Security – auch in kleinen Entwicklerteams

Dr. Stefan Schlott, BeOne Stuttgart GmbH

Am Thema „Security" kommt inzwischen keiner mehr vorbei – das steht außer Frage. Große Firmen und Konzerne etablieren für ihre Entwicklungsabteilungen entsprechende Prozesse, die einzuhalten sind, und bieten zentrale Anlaufstellen für Rückfragen, Reviews oder Penetration Tests. Kleinere Firmen hingegen können sich den Aufbau solch großer Strukturen kaum leisten. Dennoch ist es möglich, sich in kleinen Schritten einem adäquaten Sicherheitsniveau zu nähern.

Natürlich macht kein Prozess eine Anwendung automatisch sicher und selbstverständlich kommt es auch immer auf den jeweiligen Anwendungsfall an, welchen Aufwand man in die Sicherheit einer Anwendung investieren sollte – es käme ja auch niemand auf die Idee, sich mit dem „1x1 gegen Wohnungseinbruch" für Fort Knox zufriedenzugeben. Aber zumindest die typischen Fehler und „Low Hanging Fruits" für Penetration-Tester und Angreifer lassen sich vermeiden.

Auf die wesentlichen Ziele beschränken

Die Bandbreite von Security-Maßnahmen reicht von vollkommener Ignoranz bis zur vollständigen Paranoia, und beide Extreme sind Projekten nicht zuträglich. Sinnvoll ist es, wenn wir uns auf die wesentlichen Sicherheitsziele einigen – ganz genauso, wie wir mit dem Kunden besprechen, welche Features besonders wichtig sind und welche erst in einer späteren Version umgesetzt werden. Hier geht es uns zunächst um das „Was": Vor was möchten wir uns schützen? Was sind Gefahren für den Anwender unserer Software beziehungsweise für unseren Kunden? Antworten können die Sorge vor finanziellen Verlusten sein (etwa der Verlust irgendwelcher Geschäftsgeheimnisse, drohende Strafzahlungen etc.), aber auch die Sorge um die Reputation (wer will schon mit einer Negativschlagzeile auf dem Titelblatt der Tageszeitung stehen) oder das Einhalten gesetzlicher Rahmenbedingungen (die DSGVO ist da häufig nur ein Aspekt).

Diese Diskussion lässt sich sehr untechnisch und damit für den Kunden sehr verständlich führen. Wenn man diese Sicherheitsziele später mit entsprechenden Gegenmaßnahmen und dahinter steckenden Aufwänden verknüpft, kann der Kunde mit üblichen Methoden des Risikomanagements entscheiden (vereinfacht: Kosten vs. potenzieller Schaden und dessen Eintrittswahrscheinlichkeit), welche der Sicherheitsziele umgesetzt werden sollen und welche Risiken er bereit ist, in Kauf zu nehmen.

Der nächste Schritt des Threat Modeling ist die Frage, mit welchen Mitteln ein Angreifer unsere Sicherheitsziele verletzen kann. Diese geben uns dann einen direkten Hinweis auf technische Gegenmaßnahmen. Eine gebräuchliche Klassifikation sind die „STRIDE"-Kategorien *(siehe Tabelle 1)*.

Bei komplexeren Systemen lohnt es sich, dies in seine groben Komponenten aufzugliedern (Application Decomposition). Durch Festlegen von Trust Boundaries wird nochmals klargestellt, innerhalb welcher Bereiche wir uns auf die Korrektheit welcher Informationen verlassen können und wo Angaben gegebenenfalls nochmals überprüft werden müssen. Hilfreich können hier Tools wie das kostenlose Microsoft Threat Modeling *(siehe „https://www.microsoft.com/en-us/download/details.aspx?id=49168")* sein. Es liefert nicht nur ansehnliche Grafiken für die Architektur-Dokumentation, sondern auch eine generische Liste von Sicherheitsrisiken (passend zur „STRIDE"-Klassifikation) für jede Komponente.

Wann was zu tun ist

Je früher man ein Problem erkennt, desto geringer ist der Aufwand, es zu beheben. Was für architekturelle Fehler oder fachliche Missverständnisse gilt, gilt ebenso für Security-Aspekte. In einer idealen Welt hätte man in jeder Phase der Entwicklung einen Vollblut-Security-Experten im Team – ein Wunsch, der sich leider nicht überall realisieren lässt. Was sich allerdings in jedem Team etablieren lässt, ist das Bewusstsein dafür, dass es zu jeder Phase der Software-Entwicklung entsprechende Sicherheitsaspekte gibt.

Viele Anforderungen sind mit einem der Sicherheitsziele verknüpft oder können mit den daraus abgeleiteten Schutzmaßnahmen in

Angriff	Vorgehen	Verletztes Prinzip
Spoofing	Vortäuschen einer Identität	Authenticity
Tampering	Verändern von Daten	Integrity
Repudiation	Abstreiten von Aktionen	Non-Repudiation
Information Disclosure	Unberechtigte Kenntnisnahme von Informationen	Confidentiality
Denial of service	Verhinderung der Dienste-Nutzung	Availability
Elevation of privilege	Unberechtigte Nutzung erhöhter Systemprivilegien	Authenticity

Tabelle 1

Verbindung gebracht werden. Umgekehrt gilt, dass neue oder geänderte Anforderungen in Bezug auf die Sicherheitsziele hinterfragt werden müssen. Gegebenenfalls sind die Sicherheitsziele und daraus folgende Maßnahmen zu ergänzen oder anzupassen. Im Scrum-Prozess sind dies Aufgaben, die bei der ersten Befüllung des Backlogs sowie bei der Backlog-Pflege anfallen.

