IBM Software
Rational

Dezember 2009

Der Weg zur sicheren Anwendung

Checkliste für Quellcode-Sicherheitsprüfungen

IBM

Inhalt

I. Einführung

Die immer häufigeren Hinweise auf Datenverletzungen aufgrund aktueller Bestimmungen zur Bekanntmachung von Datenverletzungen führen uns die Unsicherheiten in vielen der heutigen Anwendungen schmerzlich vor Augen. Wie können Unternehmen sicherstellen, dass ihre Anwendungen sicher sind und darüber hinaus die Kosten, das schlechte Bild in der Öffentlichkeit und den Rückgang des Aktienkurses vermeiden, die mit der Bereitstellung immer neuer Sicherheitspatches verbunden sind? Wie vermeiden Sie es, Endbenutzern und den zuständigen Regulierungsbehörden erklären zu müssen, dass Codefehler Hackern die Möglichkeit boten, sensible und unter Umständen regulierte Informationen zu stehlen?

Am Anfang der Entwicklung sicherer Anwendungen stehen umfassende Tests des Quellcodes auf alle Sicherheitslücken, um sicherzustellen, dass die jeweilige Anwendung nicht den Datenschutz und die Datenintegrität beeinträchtigt oder anderen Personen dies ermöglicht.

Für Unternehmen, die intern mit speziell angepassten, ausgelagerten oder Open-Source-Anwendungen arbeiten, stellt die Gewährleistung, dass der gesamte aktuelle und vorhandene Code sicher ist, allerdings eine enorme Herausforderung dar. Das Erkennen und Beseitigen von Sicherheitslücken war schon immer extrem schwierig. Viele Unternehmen verließen sich bisher auf manuelle Codeprüfungen, die teuer und arbeitsaufwendig sind, sowie auf das ethische Hacken, bei dem ein Teil der möglichen Schwachstellen in einer Anwendung geprüft werden.

Es ist bekannt, dass Unternehmen durch die frühzeitige Erkennung und Beseitigung von Sicherheitslücken in der Softwareentwicklung Risiken und Kosten erheblich verringern können.

Während beide Vorgehensweisen sicherlich nützlich sind, können Unternehmen auf automatische Tools zur Prüfung auf Sicherheitslücken in der Software zurückgreifen, um die Entwicklung sicheren Codes systematischer, automatisierter und erfolgversprechender auf den Weg zu bringen. Mit diesen automatischen Prüftools lassen sich Abläufe und die Genauigkeit von Codeprüfungen spürbar beschleunigen und verbessern. Sie können außerdem nahtlos in den Lebenszyklus der Softwareentwicklung integriert werden. Moderne Tools weisen hierbei auf Schwachstellen in einer bestimmten Codezeile hin und liefern detaillierte Informationen zur Art der Schwachstelle, den damit verbundenen Risiken und Möglichkeiten zur Fehlerbehebung.

II. Kostendruck zwingt Unternehmen zur Umstellung auf sichere Entwicklungsverfahren

Angesichts des immer schnelleren Aufkommens neuer Technologien ist die Notwendigkeit zur Entwicklung sicheren Codes dringlicher denn je. Dies schließt auch Web-Services und umfassende Internetanwendungen sowie die Notwendigkeit ein, die Integrität von vorhandenen, herkömmlichen und Mid-Development-Anwendungen in einem netzwerkorientierten Geschäftsumfeld sicherzustellen. Unternehmen sorgen auch weiterhin für die Integration ihrer Systeme in Systeme von Business Partnern, um den Austausch von Informationen zu beschleunigen. Unter den derzeitigen Bedingungen müssen die Unternehmen sicherstellen, dass der Code das nötige Maß an Sicherheit aufweist, um den Datenschutz zu gewährleisten, die bisherige Kundenbindung aufrechtzuerhalten, sensible Informationen zu schützen und die Integrität von Geschäftsprozessen zu erhalten.

Eine einzige Schwachstelle in der Software kann bereits zu Datenverletzungen führen. Einer der schwersten Fälle im Hochschulbereich wurde im Jahr 2006 an der University of California in Los Angeles (UCLA) festgestellt, als die dortige Datenbank mit persönlichen Informationen von 800.000 Personen Ziel eines Hackerangriffs war. Jim Davis, Associate Vice Chancellor of Information Technology (IT) an der UCLA, stellte fest, dass der Hacker eine einzige Schwachstelle in der Software für die News-Accounts nutzte, um Zugang zu erhalten. Der Hacker hat seinen Angriff darüber hinaus gut getarnt, denn er wurde möglicherweise erst nach einem Jahr überhaupt aufgedeckt. Die versehentliche Offenlegung sensibler oder privater und regulierter Informationen in einem Unternehmen kann Bußgelder, einen niedrigeren Aktienkurs und Imageverluste nach sich ziehen.

Zahlreiche Studien belegen, dass die Erkennung und Beseitigung von Sicherheitslücken im Code zu einem früheren Zeitpunkt im Lebenszyklus der Softwareentwicklung erheblich weniger kostet. Angesichts der Kosten eines einzigen Fehlers in veröffentlichtem Code, der zu Datenverletzungen führt, lassen sich problemlos die Gesamtkosten errechnen, die entstehen, wenn dieser Fehler nicht festgestellt wird. Studien belegen diesen Aspekt. Im Rahmen einer Umfrage unter 31 Unternehmen, in denen Datenverletzungen auftraten, wurden durchschnittliche Kosten von 3 Mio. GBP für folgende Aspekte ermittelt: Bereinigung von IT-Systemen, Anwaltskosten, Benachrichtigungen, Verlust von Kunden, Kosten für Überwachungsservices bei betroffenen Endkunden und höhere Aufwände für den Kundenservice. Durch diese Umfrage des Ponemon Institute© wurde außerdem festgestellt, dass aufgrund der Datenverletzungen durchschnittlich zwei Prozent der Kunden abwanderten, in einigen Fällen sogar bis zu sieben Prozent der Kunden.[1]

III. Der Weg zur Entwicklung sicheren Quellcodes

Welches ist der beste Weg sicherzustellen, dass der Code sicher ist? Um ein hohes Maß an Effektivität und Sicherheit in der Softwareentwicklung zu erreichen, müssen Prozesse zur Prüfung des Quellcodes im Hinblick auf folgende drei Ziele auf den Weg gebracht werden:

- **Einheitlichkeit schaffen:** Bei der Entwicklung von Code müssen die Entwickler einheitliche Prozesse, Richtlinien und die Rahmenbedingungen für ein höheres Maß an Sicherheit schaffen.
- **Zusätzliche sicherheitsspezifische Informationen zur Verfügung stellen:** Bei gefährlichen Sicherheitslücken sind größere Schwachstellen im Design normalerweise auf mehr als nur einzelne Codierungsfehler zurückzuführen. Die Beseitigung einzelner Sicherheitslücken ist nur wenig wirksam, wenn die Daten nicht verschlüsselt sind, die Authentifizierung schlecht ist oder die Anwendung Möglichkeiten zur Umgehung bietet.
- **Fehlerbehebung priorisieren:** Bei der Überprüfung des vorhandenen Codes müssen die Entwickler alle Schwachstellen im Code ermitteln und die größten Risiken zuerst beseitigen.

A. Der Weg zur sicheren Entwicklung: Stellen für die Suche nach Sicherheitslücken

Um sicherzustellen, dass der Code sicher ist, müssen alle Stellen geprüft werden, an denen Sicherheitslücken auftreten können. Selbst bei der Verwendung automatisierter Tools muss den Entwicklern bekannt sein, dass die Entwicklung sicherer Anwendungen möglicherweise auch die Überprüfung von Implementierungs- und Designverfahren umfasst (z. B. Verfahren zur Verwendung externen Codes und zur Wiederverwendung von Code), die anfänglich unter Umständen nicht als Schwachstelle in Betracht gezogen wurden. Unternehmen müssen bei der Entwicklung sicheren Codes daher sorgfältig vorgehen und sicherstellen, dass die Vielzahl von Stellen geprüft werden, an denen Sicherheitslücken in der Software auftreten können.

