Checkmarx

The world runs on code. We secure it.

Die Risiken von Modern Application Development

Leitfaden für Führungskräfte und Experten aus der Praxis – Teil 2

Inhaltsverzeichnis

Einleitung

In Teil 1 dieses eBooks haben wir uns mit den vielen Facetten von Modern Application Development (MAD) befasst, einschließlich der Vorteile und Herausforderungen, die zu erwarten sind, wenn Unternehmen auf diesen neuen Entwicklungs- und Bereitstellungsansatz umstellen.

In Teil 2 werden wir uns eingehender mit den Sicherheitsrisiken beschäftigen, die MAD mit sich bringt. Stand heute gibt es keine andere Publikation, die diese Risiken umfassender behandelt. Entwickler und Security-Experten können das, was wir hier betrachten, als Ausgangspunkt für weitere Diskussionen, Risikoanalysen und das Risikomanagement nutzen.

Neue Risiken tauchen im MAD vor allem aufgrund der „legolisierten“ Methodik auf, die im Hinblick auf die Risiko-Nutzen-Analyse zahlreiche Variablen ins Spiel bringt. Obwohl die Vorteile von MAD messbar sind und in der Softwareentwicklungsbranche breite Akzeptanz finden, müssen Unternehmen auch die zusätzlichen Risiken anerkennen und adressieren – und das ist nur dann effektiv möglich, wenn ihnen bewusst ist, dass sie überhaupt einem Risiko ausgesetzt sind.

Deep Dive in die Security-Risiken von MAD

Unternehmen müssen heute die von OWASP, SANS und anderen Anbietern bereitgestellten Listen bekannter Softwarerisiken im Auge behalten – im Zuge von MAD-Initiativen tauchen aber auch neue Gefahren auf. Gehen wir näher auf die Risiken ein, die wir in Teil 1 dieses eBooks behandelt haben.

Risiken von Open Source

Warum Open Source so beliebt ist, ist schnell erklärt: Durch den Import von Open-Source-Libraries, Erweiterungen und anderen Ressourcen können Entwickler Code wiederverwenden, den andere schon geschrieben haben – und haben mehr Zeit für die Entwicklung innovativer, neuer Funktionen.

Open Source hat jedoch auch Schattenseiten. Um Open Source verantwortungsvoll zu nutzen und die damit oft einhergehenden Sicherheits- und Compliance-Probleme zu vermeiden, benötigen Entwickler einen lückenlosen Überblick über die von ihnen verwendete Open-Source-Software und die damit verbundenen Risiken. Hier sind die drei wichtigsten:

Risiko 1: Inkonsistente Sicherheitsstandards

Das Sicherheitsniveau von Open-Source-Code schwankt enorm. Einige Projekte – etwa der Linux Kernel – werden sehr hohen Sicherheitsstandards gerecht (und selbst da bleiben einige Schwachstellen unentdeckt). Andere auf GitHub verfügbare Tools setzen die Messlatte nicht ganz so hoch an. Open-Source-Software kann sehr sicher sein, manchmal sieht es aber auch ganz anders aus.

Entwickler müssen die Sicherheit von Open-Source-Code also von Fall zu Fall überprüfen. Auch wenn Open-Source-Befürworter gerne behaupten, dass „viele Augenpaare alle Bugs sichtbar machen“ – sprich: die Community Schwachstellen schnell entdeckt und behebt –, kann die Anzahl an Augenpaaren je nach Open-Source-Codebasis erheblich variieren. Je weniger Aufmerksamkeit ein Open-Source-Projekt erhält und je weniger Erfahrung seine Entwickler haben, desto wahrscheinlicher ist es, dass es niedrige Sicherheitsstandards aufweist.

Risiko 2: Unbekannte Herkunft des Quellcodes

Die Sicherheit von Open-Source-Code zu überprüfen, ist besonders schwierig, wenn sein Ursprung unklar ist. Sie könnten zum Beispiel einen Quellcode-Tarball auf einer Website finden, die kaum Informationen zum Entwickler liefert. Und selbst wenn ein README im Tarball den Autor benennt, haben Sie oft keine Möglichkeit, die Authentizität zu überprüfen. Vielleicht klonen Sie auch Code aus einem Git-Repository und gehen davon aus, dass der Entwickler das Repo pflegt – während der Code von jemand anderem kopiert wurde. Auch hier ist die Urheberschaft schwer zu verifizieren.

Es ist einfacher, dem Code von Dritten zu vertrauen, wenn man weiß, dass er von erfahrenen, vertrauenswürdigen Entwicklern geschrieben wurde. Trotzdem sollten Sie den Code auf Schwachstellen überprüfen. Aber wenn Sie wissen, woher er stammt, können Sie fundierter entscheiden, ob Sie ihn verwenden wollen.

