sie sich die lange Liste der Risiken ansehen, die wir in Teil 2 vorgestellt haben, werden Sie feststellen, dass Sie einige Risiken mit besseren AppSec-Methoden, andere mit verbesserten Ansätzen für Application Security Testing (AST) und wieder andere mit höherer AppSec-Awareness und Entwickler-Trainings in den Griff bekommen können – aber kein Rezept stoppt alle Risiken. Um alle Aspekte abzudecken, brauchen Sie ein breites Set von Strategien und Lösungen.

Umgang mit unsicherem Anwendungscode im MAD

Von allen Codeelementen in einer modernen Anwendung ist der erste Typ, der einem in den Sinn kommt, der Anwendungscode: der Quellcode, der alle Funktionen enthält, die die Software ausführt. Der Anwendungscode kann in-house entwickelt oder von Dritten übernommen werden (zum Beispiel Open Source). Bei MAD ist Open Source jedoch mit einigen Vorbehalten verbunden.

Open-Source-Code

Es ist leicht zu verstehen, warum Open Source so weit verbreitet ist. Durch den Import von Open Source Libraries, Komponenten und anderen Ressourcen in Anwendungen vermeiden Entwickler, dass sie das Rad neu erfinden müssen. Sie können bereits geschriebenen Code wiederverwenden und haben so mehr Zeit, um innovative Funktionen zu programmieren, die noch nicht existieren.

Open Source hat jedoch auch Nachteile. Um Security- und Compliance-Probleme zu vermeiden, die häufig mit Open Source einhergehen, brauchen Entwickler und Security-Teams volle Transparenz über den Open-Source-Code, den sie verwenden – einschließlich der damit verbundenen Risiken. Daher (und auch mit Blick auf die jüngsten Supply-Chain-Angriffe und neue Gesetzgebung) gewinnt das Konzept einer Software Bill of Materials (SBOM) im Kontext von Open Source verstärkt an Bedeutung.

> Was eine SBOM ist und warum Sie eine brauchen

„SBOM“ ist ein aus der industriellen Fertigung übernommener Begriff. Eine „BOM“ ist eine Liste von Rohstoffen oder Bauteilen mit den jeweiligen Mengen, die für die Herstellung eines kompletten Produkts benötigt werden. Man kann sie sich als eine Liste von Zutaten vorstellen. Diese Stückliste dient als Nachweisdokument und stellt sicher, dass alle erforderlichen Teile vorhanden und in der richtigen Menge verfügbar sind. Wenn beispielsweise ein Bauteil fehlt oder sich seine Lieferung verzögert, wird dies in der Stückliste vermerkt.

In der Softwareentwicklung listet die SBOM in der Regel die Open-Source-Pakete, Libraries und Komponenten auf, die in einer Anwendung enthalten sind. Ein Beispiel: Sie haben ein node.js-Projekt, bei dem der node_modules-Ordner alle Pakete enthält, die zum Erstellen und Ausführen der Anwendung erforderlich sind.

Stellt sich heraus, dass ein Element in der SBOM Schwachstellen oder Schadcode enthält, ist Ihre Software Pipeline gefährdet und die betroffenen Elemente müssen aktualisiert oder ersetzt werden. Wenn die Anwendung Libraries von NPM, Maven Central oder einer anderen Registry importiert, können Sie diese als Open-Source-Bestandteile Ihrer Anwendung zählen. Entscheidend an dieser Stelle ist, zu verstehen, was wirklich in einer Anwendung enthalten ist, auf die Ihr Unternehmen baut.

> Warum Sie eine genaue Stückliste für Ihre Anwendungen benötigen

Wenn Sie alle Teile Ihrer Anwendungen kennen und wissen, was für deren Programmierung oder Kompilierung erforderlich ist, bekommen Sie viele Probleme und Risiken in den Griff. Wenn etwa eine neue Open-Source-Schwachstelle bekannt wird, können Sie die betroffenen Versionen des Codes mit Ihrer SBOM abgleichen. Stellen Sie Übereinstimmungen fest, können Sie die betroffenen Systeme identifizieren und Maßnahmen zur Behebung der damit verbundenen Risiken ergreifen.