Damit alle mit einer bestimmten Anwendung verbundenen Risiken effektiv festgestellt werden, müssen Sicherheitsanalysten und Entwickler auf zwei Arten von Fehlern achten:

- **Fehler bei der Implementierung:**
  Diese Qualitätsfehler im Code treten recht selten auf und wirken sich normalerweise bei der Ermittlung und Beseitigung nicht auf andere Komponenten aus. Fehler bei der Implementierung werden durch fehlerhafte Programmierungsverfahren verursacht. Hierzu gehören z. B. „Buffer Overflows“ (Pufferüberläufe) aufgrund von schlechter Speicherverwaltung und der falschen Einschätzung von Wettbewerbssituationen, die aus Diskrepanzen beim Aufruftiming resultieren.
- **Designfehler:**
  Zu diesen Fehlern gehört die Nichtverwendung oder ungeeignete Implementierung sicherheitsspezifischer Funktionen. Beispiele: Authentifizierung, Verschlüsselung, Verwendung nicht sicherer externer Codetypen, Prüfung von Dateneingaben und Anwendungsausgaben.

Die häufigsten Fehler treten zwar bei der Implementierung auf, dennoch sind Sicherheitslücken im Design in den heutigen webfähigen Anwendungen mit den größten Risiken verbunden. Gibt es eine Zugriffskontrolle? Werden Verschlüsselungsfunktionen eingesetzt und sind diese zuverlässig genug? Erfolgt der Zugriff auf Datenbanken sicher und auf der Grundlage von Richtlinien? Diese Sicherheitsmängel auf Designebene müssen auf dem Weg zur sicheren Anwendungsentwicklung überprüft und beseitigt werden.

B. Der Weg zur sicheren Entwicklung: Vorgehensweise bei der Suche nach Sicherheitslücken

Bei der Ermittlung von Fehlern geht es nicht nur darum, die Notwendigkeit zur Sicherheit im Entwicklungsprozess besser zu definieren, sondern auch darum, alle Stellen im Code zu überprüfen, an denen Sicherheitslücken im Design auftreten können oder bereits auftreten. Von diesen Stellen gibt es üblicherweise weitaus mehr, als selbst erfahrene Entwickler vermuten würden. Bei einer umfassenden Analyse des Quellcodes müssen die Tester möglicherweise unvorhergesehene Stellen bearbeiten.

Um sicherzustellen, dass neuer und vorhandener Code auf sichere Weise entwickelt wird, sind Prozesse und Prozeduren für die Ermittlung von Schwachstellen und nützliche Tools erforderlich. Welche Tools eignen sich für die Entwicklung sicherer Software am besten?

Am häufigsten wird auf manuelle Codeprüfungen und ethisches Hacken zurückgegriffen. Beide Vorgehensweisen sind natürlich sinnvoll, reichen aber nicht aus, um alle vorhandenen und potenziellen Designfehler aufzudecken, und bieten daher keine Gewährleistung, dass der Code sicher ist. Manuelle Codeprüfungen sind z. B. sehr zeitaufwendig und teuer. Die Ermittlung der genauen Codezeilen, die Sicherheitslücken aufweisen oder zu Fehlern in Geschäftsprozessen führen können, ist äußerst schwierig. Mit ethischem Hacken lässt sich nur ein Teil der Fehler ermitteln, die in einer Anwendung enthalten sein können. Diese Vorgehensweise bietet sich zwar für das Aufdecken solcher Fehler an, liefert aber nur einen unvollständigen Überblick über den gesamten Grad an Sicherheit einer Anwendung.

Angesichts der Vielzahl an vorhandenen Designfehlern oder potenziellen Fehlern weisen Experten darauf hin, dass Unternehmen automatische Tools zur Ermittlung von Sicherheitslücken in der Software einsetzen sollten, um tatsächlich alle potenziellen Sicherheitslücken zu erkennen. Sie stimmen darin überein, dass für Unternehmen bei der Entwicklung eigener Software die effektivste Vorgehensweise darin besteht, Prüfungen auf Sicherheitslücken im Quellcode in Prozesse für Anwendungsentwicklung, Integration und Tests aufzunehmen. Nur mit hochentwickelten Testtools zum Auffinden von Schwachstellen im Quellcode und den zugehörigen Verfahren im Lebenszyklus der Softwareentwicklung kann effizient und effektiv sichergestellt werden, dass der Code sicher ist.

III. Schritte auf dem Weg zum sicheren Quellcode: Zu prüfende Komponenten

Das effizienteste und effektivste Verfahren zur Entwicklung sicheren Quellcodes besteht darin, vorhandene Anwendungen und den in Entwicklung befindlichen Code auf der Grundlage der folgenden fünf Kategorien von Sicherheitslücken im Code zu beurteilen:

1. Sicherheitsspezifische Funktionen
2. Fehler bei der E/A-Prüfung und -Verschlüsselung
3. Sicherheitslücken bei der Fehlerbehandlung und -protokollierung
4. Nicht sichere Komponenten
5. Codierungsfehler

Durch die Verfolgung sicherheitsspezifischer Fehler im Quellcode einer Anwendung verringern sich die Zahl der Sicherheitslücken in der Anwendung und die Menge an damit verarbeiteten und geschützten kritischen Daten erheblich.

Im Folgenden wird jede Fehlerkategorie ausführlicher erläutert.

1. Sicherheitsspezifische Funktionen

Anwendungen setzen sich aus zahlreichen separaten Funktionen zusammen, die häufig unbedenklich erscheinen. Wenn diese Funktionen allerdings auf unvorsichtige Weise miteinander kombiniert werden oder die Implementierung dieser Funktionen nicht dokumentiert wird, können sich leicht Sicherheitsrisiken ergeben, die zu Mängeln in puncto Datenschutz, Vertraulichkeit oder Systemintegrität führen.

Diese Risiken ergeben sich aufgrund der häufigen Verwendung höherer Programmiersprachen sowie insbesondere vordefinierter Module und vorab kompilierter Bibliotheken. Mithilfe dieser Tools können Entwickler Anwendungen mit vollem Funktionsumfang innerhalb kurzer Zeit implementieren, über die auf eine Reihe von Services und Datenquellen zugegriffen wird. Mit den Tools lassen sich außerdem viele Sicherheitsprobleme vermeiden, z. B. indem die Entwickler von Problemen mit der Speicherverwaltung und Ressourcenkontrolle durch die Verwendung allgemeiner Schnittstellen entlastet werden.

Für die Verwendung dieser Tools sind keine umfangreichen Kenntnisse zur sicheren Nutzung von Services und Daten erforderlich, bzw. darüber, ob dies zu Konflikten mit bestehenden Geschäftsprozessen oder der Infrastruktursicherheit im Unternehmen führt. Dies bedeutet, dass die meisten Sicherheitsprobleme in heutigen Anwendungen eigentlich auf die nicht sichere Implementierung dieser Bibliotheken oder Module zurückgeführt werden kann. Außerdem stellen diese Arten von Fehlern im Sicherheitsdesign häufig erheblich größere Risiken dar als die bisherigen Arten von Codierungsfehlern. Der Grund hierfür ist, dass die heutigen Anwendungen in vielen Fällen sowohl mit nachgelagerten Prozessen im Unternehmen als auch mit dem Internet verknüpft sind, und damit eine potenzielle Quelle für den Verlust sensibler Daten schaffen.

Bei einer umfassenden Ermittlung des Sicherheitsstatus einer Anwendung muss eine Analyse der folgenden kritischen Schwachstellen im Design durchgeführt werden:

a. Schlechte oder vom Standard abweichende Verschlüsselung

Eine der grundlegenden Komponenten bei der Anwendungssicherheit ist die Verschlüsselung von Daten. Nicht öffentliche oder sensible Daten werden verschlüsselt, um sie besser schützen zu können. Wenn Hacker es schaffen, die Verschlüsselungsalgorithmen zu knacken, können sie sensible Daten stehlen. Zwei weit verbreitete Schwachstellen bei der Verschlüsselung, die es Hackern ermöglichen, sensible Daten zu stehlen, sind die Verwendung schlechter Zufallszahlengeneratoren und vom Standard abweichender Verschlüsselungsalgorithmen.

