Andreas Falk
(andreas.falk@novatec-gmbh.de)
hat über 20 Jahre Erfahrung in der Entwicklung von Unternehmensanwendungen im Java-Umfeld. Seit mehr als drei Jahren ist er als Managing Consultant bei der NovaTec Consulting GmbH tätig. Seine Schwerpunkte sind die Architektur, Entwicklung und das agile (Security) Testen von Anwendungen.

SecDevOps – Security und DevOps gehören zusammen

Es vergeht inzwischen kaum ein Tag ohne eine neue Meldung einer gehackten Webanwendung bzw. Datenbank, einhergehend mit dem Diebstahl von Kreditkartendaten oder Passwörtern. Mit dem Internet der Dinge (Internet of Things – IoT) hält die Software nun auch zunehmend Einzug in unsere physische Welt. Auch hier wurden erste Angriffsversuche bekannt, wie u. a. ferngesteuerte Jeeps oder fremde Stimmen aus internetfähigen Babyphones. Gartner zählt daher Security inzwischen zu den Top 10 der Technologietrends für 2016. DevOps und Continuous Delivery sind zusammen mit agilen Vorgehensmodellen aktuell bereits weit verbreitet in der Softwareentwicklung. SecDevOps ist eine neue Bewegung als Reaktion auf die stetig wachsenden Sicherheitsanforderungen in Entwicklung, Deployment und Betrieb von Software. Dieser Artikel gibt eine Einführung in die Thematik und stellt die wichtigsten Konzepte vor.

Einleitung

Continuous Delivery wird erst durch die Kommunikation, Zusammenarbeit und Integration zwischen Entwicklung und Betrieb ermöglicht. Diese DevOps-Prinzipien sind bekannt und werden entsprechend in Kombination mit agilen Vorgehensmodellen wie Scrum oder Kanban in Projekten praktiziert.

Allerdings wird die Luft erheblich dünner, wenn es dann in Richtung Security geht. Häufig wird tatsächlich regelmäßig Software in kurzen Zeitabständen in Produktion gebracht, die dann allerdings nur sporadisch durch einen teuren externen Penetrationstester unter die Lupe genommen wird und dann für die weitere Zukunft als sicher eingestuft wird.

In Abbildung 1 wird die Diskrepanz zwischen der Zeitplanung der Angreifer und der Taktung von Deployments und Penetrationstests deutlich. Während die Angreifer praktisch rund um die Uhr versuchen, in das System einzudringen und bei lohnenswerten Zielen dabei immer schwerere Geschütze auffahren, wird die Anwendung selbst nur vergleichsweise selten auf Sicherheitslücken getestet.

Wie in Abbildung 1 zu sehen ist, muss die Sicherheit der Anwendung unbedingt mit der Deployment-Frequenz Schritt halten. Dies kann nur gewährleistet werden, wenn der Security-Aspekt auch in alle Aktivitäten innerhalb des Continuous Delivery mit integriert wird.

Dies beginnt bereits bei der Vision und beinhaltet, wie in Abbildung 2 mit Scrum als Vorgehensmodell dargestellt, die Befül-

