A4Q
Security Essentials

Lehrplan V. 1.5.1d 2020 Deutscher Lehrplan A4Q Security Essentials

Version 1.0: 03.08.2017
Letzte englische Version 2020 1.5.1: 04.12.2020
Finale Deutsche Version: Version 2020 1.5.1d: 14.01.2022

Inhaltsverzeichnis

0. Einleitung

0.1 Zweck und geschäftlicher Nutzen

Sowohl der Ansatz des Microsoft Security Development Lifecycle (MSDL) [URL:MSDL] als auch der des Open-SAMM-Bewertungsmodells [URL:OpenSAMM] empfehlen, dass jedes Teammitglied eines Softwareprojekts eine vollständige Grundlagenschulung zur IT-Sicherheit, sowie eine Fortbildung gemäß seiner Position im Unternehmen erhalten sollte.
Das Ziel dieses Lehrplans ist es, jedem Teammitglied, das an der Entwicklung eines IT-Systems, einer Software oder eines eingebetteten Systems beteiligt ist, eine fundierte Einführung in die IT-Sicherheit zu bieten. Personalschulungen, die sich an diesem Lehrplan orientieren, erfüllen die grundlegenden Ansprüche an Sicherheitsschulungen, wie sie in [ISO/IEC 27034-1] und im MSDL empfohlen werden. Der Lehrplan orientiert sich ebenfalls an den Bedürfnissen von Projektmanagern, Requirements-Management-Spezialisten, Softwareentwicklern und Testern im gleichen Maße.
Der allgemeine Nutzen dieser Schulung besteht in Folgendem:

- die gängigsten sicherheitsrelevanten Begriffe, Konzepte und Prozesse zu verstehen
- eine aktive, beitragende Rolle bei sicherheitsrelevanten Risikomanagement-Aktivitäten übernehmen zu können

Der spezifische Gewinn für Projektmanager:

- Projektaktivitäten mit geforderten oder empfohlenen sicherheitsrelevanten Aktivitäten abgleichen zu können
- grundlegende Sicherheitsanforderungen, die ein bestimmtes System erfüllen muss, zu verstehen und erklären zu können
- die Maßnahmen zu verstehen und erklären zu können, die für die sichere Entwicklung eines bestimmten Systems oder einer Anwendung erforderlich sind

Der spezifische Gewinn für Entwickler:

- die Maßnahmen zu verstehen und erklären zu können, die für die Entwicklung sicherer Systeme und Anwendungen erforderlich sind
- häufige sicherheitsrelevante Fehler in der Entwicklung zu verstehen und zu vermeiden

Der spezifische Gewinn für Anforderungsentwickler:

- zu verstehen und erklären zu können, wie grundlegende Sicherheitsanforderungen festgelegt werden
- häufige sicherheitsrelevante Fehler in der Anforderungsentwicklung zu verstehen

Der spezifische Gewinn für Tester:

- die Bedeutung von Tests als Teil eines Security Development Lifecycle zu verstehen
- verschiedene Arten von Sicherheitstests zu verstehen und erklären zu können

Der spezifische Gewinn für IT-Risikomanager und IT-Sicherheitsexperten[1]

- einen Leitfaden zur Erstellung einer umfassenden Grundlagenschulung zur IT-Sicherheit zur Hand zu haben

[1] Dies ist eigentlich als Nutzen des Lehrplans an sich gedacht. Mitglieder dieser Gruppe haben i.d.Regel bereits an Schulungen für Fortgeschrittene zum Thema Sicherheit teilgenommen und benötigen daher keine Grundlagenschulung mehr.

1. Grundlagen der IT-Sicherheit (110 Minuten)

Grundbegriffe

Zurechenbarkeit, Asset (Gut/Güter), Echtheit (Authentizität), Authentifizierung (Beglaubigung), Autorisierung (Bevollmächtigung), Verfügbarkeit, Vertraulichkeit, Integrität (Unversehrtheit), Nachweisbarkeit (Nicht-Abstreitbarkeit[1]), personenbezogene Daten, Informationssicherheit, Bedrohung, Bedrohungsverursacher, Sicherheitslücke, Schwachstelle

Lernziele

(1.1.1) Sich Eigenschaften der Informationssicherheit und dazugehörige Attribute in Erinnerung rufen können. (K1)