Ein anderes Beispiel: Sie stellen fest, dass eine Library oder Dependency aus irgendeinem Grund aus einer Registry entfernt wurde (z. B. Lizenzänderungen, gesetzliche Anforderungen, Abschreibung, mangelnde Unterstützung durch die Community, Entscheidung des Betreibers). Sie könnten diese Komponente nicht weiterverwenden, ohne das Risiko zu erhöhen – sprich, Sie müssten eine andere Version verwenden oder eine Alternative suchen. Wenn Sie eine genaue SBOM pflegen, können Sie die Risiko-Awareness verbessern und das Risiko effektiv minimieren.

> Minimierung der Open-Source- und Supply-Chain-Risiken

Die einfachste und zuverlässigste Methode zur Risikominimierung in der Open-Source-Software-Supply-Chain besteht darin, sicherzustellen, dass Sie Open-Source-Software von ihrem Urheber beziehen und sie anhand der geposteten Hash Images validieren. In einigen Fällen kann dieser geringe Aufwand Angriffe auf die Supply Chain verhindern, bevor sie stattfinden. Allerdings ist es nicht immer möglich, die Software direkt von der Quelle zu beziehen, zumal viele Unternehmen ihre Open Source Images in Docker Hub hochladen oder sie einem zentralen Repository wie Maven hinzufügen. Um die Sache weiter zu verkomplizieren, müssen Sie sich möglicherweise auch mit privaten in-house Registries befassen.

Letztendlich gibt Ihnen ein besserer Überblick über den Open-Source-Code, von dem Ihre Anwendungen abhängen, einen klaren Überblick über die mit dessen Nutzung verbundenen Schwachstellen und Ihr Gesamtrisiko. In MAD benötigen Sie eine Lösung, die ein Inventar des gesamten verwendeten Open-Source-Codes bereitstellt und sich problemlos in CI-/Build-Server, Artifact-Server und Entwicklungsumgebungen integrieren lässt.

Software Composition Analysis (SCA) kann Ihr Unternehmen bei der Erstellung und Pflege einer SBOM unterstützen. Mit einer für MAD konzipierten SCA-Lösung können Sie die Materialliste Ihrer Anwendungen scannen, kompilieren und vollständige Transparenz über Ihre Codebasis erlangen. Sie sollte es Ihnen auch ermöglichen, ein Zero-Trust-Modell auf Open Source anzuwenden, um potenzielle Angreifer zu identifizieren, versteckte Hintertüren ausfindig zu machen und Schadcode zu erkennen – mit dem Ziel, bösartige Pakete aus dem Verkehr zu ziehen und das Risiko zu minimieren. Dazu bedarf es aber leistungsfähiger Supply-Chain-Security-Lösungen mit Verhaltens- und Link-Analyse, Machine Learning und Threat-Informationen zur Open-Source-Supply-Chain.

Werfen wir nun einen Blick auf proprietären Code.

Proprietärer Code

Als proprietären Code bezeichnet man den von Ihren Entwicklern in-house geschriebenen Code, der auch dazu dient, die notwendige Geschäftslogik bereitzustellen, um den gesamten Open-Source-Code in der Anwendung zu operationalisieren. Wie bei Open Source können auch Sicherheitslücken im proprietären Code ausgenutzt werden. Schlimmer noch: Jede Lücke, die spät im SDLC entdeckt wird, kann katastrophale Folgen haben. Eine SAST-Lösung (Static Application Security Testing), die proprietären Code auf Schwachstellen scannt, ist daher unerlässlich.

Anwendungen, die nach MAD-Prinzipien entwickelt werden, umfassen oft multiple Microservices, ein zentrales Element von MAD. Daher muss es in diesen Umgebungen möglich sein, inkrementell nach Schwachstellen auf Quellcodeebene zu suchen, ohne den Code zu kompilieren. Microservices sind nur Teile der gesamten Anwendung, und ihr Code sollte bei jeder Änderung gescannt werden. Die Herausforderung besteht darin, dass viele SAST-Lösungen für erfolgreiche Scans eine vollständig kompilierte Anwendung benötigen, und das verträgt sich nicht gut mit MAD.