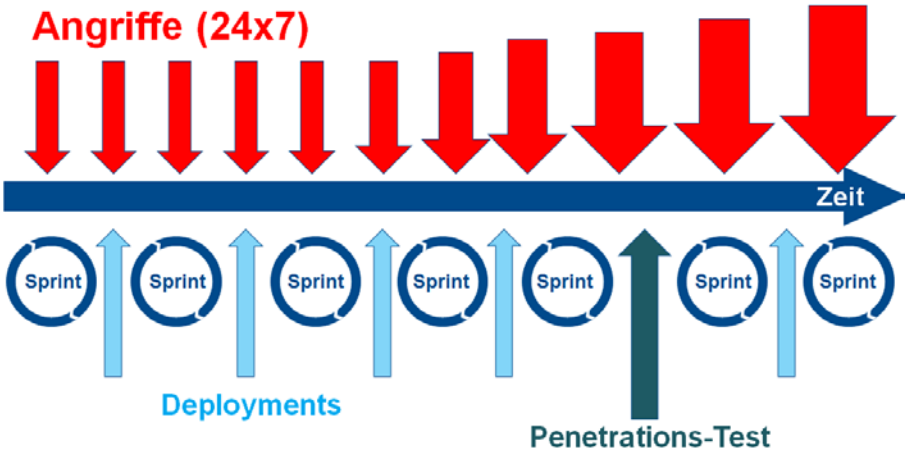

Abb. 1: 24x7 Zeitplanung der Angreifer

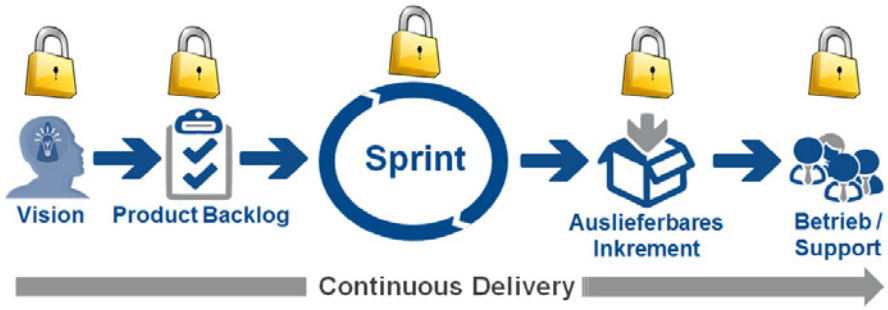

Abb. 2: Integration von Security in alle Continuous Delivery Aktivitäten

Abb. 3: AbUser (Evil) Stories

lung/Priorisierung des Product Backlogs, die Entwicklung im Sprint bis hin zur Erstellung des fertigen sicheren Inkrements und schließlich die Produktivsetzung und Wartung durch den Betrieb.

Sicherheit von Anfang an

Bereits mit Entstehung der Produktvision und den ersten Anforderungen sollten Überlegungen hinsichtlich möglicher Sicherheitsrisiken und Angriffsszenarien vorgenommen werden. Als hilfreich hat sich hier das sogenannte „Threat-Modeling“ erwiesen. Hier wird auf Basis der vorhandenen Anwendungsarchitektur ein Angriffsmodell abgeleitet, welches u. a. durch Festlegung von Vertrauensgrenzen (Trust Boundaries) zwischen den Komponenten hilft, potenzielle Angriffspunkte und -typen zu identifizieren. Eine sehr gute Einführung in das Threat-Modelling findet sich in [tre].

Aus den Diskussionen rund um das Threat-Modell ergeben sich dann entsprechende Anforderungen an die Sicherheit. Eine mögliche Kategorisierung von User Stories würde dann wie folgt aussehen:

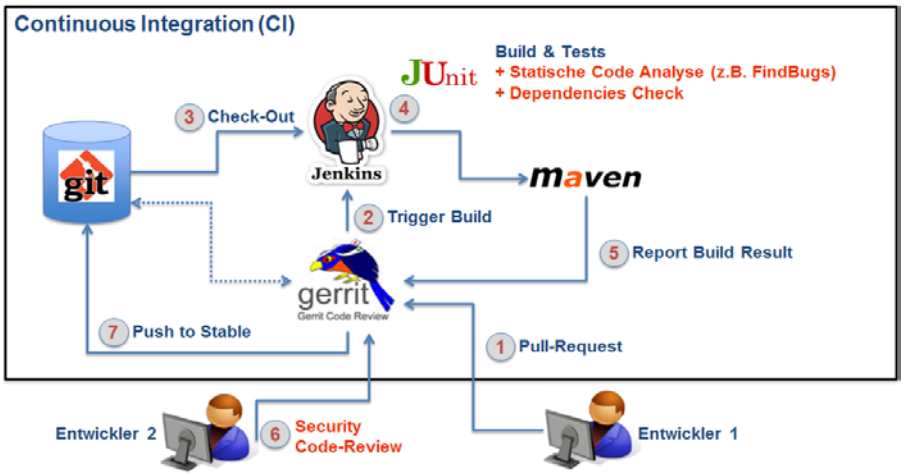

Abb. 4: Continuous Integration mit Securityaktivitäten

- User Stories für Security Features (z. B. Authentifizierung und Rollenmodell),
- Abuser Stories (siehe unten),
- User Stories mit sicherheitsrelevanten Akzeptanzkriterien,
- User Stories ohne jeglichen Security-Bezug.

Abuser Stories (auch bekannt als „Evil User Stories“ [owa1]) werden aus den fachlichen User Stories abgeleitet und beschreiben damit die verbundene gewünschte Funktionalität aus Sicht des Angreifers. Dabei sollten analog wie bei fachlichen User Stories auch „Personas“ verwendet werden, d. h. statt dem allgemeinen „Hacker“ könnte z. B. ein Skript-Kiddie, ein Firmen-Insider oder ein Wirtschaftsspion verwendet werden.

In Abbildung 3 ist ein Beispiel für eine fachliche (Business) User Story mit einer abgeleiteten Abuser Story zu sehen. Durch die Abuser Stories können dann im weiteren Verlauf für die Entwicklung im Rahmen der Sprint-Planung entsprechende Tasks erstellt werden.

Sicherheit entsteht vor allem in der Entwicklung

Gerade in der Entwicklung ist Security ein wichtiger Bestandteil. Hier sollte jeder Entwickler zumindest ein gewisses Maß an Basiswissen bezüglich Sicherheit besitzen. Durch geeignete Standardarchitekturen und „Secure Coding Design Patterns“ [owa2] können hier von Anfang an Sicherheitslöcher minimiert werden.