Um eine wirksame Verschlüsselung zu erreichen, muss die zugrundeliegende Verschlüsselungsfunktion auf einer ausreichenden Zufälligkeit basieren, damit sichergestellt ist, dass Hacker nicht problemlos die Verschlüsselungsschlüssel erraten oder reproduzieren können, die für den Austausch von Daten verwendet werden. Schlechte Zufallszahlen sind solche, die nicht ausreichend zufällig sind. Wenn solche nicht ausreichend zufälligen Zahlen als „Key Seeds“ verwendet werden, bietet der Verschlüsselungsalgorithmus die Möglichkeit, Folgenummern für verschlüsselte Daten vorherzusagen und Sitzungen zu fälschen (Spoofing). Entwickler müssen daher den Einsatz schlechter Zufallszahlengeneratoren vermeiden.

Fehlende Verschlüsselung

Achten Sie bei der Analyse der Sicherheit des Codes insbesondere auf das Vorhandensein von Verschlüsselungsfunktionen, auf deren richtige Implementierung sowie auf unsachgemäße Versäumnisse. Die fehlende Verschlüsselung sensibler Informationen hat in der Vergangenheit bereits zu zahlreichen Datenverlusten geführt. Hierzu gehört z. B. ein Zwischenfall im United States Department of Veterans Affairs (VA), der im Mai 2006 bekannt wurde. In einer der Anwendungen des Ministeriums wurden die Sozialversicherungsnummern und Adressen aller pensionierten Veteranen abgelegt. Dann wurde allerdings der Laptop eines Mitarbeiters gestohlen, auf dem diese Daten gespeichert waren, und die Daten von 38,6 Millionen Veteranen waren damit gefährdet.
Mithilfe einer Codeprüfung können alle sensiblen Informationen ermittelt werden, die auf nicht sichere Weise abgelegt wurden. Dies kann zu der Frage führen, weshalb diese Daten überhaupt auf einem Laptop gespeichert wurden. Was spricht gegen sichere Aufrufe einer zentralen Datenbank, die durch strenge Zugriffskontrollen und Überwachungsfunktionen geschützt ist? Möglicherweise die Tatsache, dass das Erstellen einer Anwendung auf einem Laptop, in der Informationen in einem nicht sicheren Format abgelegt werden, einfacher ist.

Achten Sie auf vom Standard abweichende Verschlüsselungsalgorithmen. Daten werden über Verschlüsselungsalgorithmen verschlüsselt. Es gibt aber nur wenig wirklich sichere Algorithmen, die von Verschlüsselungsexperten umfassend beurteilt wurden. Selbst für Experten ist die Entwicklung eines wirklich sicheren und akzeptablen Algorithmus extrem schwierig, der sich für die kontinuierliche Verwendung von Triple DES, Blowfish und anderer gängiger Algorithmen eignet. Auch solche Algorithmen werden in einigen Fällen zu einem späteren Zeitpunkt geknackt. Richten Sie sich dennoch nach Empfehlungen von Experten. Andere Algorithmen sind möglicherweise nicht zuverlässig genug, um Hacker von der Entschlüsselung verschlüsselter Daten abzuhalten.

b. Nicht sichere Netzwerkkommunikation
Wenn Sie berechtigte Methode zum Senden oder Empfangen von Daten verwenden, diese Prozesse aber nicht dokumentieren oder schützen, sind kritische Daten bei der Übertragung Risiken ausgesetzt, da sie auf einfache Weise abgefangen und gelesen werden können. Entwickler müssen so häufig wie möglich sichere Netzwerkübertragungsprotokolle einsetzen (z. B. Secure Sockets Layer, SSL) und diese Prozesse sorgfältig dokumentieren.

c. Sicherheitslücken in der Anwendungskonfiguration
Entwickler müssen sicherstellen, dass die Konfigurationsdateien oder -optionen zur Kontrolle der Anwendungen ebenfalls geschützt sind. Andernfalls können Hacker auf ungeschützte Dateien oder Optionen zugreifen, diese manipulieren und Softwaremerkmale oder Kontrollfunktionen für den Zugriff auf sensible Daten anpassen.

d. Sicherheitslücken bei der Zugriffskontrolle
Ohne Zugriffskontrollen können Hacker mit Netzwerkzugriff (z. B. böswillige Insider) problemlos auf vertrauliche Daten und Ressourcen zugreifen. Unternehmen müssen daher zuverlässige Verfahren einsetzen, um Benutzer zu identifizieren, identifizierte Benutzer Daten zuzuordnen und sicherzustellen, dass Benutzer nur auf die zugeordneten Daten zugreifen können.

Eine Schwachstelle hierbei besteht darin, wenn eine Anwendung einem Benutzer oder einer Anwendung umfassendere Zugriffsberechtigungen als erforderlich erteilt. Je nach Level der erteilten Zugriffsberechtigung können Benutzer dadurch auf vertrauliche Informationen zugreifen oder sogar das gesamte System kontrollieren. Achten Sie daher darauf, dass bei allen Anwendungen, über die sensible Daten verarbeitet werden, das Prinzip der geringstmöglichen Zugriffsberechtigung eingehalten wird. Erteilen Sie nur den Mindest-Zugriffslevel, der erforderlich ist, damit ein Benutzer oder eine Anwendung reibungslos arbeiten können. Hierbei müssen allerdings die unterschiedlichen Berechtigungen ermittelt werden, die eine Anwendung oder ein Benutzer dieser Anwendung benötigt, um Vorgänge durchführen zu können, und für alle zugehörigen Module und Objekte darf nur dieser Berechtigungslevel vergeben werden.

Beachten Sie bei der Beurteilung von Zugriffskontrollen auch folgende andere Sicherheitslücken:

- **Verwendung nicht geschützter Datenbank- und Dateisysteme:** Sie müssen sicherstellen, dass Tools zur Anwendungssicherheit Aufrufe von Datenbanken und Dateisystemen sorgfältig im Quellcode der Anwendung prüfen. Andernfalls können Hacker diese Aufrufe manipulieren, um auf sensible Daten zuzugreifen.
- **Sicherheitslücken in dynamischem Code:** Wird über die Anwendung dynamischer Code geladen? Hacker können bösartige Befehle in Anwendungen einfügen, über die dynamischer Code geladen wird. Sie müssen daher sicherstellen, dass diese Anwendungen zunächst dahingehend überprüft werden, ob solche Sicherheitslücken vorhanden sind.
- **Laden externen Codes:** Hacker können durch die Manipulation nicht ausreichend überprüfter Aufrufe auf Systemebene Daten ändern, offenlegen oder zerstören.
- **Sicherheitslücken im Datenspeicher:** Warum sollte ein Hacker eine Anwendung angreifen, wenn die Daten nicht geschützt werden? Hacker können auf Server zugreifen, um sensible Daten zu stehlen, wenn diese nicht sicher abgelegt werden.
- **Fehler bei der Authentifizierung:** Hacker verwenden die Anmeldedaten berechtigter Benutzer oder überlisten ein System derart, dass nur scheinbar berechtigte Anmeldedaten verwendet werden, um Daten zu stehlen oder zu manipulieren. Bei einem Angriff auf ChoicePoint, Inc© konnten Hacker z. B. tatsächliche Benutzer-Anmeldedaten erlangen und damit böswillige Aktivitäten verschleiern.