> Inkrementelle Scans sind eine Voraussetzung für MAD

Klassische SAST-Scans können Stunden dauern – und wenn lückenlose Scans zu Verzögerungen führen und den Rollout der Software bremsen, ist das ein echtes Problem. Aus diesem Grund führen Unternehmen oft weniger SAST-Scans durch, oder sie führen vollständige Scans nachts durch, um die zu erwartenden Verzögerungen abzufedern. Das sind aber nur Work-Arounds. Der Schlüssel zu schnelleren Scans liegt in der Einführung einer SAST-Lösung mit inkrementellen Scan-Funktionen, insbesondere im Hinblick auf Microservices.

Ist Ihr SAST in das Source Code Management (SCM) integriert, werden automatisch inkrementelle Scans für den Zweig des Codes gestartet, an dem ein Entwickler arbeitet. Pull Requests, Push Events, Merge Requests und andere Codeänderungen lösen alle Scans aus und liefern Ergebnisse.

Wenn Sie versuchen, vorhandene Scan-Lösungen in eine MAD-Initiative zu integrieren, werden Sie feststellen, dass SAST-Lösungen, die nur vollständige Scans unterstützen, zwangsläufig Verzögerungen verursachen. Achten Sie darauf, dass Ihr SAST inkrementelle Scans auf Quellcodeebene unterstützt, und scannen Sie so früh und so oft wie möglich.

API-Code

APIs sind für moderne Anwendungen ungemein wichtig. Sie dienen heute nicht nur dazu, Daten von Drittanbietern in eine Anwendung zu importieren, sondern sind oft ein wesentlicher Bestandteil der Architektur, die moderne Anwendungen am Laufen hält. Die meisten Microservices greifen auf interne APIs zurück, um einander zu identifizieren und miteinander zu kommunizieren.

Vielleicht erinnern Sie sich noch an die Modelle zur Veranschaulichung von Molekülen aus dem Chemie-Unterricht, bei denen die Kugeln für die Atome und die Stäbchen für die Bindungen zwischen diesen Atomen stehen. In modernen Anwendungen sind die Kugeln die Microservices und die Stäbchen die APIs. Wenn Ihre Microservices aufgrund eines Problems mit den APIs nicht miteinander kommunizieren können, funktionieren Ihre Anwendungen nicht mehr.

Andere Einsatzbereiche von APIs im MAD

Ebensowenig können Sie eine containerisierte Anwendung effektiv auf einer Orchestrierungsplattform wie Kubernetes bereitstellen, ohne die vielen beweglichen Teile des Clusters mithilfe von APIs zu managen: Worker Nodes, Master Nodes, Pods, Sidecars und so weiter. Und genauso schwierig ist es, Anwendungen skalierbar in der Cloud bereitzustellen, ohne APIs zur Automatisierung des Application Managements, zum Ausbalancieren des Traffics und so weiter zu nutzen. Obwohl Cloud-Umgebungen auch anders gemanagt werden können – etwa über ein Web Interface oder CLI – ist ein API-zentrierter Ansatz in der Regel der effizienteste.

Selbst viele API-Use-Cases aus der Dotcom-Ära sind heute noch relevant. Entwickler verlassen sich nach wie vor häufig auf APIs, um Anwendungen in externe Plattformen zu integrieren und Ressourcen von Drittanbietern zu überwachen und zu sichern. In modernen nativen Cloud-Umgebungen geht die Rolle von APIs jedoch weit über Integrationen und Monitoring hinaus.

APIs erfordern eine moderne Security

Da moderne Apps bei der Kommunikation zwischen verteilten und lose gekoppelten Anwendungen und Services in hohem Maße auf APIs angewiesen sind, steht API Security ganz oben auf der Liste der AppSec-Überlegungen. Unternehmen müssen die Themen API-Discovery, API-Accountability, API-Management und API-Security mit hoher Priorität angehen.

Die Security-Risiken von APIs unterscheiden sich grundlegend von den Risiken von Webanwendungen. Daher bedarf es dedizierter Lösungen, die interne und externe APIs identifizieren, sichere und unsichere APIs ermitteln, Shadow-APIs entdecken und fehlende oder schwache Authentifizierung und Autorisierung erkennen. Außerdem müssen Sie in der Lage sein, East-West Traffics und Service Flows zu visualisieren, um API-Beziehungen zwischen Services darzustellen und die Risiken durch vernetzte APIs zu bewerten. Kein Wunder also, dass die API-Security bei MAD-Projekten immer öfter ganz oben auf der Prioritätenliste steht.

Werfen wir als Nächstes einen Blick auf Containercode.