Risiko 3: Lizenzverstöße

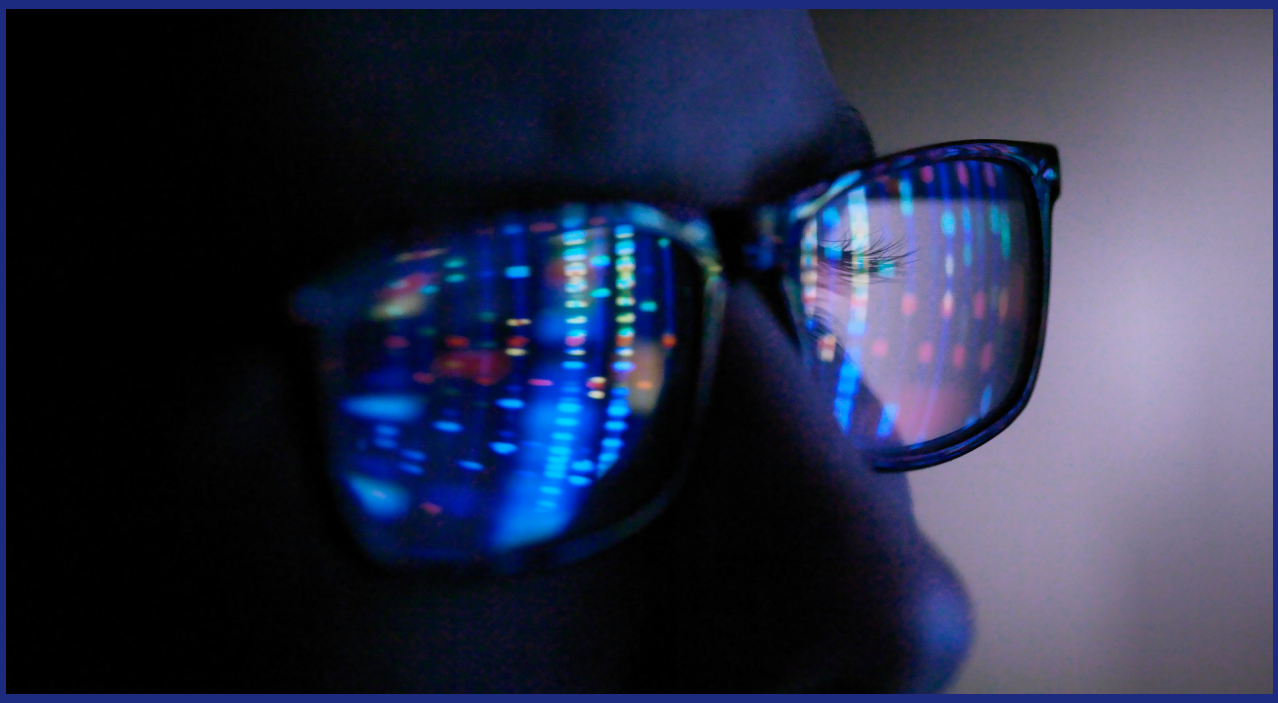
Viele Entwickler sind überzeugt, dass sie Experten für Open-Source-Lizenzen sind, aber die meisten sind es nicht. Sie misssverstehen oft die Lizenzbedingungen[1] und haben falsche Vorstellungen, etwa, dass „man GPL-Code nicht gegen Geld verkaufen darf“ oder dass man „unter der MIT-Lizenz machen kann, was man will, solange man die ursprünglichen Entwickler nennt.“

Missverständnisse wie diese bergen das Risiko von Lizenzverstößen. Wenn Entwickler die Besonderheiten der Lizenzen, die für die verschiedenen von ihnen verwendeten Open-Source-Komponenten gelten, nicht kennen, verletzen sie möglicherweise Lizenzvereinbarungen – und das setzt ihr Unternehmen dem Risiko eines möglichen Rechtsstreits aus.

Erschwerend kommt hinzu, dass Open-Source-Projekte ihre Lizenzbedingungen mitunter ändern – wie etwa Elastic im Jahr 2021. Es reicht also nicht immer aus, die Lizenzanforderungen von Open-Source-Code nur bei der ersten Verwendung zu ermitteln. Sie müssen jedes Mal, wenn der Code die Version wechselt, eine Neubewertung vornehmen.

Damit haben wir die drei größten Risiken im Zusammenhang mit der Verwendung von Open Source abgedeckt. Als Nächstes betrachten wir die Risiken im Zusammenhang mit Microservices.

[1] „Common misconceptions in licensing“, Free Software Foundation, 23. Februar 2015, https://www.fsf.org/bulletin/2014/fall/common-gpl-misconceptions.



Risiken von Microservices

Wenn Sie heute als Entwickler arbeiten, lieben Sie Microservices wahrscheinlich. Microservices-Architekturen erleichtern die Entwicklung performanter Software, da sie Anwendungen agil und resilient machen. Eine Microservices-Strategie zahlt sich aber nur aus, wenn Sie die damit verbundenen Risiken effektiv managen. Die Absicherung von Microservices birgt mitunter größere Herausforderungen als weniger komplexe monolithische Architekturen: Wenn Sie die Sicherheitsrisiken von Microservices nicht in den Griff bekommen, kann es passieren, dass Ihre Anwendung am Ende schlecht performt, weil sie kompromittiert wurde – ein Problem, das auch durch mehr Agilität nicht zu lösen ist. Hier sind die fünf häufigsten Sicherheitsrisiken bei der Entwicklung von Microservices-basierten Apps:

Risiko 1: Zunehmende Komplexität

Falls Sie jemals eine Microservices-App programmiert oder gemanagt haben, wissen Sie, dass Microservices-Architekturen die Komplexität auf eine ganz neue Ebene bringen. Das Programmieren der App ist ungleich komplexer, weil die Entwickler sicherstellen müssen, dass jeder Microservice andere Microservices effizient und zuverlässig lokalisieren und mit ihnen kommunizieren kann. Sie erschweren auch das Management, da Admins mit Service-Discoveries, verteilten Log-Daten oder Instanzen, die ständig hoch- und runterfahren, zurechtkommen müssen.

