JAVA Mag
Javamagazin™
Java | Architektur | Software-Innovation

Security agil verankern

Plädoyer für eine Securitykultur

www.javamagazin.de

Ausgabe 10.2016

Deutschland € 9,80
Österreich € 10,80
Schweiz sFr 19,50
Luxemburg € 11,15

4 194586 709801 10

Secure SDLC

Sicherheit agil verankern

Unternehmen binden heute immer neue kritische Geschäftsfunktionen an das Internet an, die auch für potenzielle Angreifer interessant sind. Erst durch die Verankerung von Sicherheit im Entwicklungsprozess sowohl auf technischer als auch auf organisatorischer Ebene lässt sich dieses Risiko adressieren. Wir zeigen Ihnen effiziente Maßnahmen, wie Projekte diesem Ziel einen Schritt näher kommen können.

von Matthias Pöpping, Matthias Rohr und Christian Schneider

Das klassische Vorgehen, bei dem Dienstleister mit Sicherheitstests, so genannten Pentests, die Sicherheit einer Anwendung ausschließlich vor der Produktivstellung analysieren, ist hoffnungslos veraltet. Besonders deutlich wird dies in Zeiten von agiler Softwareentwicklung und DevOps, die in immer mehr Unternehmen Einzug halten. Dort ist ein Umdenken erforderlich, um auch für diese neuen Entwicklungsverfahren Sicherheit zu gewährleisten.

Generell gilt: Sicherheitsprobleme können in jeder Entwicklungsphase eingebracht werden. Und je zeitnaher diese erkannt werden, desto einfacher, günstiger und vollständiger lassen sie sich in der Regel auch beheben. Daraus folgt, dass idealerweise in jeder Phase der Softwareentwicklung geeignete Verfahren und Werkzeuge integriert werden müssen, um dort entstandene Sicherheitsprobleme zu identifizieren. Ein auf diese Weise erweiterter Entwicklungsprozess wird häufig als „Secure Software Development Life Cycle“ (Secure SDLC) bezeichnet.

Ansatzpunkte für einen Secure SDLC

Seit einiger Zeit existieren mit Microsofts SDL, Cigitals Touchpoints, OWASP SAMM oder BSIMM verschiedene Prozess- und Reifegradmodelle, die wertvolle Ansatzpunkte für die Integration sinnvoller Maßnahmen in die Softwareentwicklung eines Unternehmens bieten. Häufig scheitern aber viele Versuche, die entsprechenden Aktivitäten in der Organisation umzusetzen, daran, dass diese isoliert betrachtet werden, z. B. als Schulung oder Definition von Vorgaben. Jedoch erfordert praktisch jede Maßnahme zur nachhaltigen Verankerung von Sicherheit stets deren Berücksichtigung auf vier Ebenen: Organisation, Vorgaben und Guidance, Qualifikation und (Security-)Kultur der Mitarbeiter sowie den Einsatz von Technologien. Diese zentrale Sichtweise der Applikationssicherheit veranschaulicht der Würfel in **Abbildung 1**.

Weg mit Elfenbeinturmanforderungen

Sicherheitsvorgaben beschreiben, welche Sicherheitsmaßnahmen im Endeffekt einzuhalten und umzusetzen sind. An diesen mangelt es grade in großen Unternehmen nur selten, doch sind sie vielfach sehr abstrakt. Das ist grundsätzlich auch richtig, da Vorgaben letztlich aus der Sicherheitsrichtlinie abgeleitet und vom Chief Information Security Officer (CISO) verantwortet werden müssen. Andererseits sollten Sicherheitsvorgaben auch für diejenigen zu verstehen sein, die sie umsetzen sollen – also für Entwickler. Das erfordert die Durchführung von grundlegenden Maßnahmen. Eine sinnvolle Variante besteht darin, im ersten Schritt implementierungsübergreifende Standards für bestimmte Technologien wie Mobile oder Web (z. B. TSS-WEB oder OWASP ASVS) zu definieren und daraus dann konkrete implementierungsspezifische Secure Coding Guidelines für Java, JSF oder JavaScript abzuleiten. Genau an solchen Guidelines oder zumindest an deren Qualität und Aktualität mangelt es oft. In Folge dessen existiert eine Fülle an Vorgaben, mit denen sich das Ziel, eine sichere Software zu entwickeln, aber praktisch nicht erreichen lässt. Das Erstellen der Secure Coding Guidelines sollte daher in enger Zusammenarbeit mit den Entwicklern geschehen, die diese dann auch idealerweise mit verantworten. Sicherheitsanforderungen müssen leben und genauso wie die zu entwickelnde Software laufend an neue Technologien und Use Cases angepasst werden. Sehr gute Unterstützung bei der Dokumentation bieten Wikis wie Confluence. Damit lassen sich die Guidelines leicht aktualisieren und, durch die Anbindung an Ticketsysteme wie Jira, nahtlos in die vorhandenen Entwicklungsprozesse integrieren.