Ausnutzen von Sicherheitslücken in der Zugriffskontrolle: Forced Browsing

Achten Sie bei der Überprüfung des Codes besonders auf Forced Browsing. Hierbei handelt es sich um Folgendes: Hacker führen eine Anforderung direkt für eine Webseite aus, auf die sie möglicherweise keinen Zugriff haben. Wenn die bestehende Zugriffskontrolle nicht ausreicht, können sie unter Umständen auf diese Webseite oder die nachgelagerten Ressourcen zugreifen und diese Ressourcen stehlen oder beschädigen.
Um solche Angriffe zu verhindern, müssen Entwickler sicherstellen, dass keine Webseiten mit sensiblen Informationen im Cachespeicher auf den Servern oder den lokalen PCs der Benutzer abgelegt werden. Für alle diese Daten können nur Anforderungen gestellt werden, die ein aktives und authentifiziertes Sitzungstoken aufweisen und einem Benutzer zugeordnet wurden, der die erforderlichen Berechtigungen für den Zugriff auf die Seite oder die Informationen hat.

2. Fehler bei der E/A-Prüfung und -Verschlüsselung

Die meisten Anwendungen erfordern dynamische Eingaben. Alle Eingaben können zu Sicherheitslücken führen und müssen daher geprüft werden, um sicherzustellen, dass die Anwendung aufgrund des Formats oder des Umfangs der Eingabe nicht unvorhersehbar reagiert. Alle Eingabequellen – insbesondere solche, die über Benutzertransaktionen bereitgestellt werden – müssen nach dem Einfügen in das System und vor dem Erreichen der Position, an der Sie verwendet werden, geprüft werden. Die Entwickler müssen sicherstellen, dass alle Eingaben geprüft werden und den Erwartungen entsprechen. Wenn dies nicht der Fall ist, muss eine Anwendung verhindern, dass fehlerhafte Eingaben über die Anwendung weiterverarbeitet werden.

Um ein besseres Verständnis dafür zu bekommen, wann und in welchen Fällen E/A-Prüfungen eingesetzt werden sollten und in welchen Fällen sie besonders notwendig ist, bietet es sich an, einige der wichtigsten damit zusammenhängenden Angriffe und Vorgehensweise zu deren Verhinderung zu erläutern:

a. Sicherheitslücken durch SQL-Injections (Structured Query Lanaguage)

Eine gängige Schwachstelle bei der Prüfung von Eingaben ist eine SQL-Injection. Hacker wendeten dieses Verfahren an, um eine Vielzahl von Kreditkarteninformationen aus Datenbanken zu stehlen. Wie der Name bereits vermuten lässt, führen Hacker hierbei unsachgemäße SQL-Abfragen in einer Datenbank durch, um unerlaubt auf Daten zuzugreifen oder eine unzuverlässige Funktionsweise der Datenbank zu verursachen. Die wirksamste Methode, um Angriffe durch SQL-Injections zu stoppen, besteht darin, ausschließlich gespeicherte Prozeduren oder parametrisierte Datenbankaufrufe zu verwenden, bei denen kein Code eingefügt werden kann. Mit diesem einfachen Verfahren können Angriffe durch SQL-Injections gestoppt werden.

b. XSS-Sicherheitslücken

XSS-Angriffe richten sich auf die Verwendung nicht ausreichend geschützter Ausgabeverfahren in Anwendungen. Hacker können sich diese Verfahren zunutze machen, um ahnungslose Benutzer dazu zu bringen, bösartigen Code auszuführen oder darauf zuzugreifen.

XSS-Angriffe werden im Allgemeinen in zwei Kategorien unterteilt:

- **Stored Attacks:** Der eingefügte Code wird dauerhaft in einer Datenbank, einem Nachrichtenforum oder einem Besucherprotokoll auf dem Zielserver gespeichert.
- **Reflected Attacks:** Der eingefügte Code erreicht das vorgesehene Opfer möglicherweise über eine E-Mail-Nachricht oder einen anderen Server. Sobald ein Benutzer überlistet wurde, weil er auf einen Link geklickt oder ein Formular übermittelt hat, wird der eingefügte Code auf den fehleranfälligen Web-Server übertragen, über den der Angriff im Browser des Benutzers angegeben wird. Der Browser führt diesen Code anschließend aus, da er von einem vertrauenswürdigen Server stammt.

Die schwerwiegendsten XSS-Angriffe führen zur Offenlegung des Sitzungscookies eines Benutzers, so dass ein Hacker die Möglichkeit zum Hijacking bei dieser Benutzersitzung hat und den zugehörigen Account übernehmen kann. Andere schwerwiegende Angriffe sind die Offenlegung von Endbenutzerdateien, die Installation von Trojaner-Anwendungen, das Umleiten eines Benutzers auf eine andere Webseite oder Website, auf der ein Phishing-Angriff und modifizierende Inhalte eingerichtet wurden. Um sich vor solchen XSS-Angriffen zu schützen, müssen alle Benutzerausgaben zum Kunden und Protokolldateien mit HTML Entity Encoding geschützt werden. Bei dieser Art der

Cross-Site Scripting (XSS) für Angriffe auf andere Websites

XSS-Angriffe stellen ein hohes Risiko für verbraucherorientierte Websites dar. Dies gilt insbesondere für Social Networking-Websites, auf denen bei der Kommunikation eher auf moderne Multimediafunktionen und nur ein Minimum an Sicherheit Wert gelegt wird.
Auf MySpace.com© wurde im Jahr 2006 z. B. ein Wurm festgestellt, der mit unglaublich bösartigen Merkmalen durch einen Sicherheitsanalysten synchronisiert wurde. Bei diesem XSS-Angriff wurde eine Sicherheitslücke in Apple© QuickTime ausgenutzt (obwohl Apple diese ursprünglich nicht als solche erkannte), über die JavaScript von einer externen Quelle aus ohne Vorwarnung in einem Benutzerprofil beschädigt werden konnte. Dadurch war es letztendlich möglich, die Anmeldedaten von MySpace-Benutzern zu stehlen und Spamangriffe über MySpace auszuführen. Eine Sicherheitslücke in einem Multimedia Player ermöglichte diesen umfassenden Angriff.

Verschlüsselung werden alle nicht alphanumerischen Zeichen in eine spezielle Zeichenfolge umgewandelt, die von HTML-fähigen Anzeigeprogrammen nicht interpretiert werden kann.

c. Sicherheitslücken durch Betriebssystem-Injections
Anwendungen benötigen in einigen Fällen Zugriff auf Befehle auf Betriebssystemebene. Seien Sie vorsichtig, wenn diese Zugriffe auch Benutzereingaben umfassen, da Hacker wie bei SQL-Injections fehlerhafte Eingaben verwenden können, um das Verhalten des Betriebssystems zu beeinflussen. Dies führt zu möglichen Datenverlusten oder einer Beeinträchtigung des gesamten Systems.

d. Manipulation individueller Cookies oder versteckter Felder
Cookies sind nützlich und ein gängiges Mittel, um Benutzerinformationen in Sitzungen und darüber hinaus zu verwalten. Die Erstellung individueller Cookies mit angepassten Namen und Werten ist gefährlich, da dies Entwickler häufig dazu verleitet, sich zu sehr auf diese nicht sicheren Cookies zu verlassen, denn Hacker können Cookies auf einfache Weise modifizieren. Wenn Entwickler diese Cookies nicht prüfen, schaffen Sie unter Umständen eine Möglichkeit für Hacker, SQL-Injections anzulegen oder erfolgreiche XSS-Angriffe durchzuführen.

Web 2.0: Ein Mangel an Sicherheit im World Wide Web

Durch das stark weborientierte Arbeiten im heutigen Geschäftsumfeld kann auf mehr Daten denn je über das Internet zugegriffen werden. Technologien wie z. B. AJAX ermöglichen Softwareentwicklern einen einfacheren Zugriff auf Informationen und können zu einer erheblich umfassenderen Funktionalität für den Endbenutzer beitragen. Der Faktor Sicherheit steht bei diesen Technologien aber nach wie vor nicht im Vordergrund. Unternehmen müssen bei der Einführung neuer Tools und Entwicklungsverfahren besonders auf die Codequalität und auf Sicherheitslücken achten.
Eine Neuerung bei Web 2.0 sind z. B. Mashups, in denen Inhalte von einem Host-Server mit öffentlich verfügbaren Feeds kombiniert werden. Bei neuen Funktionen können sich allerdings auch neue Formen von website-übergreifendem Missbrauch ergeben. Wenn Web 2.0 Mashups nicht auf sichere Weise erstellt werden, gibt es aufgrund erheblicher Sicherheitslücken genügend Spielraum für neue Formen von Phishing und anderen Angriffen.