Umgang mit unsicherem Containercode im MAD

Es gibt viele Gründe für den Einsatz von Containern: Sie sind flexibler und effizienter als virtuelle Maschinen (VMs), flexibler und sicherer als Anwendungen, die direkt auf dem Betriebssystem laufen, und lassen sich mithilfe von Plattformen wie Kubernetes leicht skaliert orchestrieren.

Allerdings gehen Container auch mit Sicherheitsrisiken einher. Die Vorteile überwiegen in der Regel die Risiken, aber es ist wichtig, die Probleme, die Container verursachen können, zu kennen und zu adressieren. Um die Risiken von Containern zu reduzieren, müssen Sie wissen, woher die Images stammen, was sie enthalten und welche Probleme bei der Ausführung auftreten können.

Zusätzlich muss Ihre Lösung Registry-Scans gemäß der Vorgaben der Open Container Initiative (OCI), Amazon ECR, Private Registries und OpenShift unterstützen. Und: Container-Scanning-Lösungen sollten Exploitable Path Information, handlungsrelevante Hinweise für die Behebung von Schwachstellen und Integrationen in IaC-Testing-Lösungen bieten.

Lassen Sie uns dieses Thema als Nächstes betrachten.

Umgang mit Infrastructure-as-Code-Risiken im MAD

Entwickler und DevOps-Teams verwenden IaC-Tools, um gängige Infrastrukturkomponenten in einer Konfigurationssprache zu beschreiben, die dann als Blaupause für die On-Demand-Bereitstellung von Infrastrukturservices dient. IaC gibt Entwicklern und Ops-Teams mehr Kontrolle über den Change-Prozess und hilft, Änderungen effizienter und einheitlicher bereitzustellen. Der Versuch, IaC in der Praxis zu implementieren, ist jedoch häufig mit Risiken verbunden.

Wie Entwickler gängige IaC-Risiken erkennen und adressieren

Die meisten gängigen Risiken (die wir in Teil 2 behandelt haben) lassen sich durch die Einhaltung bewährter Security Best Practices adressieren. Hier einige Empfehlungen:

> Machen Sie sich mit den Tools vertraut

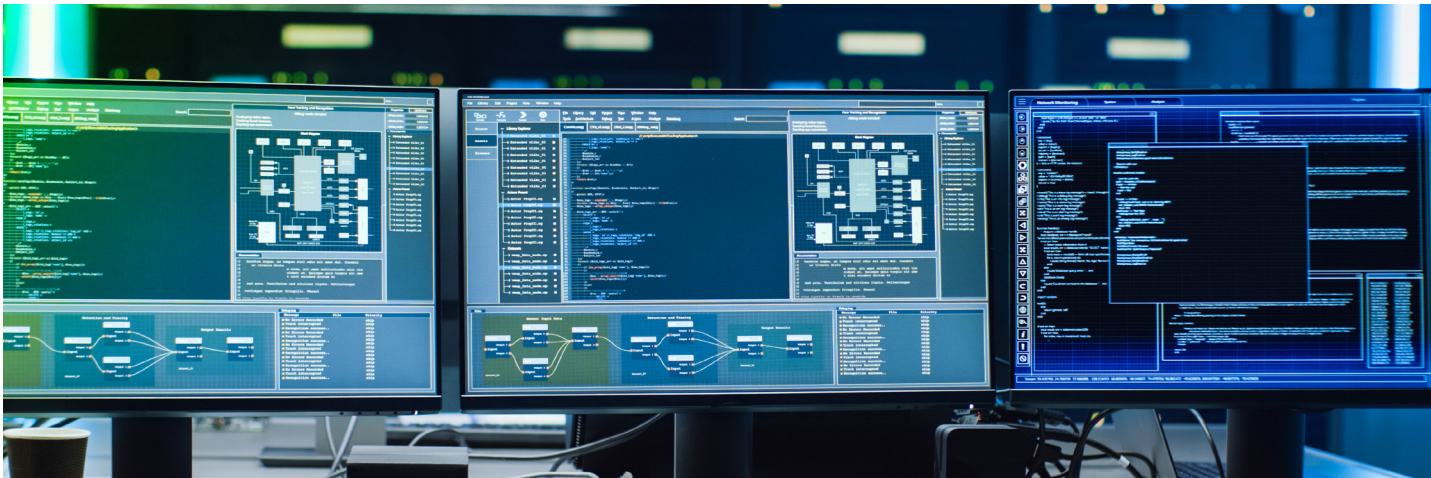
Versuchen Sie, die Funktionsweise jedes IaC-Tools vollständig zu verstehen, einschließlich seiner Eigenheiten, Probleme und Best Practices. Nehmen Sie an Meetups teil und nutzen Sie Events, um von Branchenexperten zu lernen. Das wird es Ihnen leicht machen, grundlegende und anspruchsvollere Tasks zu meistern, ohne Ihr Security-Standing zu kompromittieren.