(1.2.1) Beispiele für Assets, Sicherheitslücken und Bedrohungen bzw. Bedrohungsverursacher identifizieren. (K2)

(1.2.2) Sich Ursachen für Schwachstellen ins Gedächtnis rufen und verstehen, wann eine Schwachstelle zu einer Sicherheitslücke wird. (K1)

(1.2.3) Identifizieren, welche der Systemeigenschaften Vertraulichkeit, Integrität oder Verfügbarkeit von einer gegebenen Systemschwachstelle, einer Sicherheitslücke oder Angreiferaktion betroffen ist. (K2)

(1.2.4) Typische Aktivitäten in einem Informationssicherheitsrisikomanagement-Prozess beschreiben. (K2)

(1.2.5) Typische menschliche Bedrohungsverursacher/Angreifertypen identifizieren. (K1)

(1.3.1) Sichere Entwicklungsprinzipien auf der Grundlage von Authentifizierung, Autorisierung und Auditierbarkeit identifizieren. (K2)

(1.4.1) Sich in Erinnerung rufen, welche Eigenschaften einen sicherheitsrelevanten Systemausfall von anderen Systemausfallarten unterscheiden können. (K1)

(1.4.2) In einer Liste von Grundursachen und typischen Falschannahmen diejenigen erkennen, die typisch für Informationssicherheitsrisiken sind. (K2)

(1.5.1) Anhand einer Kurzbeschreibung einer Sicherheitsnorm die Art der Sicherheitsnorm bestimmen. (K1)

[1] Manchmal auch als Unleugbarkeit bezeichnet.

1.1 Das Konzept der IT-Sicherheit (15 Minuten)

In diesem Lehrplan behandeln wir Sicherheit im Kontext von Informationssystemen. Gemäß Definition ist ein Informationssystem ein geschlossenes oder offenes dynamisches elektronisches System mit der Fähigkeit, Informationen zu speichern und zu verarbeiten [Eck 14].
Laut [ISO 25010] ist Sicherheit *„der Grad, in welchem ein Produkt oder System Informationen und Daten schützt, sodass Personen und andere Produkte und Systeme in dem Umfang Zugriff auf die Daten haben, der ihrem Typ und ihrer Berechtigungsstufe entspricht.“*

Das „Sicherheits-Dreieck“ CIA wird von praktisch allen führenden IT-Sicherheitsorganisationen (z. B. OWASP und InfoSec) als eines der wichtigsten Sicherheitskonzepte bezeichnet.

CIA steht für die Qualitätsattribute Vertraulichkeit (confidentiality), Integrität (integrity) und Verfügbarkeit (availability):

- Vertraulichkeit in einem Informationssystem bedeutet, dass der Zugriff auf Informationen auf diejenigen beschränkt sein muss, die dazu berechtigt sind ([ISO 25010]). Daten müssen vor unbefugtem Zugriff geschützt werden, unabhängig davon, ob der unbefugte Zugriff absichtlich oder unabsichtlich erfolgt.
- Integrität ist definiert als der „Grad, zu dem ein System, Produkt oder Produktbestandteil unbefugten Zugriff auf oder die Modifikation von Computerprogrammen oder Daten verhindert“ ([ISO 25010]). Zu beachten ist, dass Informationen mehr als nur Daten sind. So kann zum Beispiel der Abruf einer verschlüsselten Datei eine Integritätsverletzung darstellen, wohingegen ein unbefugtes Abrufen der in der Datei gespeicherten Information eine Verletzung der Vertraulichkeit darstellt.
- Verfügbarkeit bedeutet [ISO 25010], dass die in einem System gespeicherten Informationen oder die von dem System geleisteten Dienste den berechtigten Anwendern zur Verfügung stehen, wenn ihre Nutzung erforderlich ist. Mit anderen Worten: Das System muss funktionsfähig sein. In Anlehnung an [ISO 25010] kann die Verfügbarkeit auch als ein Aspekt der Zuverlässigkeit betrachtet werden.