Neben allgemein gehaltenen, nicht funktionalen Sicherheitsanforderungen können auch Anforderungen spezifisch für bestimmte Projekte oder Anwendungen definiert werden. Diese können sowohl technisch als auch fachlich sein. Sehr anschaulich wird dies bei funktionalen Sicherheitsanforderungen. Diese werden etwa mittels Bedrohungsmodellierung, Security User Stories oder Evil User Stories im Rahmen einer Sprint-Planung identifiziert und erhalten dadurch auch entsprechende Story Points. Hierfür bedarf es natürlich entsprechend geschulter Teams und der Unterstützung durch Experten.

Security Gates errichten

Um zu verifizieren, dass Sicherheitsmaßnahmen vollständig und auch korrekt umgesetzt werden, erfordert es in der Regel die Durchführung entsprechender Tests. Im Fall von Sicherheitsvorgaben sind dies beispielsweise Pentests, Codescans oder Peer-Reviews. Um dies zu gewährleisten, lassen sich Security (Quality) Gates an kritischen Punkten im Entwicklungsprozess etablieren. Was genau dort durchzuführen ist, sollte davon abhängen, wie kritisch eine Anwendung ist. Auch die Strenge dieser Security Gates lässt sich danach unterscheiden, ob etwa eine Abnahme (Sign-off) des Security Officers erforderlich ist oder aber bereits ein Self-Assessment durch das Team genügt. Wichtige Security Gates sind in diesem Zusammenhang:

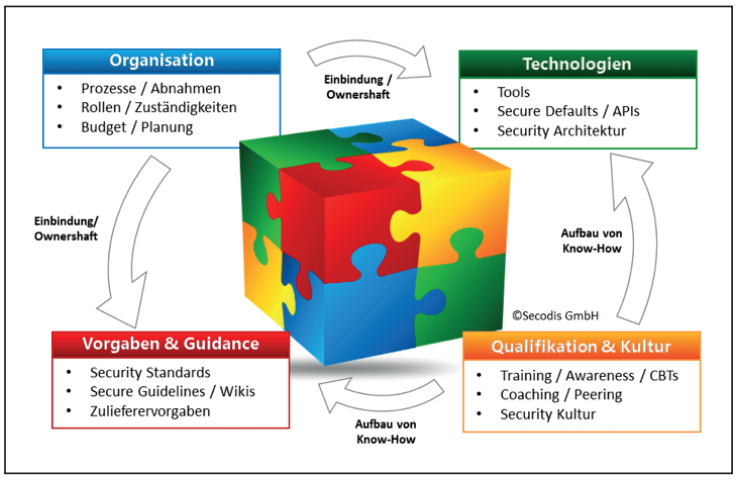

Abb. 1: Um Sicherheit nachhaltig zu verankern, müssen alle Elemente der Applikationssicherheit berücksichtigt werden

- Security Gate 1: Nach der Spezifikation
- Security Gate 2: Nach Abschluss des technischen Designs
- Security Gate 3: Vor Inbetriebnahme (auch „Final Security Sign-off“)

Gerade bei agil arbeitenden Teams sind solche Sicherheitskontrollpunkte jedoch schwierig in der Praxis umzusetzen. Verschiedene Maßnahmen unterstützen dabei. Vor allem die Automatisierung der Sicherheitstests mithilfe von Werkzeugen, Secure Defaults und das Stärken der Eigenverantwortlichkeit und des Know-hows der Teams in Bezug auf Sicherheit helfen bei der Umsetzung. Ein automatischer Codescan könnte z. B. bereits ein Security Gate erfüllen. Dadurch lassen sich Security Gates flexibel an die Arbeitsweise agiler Teams anpassen, um diese damit nicht unnötig bei ihrer Arbeit zu stören.