Diese Herausforderungen sind ein Sicherheitsrisiko: Je schwieriger es ist, in einer Umgebung den Überblick zu behalten, desto schwieriger ist es auch, Schwachstellen zu erkennen. Entwickler und Security-Teams benötigen daher wesentlich leistungsfähigere Tools für das Quellcode-Management und das Monitoring der Laufzeitumgebungen als bei monolithischen Apps.

Risiko 2: Begrenzte Kontrolle über die Umgebung

Je nachdem, wie Sie Microservices bereitstellen, haben Sie vielleicht nur begrenzte Kontrolle über die Laufzeitumgebung. Wenn Sie etwa serverlose Funktionen zum Hosten von Microservices verwenden, haben Sie wenig bis gar keinen Zugriff auf das Host-Betriebssystem (OS). Im Hinblick auf Monitoring, Zugriffskontrolle und andere Tools müssen Sie mit dem arbeiten, was die Serverless-Plattform bietet.

Das macht die Absicherung deutlich schwieriger, da Sie sich nicht auf Tools auf Betriebssystemebene verlassen können, um Ihre Microservices zu schützen, sie voneinander zu isolieren oder Daten zu sammeln, die auf Sicherheitslücken hinweisen. Sie selbst müssen alle Risiken innerhalb des Microservices adressieren. Das ist sicher möglich, erfordert aber viel Koordination und Zeit, die Entwickler von monolithischen Anwendungen nicht aufwenden müssen.

Risiko 3: Unzureichender Schutz von Daten

In einer monolithischen Anwendung ist die Speicherung der Daten normalerweise einfach und unkompliziert: entweder im lokalen Dateisystem des Servers, auf dem die Daten gehostet werden, oder in einem NAS-Speicher, der dem lokalen Speicher des Servers zugeordnet ist. Diese Daten lassen sich problemlos verschlüsseln und mit starken Zugriffskontrollen sichern.

Microservices verwenden für gewöhnlich eine völlig andere Storage-Architektur: Da Microservices oft über einen Cluster von Servern verteilt sind, können Sie sich nicht auf einen lokalen Speicher und Zugriffskontrollen auf Betriebssystemebene verlassen. Stattdessen verwenden Sie meist einen Scale-out-Speicher, der die Daten von den zugrunde liegenden Speichermedien abstrahiert.

Auch diese Speichersysteme können mit Zugriffskontrollen geschützt werden; diese Kontrollen sind aber oft komplexer als die Vergabe von Zugriffsrechten auf Dateisystemebene. Entwickler können also leichter Fehler machen, die zu Security-Breaches führen.

Sicherzustellen, dass jeder Microservice den richtigen Zugriff auf den Speicher hat, kann darüber hinaus so komplex sein, dass einige Entwickler – unverantwortlicher Weise – den Weg des geringsten Widerstands wählen, sprich: keine granularen Storage Policies konfigurieren und allen Microservices den Zugriff auf alle Daten erlauben.

In jedem Fall ist der Speicher dann weniger sicher als bei einer herkömmlichen monolithischen App. Um dies zu vermeiden, müssen Sie die Möglichkeiten der granularen Zugriffskontrolle in Ihren Speichersystemen voll ausschöpfen und konsequent nach Fehlkonfigurationen suchen.

Risiko 4: Unzureichender Schutz des Netzwerks

Der Schutz des Netzwerks ist für vernetzte Anwendungen – und das trifft heutzutage auf praktisch jede Anwendung zu – von entscheidender Bedeutung. Bei Microservices ist die Netzwerksicherheit jedoch wesentlich komplexer: Microservices kommunizieren nicht nur mit Usern oder Ressourcen von Drittanbietern über das Internet, wie es bei einem Monolithen der Fall wäre. Sie greifen in der Regel auch auf Overlay-Netzwerke zurück, um Informationen untereinander auszutauschen.

Mehr Netzwerke bieten Angreifern mehr Ansatzpunkte, um Schwachstellen zu finden und auszunutzen. Sie können sensible Daten abfangen, die Microservices austauschen, interne Netzwerke nutzen, um Breaches von einem Microservice auf andere zu eskalieren und mehr.

Nachdem wir nun die häufigsten Risiken im Zusammenhang mit Microservices betrachtet haben, wollen wir uns nun die Risiken von Containern ansehen.

Risiken von Containern

Container sind flexibler und verbrauchen weniger Ressourcen als virtuelle Maschinen (VMs). Im Vergleich zur Ausführung von Anwendungen auf dem Betriebssystem bieten sie mehr Flexibilität und Sicherheit und lassen sich mit Plattformen wie Kubernetes leicht skalieren und orchestrieren.