> Entwickeln Sie gemeinsame Engineering-Prozesse und Best Practices

Führen Sie Peer Code Reviews, CI/CD-Checks, Linting und Verifikation durch. Dadurch lässt sich die Zahl der häufigsten unbeabsichtigten Zwischenfälle und Fehler reduzieren.

> Integrieren Sie dedizierte Security-Tools

Implementieren Sie für Ihre IaC-Umgebung zusätzliche Security-Tools, um eine sichere und effiziente IaC-Pipeline aufzubauen. So profitieren Sie von einer skalierbaren, für multiple Cloud-Provider optimierten Lösung, die Sie passgenau gemäß Ihrer Securit-Policy konfigurieren können.



Awareness- und AppSec-Trainings für Entwickler in MAD-Umgebungen

Unternehmen setzen auf immer kürzere Entwicklungszyklen, häufigere Releases und immer komplexere Anwendungsarchitekturen. Daher ist es wichtiger denn je, Application Security in allen Phasen des DLC zu verankern und Secure Coding Skills zur obersten Priorität zu machen.

Dafür muss sich die Denkweise der Development-Teams ändern: Diejenigen, die Ihre Software entwickeln, sollten mehr Verantwortung für die Sicherheit des Endprodukts übernehmen. Anstatt sich auf umfangreiches Post-Development-Testing zu verlassen, um Sicherheitslücken aufzuspüren und provisorisch zu flicken, müssen Teams einen „Shift-Left"-Ansatz verfolgen und Security Skills dann trainieren, wenn die Entwickler sie am dringendsten brauchen: während sie Code schreiben.

Effektive Programme für Awareness- und AppSec-Trainings sollten sich alle Vorteile moderner Schulungstechnologien zunutze machen. Ähnlich wie eine attraktive Mobile App das Verhalten der User beeinflusst, können Gaming-Prinzipien und technologiegetriebene Abläufe die Grundlage für effiziente Secure Coding Practices bilden – und dazu beitragen, User langfristig zu binden.

Gamification, die Anwendung von Game-Design-Elementen und -Prinzipien in neuem Kontext, bietet bekanntermaßen viele Vorteile. Doch die meisten Lösungen für Awareness- und AppSec-Trainings machen sich diese nicht zunutze. Gamification-Elemente können in unteschiedlichsten Phasen des Programms integriert werden: von simulierten Attacker-vs.-Defender-Szenarien bis hin zum Freischalten von „Trophäen".

Ein konsequenter Shift-Left-Ansatz für AppSec im MAD setzt voraus, dass sich Entwickler in allen Phasen der Sicherheitsrisiken bewusst werden und bei der Entwicklung bewährte Secure Coding Practices einhalten. Machbar in der Theorie – aber um einen solchen Kulturwandel herbeizuführen, muss das Development-Team mitziehen. Eine Möglichkeit, um das zu bewerkstelligen, ist es, ausgewählte Entwickler zu Security Champions zu machen.

Die Rolle des Security Champions

Security Champions sind im MAD motivierte Entwickler, die daran interessiert sind, kontinuierlich Best Practices für das Secure Coding zu entwickeln und zu implementieren. Dafür müssen sie bereit sein, in ihrem Team und im gesamten Unternehmen die Rolle eines Influencers zu übernehmen.

Security Champions sind auch für das Security-Team eine wertvolle Ressource, da sie ein tieferes Verständnis für die Prozesse in der Entwicklung ermöglichen. Sie sollten zudem bereit sein, andere Entwickler bei sicherheitsrelevanten Fragen zu unterstützen, ihnen Secure Coding Practices zu vermitteln und sie bei der Entwicklung sicherer Anwendungen anzuleiten. Auf Entscheiderebene sind die Security Champions an der Definition unternehmensweiter Standards und Policies beteiligt und schaffen stärkere Awareness für das Thema Security im Unternehmen.