So lassen sich beispielsweise in der Definition of Done (DoD) auch die Durchführung von bestimmten Sicherheitsprüfungen fallabhängig einfordern, etwa die Durchführung eines Codescans oder Peer-Reviews. Hierbei helfen Securityindikatoren. Mit diesen lassen sich sicherheitsrelevante Änderungen (Security Changes) an einer Anwendung identifizieren, z. B. das Anbinden einer neuen Schnittstelle oder das Anpassen einer Sicherheitskomponente.

Des Weiteren können Projektleiter oder Product Owner diese Änderungen gezielt im Sprint Backlog planen und sie in einem „Security Sprint“ bündeln. Auch das Herauslösen von sicherheitskritischen Komponenten, z. B. ein Security-API, aus einer agil entwickelten Anwendung kann hier helfen. So lässt sich eine solche Komponente dann über ein Vorgehensmodell mit weniger Releases im Jahr entwickeln, um dadurch wiederum die erforderlichen Sicherheitsprüfverfahren einfacher umsetzen zu können. Wie die Securityintegration in einen agilen Prozess aussehen kann, zeigt **Abbildung 2**.

Jedes Team braucht einen Security Champion

Ein zentrales Grundkonzept von agil arbeitenden Teams ist es, Kompetenzen und Verantwortung für bestimmte Themen in das Team zu verlagern. Genau dies muss auch in Bezug auf die IT-Sicherheit erfolgen. Die Verantwortung hierfür wird dabei auf einen oder mehrere Teammitglieder übertragen, den so genannten Security Champion. Je nach Größe des Teams und Sicherheitsbezug der dort entwickelten Anwendungen kann diese Rolle unterschiedlich stark ausgeprägt sein. Security Champions werden entsprechend für das Thema geschult, können kleine Sicherheitstests eigenständig durchführen und Sicherheitsrelevanz von User Stories identifizieren. Weiterhin dienen sie als Ansprechpartner nach außen und agieren als Multiplikator für Sicherheitsthemen nach innen.

Die Security Champions aus allen Teams sind untereinander in einer Community vernetzt, der Software Se-