Sie stellen die Unternehmen aber auch vor Probleme, nicht zuletzt mit Blick auf die Security. Obwohl die Vorteile die Risiken meist überwiegen, ist es wichtig, die Herausforderungen zu kennen, die Container in Ihrem Software-Stack verursachen können – und Maßnahmen zu ergreifen, um diese zu adressieren. Hier sind die sieben gängigsten Sicherheitsrisiken.

Risiko 1: Ausführen von Containern aus unsicheren Quellen

Container sind nicht zuletzt deshalb so beliebt, weil Admins jederzeit Container-Images aus einer Public Registry abrufen und mit wenigen Befehlen bereitstellen können. Das ist flexibel und schnell, kann aber auch Probleme verursachen, wenn das Container-Image Malware enthält.

Das ist kein theoretisches Risiko: Hacker haben schadhafte Container Images auf Docker Hub (die meistgenutzte öffentliche Container Registry) hochgeladen und ihnen Namen gegeben, die Entwicklern vorgaukeln, dass sie aus vertrauenswürdigen Quellen stammen. Eine Studie von Prevasio aus dem Jahr 2020 ergab, dass 51 Prozent der Container Images auf Docker Hub mindestens eine Schwachstelle aufweisen.[2]

Die Lehre hieraus: Es ist unerlässlich, die Herkunft der Container-Images, die Sie abrufen, gegenzuchecken – insbesondere wenn Sie es mit öffentlichen Registries zu tun haben.

[2] 2 „Operation Red Kangaroo“, Prevasio, Dezember 2020, https://knowledge-base.prevasio.io/pdf.html?file=Red_Kangaroo.pdf

Risiko 2: Unbekannte Herkunft des Quellcodes

Risiken im Zusammenhang mit Container Registries gibt es auch in umgekehrter Richtung: Sie laden Daten in eine Private Registry hoch, von der Sie annehmen, sie sei sicher, nur um dann festzustellen, dass sie – inkusive der dort gespeicherten sensiblen Daten – für die ganze Welt zugänglich ist.

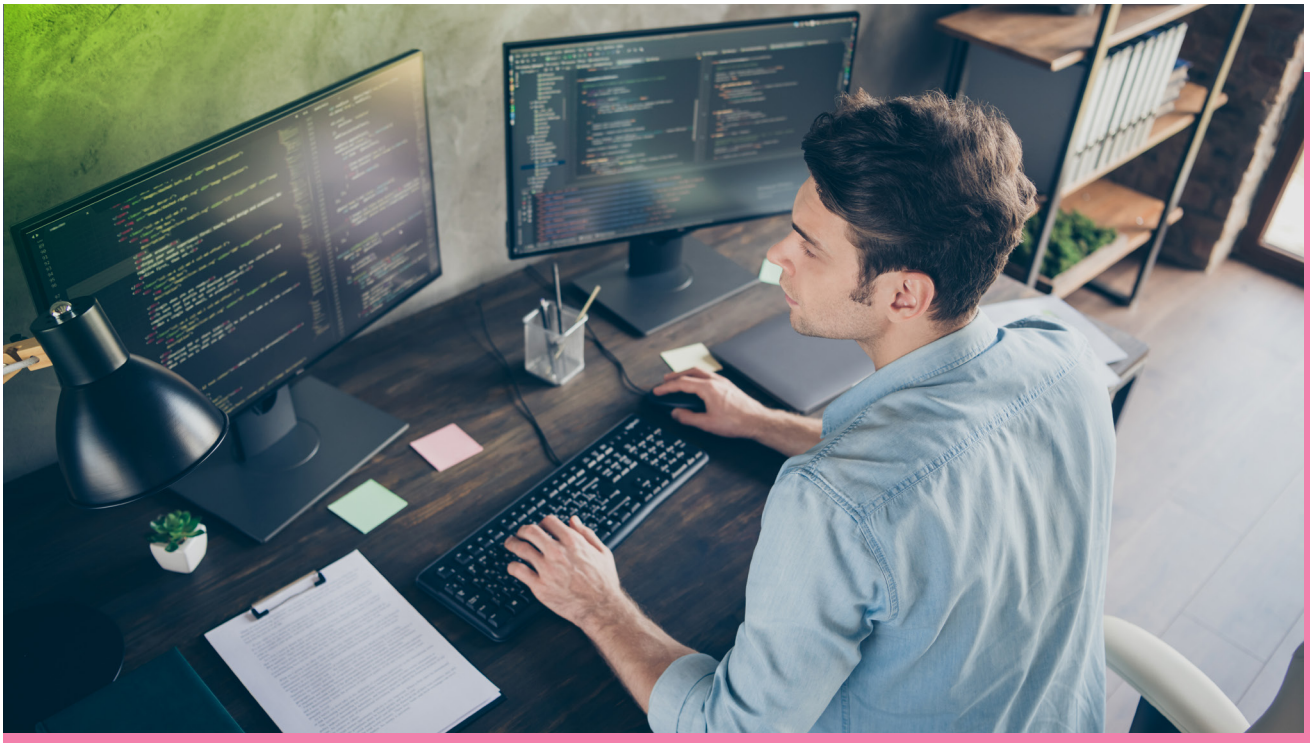
So geschehen bei Vine im Jahr 2016.[3] Das Unternehmen lud ein Container-Image mit dem Quellcode seiner gesamten Plattform in eine unsichere Registry hoch. Die URL der Registry war nicht öffentlich, aber jeder, der sie erraten konnte, hatte uneingeschränkten Zugriff auf die Images in der Registry.