> Die Vorteile eines Security Champions

Entwickler, die die wichtige Rolle als Mentor und Evangelist annehmen, können mit der zusätzlichen Verantwortung ihre Karriere voranbringen. Wie jede Qualifizierung kann sie Anerkennung auf Führungsebene einbringen, und damit ein Türöffner beim weiteren Aufstieg werden. Security Champions positionieren sich als Experten für sichere Entwicklungsprozesse – und machen sich damit für das Unternehmen unentbehrlich.

Neben diesen Karrierevorteilen kann auch das Mentoring anderer Entwickler lohnend sein: Wer stets mit gutem Beispiel vorangeht und Kollegen in Fragen der sicheren Entwicklung unterstützt, verdient sich den Respekt des Teams und trägt gleichzeitig dazu bei, auf Entwicklerebene ein Mindset der gemeinsamen Verantwortung für Security zu etablieren. Dies kann einen positiven kulturellen Wandel anstoßen, der die Entwicklung von sichereren Endprodukten ermöglicht und Ihrem Unternehmen hilft, weniger isolierte Arbeitsumgebungen zu schaffen, in der Security- und Development-Teams besser zusammenarbeiten.

> Die Herausforderung des kulturellen Wandels im MAD

Niemand wird den Stellenwert von AppSec bestreiten, aber Security Champions können auf den Widerstand anderer Entwickler stoßen, die der Meinung sind, dass der Fokus auf die Security die Design- und Entwicklungsphasen verzögert. Diese Entwickler sehen vielleicht die Investitionskosten und den Overhead, die mit der Integration von Security in ihre Prozesse einhergehen – und sind der Meinung, dass es dies dem Team schwermacht, Änderungen effizient durchzuführen. Was sie dabei übersehen, ist, dass dieser kulturelle Wandel zu erheblichen Einsparungen am Ende des Development Cycle führen wird.

Wenn Security als abteilungsübergreifende Aufgabe angesehen und zu einem frühen Zeitpunkt im Entwicklungsprozess verankert wird, lassen sich viele Risiken minimieren: etwa das Risiko, dass Sicherheitslücken in die Produktivumgebung gelangen, oder die Gefahr, dass der Terminplan nicht eingehalten werden kann. Mit anderen Worten: Mit dem kontinuierlichen Bewerten und Adressieren von Sicherheitsproblemen während des Entwicklungsprozesses sinkt die Wahrscheinlichkeit, dass Sie schwerwiegende Sicherheitslücken entdecken (die Art von Sicherheitslücken, die den Release-Zeitplan über den Haufen werfen können), wenn kritische Liefertermine näher rücken.

Das sind eine ganze Reihe von AppSec-Überlegungen, die Sie im Hinterkopf behalten sollten. Werfen wir kurz einen Blick auf AST im Kontext von MAD.

