Dr. Achim D. Brucker
(a.brucker@sheffield.ac.uk)
ist ein Senior Lecturer an der Fakultät für Informatik der University of Sheffield in Großbritannien. Bis Dezember 2015 arbeitete er als Strategie und Research-Architekt im Bereich Sicherheitstest im globalen Sicherheitsteam der SAP SE, wo er unter anderem die risikobasierte Sicherheitsteststrategie der SAP ausgearbeitet hat sowie an Standardisierungsprozessen der OMG teilgenommen hat.Er arbeitet bereits seit mehr als zwei Jahrzehnten sowohl im industriellen Umfeld als auch in der akademischen Forschung im Bereich der statischen und dynamischen Testverfahren von IT-Systemen. Er hat mehr als 60 Artikel und mehrere Patente veröffentlicht.

Sicherheitstests für Secure DevOps (SecDevOps)

Eine erfolgreiche Anwendung der DevOps-Strategie erlaubt Organisationen eine deutlich schnellere Auslieferung an den Kunden. Dies ist nur durch einen hohen Grad an Testautomatisierung möglich. Dies gilt nicht nur für die Tests funktionaler Eigenschaften, sondern auch für die Sicherheitstests. Da diese meist nur halbautomatisiert oder gar vollständig manuell erfolgen, stellt die Einführung des DevOps-Modells eine große Herausforderung für die Sicherheitsexperten innerhalb der Softwareentwicklung dar. Zeitgleich bietet das Zusammenwachsen von Entwicklung und Betrieb aber auch eine neue Chance: die flexible Nutzung des Spielraumes zwischen defensiven Entwicklungstechniken sowie dem Reagieren auf Angriffe während des Betriebes. Deswegen ist die Einführung einer DevOps-Strategie für jede Softwareentwicklungsabteilung ein guter Zeitpunkt, die Sicherheitsteststrategie zu überdenken und an die neuen Anforderungen anzupassen.

Eine erfolgreiche Anwendung der DevOps-Strategie erlaubt Organisationen eine deutlich schnellere Auslieferung an den Kunden. Teilweise werden dabei, in der Ausprägung "Continuous Delivery", über 50 Deployments pro Tag erreicht. Diese hohen Auslieferungsfrequenzen sind nur mit einem hohen Grad an Testautomatisierung möglich.

Neben der Automatisierung spielt hierbei auch die Qualität der Tests eine große Rolle, da nach erfolgreichem Durchlauf der Tests ein automatisches Deployment (ohne weitere manuelle Kontrolle) angestoßen wird.

Dies gilt nicht nur für die Tests funktionaler Eigenschaften, sondern auch für die Sicherheitstests. Da diese meist nur halbautomatisiert (z. B. statische Sourcecode-Analyse, welche eine Analyse der Ergebnisse verlangt) oder gar vollständig manuell (z. B. Code Reviews oder Penetrationstests) erfolgen, stellt die Einführung des DevOps-Modells eine große Herausforderung für die Sicherheitsexperten innerhalb der Softwareentwicklung dar.

Zeitgleich bietet das Zusammenwachsen von Entwicklung und Betrieb aber auch eine neue Chance: die flexible Nutzung des Spielraumes zwischen defensiven Entwicklungstechniken, um Sicherheitsfehler von vorneherein auszuschließen, und dem Entdecken von sowie dem Reagieren auf Angriffe während des Betriebes.