Fehler wie dieser passieren schnell. Wenn man mit Dutzenden oder Hunderten von Container-Images jongliert, kann man leicht aus Versehen ein sensibles Image in einer ungesicherten Registry ablegen oder sogar vergessen, dass ein Image überhaupt sensible Daten enthält.

Risiko 3: Zu viel Vertrauen in Image-Scanner

Image-Scanner sind ein wertvolles Tool, um automatisch festzustellen, ob Container bekannte Schwachstellen enthalten – bieten aber keine Garantien. Da sie den Inhalt von Container-Images mit Listen bekannter Schwachstellen abgleichen, werden sie keine Sicherheitslücken entdecken, die noch nicht öffentlich bekannt sind. Scanner können Schwachstellen auch übersehen, wenn Container-Images ungewöhnlich strukturiert sind oder ihr Inhalt nicht so gelabelt ist, wie es der Scanner erwartet.

[3] „Hacker Downloaded Vine's Entire Source Code. Here's How...", The Hacker News, 23. Juli 2016, https://thehackernews.com/2016/07/vine-source-code.html



Risiko 4: Größere Angriffsfläche

Container erfordern mehr Tools und Software-Ebenen als eine herkömmliche Anwendung. Das bedeutet auch, dass die Angriffsfläche größer wird.

Wenn Sie einen Container bereitstellen, sollten Sie sich nicht nur um die Sicherheit der Anwendung und des Betriebssystems kümmern, das sie hostet, sondern auch um die Container-Laufzeit, den Orchestrator und möglicherweise auch um die Plug-ins, die der Orchestrator verwendet, um das Netzwerk und den Speicher zu managen. Wenn Sie „Sidecar“-Container zur Unterstützung von Tasks wie dem Logging einsetzen, können auch diese ein Sicherheitsrisiko darstellen.

All das ist lösbar. Es erfordert aber größere Investitionen in die Sicherheit und eine breitere Palette an Security-Tools als ein herkömmlicher, nicht containerisierter Anwendungs-Stack.

Risiko 5: Aufgeblähte Base Images

Base Images dienen Entwicklern als Grundlage für die Erstellung eigener Images. Meist umfasst ein Base Image eine Art Betriebssystem sowie alle gängigen Libraries und anderen Ressourcen, die für die Ausführung der von Ihnen bereitgestellten Anwendungen erforderlich sind.

Es kann verlockend sein, mehr als nur das Nötigste in die Base Images zu packen. Sie wissen nie, was Sie in Zukunft für Ihre Anwendungen benötigen. Also entschließen Sie sich vielleicht, Libraries hinzuzufügen, die für Ihre Anwendungen heute nicht unbedingt erforderlich sind.

Je größer die Base Images werden, desto größer ist auch das Risiko von Schwachstellen für Ihre Container oder Anwendungen. Im besten Fall beschränken Sie sich bei den Base Images daher auf das absolute Minimum – auch wenn das bedeutet, dass sie öfter aktualisiert werden müssen oder verschiedene Base Images für verschiedene Anwendungen verwendet werden.

Risiko 6: Mangelnde Isolation

Container sollten Anwendungen auf Prozessebene isolieren, doch in der Praxis ist das oft nicht der Fall. Da Container denselben Kernel nutzen, können Bugs in der Laufzeitumgebung oder Fehlkonfigurationen einem Container-Prozess dazu führen, dass Dritte auf Ressourcen in anderen Containern zugreifen oder sogar Root-Zugriff auf den Host erhalten können.

Daher ist es besonders wichtig, neben dem Monitoring der Laufzeitumgebung auch Ihre Konfigurationen auf Sicherheit zu überprüfen. Bei Containern ist das Risiko von Privilege Escalation und ähnlichen Problemen deutlich größer als bei VMs.

Risiko 7: Fehlende Transparenz

Je schwieriger es ist, eine Umgebung zu überwachen, desto schwieriger ist es, sie zu schützen – und bei Containern sind Transparenz und Monitoring besonders problematisch. Es ist nicht so, dass die Daten, die Sie für das Tracking von Containern benötigen, nicht vorhanden wären. Diese Daten sind aber oft über mehrere Lokationen verteilt: Sie befinden sich in Containern, auf Kubernetes Worker Nodes und Master Nodes – und sind auch nicht immer persistent: Wenn Sie zum Beispiel Ihre Container-Logs nicht rechtzeitig verschieben, sind diese für immer verloren, sobald die Container-Instanz heruntergefahren wird.

Diese Herausforderungen sind lösbar, aber sie erfordern im Hinblick auf das Monitoring ihrer Umgebung eine umfassendere Strategie als bei einem einfacheren Anwendungs-Stack.