3. Sicherheitslücken bei der Fehlerbehandlung und -protokollierung

Ist Ihre Anwendung kontrolliert ausgefallen? Fehlerbehandlung und -protokollierung sind für Entwickler besonders schwierig: Wer kann alle Arten oder die damit verbundenen Konsequenzen vorhersagen, auf die eine Anwendung ausfallen kann?

Die Verfolgung der Fehlerbehandlung und -protokollierung für Anwendungen ist jedoch von entscheidender Bedeutung, da enorm viele Angriffe heutzutage erfolgreich sind, indem fehlerhafte Daten in Anwendungen eingefügt und das daraus resultierende Fehlverhalten der Anwendung ausgenutzt wird.

Berücksichtigen Sie bei der Beurteilung einer Anwendung nach der zugehörigen Fehlerbehandlung zunächst folgende zwei Aspekte:

a. Nicht sichere Fehlerbehandlung
Eine schlechte Fehlerbehandlung kann Hackern wichtige Informationen zur Durchführung von Angriffen liefern. Nicht voneinander getrennte Fehlerroutinen liefern z. B. wertvolle Einblicke in die Verarbeitung von Eingaben durch eine Anwendung, insbesondere, wenn die Fehlerroutine im Hinblick auf bestimmte Fehler und Datenelemente individuell erstellt wurde. Entwickler müssen die Anzahl an Details begrenzen, die für Benutzer der Anwendung transparent sind. Arbeiten Sie nicht mit Anwendungen, die in Verknüpfung mit einem Web-Server Fehler auf dem Server anzeigen. Ein externer Hacker kann die daraus resultierenden Informationen nutzen, um Zugriff auf die Systeme zu erhalten.

b. Nicht sichere oder unzureichende Protokollierung
Protokolldateien können für das Tracking des Verhaltens von Anwendungen von entscheidender Bedeutung sein, sie stellen aber auch eine riesige Informationsquelle für Hacker dar. Achten Sie darauf, keine Zugriffsberechtigungen für Protokolldateien einzurichten. Protokollieren Sie gegebenenfalls das Verhalten von Anwendungen, mit denen sensible Daten verarbeitet werden, um sicherzustellen, dass Hacker ihre Spuren nicht verwischen können.

4. Nicht sichere Komponenten

Fehlerhafter Code kann entweder durch Böswilligkeit oder versehentlich mangelhafte Verfahren bei der Codeentwicklung Teil einer Anwendung werden. So kann ein Hacker z. B. bösartigen Code in eine Anwendung einfügen, um bestehende Sicherheitsmaßnahmen zu umgehen. Unglücklicherweise entspricht bösartiger Code in vielen Fällen exakt dem nicht bösartigen Code. Über beide Arten von Code ist der Zugriff auf Netzwerke und Daten möglich, und beide Arten funktionieren üblicherweise wie andere Teile einer Anwendung.

Die automatische Identifizierung bösartigen Codes ist daher extrem schwierig, wenn Entwickler ausschließlich die Funktionalität analysieren. Sie müssen ihr Augenmerk vielmehr auf den Ort richten. Identifizieren Sie zunächst bestimmte Arten von Funktionen (z. B. Netzwerkübertragungen, zeitlich gesteuerte Ereignisse und Änderungen an Berechtigungen) und ordnen Sie diese anschließend jeweils dem Anwendungsmodul zu, in dem sie eingesetzt werden. Achten Sie dabei auf Diskrepanzen, wie z. B. eine Grafikbibliothek, die abgehende Netzwerkvorgänge durchführt, oder fest codierte zeitlich gesteuerte Ereignisse in einer weitestgehend in Echtzeit arbeitenden Anwendung. Solche Diskrepanzen sind Warnhinweise auf böswillige Absichten.

Die heutigen Verfahren bei der Anwendungsentwicklung sind stark komponentenbezogen. Während dies die Möglichkeit bietet, sichere Anwendungen schneller zu entwickeln, stellen zwei Arten von Komponenten erhebliche Sicherheitsrisiken dar:

a. Nicht sichere Java Native Interface-Methoden
Wenn Entwickler nicht sichere Java Native Interface-Methoden (JNI) für ihren Code verwenden, könnten Hacker auf einfache Weise auf kritische Ressourcen zugreifen, z. B. den System- oder Umgebungsspeicher.

Was sind JNI-Methoden? Hierbei handelt es sich um allgemeine Sprachen, die Schnittstellen zur Prüfung aller Zugriffe auf Ressourcen bereitstellen. Im Gegensatz dazu sind JNI-Methoden einer eher grundlegendes Mittel für den Zugriff auf Ressourcen, die diese Schnittstellen umgehen und damit bessere Leistungsmerkmale bieten. Da der Code ohne Prüfungen auf Schnittstellenebene geschrieben wird, ist das Risiko für fehlerhaften Code extrem hoch. Im Allgemeinen sollten Sie die Verwendung von JNI-Methoden vermeiden, außer in Ausnahmefällen, in denen Leistungsmerkmale die entscheidende Rolle spielen. Lokalisieren Sie in jedem Fall die verwendeten JNI-Methoden und testen Sie diese sorgfältig. Führen Sie sich darüber hinaus die vielen Möglichkeiten vor Augen, in denen JNI-Bibliotheken interne Fehler aufweisen können. Dies gilt insbesondere für die Programmierung in C oder C++, die anfälliger für Probleme durch Buffer Overflows oder Konkurrenzsituationen (Race Conditions) ist.

b. Nicht unterstützte Methoden
Entwickler entscheiden sich in einigen Fällen für Schnellverfahren und verlassen sich bei der Entwicklung von Anwendungen auf nicht unterstützte Methoden oder Aufrufe. Die Verwendung nicht dokumentierter Funktionen oder Routinen kann jedoch versteckte Risiken nach sich ziehen, die es Hackern ermöglichen, eine Anwendung auszunutzen. Stellen Sie daher sicher, dass bei allen Softwareprojekten in allen Vereinbarungen angegeben ist, dass ausschließlich unterstützte Methoden verwendet werden. Ermitteln Sie für die vorhandenen Anwendungen alle nicht unterstützten Methoden, entfernen Sie diese und ersetzen Sie sie durch unterstützte Methoden.

5. Codierungsfehler

Codierungsfehler, die auch als Implementierungsfehler bezeichnet werden, können durch nicht ausreichend geschulte Entwickler, dicht gedrängte Projektpläne, eine unzureichende Priorisierung der Projektvoraussetzungen oder die Wiederverwendung von Code mit fraglicher oder nicht bekannter Qualität verursacht werden.

Codierungsfehler in Anwendungen können zu unerwarteten Funktionsweisen führen, z. B. Übertragung der Kontrolle für ein System oder einen Prozess an einen Hacker oder das Beenden einer Anwendung. Aufgrund der mit Codierungsfehlern verbundenen Sicherheitsrisiken müssen solche Fehler aus dem Code entfernt werden, unabhängig davon, ob dies in der Entwicklung oder bereits in der Produktionsumgebung erfolgt.

Achten Sie bei der Analyse von Code auf folgende codespezifische Sicherheitsrisiken:

a. Sicherheitslücken durch „Buffer Overflow“ (Pufferüberlauf)
Buffer Overflows treten auf, wenn mehr Daten in einen Puffer kopiert werden, als dieser aufnehmen kann. Obwohl es bereits seit mehr als 20 Jahren Erkenntnisse zu Buffer Overflows gibt, sind sie weiterhin ein gängiges Problem, das extrem hohe Risiken mit sich bringt. Hacker können sich diese Art der fehlerhaften Speicherverwaltung zunutze machen, um Code in

Verdächtige Merkmale

Hinweise, dass eine Anweisung möglicherweise bösartigen Code enthält:

Merkmal	Hinweis auf...
Raw-Socket-Zugriff	Mögliche versteckte Zugriffe
Timer oder Get Time-Funktion	Trigger
Änderungen an Berechtigungen	Nicht autorisierte Zugriffsstufen

den Computerspeicher zu laden und auszuführen, der es dem Hacker ermöglicht, die Kontrolle über das gesamte System zu erlangen.