Unterstützend sollten auch nur erprobte Standardbibliotheken für Security-Funktionen eingesetzt werden, wie beispielsweise Spring Security [spr] oder Apache Shiro [shi]. Versäumnisse in der Entwicklung sind später durch Security-Tests nur mit erheblichem Aufwand korrigierbar sofern sie überhaupt rechtzeitig entdeckt werden.

In der Entwicklung hat sich eine „Continuous Integration Stage“ wie in Abbildung 4 bewährt. Neben der automatisierten statischen Code-Analyse und Prüfung von 3rd-Party-Bibliotheken auf bekannte Sicherheitslöcher ist gerade das manuelle Code-Review durch einen zweiten Entwickler integraler Bestandteil (hier z. B. mit Gerrit [ger]).

Darüber hinaus gibt es inzwischen mit OWASP Zap [zap] ein sehr gutes Open-Source-Werkzeug, mit welchem sich auch automatisierte dynamische Security-Tests aufsetzen lassen.

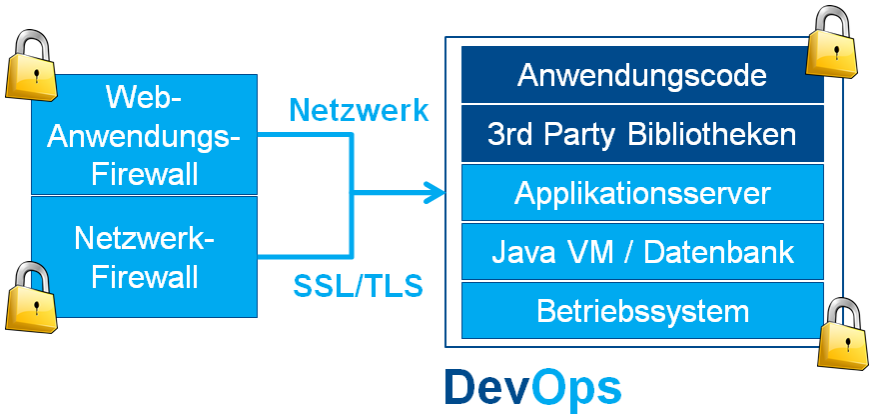

Abb. 5: Abzusichernde Assets im DevOps

Betrieb: Mehr als nur Netzwerksicherheit

In vielen Unternehmen kümmert sich der Betrieb häufig nur um die Absicherung der Netzwerke durch entsprechende Netzwerk- und Webanwendungsfirewalls sowie die Transportsicherung durch SSL bzw. TLS. Auch bzgl. SSL/TLS werden häufig Konfigurationsfehler gemacht (SSLv1 oder unsichere Cipher-Suites). Dabei kann auch dieses mit automatischen Prüfservices wie z. B. von den SSL Labs [ssl] verbessert werden.

Daneben wird meist das Betriebssystem regelmäßig mit Sicherheits-Updates versorgt. Leider lässt die Patchfrequenz nach, je höher man den Stack hinauf geht (siehe Abbildung 5). Applikationsserver werden nur ungern mit neuesten „Patchleveln" versehen, da man anschließend alle Anwendungen erneut testen muss.

Neben dem eigenen Anwendungscode wird auch häufig das Monitoring der 3rd-Party-Bibliotheken von laufenden Produktivanwendungen vergessen. Gerade hier ist eine enge Zusammenarbeit zwischen Entwicklung und Betrieb immens wichtig.

Meldet der Betrieb eine erkannte Schwäche an die Entwicklung, dann sollte dort die betroffene Komponente schnellstmöglich aktualisiert werden und das neue Software-Artefakt durch den Betrieb zeitnah produktiv gesetzt werden.

Fazit

In diesem Artikel wurden wesentliche Aspekte der SecDevOps dargestellt. Damit wird statt der punktuellen Einzelprüfung der Security durch externe Penetrationstester eine kontinuierliche Integration des Security-Aspekts in jeden Bereich von DevOps erreicht. Dies führt gerade bei Continuous Delivery zu entsprechend sicheren auslieferbaren Inkrementen. Einen guten externen Penetrationstester kann man dann stattdessen sinnvoller als Trainer/Coach für die Entwicklung oder für das Testen von Härtefällen einsetzen. ■

Referenzen

[tre] http://threatmodelingbook.com/index.html
[owa1] https://owasp.org/index.php/Agile_Software_Development:_Don%27t_Forget_EVIL_User_Stories
[owa2] https://www.owasp.org/index.php/OWASP_Proactive_Controls
[spr] http://projects.spring.io/spring-security
[shi] https://shiro.apache.org
[ger] https://www.gerritcodereview.com
[zap] https://github.com/zaproxy/zaproxy
[ssl] https://www.ssllabs.com/ssltest