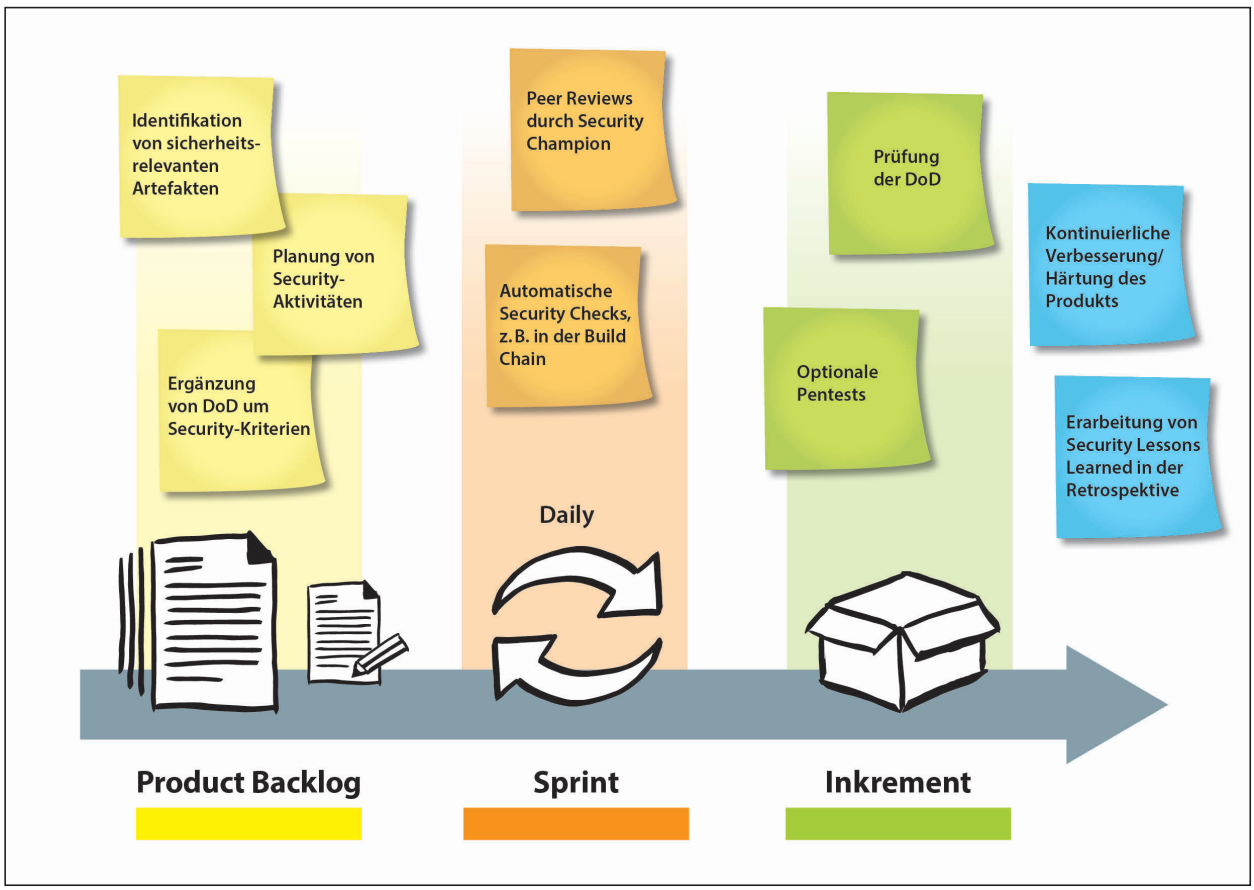

Abb. 2: Verschiedene Maßnahmen helfen dabei, Security in agile Prozesse einzubinden

Ein zentrales Konzept von agil arbeitenden Teams ist es, Kompetenzen und Verantwortung in das Team zu verlagern. Genau dies muss auch in Bezug auf die IT-Sicherheit erfolgen.

curity Group. Diese trifft sich regelmäßig. In ihr wird gemeinsam an bestimmten Themen wie Toolevaluierung oder Guidelines gearbeitet. Unterstützt und eingebunden werden die Security Champions von der zentralen IT-Sicherheitsabteilung.

Ohne Securitywerkzeuge geht es nicht

Gerade bei agiler Entwicklung lässt sich kaum jedes Release einem umfangreichen Pentest und Codereview unterziehen. Daher führt praktisch kein Weg daran vorbei, Sicherheitsprüfungen soweit wie möglich zu automatisieren; auch, um die Arbeit der Security Champions in den Teams zu unterstützen. Hierzu existieren verschiedene Arten von Werkzeugen, die mal mehr, mal weniger gut als Teil einer Security-Build-Pipeline eingesetzt werden können. Grob unterscheidet man nach den folgenden Kategorien:

- **DAST** (Dynamic Application Security Testing): DAST-Werkzeuge betrachten die zu prüfende Webanwendung aus Sicht eines Angreifers von außen, also als Black Box. Je nach Konfigurationsmöglichkeiten des Werkzeugs können bei einfacheren Anwendungen auch Tests im eingeloggten Bereich durchgeführt werden. Im Falle von komplexen Anwendungen können jedoch die Anpassungsaufwände des Werkzeugs nicht unerheblich sein, insbesondere wenn fachlich zu füllende Dialogstrecken abzubilden sind, bei denen der Crawler an seine Grenzen stößt. Prominente Open-Source-DAST-Werkzeuge sind z. B. OWASP ZAP und Arachni Scanner.
- **SAST** (Static Application Security Testing): SAST-Werkzeuge scannen den Quellcode oder Bytecode der Anwendung auf Sicherheitslücken. Einfachere SAST-Werkzeuge betrachten nur die Sinks, also die potenziell unsicheren Stellen, ohne eine Verbindung von potenziell manipulierbaren Daten herzustellen. Professionellere Werkzeuge prüfen auch diese Verbindung und können damit genauere Aussagen zur tatsächlichen Ausnutzbarkeit treffen. Im Vergleich zu DAST-Werkzeugen haben SAST-Werkzeuge einen höheren Abdeckungsgrad der Anwendung, weil der gesamte Quell- oder Bytecode gescannt werden kann. Zusätzlich zu kommerziellen Werkzeugen existieren im Open-Source-Bereich z. B. das FindBugs-Plug-in Find Security Bugs oder auch erweiterte Sonar-Regeln für Java-Anwendungen. Für JavaScript kann das in ESLint aufgehende ScanJS verwendet werden. Weiterhin gibt es Brakeman für Ruby-on-Rails-Anwendungen. Ebenfalls existieren hier Werkzeuge zur Prüfung der Abhängigkeiten, um das Risiko durch veraltete und unsichere Bibliotheken zu reduzieren. Diese gleichen die Listen bekannter verwundbarer Komponenten gegen die verwendeten Bibliotheken ab. Als Open-Source-Lösungen, die im Rahmen des Builds eingesetzt werden können, existieren hier OWASP Dependency Checker für Java-Abhängigkeiten sowie Retire.js zur Prüfung der eingebundenen JavaScript-Bibliotheken.
- **IAST** (Interactive Application Security Testing): Diese im Vergleich neuere Kategorie kann als dynamische Codeanalyse betrachtet werden: Hierbei wird mittels Instrumentierung einer Testumgebung die Anwendung zur Laufzeit einer White-Box-Analyse unterzogen. Bemerkenswert bei manchen IAST-Werkzeugen ist, dass die Sicherheitsüberprüfung parallel zur fachlich getriebenen Benutzung der Anwendung stattfinden kann. In diesem Fall ist die Definition von Tests speziell für die Anwendungssicherheit nicht notwendig. Das gestaltet die Integration in die vorhandenen Testprozesse einfacher. Andere IAST-Werkzeuge setzen hingegen auf eigene Testtreiber, die das Crawler-Problem mit entsprechenden Anpassungsaufwänden mitbringen.

Diese Werkzeugkategorien zur direkten Analyse der Anwendung oder des Anwendungscodes bilden unterschiedliche Schwerpunkte bei den auffindbaren Sicherheitslücken. Somit kann der parallele Einsatz von Werkzeugen mehrerer Kategorien durchaus sinnvoll sein.

Daneben existieren noch eine Reihe weiterer Securitytools, die sich nicht in die obigen Kategorien einordnen lassen. Prinzipiell lassen sich Sicherheitsaktivitäten in allen Phasen des Lebenszyklus einer Anwendung mit entsprechenden Werkzeugen unterstützten. Angefangen von der Spezifikation, wo sich mit Threat Modeling Tools (z. B. Microsoft Threat Modeling Tool 2016) potenzielle Sicherheitsprobleme frühzeitig identifizieren lassen, bis hin zum Betrieb, wo sich Werkzeuge eher auf das Erkennen oder Vermeiden aktiver Angriffe (z. B. mittels OWASP AppSensor) beziehen.

Werkzeuge in den Entwicklungsprozess integrieren

Der alleinige Einsatz von Werkzeugen löst das Grundproblem jedoch nicht, da die Integration in den Entwicklungsprozess effizient gestaltet werden muss. Dies umfasst auch die Klärung von Fragestellungen wie „Wer bedient und verantwortet die Werkzeuge?“. Sollen Entwickler mit solchen Werkzeugen direkt arbeiten, ist vor