In strukturierten Programmiersprachen wie C und C++ gibt es in der Tat Hunderte von Aufrufe oder Kombinationen aus Aufrufen, bei denen Speicher falsch zugeordnet werden kann oder es keine ausreichenen Erkenntnisse zum gesamten Anwendungsverhalten gibt. Diese Komplexität sorgt für ideale Bedingungen bei Angriffen über Buffer Overflows.

In höheren Programmiersprachen wie Java, JavaServer Pages (JSP), C# und .Net sind Sicherheitsrisiken weit weniger geläufig, da die gesamte Speicherverwaltung auf Systemebene über die Programmiersprache und deren Interpretation bei Laufzeit erfolgt und von Einflüssen durch die Programmierer geschützt ist. Aber selbst in höheren Programmiersprachen kann es bestimmte Aufrufe geben, die in einigen Fällen nicht dokumentiert sind, die ein Sicherheitsrisiko durch Buffer Overflows darstellen, das dem Programmierer nicht bekannt ist. Achten Sie daher in allen höheren Programmiersprachen insbesondere auf Buffer Overflows.

b. Sicherheitslücken durch Formatierungszeichenfolgen
Sicherheitslücken durch Formatierungszeichenfolgen verdeutlichen die Abhängigkeiten zwischen Implementierungsfehlern sowie die Codequalität insgesamt. Häufig werden Sicherheitslücken durch Formatierungszeichenfolgen als eine Art „Buffer Overflow“ betrachtet, dieser Vergleich trifft aber nicht hundertprozentig zu. Diese Sicherheitslücken können zwar zu einem Buffer Overflow führen, allerdings auch ohne Buffer Overflow eine Offenlegung von Informationen nach sich ziehen.

Bei der Codeentwicklung kann die unvollständige Verwendung bestimmter Aufrufe zu Sicherheitslücken durch Formatierungszeichenfolgen führen. Bei guten Codierungsverfahren ist die einheitliche Verwendung bestimmter Argumente (z. B. Feldkennungen) Voraussetzung. Wenn Entwickler Code dagegen Code auf der Grundlage unvollständiger Aurufe entwickeln, werden diese Aufrufe nicht ausreichend verknüpft, so dass Hacker gegebenenfalls zusätzliche Daten, Argumente oder Informationsanforderungen in diese Aufrufe einfügen können. Überprüfen sie daher den gesamten Code auf unvollständige Aufrufe und vervollständigen Sie diese.

c. Denial-of-Service-Fehler
Alle Anwendungen, die den Zugriff auf kritische Daten oder Services bereitstellen, können dies nur ermöglichen, wenn sie ausgeführt werden. Bei Denial-of-Service-Angriffen (DoS) werden Anwendungen beeinträchtigt und die Bereitstellung kritischer Daten oder Services wird beeinflusst.

Sicherheitsrisiken durch Denial-of-Service-Fehler beziehen sich auf Implementierungsfehler, die entweder zur Ausschöpfung knapper, begrenzter oder nicht erneuerbarer Ressourcen oder zur Zerstörung oder Änderung von Konfigurationsinformationen führen. Um solche Fehlerbedingungen zu vermeiden, müssen Entwickler Anwendungen so konzipiert werden, dass sie selbst in Worst-Case-Szenarios ausgeführt werden.

d. Sicherheitslücken durch zusätzliche Berechtigungen
In vielen Fällen zielen Angriffe durch Hacker letztendlich auf zusätzliche Berechtigungen ab. Ein Benutzer mit nicht ausreichenden Berechtigungsnachweisen erhält Sonderzugriffsrechte, so dass Hacker auf vertrauliche Daten und Ressourcen zugreifen und sogar die Kontrolle über das gesamte System übernehmen und es zerstören können. Sie sind hierbei die ganze Zeit als vertrauenswürdiger Benutzer eingestuft.

Hacker weiten ihre Berechtigungen üblicherweise durch die Ausnutzung von Programmabschnitten aus, in denen eine Anwendung umfassendere Berechtigungen erteilt oder erhält. Anwendungen müssen in einigen Fällen Prozesse mit unterschiedlichen Benutzernamen oder Zugriffsrechten erstellen. Es liegt in der Verantwortlichkeit der Entwickler sicherzustellen, dass der Prozess zur Erteilung zusätzlicher Berechtigungen nicht von betrügerischen Programmen ausgenutzt werden kann. Auch der Prozess zur Einschränkung von Berechtigungen muss natürlich umfassend getestet werden.

Wenn ein Prozess mit niedrigerer Priorität die Kontrolle wieder an ein Programm mit höherer Berechtigung abgibt, muss die Anwendung durchgängig prüfen, ob gültige Rückkehrcodes, Fehlerbedingungen und Vorgänge zur Herabsetzung der Berechtigung durch Fehler ausgelöst wurden. Wenn einer dieser Vorgänge fehlschlägt, können die Programme auf einem anderen Level oder mit einer anderen Berechtigung weiterhin ausgeführt werden, der/die beabsichtigt oder erforderlich war.

e. Konkurrenzsituationen (Race Conditions)
Zwei Prozesse können die Steuerung gemeinsam übernehmen oder Daten gemeinsam nutzen. Wenn bei der gemeinsamen Nutzung Konflikte auftreten, wird dies als Konkurrenzsituation betrachtet. Sie resultiert normalerweise aus Fehlern bei der Synchronisation, wenn Prozesskonflikte auftreten können und sich daraus eine Sicherheitslücke ergibt. Die Ausnutzung solcher Situationen geschieht üblicherweise, indem ein Paar aufeinanderfolgender Aufrufe unterbrochen werden, die zusammen mit einem dritten Prozess automatisch und ohne Unterbrechung durch einen anderen Thread oder Prozess auf dem System ausgeführt werden sollen.

Ein Beispiel hierfür ist das kombinierte Prüfen von Zugriffsrechten für eine Datei, gefolgt von einem nachfolgenden Aufruf, Daten in diese Datei zu schreiben oder daraus zu lesen. Durch die Unterbrechung des Prozesses zwischen den beiden Aufrufen können Hacker den Inhalt der Datei umschreiben oder ändern, da dieses Verhalten erwartet wird. Hacker können in diesem Fall schädliche Informationen in eine Datei schreiben oder möglicherweise unberechtigt auf eine Datei zugreifen.

Gängige Denial-of-Service-Fehler (DoS)

Denial-of-Service durch Herunterfahren der Anwendung: Wie erfolgt das Herunterfahren der Anwendung? Einige Anwendungsentwickler führen diesen Vorgang bei der Implementierung von Funktionen zur Beendigung einer Anwendung zu häufig durch. Wenn eine Anwendung z. B. aufgrund von Eingabefehlern über eine Systemexit-Funktion automatisch geschlossen wird, können Hacker ähnliche Ereignisse verursachen, die auf unnatürliche Weise und unnötigerweise zu einer Beendigung der Anwendung führen.
Denial-of-Service durch nicht beendete Datenbankverbindungen: Wie ordnungsgemäß stellt die Anwendung Verbindungen zu Datenbanken her? Damit einem Prozess der Zugriff auf Daten gewährt werden kann, muss zunächst eine Verbindung zwischen Anwendung und Datenquelle hergestellt werden. Wenn dieser Prozess in ungeeigneter Weise codiert wurde, kann dies zu einer vollständigen Auslastung der Warteschlange mit Anforderungen zur Erfassung und letztendlich zu einer Überlastung aufgrund fehlgeschlagener Versuche, die Verbindung zur Datenbank herzustellen, führen.