Deswegen ist die Einführung einer DevOps-Strategie für jede Softwareentwick-

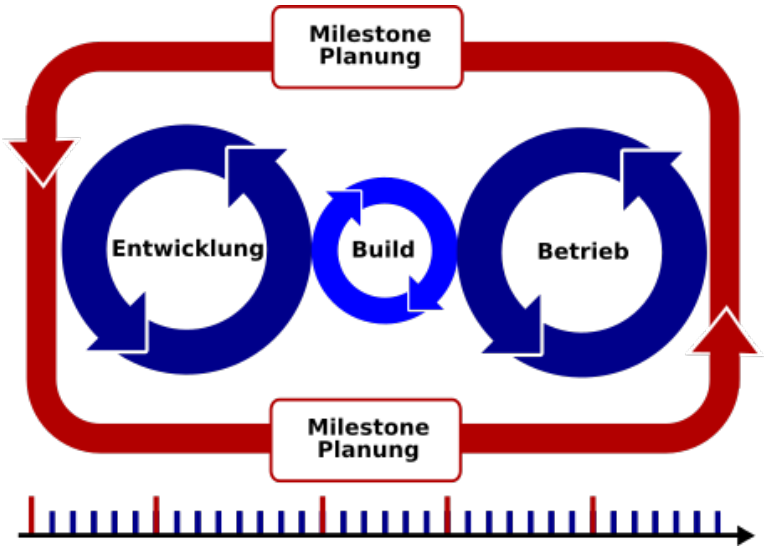

Abb.: Der DevOps-Entwicklungs- und Betriebs-Lifecycle

lungsabteilung ein guter Zeitpunkt, die Sicherheitsteststrategie zu überdenken und an die neuen Anforderungen anzupassen.

Der DevOps-Entwicklungs- und Betriebsprozess

Traditionell sind die Sicherheitstest in einem Secure Development Lifecycle (SDL), wie in dem als Referenz geltenden SDL von Microsoft [MSDF, MSDL], in zwei Bereichen der Softwareentwicklung angesiedelt:

1) Techniken, wie statische Code-Analysen, welche die defensive Programmierung unterstützen und somit helfen, Sicherheitslücken in der Entwicklung zu vermeiden.
2) Techniken, wie der manuelle Penetrationstest, welche am Ende der Entwicklung dazu dienen, sicherzustellen, dass keine Sicherheitslücken im fertigen Produkt vorhanden sind.

Diese Einteilung ist in der Regel nicht adäquat für die schnellen Releasezyklen der DevOps-Entwicklungsstrategie. Generell ist es nicht einfach, Sicherheitstestwerkzeuge welche, wie z. B. die statische Codeanalyse, sehr erfolgreich in Entwicklungsstrategien mit längeren Releaseszyklen eingesetzt werden [BaS14], für den Einsatz in der DevOps-Welt „fit zu machen".

Ein wichtiges Ziel der DevOps-Strategie ist die Überwindung der Kluft zwischen der Entwicklung und dem Betrieb einer Anwendung. Ein weiteres wichtiges Ziel ist es, neue Releases in rascher Abfolge dem Kunden zur Verfügung zu stellen.

Da weder alle Aktivitäten der Softwareentwicklung vollautomatisch geprüft und freigegeben werden können und zudem sich auch nicht alle Aktivitäten in so kleine „Häppchen" aufteilen lassen, wie sie zumindest die Continuous Delivery-Strategie erfordert, ist es zumindest konzeptionell hilfreich, zwischen *normalen Releases* und *Milestone-Releases* zu unterscheiden. Abstrakt kann dies durch das in der Abbildung gezeigte Prozessmodell dargestellt werden.

Hierbei beschreiben die inneren Kreise den schnellen Entwicklungs-, Release- und Betriebszyklus, wie er für DevOps bzw. Continuous Delivery typisch ist. Der äußere Kreis lässt einen längeren Planungs- und Entwicklungszyklus zu, der es erlaubt, gesammeltes Feedback aus dem Betrieb gezielt und planbar in den Prozess einzubringen. Zudem erlaubt die Aufteilung in Releases und Milestone-Releases gewisse Tätigkeiten nur bei jedem Milestone-Release und somit in regelmäßigen und größeren Zeitabständen durchzuführen.

Anforderungen an Sicherheitstestwerkzeuge

Damit eine erfolgreiche Sicherheitsstrategie werkzeugunterstützt umgesetzt werden kann, müssen die traditionell eingesetzten Werkzeuge, bzw. deren Konfigurationen, angepasst sowie mit neuen Methoden ergänzt werden. Damit die Aktivitäten zur Sicherstellung der Produktsicherheit nicht als Hindernis bei der Anwendung der DevOps-Strategie wahrgenommen werden und auch die Sicherheitsaktivitäten von den Vorteilen profitieren, müssen Sicherheitstests zu verschiedenen Zeiten innerhalb der DevOps-Strategie eingesetzt werden:

- **Entwicklung:** Um Sicherheitsschwachstellen von Beginn an zu vermeiden, sollten Entwicklern (statische) Werkzeuge an die Hand gegeben werden, die in der Entwicklungsumgebung (IDE) sofort mögliche Schwachstellen anzeigen. Hierbei ist eine sofortige Anzeige und Korrekturmöglichkeit, ähnlich der Rechtschreibkorrektur in einer Textverarbeitung, besonders wichtig.
- **Build:** Im Bereich des Builds sind zwei Strategien möglich, die auch kombiniert eingesetzt werden können:
  - *Commit-Checks:* Um sicherzustellen, dass die von den in der Entwicklungsumgebung integrierten Werkzeugen gefundenen Schwachstellen in das Code-Repository eingecheckt werden, empfiehlt es sich, diese Checks bei jedem Commit erneut auszuführen. Zusätzliche können an dieser Stelle dynamische Tests (ähnlich zu Unit-Tests) ausgeführt werden.
  - *Nightly-Build-Checks*: Insbesondere komplexe statische Prüfungen, aber auch aufwendige dynamische Tests, benötigen in der Regel mehr Zeit als für Commit-Checks akzeptabel ist. Um einen guten Kompromiss zwischen einer möglichst zeitnahen Prüfung und den schnellen Releasezyklen zu ermöglichen, empfiehlt es sich, diese Tests "über Nacht" durchzuführen und eine Entwicklungsroutine zu etablieren, bei der die nächtlich gefundenen Probleme am nächsten Morgen behoben werden.
- **Milestone Releases:** Die bisher besprochenen Teststrategien finden in der Entwicklung statt und müssen dementsprechend für Entwickler, welche keine Sicherheitsexperten sind, zugänglich sein. Deswegen sind im normalen Releasezyklus manuelle Sicherheitstests (z. B. Penetrationstest) nur schwierig durchzuführen. Im Rahmen der Milestone Releases lassen sich durch Sicherheitsexperten alle klassischen Sicherheitsüberprüfungen durchführen. Die Korrektur der dabei gefundenen Schwachstellen kann im Rahmen des Milestone-Release-Zyklus geplant und durchgeführt werden.
- **Betrieb:** Die Tatsache, dass dasselbe Team die Entwicklung als auch den Betrieb einer Anwendung durchführt, ermöglicht eine Erweiterung der Testaktivitäten und somit eine bessere Risikoabdeckung und einen optimalen und anwendungsspezifischen Einsatz des Sicherheitsaufwandes. Hier bieten sich besonders zwei Techniken an:
  - *Monitoring und Intrusion-Detection* werden bereits beim Betrieb eingesetzt. Allerdings erfolgt nur in sehr seltenen Fällen ein Austausch zwischen dem Betrieb und der Entwicklung. Ein solcher Austausch ist aber zwingend notwendig, um die Sicherheitsaktivitäten und den -aufwand in der Entwicklung fokussieren zu können. Die Erkenntnisse aus dem Monitoring und der Meldung von Intrusion-Detection-Systemen führen im Idealfall zu verbesserten Sicherheitstestwerkzeugen (bzw. deren Konfiguration) in der Entwicklung. Umgekehrt ermöglicht die DevOps-Strategie die Anforderungen, welche eine umfängliche Monitoringlösung an eine Anwendung stellen, bereits in der Entwicklung zu berücksichtigen.
  - Eine *aktive Verteidigung*, wie sie z. B. Runtime Application Self Protection (RASP) bietet, ist erst im Rahmen einer DevOps-Strategie erfolgreich umsetzbar. Hierbei ist die enge Verzahnung von Betrieb und Entwicklung besonders wichtig, da die von RASP-Werkzeugen eingesetzte Instrumentierung unter Umständen bereits bei der Entwicklung berücksichtig werden muss.