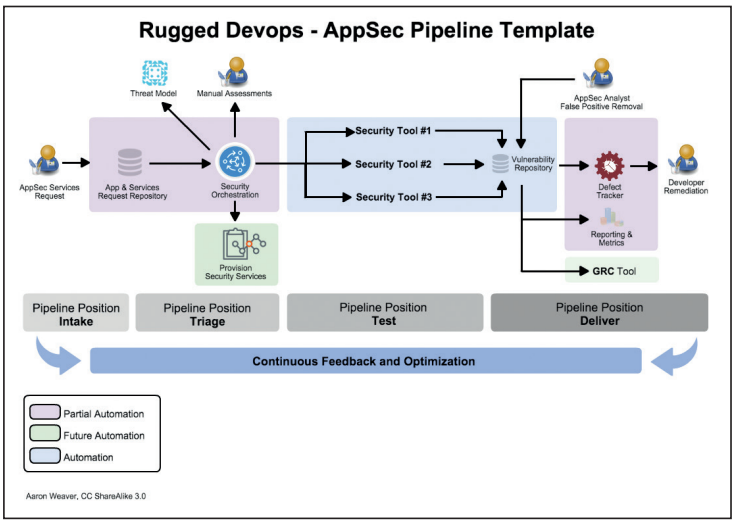

Abb. 3: Exemplarische Integration einer Security-Build-Pipeline (Quelle: www.OWASP.org, Aaron Weaver, CC ShareAlike 3.0)

allem ein Faktor entscheidend, damit dies auch in der Praxis erfolgt: die Bedienbarkeit. Zu komplizierte Tools oder solche, die viele Fehlalarme ausgeben, werden schnell lästig und deren Ergebnisse ignoriert. Daher gilt auch hier der Satz „Weniger ist mehr".

Gerade für die Einführungsphase empfiehlt es sich, Baseline-Scans durchzuführen, die nur Findings mit einer hohen Zuverlässigkeit und Kritikalität ausgeben. Dabei muss jedes Finding eines Tools auch unbedingt zunächst verifiziert werden, bevor dieses als Schwachstelle bewertet und hierfür ein Ticket eingestellt wird. Hier könnte einer der Security Champions unterstützend tätig werden. Sicherheitsprobleme mit geringerer Kritikalität, die nicht sofort beseitigt werden, lassen sich im Rahmen periodisch durchgeführter Aufräumarbeiten nachziehen, z. B. im hierfür speziell angesetzten Security Sprint.

Ein weiterer wichtiger Schritt ist die Integration von Securitytools in die Build-Umgebung. Zu vielen gängigen Tools existieren hierfür Plug-ins, etwa für die Einbindung in Jenkins. Wenn nicht mithilfe eines Plug-ins, so sollte sich zumindest über die Kommandozeile jede Art von Werkzeug einbinden und in der Build-Kette automatisch ausführen lassen. Dabei stellt sich natürlich schnell die Frage, wie die Ergebnisse der verschiedenen Tools im weiteren Verlauf ausgewertet und weiterverarbeitet werden. Hier bieten sich Tools wie ThreadFix an. Solche Vulnerability-Management-Systeme können die Ausgaben zahlreicher gängiger Securitywerkzeuge per API entgegennehmen, zusammenführen und auch Duplikate entfernen. Per Web GUI lassen sich dann, z. B. von einem Security Champion, die offenen Findings analysieren, False Positives entfernen und schließlich vom Tool entsprechende Tickets oder Stories für deren Beseitigung im Ticketsystem oder im Backlog einstellen. So lässt sich das Beheben von Sicherheitsproblemen durchsetzen und später auch nachvollziehen. In **Abbildung 3** ist eine exemplarische Integration einer solchen Security-Build-Pipeline gezeigt. Je nach Aufbau der eigentlichen Build-Kette lassen sich dort Sicherheitstests aber auch in anderer Form abbilden.

Mithilfe geeigneter Securitywerkzeuge lassen sich die definierten Security Gates im Hintergrund (teil-)automatisiert abbilden: Es laufen bestimmte Prüfungen permanent während der Entwicklung mit und bieten somit eine Basisabsicherung bei häufigen Rollouts.

Die meisten etablierten Ticketsysteme, z. B. Jira, bieten die Möglichkeit, zusätzliche Prozessschritte wie die Security Gates in den Bearbeitungsworkflow zu integrieren. Dadurch kommt es zu keinem Bruch bei den im Team bereits eingesetzten Werkzeugen. Dies trifft ebenfalls auf den Umgang mit Security Findings zu. Hierzu lässt sich ein neuer Tickettyp Security Issue anlegen und über diesen den Fortschritt bei deren Behebung tracken. Zudem lassen sich auch häufig vorkommende Risikoübernahmen dokumentieren.