IV. Checkliste für Quellcode-Sicherheitsprüfungen anwenden

A. Anwendungen gelten als fehlerhaft, bis das Gegenteil eindeutig festgestellt wurde

Die oben beschriebenen fünf häufigsten Arten von Sicherheitslücken im Code stellen die wahrscheinlichsten und gefährlichsten Risiken im aktuellen und vorhandenen Code dar. Unternehmenskunden, Projektmanager in der Softwareentwicklung und Entwickler müssen sicherstellen, dass der Code auf diese fünf Kategorien von Sicherheitslücken hin überprüft wird.

Angesichts der Vielzahl von Risiken, die sich aus diesen Sicherheitslücken ergeben können, und der Wahrscheinlichkeit ihres Vorkommens in zahlreichen Anwendungen, müssen Projektteams bei jeder Anwendung, die Sie beauftragen, erstellen oder erneut analysieren, in puncto Sicherheit das nötige Maß an Skepsis walten lassen. Eine solche Haltung wäre eine klare Veränderung weg vom bisherigen Entwicklungskonzept, eine Anwendung basierend auf Geschwindigkeit, Features und Einsatzbereichen zu analysieren.

Heutzutage behandeln Entwicklungsteams alle vorhandenen und in Entwicklung befindlichen Anwendungen als Sicherheitsrisiko, bis das Gegenteil eindeutig festgestellt werden konnte. Die Mitarbeiter müssen aufgrund der Risiken, die solche Sicherheitslücken für das Unternehmen darstellen, auf diese Weise reagieren. Wie eine Arbeitsgruppe des Institute of Electrical and Electronics Engineers (IEEE) feststellte, die die neue Rolle der Sicherheit im Lebenszyklus der Softwareentwicklung untersuchte, „können Anwendungen heute langfristig einen weitaus größeren Einfluss auf die Zustimmung durch den Endbenutzer, die Unternehmensstabilität und die Rentabilität haben“.[2]

Als Ergebnis daraus empfiehlt die Arbeitsgruppe des IEEE, dass „Projektteams, die sich bisher ausschließlich auf die Bearbeitung von Kundenanforderungen konzentrierten, jetzt lernen müssen, dem Projektmanagement Sicherheitsrisiken vorab zu erläutern, um das erforderliche Budget und die Projektanforderungen zu rechtfertigen.“ Solche Veränderungen sind zwar nicht populär, die heutigen Sicherheitsrisiken und die Notwendigkeit, sensible und persönliche Informationen zu schützen, erfordern allerdings, dass sich alle Mitglieder eines Projektteams auf den Faktor Sicherheit konzentrieren.

B. Kontinuierliche Schwachstellentests für den Quellcode

Tools zum Testen von Sicherheitslücken alleine machen Software nicht sicher. Solche Tools sind ein Hilfsmittel für Entwickler und Codeprüfer bei der Analyse von Anwendungen – selbst solchen mit vielen Millionen Codezeilen –, um Schwachstellen mit dem potenziell größten Sicherheitsrisiko zu ermitteln. Dank dieser Tests können Mitarbeiter in der Entwicklung und Fehlerbehebung ihre Maßnahmen priorisieren und den Code risikobasiert verbessern. Hierbei werden zunächst die kritischsten Probleme bearbeitet.

Das Erstellen sicherer Anwendungen erfordert, dass Unternehmen sicheren Code entwickeln und kontinuierliche Schwachstellentests einplanen. Führen Sie daher als Voraussetzung für die Freigabe von Code ein, dass dieser für eine Anwendung sicher sein muss. Aufgrund der starken Zunahme gezielter Gefährdungen sollten sich alle Unternehmen auf die Einführung verbindlicher Schwachstellentests zur Anwendungssicherheit konzentrieren.

Dieser Prozess ist anspruchsvoll und kompliziert aufgrund der Tatsache, dass komplexe Anwendungen extrem komplexe und nur schwer zu identifizierende Codierungsfehler und Sicherheitslücken enthalten können. Mitarbeiter für die Codeprüfung weisen daher auf die Notwendigkeit hochentwickelter Codeanalysen hin. Dank der technischen Merkmale und den bekannten Ursachen für Implementierungs- und Codierungsfehler betrachten in der Tat viele Unternehmen Codeprüfungen als logischen Ausgangspunkt zur Verbesserung der Anwendungssicherheit insgesamt.

C. Auswahl des richtigen Tools

Bei der Auswahl eines automatischen Tools für Schwachstellentests für den Quellcode, das während des gesamten Lebenszyklus in der Softwareentwicklung zum Einsatz kommen soll, müssen Unternehmen zunächst die vorhandenen Ressourcen für die Codeentwicklung beurteilen. Hierzu gehören unternehmensinterne Sicherheitsexperten, Technologien und Servicepartner sowie Zielsetzungen für Verbesserungen im Lebenszyklus in der Softwareentwicklung. Diese Zielsetzungen können Folgendes umfassen: Reduzierung der Anzahl an freigegebenen Sicherheitspatches, Verringerung der Möglichkeiten für kostspielige Datenverluste, Senkung der Kosten im Lebenszyklus in der Softwareentwicklung, durchgängigere Einhaltung gesetzlicher Bestimmungen und Sicherstellung von Wettbewerbsvorteilen. Unternehmen müssen sicherstellen, dass das ausgewählte Tool bestmöglich zusammen mit den vorhandenen Coderessourcen eingesetzt werden kann und die Möglichkeit bietet, die gesetzten Ziele in puncto Lebenszyklus in der Softwareentwicklung zu erreichen. Sie müssen darüber hinaus sicherstellen, dass das ausgewählte Tool für alle Arten von Codierungsfehlern und Schwachstellen im Design eingesetzt werden kann, die bei der Entwicklung einer sicheren Anwendung ermittelt und beseitigt werden müssen.

Die zahlreichen bisher bekannt gewordenen Datenverluste sind häufig das Resultat von Schwachstellen im Code und ein Beleg dafür, wie wichtig die Beseitigung von Sicherheitslücken ist, um die versehentliche oder böswillige Offenlegung sensibler oder regulierter Informationen zu vermeiden. Die schnelle Zunahme stärker miteinander verknüpfter Tools, Technologien und Programmiersprachen (z. B. Web 2.0-Anwendungen) machen alle Unternehmen, die mit solchen Technologien arbeiten, anfällig. Dies gilt auch für Unternehmen, die derzeit nicht an solche Regelungen gebunden sind. Angesichts der erheblichen finanziellen Aufwände und Auswirkungen auf das Unternehmensimage, die sich aus Datenverlusten ergeben, befassen sich Unternehmen immer intensiver mit der Suche nach der effizientesten und effektivsten Möglichkeit, um die Sicherheit des Quellcodes sicherzustellen, sichere Anwendungen zu entwickeln und Fehler so früh wie möglich im Entwicklungsprozess und vor Auslieferung des Codes zu beseitigen.

Unternehmen, die in der Lage sind, Schwachstellentests für den Quellcode effizient und effektiv in den Lebenszyklus in der Softwareentwicklung zu integrieren, können die negativen Auswirkungen von Sicherheitsrisiken vermeiden. Solche verbesserten Verfahren führen zu erheblichen Einsparungen im Unternehmen sowie für Kunden, Geschäftspartner und andere Beteiligte, die auf die Software des Unternehmens angewiesen sind. Diese Einsparungen sind ein klares Zeichen für ein wirksames Sicherheitsprogramm für den Quellcode.