AST-Herausforderungen im MAD

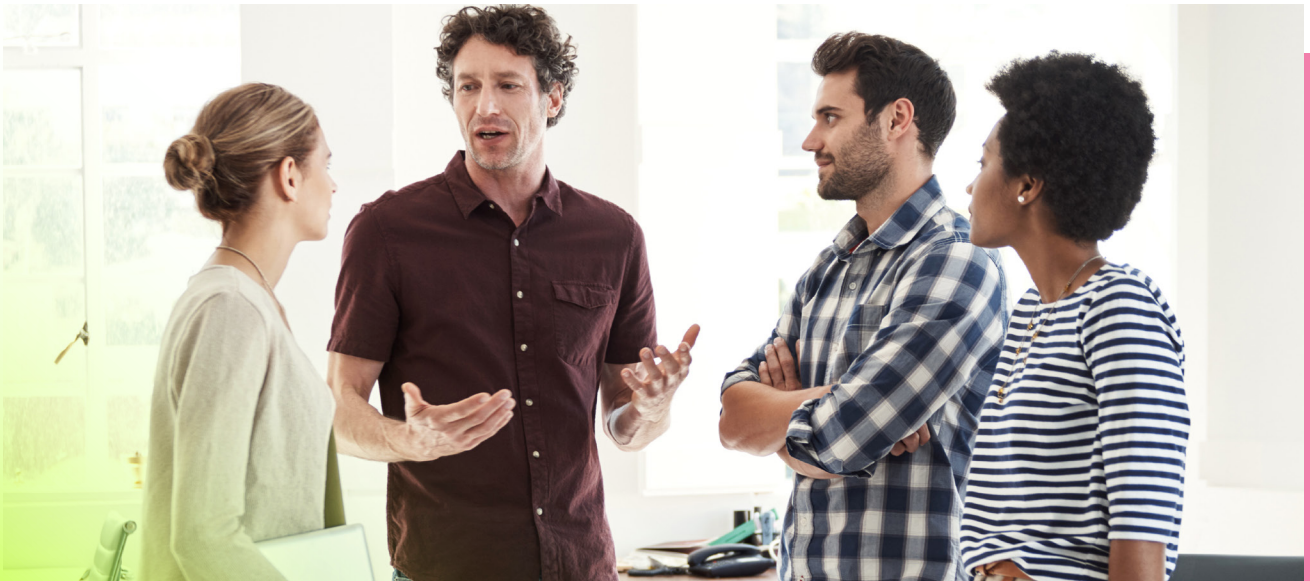
Die Techniken und Tools im MAD bieten Entwicklern ein hohes Maß an Flexibilität und Kontrolle. Doch trotz all dieser Fortschritte werden die gleichen Kernprobleme, mit denen Anwendungen schon immer konfrontiert waren (z. B. Sicherheit), immer wichtiger. Seit Microservices-Architekturen zu einem der meistgenutzten Entwicklungsmodelle geworden sind, hat sich die Angriffsfläche im MAD exponentiell vergrößert. Darüber hinaus sind viele Services containerisiert und greifen verstärkt auf Open-Source-Projekte und IaC zurück.

Angesichts dieser neuen Angriffsvektoren sind die Risiken größer denn je. Dazu gehören etwa lange Supply Chains, die außerhalb des Einflussbereichs des Development-Teams liegen – und die eine schier endlose Liste von Open Source Libraries und Frameworks umfassen, die in Tausenden von APIs für die Bereitstellung wichtiger Web- und Mobilanwendungen verwendet werden.

Hier kommen auch Konzepte wie AppSec, DevSecOps und SecOps ins Spiel. Die Teams sind zwar unterschiedlich, haben aber alle das gleiche Ziel: Risiken im Anwendungsportfolio und in der Infrastruktur des Unternehmens zu identifizieren und zu minimieren. Die Teams nutzen MAD-Prozesse, um moderne AST-Lösungen in ihre SCMs und CI/CD-Pipelines zu integrieren. Einfach ausgedrückt: MAD braucht moderne Security – das heißt: modernes AST.

Diese AST-Lösungen werden mit konsequenter Shift-Left-Mentalität in die Pipelines integriert und so nah wie möglich bei den Entwicklern platziert. Je früher im Life Cycle der Anwendung ein Bug, ein Defect oder eine Schwachstelle erkannt wird, desto schneller können Sie sie beheben. Im Idealfall ist Ihr Produkt bereits mehrfach und mit verschiedenen Verfahren vollständig gescannt worden, bevor es für die endgültige Freigabe vor der Produktion bereit ist.

Lassen Sie uns noch einige Tipps rund um AST im MAD betrachten, die Sie beachten sollten.



Sieben AST-Tipps für MAD-Umgebungen

Wenn Sie nach AST-Lösungen für Ihre MAD-Umgebung suchen, müssen Sie sicher sein, dass Sie die Digitalisierung voranbringt und nicht behindert. Eine moderne AST-Lösung muss:

auf Cloud Development ausgelegt und vollständig auf moderne Tech-Stacks, Architekturen, Prozesse und Schwachstellen ausgerichtet sein.

in die Arbeitsabläufe von Entwicklern integrierbar und in der Lage sein, Scans über den SDLC zu automatisieren, Ergebnisse zu korrelieren und Risiken transpartent zu identifizieren.

auf Knopfdruck cloudbasierte Scans für einen einzelnen Stakeholder mit einem Prozess, von einer Plattform aus, ohne Installationen und Scanning-Server bieten.