Neben Build-Systemen lassen sich einige Securitywerkzeuge aber auch direkt in eingebundenen Code Repositories einhängen, nicht zuletzt, um darüber teamübergreifende Prüfungen durchzuführen. Etwa durch den Einsatz von OWASP Dependency Checks, die nach unsicheren Abhängigkeiten suchen. Oder Peer-Review-Tools, die, eingebunden im Code Repository, bei Änderungen an bestimmten, als sensibel markierten, Codebereichen eine Freigabe durch den Security Champion erforderlich machen. Die Bereitstellung von Tools, entsprechender Integrationen und Templates kann über die zentralen Securityteams erfolgen.

Egal welche und wie viele Tools jedoch am Ende eingebunden werden. Diese können nicht den Einsatz von manuellen Sicherheitsprüfungen ersetzen. Allerdings müssen diese nicht zwangsläufig im Rahmen jedes Sprints, sondern lediglich periodisch und bei größeren Änderungen an der Anwendung durchgeführt werden. Hier kommt es auch auf die richtige Planung durch den Projektleiter an, der gezielt sicherheitsrelevante Stories, z. B. neue Schnittstellen oder Änderungen am Session Management, in eigenen Sprints schiebt und für diesen dann auch z. B. die Durchführung eines Pentests einplant.

Secure Defaults sind die Basis

Bei der Anwendungssicherheit alleine auf die rechtzeitige Identifikation möglicher Sicherheitsprobleme zu setzen, reicht jedoch nicht aus. Zusätzlich sollte der Fokus stets darauf gelegt werden, wie sich von vornherein vermeiden lässt, dass solche Probleme überhaupt auftreten. Mithilfe von sicherheitstechnischen Leitplanken in Form von Architekturvorgaben lässt sich bereits viel erreichen: So können z. B. sicherheitskritische Komponenten zur Authentifizierung, zur Backend-Anbindung oder zur Kommunikation zwischen Microservices von der änderungsintensiven agilen Frontend-Entwicklung abgekoppelt und durch eigene Teams mit höherem Sicherheitsfokus und eventuell sogar einem anderen Entwicklungsvorgehen entwickelt werden.

Eine weitere Fehlervermeidungsstrategie besteht im Einsatz von speziell gehärteten Frameworks, durch die

sich manche Sicherheitsprobleme bereits vollständig implizit behandeln lassen. So bieten z. B. bestimmte Webframeworks Härtungsmaßnahmen gegen Cross-site Request Forgery (CSRF) in Form von automatisch eingebauten Anti-CSRF-Tokens out of the box an. Ebenso existieren Template-Engines, die automatisch kontextabhängig das richtige Output Encoding zum Schutz gegen Cross-site Scripting (XSS) anwenden. Auch die Kompatibilität zum Content Security Policy (CSP) und als sekundäre Härtungsmaßnahme sowie Bean-Validation-Standard zur Eingabevalidierung sollten bei der Auswahl eines Webframeworks beachtet werden.

Mit solchen Secure Defaults lässt sich die Fehleranfälligkeit deutlich reduzieren. Teilweise lässt sich das Auftreten bestimmter Sicherheitsprobleme hierdurch sogar ganz ausschließen und der Einsatz entsprechender Sicherheitstests reduzieren.

Security-Know-how aufbauen

Der Erfolg dieser Maßnahmen hängt maßgeblich vom Verständnis der Sicherheitsziele, der Umsetzbarkeit der Vorgaben und der Werkzeugbenutzung in den einzelnen Teams ab. Ohne entsprechende Schulungen besteht das Risiko, dass ein Großteil der getroffenen Maßnahmen langfristig keine nennenswerten Effekte zeigt. Sinnvolle Schulungsformen umfassen sowohl Entwicklertrainings in der defensiven Programmierung und der gezielten Werkzeugbenutzung, als auch Awareness-Maßnahmen für die Fachseite bzw. Businessanalysten. Gerade im Bereich des Requirements Engineering lassen sich mit vergleichsweise geringen Aufwänden sicherheitstechnische Maßnahmen für die Entwicklung spezifizieren – optimalerweise auf identifizierte Abuse Cases zielend. Ergänzend zu Hands-on-Trainings mit Übungen für Entwickler können auch Security-Coaches zeitweise in den Teams mitarbeiten und dort das Wissen bei den Teammitgliedern on the Job aufbauen. Auch über die Security Champions lassen sich die jeweiligen Teams laufend mit neuen Informationen zu Sicherheitsthemen versorgen.