V. ANHANG: Checkliste für Quellcode-Sicherheitsprüfungen

Bei Sicherheitsprüfungen von Anwendungen müssen zunächst folgende Schwachstellen bearbeitet werden, um die damit verbundenen Risiken für die Anwendung und die Datenintegrität einzugrenzen:

Kategorie	Schwachstelle	Risiko
Sicherheitsspezifische Funktionen	Schlechte oder vom Standard abweichende Verschlüsselung	Hacker können Algorithmen knacken und sensible Daten stehlen.
	Nicht sichere Übertragungen über das Netzwerk	Die Legitimierung von Methoden zum Senden von Informationen wird nicht dokumentiert oder geschützt, so dass kritische Daten Risiken ausgesetzt sind.
	Sicherheitslücken in der Anwendungskonfiguration	Der Zugriff auf nicht geschützte Konfigurationsdateien oder -optionen ermöglicht die Manipulation von Softwaremerkmalen oder Daten.
	Sicherheitslücken bei der Zugriffskontrolle	Nicht autorisierter Zugriff auf vertrauliche Daten und Ressourcen.
	Verwendung nicht geschützter Datenbank- und Dateisysteme	Durch Hijacking und die Änderung von Aufrufen von Datenbanken und Dateisystemen werden Daten Risiken ausgesetzt.
	Sicherheitslücken in dynamischem Code	Erfolgreiches Einfügen bösartiger Befehle in Anwendungen, über die dynamischer Code ohne ausreichende Prüfung geladen wird.
	Laden lokalen Codes	Die Änderung dieser Aufrufe auf Systemebene ermöglicht die Manipulation, Gefährdung oder Zerstörung von Daten.
	Sicherheitslücken im Datenspeicher	Auf nicht sichere Weise gespeicherte Daten können problemlos gestohlen werden.
	Fehler bei der Authentifizierung	Hacker verwenden die Anmeldedaten berechtigter Benutzer, um Daten zu stehlen oder zu manipulieren.
Fehler bei der E/A-Prüfung und -Verschlüsselung	Sicherheitslücken durch SQL-Injections	Senden von SQL-Befehlen direkt an Datenbanken, um Daten zu stehlen oder zu manipulieren.
	XSS-Sicherheitslücken	Benutzer sind unbewusst dem Hijacking von Sitzungen ausgesetzt, laden Trojaner herunter oder werden Opfer von Phishing-Angriffen.
	Sicherheitslücken durch Betriebssystem-Injections	Hacker ändern oder missbrauchen Betriebssystembefehle, um Daten und Ressourcen zu steuern.
	Manipulation individueller Cookies oder versteckter Felder	Schafft ein Maß an Vertrauen, das Hacker zur Durchführung von Angriffen (z. B. SQL-Injections oder XSS) manipulieren können.

Kategorie	Schwachstelle	Risiko
Sicherheitslücken bei der Fehlerbehandlung und -protokollierung	Nicht sichere Fehlerbehandlung	Liefert Hackern Informationen, die diese für Angriffe nutzen können.
	Nicht sichere oder unzureichende Protokollierung	Über zugängliche Protokolldateien erhalten Hacker nützliche Informationen, und eine unzureichende Protokollierung ermöglicht Hackern, Spuren zu verwischen.
Nicht sichere Komponenten	Bösartiger Code	Über scheinbar berechtigten Code, der in die Software eingefügt wurde, können Hacker Sicherheitsmaßnahmen umgehen.
	Nicht sichere lokale Methoden	Die ungeprüfte Verwendung lokaler Methoden liefert Hackern einen Einstieg für den Zugriff auf kritische Ressourcen (z. B. System- oder Umgebungsspeicher).
	Nicht unterstützte Methoden	Nicht dokumentierte Funktionen oder Routinen sind eine versteckte Sicherheitslücke, die ausgenutzt werden kann.
Codierungsfehler	Sicherheitslücken durch Buffer Overflow	Hacker können ein Hijacking auf Systemressourcen durchführen.
	Sicherheitslücken durch Formatierungszeichenfolgen	Dies kann zu einem Buffer Overflow oder Risiken für die Daten führen.
	Denial-of-Service-Fehler	Verhindert die ordnungsgemäße Funktionsweise der Software.
	Sicherheitslücken durch zusätzliche Berechtigungen	Hacker können auf vertrauliche Daten und Ressourcen zugreifen.
	Konkurrenzsituationen (Race Conditions)	Umgehen eines Anwendungsprozesses, um Geschäftsprozesse zu manipulieren.
	Nicht sichere Verwendung lokaler Methoden	Dies kann sich nachteilig auf die Sicherheit zugunsten der Leistung auswirken und den nicht sicheren Zugriff auf den System- oder Umgebungsspeicher ermöglichen.
	Nicht unterstützte Methoden	Berechtigte Prozesse können unbewusst Aufrufe an ungeschützten Code durchführen.

Weitere Informationen

Wenn Sie mehr über die IBM Rational AppScan Source Edition erfahren möchten, wenden Sie sich bitte an den zuständigen IBM Vertriebsbeauftragten oder IBM Business Partner, oder besuchen Sie uns unter: **ibm.com**/software/rational/products/appscan/source/

Informationen über den Autor

Ryan Berg ist Senior Security Architect bei IBM. Er ist ein bekannter Referent, Schulungsleiter und Autor für Sicherheits-, Risikomanagement- und sichere Entwicklungsprozesse. Er hält in folgenden Bereichen Patente bzw. hat Patente angemeldet: Sicherheitsanalysen in mehreren Sprachen, Sicherheit auf Kernel-Ebene, Sprachen für zwischengeschaltete Sicherheitsanalysen und sichere Remote-Übertragungsprotokolle.

[1] Ponemon Institute, „2006 Annual Study: Cost of a Data Breach," Oktober 2006.

[2] Biscick-Lockwood, Bar, „The Benefits of Adopting IEEE P1074-2005," 2. April 2006.

IBM Deutschland
IBM-Allee 1
D-71139 Ehningen
Germany
ibm.com/de

IBM Österreich
Obere Donaustraße 95
1020 Wien
ibm.com/at

IBM Schweiz
Vulkanstrasse 106
8010 Zürich
ibm.com/ch

Die IBM Homepage finden Sie unter: **ibm.com**

IBM, das IBM Logo und ibm.com sind eingetragene Marken der IBM Corporation. AppScan und Rational sind Marken oder eingetragene Marken der International Business Machines Corporation in den USA und/oder anderen Ländern. Sind diese und weitere Markennamen von IBM bei ihrem ersten Vorkommen in diesen Informationen mit einem Markensymbol (® oder ™) gekennzeichnet, bedeutet dies, dass IBM zum Zeitpunkt der Veröffentlichung dieser Informationen Inhaber der eingetragenen Marken oder der Common-Law-Marken (common law trademarks) in den USA war. Diese Marken können auch eingetragene Marken oder Common-Law-Marken in anderen Ländern sein.

Eine aktuelle Liste der IBM Marken finden Sie auf der Webseite „Copyright and trademark information" unter: **ibm.com**/legal/copytrade.shtml

Java und alle auf Java basierenden Marken und Logos sind Marken von Sun Microsystems, Inc. in den USA und/oder anderen Ländern.

Vertragsbedingungen und Preise erhalten Sie bei den IBM Geschäftsstellen und/oder den IBM Business Partnern. Die Produktinformationen geben den derzeitigen Stand wieder. Gegenstand und Umfang der Leistungen bestimmen sich ausschließlich nach den jeweiligen Verträgen.

Hinweise auf IBM Lizenzprogramme oder andere IBM Produkte bedeuten nicht, dass nur Programme, Produkte oder Services von IBM verwendet werden können. Stattdessen können andere, diesen funktional entsprechende Produkte verwendet werden.

Diese Veröffentlichung dient nur der allgemeinen Information. Die in dieser Veröffentlichung enthaltenen Informationen können jederzeit ohne vorherige Ankündigung geändert werden. Aktuelle Informationen zu IBM Produkten und Services erhalten Sie bei der zuständigen IBM Verkaufsstelle oder dem zuständigen Reseller.

IBM erteilt keine Rechts- oder Steuerberatung und gibt keine Garantie bezüglich der Konformität von IBM Produkten oder Services mit den geltenden Gesetzen und gesetzlichen Bestimmungen. Der Kunde ist für die Einhaltung anwendbarer Sicherheitsvorschriften und sonstiger Vorschriften des nationalen und internationalen Rechts verantwortlich.

© Copyright IBM Corporation 2009
Alle Rechte vorbehalten.

Bitte der Wiederverwertung zuführen