SAST, SCA, Container-, IaC-, API- und Supply-Chain-Security, einen Orchestration- und Correlation-Layer, Reporting und Developer-Training unterstützen.

automatisierte Scans mit multiplen Engines durchführen, Ergebnisse entlang der Codebasis korrelieren und durch einen Plattform-Ansatz vollständige und genaue Ergebnisse liefern.

Standard-Support, Optionen für Premium-Support, unbegrenzte Scans, simultane Scans, inkrementelle Scans, Customizations, Plugins und Add-ons bieten.

flexible Deployments mit identischen Kapazitäten ermöglichen, unabhängig davon, ob sie On-Prem, in der Cloud oder in hybriden Umgebungen genutzt wird.

Fazit

Wir haben uns bemüht, diese dreiteilige Serie mit fast allem zu füllen, was Sie wissen müssen, um Modern Application Development mit Selbstvertrauen, Skill und Know-how anzugehen – und wir hoffen, dass Sie sie mit anderen Führungskräften, AppSec-Experten und Entwicklern in Ihrem Unternehmen und anderswo teilen werden.

Eines ist sicher: Mit Blick auf die rasante Einführung neuerer Softwareentwicklungs- und Bereitstellungsmodelle ist AppSec ein bewegliches Ziel. Inmitten von nahezu vollständigem Cloud Native, unzähligen Microservices, einer riesigen Anzahl an APIs, allgegenwärtiger Nutzung von Open Source, Containern in Hülle und Fülle sowie omnipräsenter IaC, geht es für MAD nur in eine Richtung: nach oben.

Über Checkmarx

Checkmarx setzt im Application Security Testing immer neue Maßstäbe, um Security für Entwickler auf der ganzen Welt einfach und intuitiv zu halten und CISOs das notwendige Vertrauen und die richtigen Werkzeuge an die Hand zu geben. Als Marktführer im Bereich AppSec-Testing entwickeln wir bedienfreundliche Lösungen, die Developern und Security-Teams höchste Zuverlässigkeit, einen breiten Leistungsumfang, lückenlose Transparenz und handlungsrelevante Hinweise für die Behebung gefährlicher Schwachstellen in allen Komponenten moderner Software bieten – sowohl im eigenen Code als auch in Open Source, APIs und Infrastructure-as-Code. Mehr als 1.600 Kunden, darunter nahezu die Hälfte der Fortune 50, verlassen sich auf unsere Security-Technologie, unsere Security Research und unsere globalen Services, um sicher, schnell und skaliert zu entwickeln. Für mehr Informationen besuchen Sie unsere Website, lesen unseren Blog oder folgen uns auf LinkedIn..

Checkmarx

Checkmarx auf einen Blick

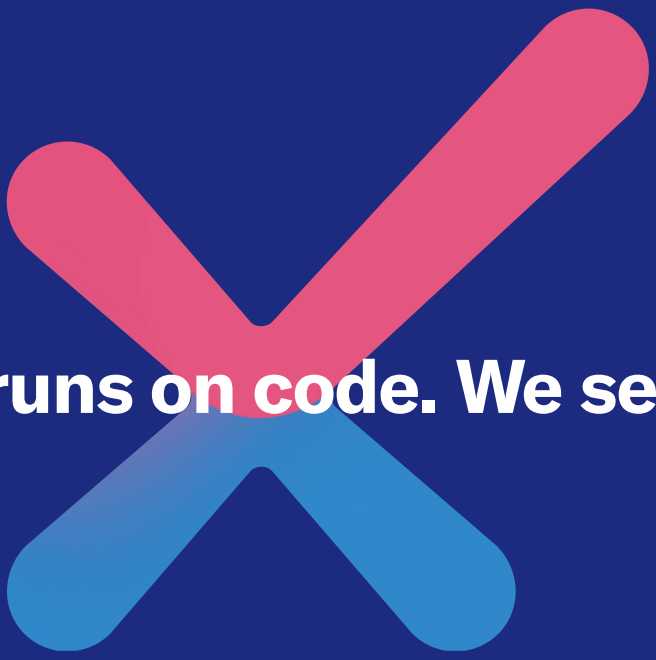
1,675+
Kunden in 70 Ländern

750
Mitarbeiter in 25 Ländern

45%
der Fortune 50 verlassen sich auf uns

30+
Sprachen & Frameworks

500k+
KICS-Downloads in 2021



The world runs on code. We secure it.