Um nachhaltigen Erfolg zu haben, muss das Erlernte kontinuierlich aufrechterhalten und aktualisiert werden, was z. B. durch die oben erwähnten Security Guidelines oder Updates in Form von Self-Training aufbauend auf einem initialen Hands-on-Training stattfinden kann. Als ebenfalls interessanter Ansatz zum kontinuierlichen Lernen lässt sich das Bewusstsein für unsicheren Code und dessen Absicherung im Rahmen von Gamification schaffen: z. B. in Form von Game-of-Hacks, in dem Entwickler in einer unternehmensinternen Challenge einen Score in der sicherheitstechnischen Bewertung von beispielhaft erstellten unsicheren Code-Snippets zu erreichen versuchen.

Fazit

Sicherheit und agile Softwareentwicklung sind keinesfalls Widersprüche. Damit beides miteinander funktioniert, ist jedoch vielfach ein Umdenken, insbesondere in Bezug auf Freigaben (Security Gates) und Securitytests erforderlich. Zentrale Aspekte sind hierbei Secure Defaults, Securitytestautomatisierung sowie die Verlagerung der Zuständigkeit für Sicherheit in die einzelnen Teams.

Ohne entsprechende Schulungen besteht das Risiko, dass viele Maßnahmen langfristig keine nennenswerten Effekte zeigen.

Die eigenen Entwicklungsteams nicht einzubeziehen, auch in die Verantwortlichkeit für Sicherheit, bedeutet die Sicherheit in einer agilen Entwicklung zum Scheitern zu verurteilen. Daher ist es wichtig, eine lokale Securitycommunity innerhalb der Entwicklung aufzubauen, in die auch die agilen Teams durch ihre eigenen Security Champions eingebunden sind und dort aktiv mitarbeiten.

Derart umfassende Maßnahmen lassen sich natürlich nicht über Nacht umsetzen. Daher nehmen Sie sich Zeit für die Entwicklung ihrer Organisation und das Schaffen einer Securitykultur. Beginnen Sie mit wenig und einfach gehaltenen Tests, die Sie dann schrittweise aufbauen. Gleiches gilt für das Einbeziehen der lokalen Security Champions. Sicherheit muss auch Spaß machen. Dies ist gerade im Fall der Integration von Sicherheit in agile Entwicklung der Fall.

Natürlich muss auch das Management in dieses Vorhaben eingebunden werden, nicht zuletzt um diesem die notwendige Priorität und das erforderliche Budget einzuräumen. Machen Sie diesem die Notwendigkeit für diese Anstrengungen deutlich und Erfolge in diesem Prozess sichtbar.

Matthias Pöpping ist Softwareentwickler und Information Security Officer bei einem Kölner Versicherungskonzern. Er hat mehrere Jahre in einem Scrum-Team Webanwendungen programmiert und berät nun agile Teams zum Thema IT-Sicherheit. Außerdem ist er nebenberuflicher Student im Fach Informatik/IT-Sicherheit.

Matthias Rohr beschäftigt sich seit über zehn Jahren mit Lösungen zur Erhöhung der Sicherheit von Enterprise-Webanwendungen. Er ist Autor des Buchs „Sicherheit von Webanwendungen in der Praxis“ und Geschäftsführer der Secodis GmbH, einem Beratungshaus, das Unternehmen speziell bei der Integration von Sicherheit in ihre Entwicklungsprozesse unterstützt.

Christian Schneider ist als freiberuflicher Softwareentwickler, White Hat Hacker und Trainer tätig. Er unterstützt Kunden im Bereich der Websecurity durch Penetrationstests und „Security Architecture Consulting“ sowie agile Teams bei der Einführung von „Security DevOps“. In dieser Rolle ist er regelmäßig als Trainer tätig und spricht auf Konferenzen.

@cschneider4711 **www.Christian-Schneider.net**