Einsatzzeitpunkt	Entwicklung	Build (Commit)	Build (Nightly)	Milestone Release	Betrieb
Automatisierung	Hoch	hoch	hoch	niedrig	hoch
Max. Laufzeit	< 1s	< 10s	< 12h	mehrere Tage	< 1ms
False positives Rate	sehr niedrig	sehr niedrig	niedrig	hoch	mittel
False negatives Rate	Hoch	noch	mittel	niedrig	mittel
Integration	IDE	Code-Repository	Build-Server	nicht zwingend notwendig	Laufzeitumgebung
Sicherheits-expertise	Niedrig	niedrig	mittel	hoch	mittel
Anwendergruppe	Entwickler	Entwickler	Entwickler (und Sicherheitsexperten)	Sicherheitsexperten	Entwickler, Sysadmins
Beispiel	Find Security Bugs [FSB]	BDD [BDD]	Checkmarx (limited scope) [CHE]	Checkmarx (full scope) [CHE], man. Penetrationstests	HPE Application Defender [HPE]

Tab.: Anforderungen an Sicherheitstestwerkzeuge im Rahmen der SecDevOps-Strategie

Selbstverständlich sind nicht alle Sicherheitstests für jedes Produkt notwendig. Der genaue Einsatz muss bzgl. des Sicherheitsrisikos, den gesetzlichen und regulatorischen Anforderungen, sowie dem vorhandenen Budget abgewägt werden.

Die Tabelle fasst die verschiedenen Anforderungen an Sicherheitstests im Rahmen der (Sec)DevOps-Strategie zusammen. Hierbei beschreibt die „False-Positive-Rate“ den Anteil an falschen Meldungen eines Sicherheitstests, d. h. Meldungen, bei denen die detaillierte Analyse zu dem Schluss kommt, dass sie nicht sicherheitsrelevant sind.

Umgekehrt beschreibt die „False-Negative-Rate“ den Anteil an Sicherheitsfehlern, welche nicht von einem Sicherheitstest berichtet werden. Selbstverständlich ist eine sehr niedrige „False-Positive-Rate“ bei einer gleichzeitig sehr niedrigen „False-Negative-Rate“ wünschenswert, dies kann aber aus prinzipiellen Gründen von keinem Werkzeug erreicht werden.

Deswegen sollten Werkzeuge für Entwickler und die „Build Integration“, welche die automatische Release-Entscheidung trifft, den Fokus auf eine niedrige „False-Positive-Rate“ legen. Das heißt, jede Meldung ist mit hoher Wahrscheinlichkeit eine Sicherheitsschwachstelle. Im Gegenzug sollten Werkzeuge für Sicherheitsexperten den Fokus auf eine niedrige „False-Negative-Rate“ (keine Sicherheitsschwachstellen werden übersehen) legen.

Darüber hinaus gelten die generellen Anforderungen, bzw. Evaluierungskriterien für Sicherheitstesttools (z. B. [Che]). Selbstverständlich ist eine SecDevOps-Strategie, welche alleine auf Sicherheitstests aufbaut, nicht ausreichend. Softwaresicherheit beginnt auch bei Anwendung der DevOps-Strategie bereits in der Schulungs- und Planungsphase und endet frühestens mit einem umfassenden Response- sowie Patch- und Updateprozess.

Fazit

DevOps und sichere Softwareentwicklung sind keine Gegensätze. Aber insbesondere die Einführung von Sicherheitstestwerkzeugen muss noch wohlüberlegter erfolgen als in eher traditionellen Entwicklungsstrategien. Insbesondere müssen, im Sinne einer “Customer First”-Strategie, die Bedürfnisse der Softwareentwickler im Mittelpunkt stehen: Sicherheitstestwerkzeuge sollen Entwickler unterstützen und ihnen helfen, sichere Software zu entwickeln – und nicht als Hindernis empfunden werden. ■

Referenzen

[MSDF] Microsoft Security Development Lifecycle https://www.microsoft.com/en-us/sdl/
[MSDL] SDL for Agile https://www.microsoft.com/en-us/SDL/Discover/sdlagile.aspx
[BaS14] A. D. Brucker and Uwe Sodan. Deploying Static Application Security Testing on a Large Scale. In GI Sicherheit 2014. LNI 228, 2014.
[FSB] Find Security Bugs. http://find-sec-bugs.github.io/
[BDD] BDD-Security http://www.continuumsecurity.net/bdd-intro.html
[CHE] Checkmarx SAST Solution https://www.checkmarx.com
[HPE] HPE Application Defender http://www8.hp.com/us/en/software-solutions/appdefender-application-self-protection/
[Che] The AppSec How-To: Choosing a SAST Tool http://www.infosecurityeurope.com/__novadocuments/126614?v=635863258275630000