Nachdem wir nun einige der gängigen Risiken im Zusammenhang mit Containern behandelt haben, wollen wir uns nun dem Thema Infrastructure-as-Code (IaC) zuwenden.

Risiken von IaC

Der Begriff IaC (Infrastructure-as-Code) bezeichnet Softwarelösungen, mit denen Entwickler und DevOps-Teams gängige Infrastrukturkomponenten wie Server, Virtual Private Clouds, IP-Adressen und VMs in einer aus Codezeilen bestehenden Konfigurationssprache beschreiben. Nach der Bereitstellung dient diese Konfiguration als Blaupause für die Provisionierung tatsächlicher Infrastruktur-Services On-Demand.

Mit IaC profitieren Sie von einer besseren Kontrolle über den Änderungsprozess, einer höheren Effizienz und mehr Konsistenz bei der Bereitstellung von Änderungen. Der Teufel steckt jedoch im Detail, und der Versuch, diese Tools in der Praxis zu implementieren, birgt oft neue Risiken.

Risiko 1: Steile Lernkurve

Wer IaC-Tools verwendet, ohne sich mit ihren Eigenheiten und Einschränkungen vertraut zu machen, riskiert ernsthafte Probleme. Einige der meistgenutzten IaC-Tools wie Terraform und AWS CloudFormation sind auf den ersten Blick unglaublich einfach zu bedienen – werden aber mit der Zeit und mit neuen Anforderungen viel komplexer. Versucht man zum Beispiel, in CloudFormation benutzerdefinierte Ressourcen zu verwenden, muss man genau wissen, wie das Tool funktioniert. Wenn Sie sich nicht mit den Details befassen, gehen Sie ein hohes Risiko ein, etwa mit Blick auf Configuration Drifts, bei denen die falschen Infrastrukturkomponenten gepusht werden oder das System unvollständig ausgeliefert wird.

Terraform-Importe sind ein weiteres Beispiel, das einen tiefen Einblick in die Funktionsweise von Terraform erfordert. Die offizielle Seite warnt: „Wenn Sie das gleiche Objekt mehrmals importieren, kann Terraform ein unerwünschtes Verhalten zeigen.“[4] Um dieses „unerwünschte Verhalten“ wirklich zu verstehen, müssen Sie sich jedoch die Details genau ansehen:

- **Was bedeutet es wirklich?**
- **Verliere ich die lokale TFSTATE-Datei?**
- **Kommt es zu einer Configuration Drift?**
- **Wie verhindert man, dass das gleiche Objekt mehrfach importiert wird?**

Diese Fragen zu beantworten, erfordert Zeit und Comittment. Als Entwickler sollten Sie diese Zeit für den Lernprozess einplanen, wenn Sie es für Ihre Projekte verwenden wollen.

[4] „Import Usage“, Terraform, Zugriff am 16. September 2021, https://www.terraform.io/docs/cli/import/usage.html.

Risiko 2: Menschliches Versagen

Bei einigen IaC-Tools müssen Sie eine bestimmte Abfolge von Phasen oder Schritten in einer vorgegebenen Reihenfolge ausführen. In Terraform gibt es zum Beispiel folgende Phasen:

> Die Phase **terraform plan,** in der ein Ausführungsplan dafür erstellt wird, was Terraform mit der Infrastruktur machen soll.

> Die Phase **terraform apply,** in der der Terraform-Plan umgesetzt wird.

Wenn Sie den Plan vor der Anwendung nicht überprüfen, kann dies zu destruktiven Änderungen führen. Bei der Sichtkontrolle haben Sie noch einmal die Chance, die Änderungen zu überprüfen. Diese Reihenfolge muss nicht zwingend eingehalten werden – Änderungen direkt anzuwenden und den ersten Schritt zu überspringen, kann allerdings zu Problemen führen.

Übersehen Sie eine destruktive Änderung, können die Folgen gravierend sein. Wir empfehlen daher, automatisierte Schutzmaßnahmen für die Durchführung von Änderungen an der Infrastruktur einzurichten. Anstatt beispielsweise Terraform-Plan-Outputs manuell zu überprüfen (was langwierig sein kann), sollten Sie in Pull-Request-Tools wie Atlantis investieren, um unbeabsichtigte destruktive Änderungen frühzeitig zu erkennen.

Risiko 3: Configuration Drift

Terraform und andere Tools helfen Ihnen, Infrastrukturkomponenten und deren Attribute zu managen – sofern dies die einzigen Tools sind, die Sie verwenden. Nehmen Sie Änderungen mit anderen Lösungen wie Chef oder Puppet vor, wird Terraform die Drifts von Ressourcen aber nicht erkennen. Für Unternehmen, die mit multiplen, nicht orchestrierten IaC-Tools arbeiten (oder in denen sich die Arbeitsbereiche überschneiden), kann dies zu einem echten Problem werden.