Architektur-Entscheidungen sind ein weiterer Moment, an dem das Thema „Security" auf dem Radar erscheint: Hier sind auch konzeptionelle Security-Entscheidungen zu treffen – sei es bei der Strukturierung der Anwendung oder der Auswahl der verwendeten Frameworks und Bibliotheken. Aus Security-Sicht sind solche Bibliotheken zu bevorzugen, die zum einen gut gepflegt sind und zum anderen möglichst gute Voreinstellungen haben („Secure by default"). Es soll den Entwicklern später so leicht wie möglich gemacht werden, Dinge richtig zu tun.

Bleibt noch das eigentliche Doing, also Implementierung und Testing. Hier ist das Team so zu schulen, dass es die verwendeten Tools und Bibliotheken richtig (also sicher) verwendet und auch ansonsten handwerkliche Fehler vermeidet. Dies wird nur gelingen, wenn bei allen Entwicklern ein Bewusstsein für die Probleme vorhanden ist. Einen guten Anhaltspunkt für einen Einstieg und die brennendsten Themen liefert das Open Web Application Security Project (OWASP): Wirft man einen Blick auf das Top-10-Projekt *(siehe „https://www.owasp.org/index.php/Category:OWASP_Top_Ten_2017_Project* und Vorjahre"), sind Probleme mit Injection-Angriffen und der Authentisierung am häufigsten. Cross-Site-Scripting (XSS) und Cross-Site-Request-Forgery (XSRF) spielen in der letzten Ausgabe der Top 10 in den hinteren Plätzen mit, bei Anwendungen mit großem Logik-Teil auf dem Browser (etwa bei der Anwendungsentwicklung mit Frameworks wie Angular) sollte ihnen jedoch ein besonderes Augenmerk zukommen.

Coaching mit Spaß

Ein Augenöffner ist häufig eine Demonstration, was mit Lücken wie SQL-Injection oder XSS möglich ist: Es ist schließlich eine Sache, ob man sich ein passwortloses Login erschleichen oder ein nerviges JavaScript-Browser-Popup öffnen kann – oder aber (mithilfe frei verfügbarer Tools) ein kompletter SQL-Dump extrahiert oder der Browser des Opfers als Angriffs-Trittbrett ins Firmen-Intranet oder auf weitere Webseiten genutzt wird.

Ebenfalls spannend ist der Wechsel in die Angreifer-Perspektive: Mit wenig Aufwand lassen sich Übungsumgebungen bereitstellen, mit denen man obige Angriffe einmal selbst ausprobieren kann. Entsprechend präparierte Anwendungen wie Damn Vulnerable Web Application (DVWA, siehe *„http://www.dvwa.co.uk")* oder OWASP WebGoat *(siehe „https://github.com/WebGoat/WebGoat")* lassen sich als Docker-Container rasch an den Start bringen. Wenn das Team daran etwas Spaß findet: Warum nicht einen kleinen Wettbewerb starten? Auf einem CTF-Server (Capture The Flag, z.B. FBCTF oder CTFd) sammeln Teams Punkte für sogenannte „Flags", die sie durch Ausnutzen von Lücken in einer anfälligen Anwendung finden. Der OWASP Juice Shop *(siehe „https://github.com/bkimminich/juice-shop")* bietet für eine solche Integration einen Extra-Startmodus.

Auch für die sehr trocken anmutende Phase des Threat Modeling gibt es spielerische Unterstützung: Ebenfalls auf den Seiten der OWASP findet man das Kartenspiel „Cornucopia" *(siehe „https://www.owasp.org/index.php/OWASP_Cornucopia").* Es ist ein einfaches Ablegespiel – möglicherweise nicht so spannend wie eine James-Bond-Poker-Runde, aber dennoch ein gutes Mittel, um die Diskussion über Sicherheitsanforderungen an ein Testprojekt (oder das eigene Projekt) anzuregen.

Fazit

Es wäre übertrieben zu erwarten, dass ein Entwicklerteam von heute auf morgen nur noch fehlerfreie Software schreibt; auch Security ist ein Lernprozess. Das Bewusstsein für die Probleme ist jedoch eine Grundvoraussetzung, die gelegt werden kann. Die hier vorgeschlagenen Methoden können dabei helfen, die ersten Schritte zu gehen.

Dr. Stefan Schlott
stefan.schlott@beone-group.com

Dr. Stefan Schlott ist Advisory Consultant bei der BeOne Stuttgart GmbH und dort als Entwickler, Architekt und Trainer im Java-Umfeld tätig. Security und Privacy gehören ebenso zu seinen Schwerpunkten wie skalierbare Architekturen und das breite Feld der verschiedenen Web-Technologien. In seiner Freizeit ist er auch als Dozent an der Dualen Hochschule Baden-Württemberg aktiv. Er begeistert sich für funktionale Programmierung sowie die Sprache Scala und ist überzeugter Open-Source-ler.