Drift-Scans helfen Ihnen, Abweichungen zwischen Ihren IaC-Tools und der realen Infrastruktur festzustellen. Da jeder Fall anders ist, gibt es aber kaum öffentliche oder private Services, die dies fehlerfrei tun. Um Configuration Drift erfolgreich zu managen, solten Sie dafür sorgen, dass:

> es keine Überschneidungen zwischen Ihren IaC-Tools gibt
> Infrastrukturänderungen mit einem einzigen IaC-Tool überwacht werden
> Reporting und Health Check integriert sind (externe Checks können bei der Verifizierung helfen)

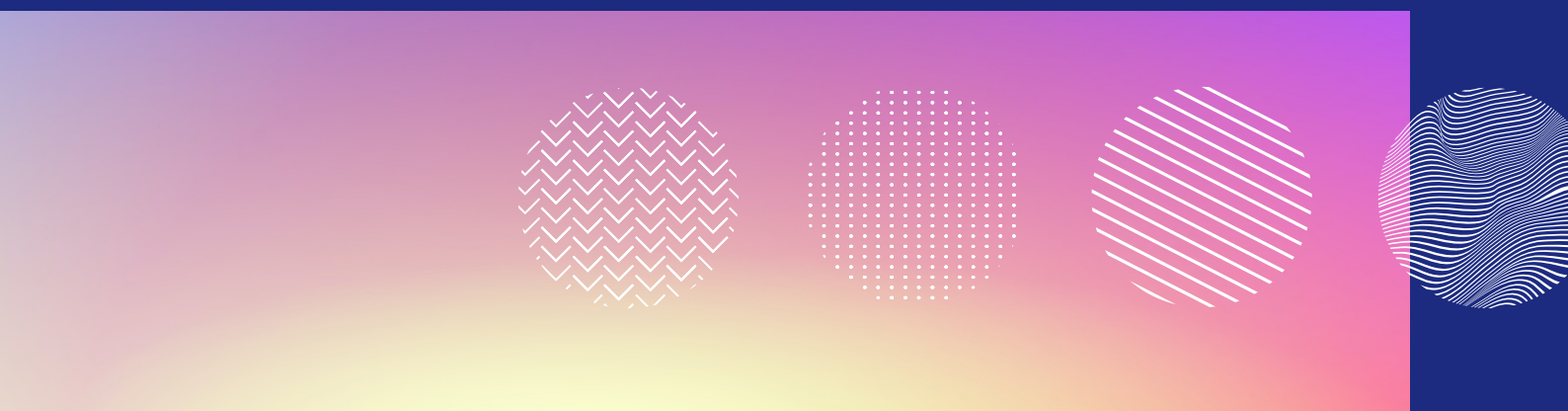
Risiko 4: Exponierte sensible Daten und Ports

Obwohl Terraform und AWS CloudFormation Infrastrukturkomponenten bereitstellen können, müssen Sie bei der Verwendung dieser Tools einige wichtige Fragen beantworten. Zum Beispiel:

> Wie können Sie während des Prozesses feststellen, ob sensible Daten nach außen dringen?
> Woher wissen Sie, dass der S3-Bucket die Sicherheitsprofile enthält, die Sie vorgegeben haben?
> Leitet Ihr Anbieter sensible Daten an stdout weiter?
> Haben Sie unerwartet einen Port zum Internet geöffnet?

Das sind nur einige Fragen, die Sie beantworten müssen, wenn Sie das Management Ihrer Infrastruktur an IaC-Tools delegieren. Es ist leichter, versehentlich Ressourcen über das öffentliche Internet zu teilen, als das Gegenteil zu tun – es steht also eine Menge auf dem Spiel.

Sehen wir uns nun einige neue API-Risiken im Vergleich zur OWASP API Security Top 10 an.



Risiken von APIs

Das Open Web Application Security Project (OWASP) veröffentlicht regelmäßig Listen von Sicherheitsrisiken für Webanwendungen und APIs. Natürlich gibt es Überschneidungen zwischen diesen Kategorien – schließlich nutzen viele Webanwendungen heute APIs – aber OWASP bewertet die Risiken getrennt, da nicht alle WebApps APIs nutzen und nicht alle APIs in Webanwendungen verwendet werden. Hier finden Sie eine Zusammenfassung der neuesten OWASP-API-Risikoliste, die wir auch in Teil 1 dieses eBooks vorgestellt haben.

Auch wenn die OWASP API Security Top 10 ein guter Ausgangspunkt für Best Practices ist, die Entwickler bei der Arbeit mit APIs befolgen sollten, sind viele sicherheitsbewusste Entwickler und AppSec-Experten der Meinung, dass die Liste einige Schwächen aufweist, auf die wir im Folgenden näher eingehen werden. Ihr Unternehmen sollte diese Punkte berücksichtigen, wenn es sich mit API-Risiken befasst, die über die OWASP-Liste hinausgehen.

Neues Risiko 1: Third-Party APIs

Die OWASP API Security Top 10 unterscheidet nicht zwischen internen und externen APIs. Obwohl beide in technischer Hinsicht ähnlich funktionieren, sind Third-Party APIs ein zusätzliches Risiko, da Entwickler im Hinblick auf das Monitoring sowie in vielen Fällen auch das Management von Authentifizierung und Autorisierung nur begrenzte Möglichkeiten haben.

Programmieren Entwickler APIs nicht selbst, können sie nur so damit interagieren, wie diese es vorsehen. Daher sollten Entwickler bei der Verwendung von Third-Party APIs besonders vorsichtig sein. Sie müssen sie nicht meiden, sollten aber ihren Ursprung und ihre Grenzen kennen.

Neues Risiko 2: Überlegungen zur Redundanz der API-Liste

Wie viele „Top-10“-Listen ist auch die API-Risikoliste von OWASP stellenweise redundant. Das erschwert die Arbeit mit ihr und gibt Entwicklern einen falschen Eindruck von den Prioritäten: Da etwa die Punkte 2 und 5 beide die Authentifizierung und Autorisierung betreffen, könnten sie wahrscheinlich in einem Punkt zusammengefasst werden, der sicherstellt, dass APIs alle Ressourcen ordnungsgemäß authentifizieren und autorisieren.

Einige behaupten, fehlerhafte Benutzerauthentifizierung sei ein größeres Risiko als fehlerhafte Autorisierung auf Funktionsebene; andere behaupten das Gegenteil. In jedem Fall scheint der Unterschied groß genug zu sein, um zwei separate Punkte zu rechtfertigen. Die wichtigste Erkenntnis ist, dass eine zuverlässige, granulare Authentifizierung und Autorisierung für jede API-Interaktion unerlässlich ist, und dass die OWASP-Liste dies nicht so deutlich ausdrückt, wie es möglich wäre.

Neues Risiko 3: Mangelnder Fokus auf das API-Monitoring

Nur die letzten zwei Punkte auf der OWASP-Liste betreffen das Monitoring der API-Ressourcen und des API-Verhaltens. Mit Blick auf die Schlüsselrolle, die APIs beim MAD zukommt, sollte ein lückenloses Monitoring und Logging aber ganz oben auf der Liste stehen. Einfach ausgedrückt: Sie können Ihre APIs nicht schützen, wenn Sie die Risiken nicht kennen. Zugegeben: Logging und Monitoring sind eher eine Aufgabe für die IT als für die Entwickler. Vielleicht ist das der Grund, warum die OWASP ihnen in dieser Liste relativ wenig Bedeutung beimisst.

Dennoch leben wir in einer DevOps-zentrierten Welt, in der die Grenzen zwischen Entwicklern und IT-Teams fließend sind. Außerdem können Sie die API-Ressourcen und das API-Verhalten nicht tracken, wenn die APIs keine geeigneten Möglichkeiten für Logging und Monitoring bieten.

New Risk 4: API-Training in der MAD-Kultur

Diskussionen über die „Security-Kultur“ oder „Security-Trainings für Entwickler“ klingen oft oberflächlich, vor allem, wenn konkrete Tipps zum Erreichen dieser Ziele fehlen. Dennoch enthält die OWASP-Liste keine Warnung vor fehlender Transparenz oder Zusammenarbeit zwischen Entwicklern und Security-Teams, was ebenfalls ein Risiko darstellen kann – insbesondere bei MAD.

Dieses Thema könnte spezifischer und konkreter aufgenommen werden: Die Liste könnte darauf eingehen, wie wichtig es ist, API-Monitoring- und Log-Daten sowohl mit Security-Teams als auch mit Entwicklern zu teilen – das wäre ein Schritt hin zum Aufbau einer modernen Sicherheitskultur.

Die OWASP API Security Top 10 sind eine Pflichtlektüre für jeden Entwickler, dem die sichere Entwicklung und Nutzung von APIs am Herzen liegt. Aber sie stellen letztlich nur einen Ausschnitt der API-Security dar. Wichtig ist, dass Sie den Leitfaden selbst überdenken und die Empfehlungen an Ihre eigenen Anwendungen und Umgebungen anpassen.

MAD – Nächste Schritte

Wir haben uns genauer angesehen, welche Risiken beim Einsatz einiger gängiger Komponenten der modernen Anwendungsentwicklung zu erwarten sind. Obwohl jedes der präsentierten Risiken wichtig ist, erhebt diese dabei Liste keinen Anspruch auf Vollständigkeit. Im Gegenteil: Wenn sich die modernen Entwicklungsmethoden weitwerentwickeln, wird diese Liste zweifellos länger werden.

Um es zusammenzufassen: Wir haben zunächst die Risiken im Zusammenhang mit Open Source beleuchtet und sind dann zu Microservices übergegangen – zwei Technologien, denen im MAD eine Schlüsselrolle zukommt. Im Anschluss daran haben wir uns mit den Gefahren beim Einsatz von Containern, von Infrastructure-as-Code und schließlich von APIs beschäftigt.

Checkmarx-Lösungen sind von Entwicklern für Entwickler gemacht, und wir sind uns bewusst, dass jede Software Risiken birgt, unabhängig davon, wie, wann oder wo sie entwickelt wurde. Daher sollten Entwickler und Security-Experten die Risiken, die wir hier zusammengestellt haben, unbedingt im Blick behalten. Mehr Awareness trägt unmittelbar zu besseren Software Security Practices in der gesamten Branche bei.

In Teil 3 dieses eBooks werden wir uns mit AppSec-Überlegungen im Kontext von MAD befassen und aufzeigen, wie sich moderne Anwendungen bestmöglich schützen lassen.

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.