Weitere Eigenschaften von Sicherheit gemäß [ISO 25010]:

- Nachweisbarkeit – ein System muss über geeignete Mechanismen (z. B. Protokollierung) verfügen, damit nachgewiesen werden kann, dass Aktionen und Ereignisse stattgefunden haben. Diese Mechanismen (z. B. die Protokolle) müssen gegen Manipulation geschützt sein.
- Zurechenbarkeit – der „Grad, zu welchem die Handlungen einer Entität eindeutig dieser Entität zugeschrieben werden können“. Dies erfordert in der Regel irgendeine Form von Authentifizierung und Speicherung, z. B. Identifikatoren für Anwender, Programme, Prozesse und Systeme in Protokolldateien.
- Echtheit bzw. Authentizität – bemisst den „Grad, zu welchem die Identität eines Subjekts oder einer Ressource als die behauptete nachgewiesen werden kann“.

1.2 Assets, Bedrohungen und Sicherheitslücken – IT-Sicherheit im Kontext (55 Minuten)

Die Gewährleistung der Sicherheit eines IT-Systems betrifft mehr Aspekte, als die obige Definition von Sicherheit vermuten lässt. Sie umfasst alle Dienste und Informationen, die für eine Organisation von Wert sind. Die Agentur der Europäischen Union für Cybersicherheit (European Network and Information Security Agency – ENISA) definiert Sicherheit als „alle Aspekte im Zusammenhang mit der Definition, Erreichung und Sicherung von Vertraulichkeit, Integrität, Verfügbarkeit, Verantwortlichkeit, Authentizität und Zuverlässigkeit von Daten. (…).“ [ENISA]
Beim Thema Sicherheit gilt es, immer in Betracht zu ziehen, dass ein System aus irgendeinem Grund gefährdet sein könnte.
Ein System wird als sicher bezeichnet, wenn es in der Lage ist, spezifische Assets auf angemessene Weise gegen unbefugten Zugriff, Zerstörung, Veröffentlichung, Veränderung von Daten, und/oder Denial-of-Service zu schützen. Jeder Umstand oder Vorgang, der eine der soeben genannten Konsequenzen zur Folge haben könnte, wird als Bedrohung des Systems bezeichnet. [ENISA]

Ein Asset kann nur im Kontext einer Organisation definiert werden: „Alles, was einen Wert für die Organisation, ihren Geschäftsbetrieb und dessen Kontinuität hat, einschließlich Informationsressourcen, die die Mission der Organisation unterstützen.“ [ENISA]
Beispiele für Assets von Individuen innerhalb eines Informationssystems können deren personenbezogene Daten sein, beispielsweise Kreditkartennummern oder auch jegliche Informationen, die zur Authentifizierung verwendet werden können. Bei einem Unternehmen können es beispielsweise elektronisch gespeicherte Geschäftspläne, Informationen über beantragte Patente, Gehaltslisten, jedes denkbare Geschäftsgeheimnis wie z.B. Verträge sein. Zu den Assets eines Unternehmens können auch computergesteuerte Produktionskapazitäten oder andere über das Internet bereitgestellte Dienstleistungen zählen. Für einen Staat können Assets dessen militärische Geheimnisse, kritische Infrastruktur wie Telekommunikation, Wasser- und Stromversorgung sowie Verkehrsleitsysteme sein.

Angreifer oder Widersacher (Individuen, Organisationen) können Assets bedrohen[1]. Widersacher nutzen häufig eine Schwachstelle in einem Informationssystem aus. Eine Schwachstelle kann durch einen Programmierfehler, schlechtes Design, schlechte Architektur oder fehlerhafte Anforderungen entstehen [ENISA]. Sehr häufig führen falsche Konfigurationen zu Schwachstellen.
Eine Sicherheitslücke in einem System entsteht, wenn ein Angreifer Zugang zu einer Schwachstelle hat oder erlangen kann.

Typische Schwachstellen und Sicherheitslücken können sich folgendermaßen auf das oben genannte Sicherheitsdreieck CIA auswirken:

- Vertraulichkeit, z. B.:
  - Der Heartbleed-Bug in der offenen SSL-Verschlüsselungsbibliothek ermöglichte es einem Angreifer, beliebige Speicherinhalte von Webservern auszulesen, einschließlich Passwörter und Schlüssel.

[1] Assets werden außerdem von Zufallsereignissen wie Hardwarefehlern oder Naturkatastrophen bedroht. Diese werden in diesem Lehrplan nicht behandelt.

    - Eine Fehlkonfiguration von Servern oder Firewalls kann einem Angreifer Informationen über Programmversionen oder IP-Adressen verraten, die er zur Vorbereitung künftiger Angriffe nutzen kann. Der Vorgang des Sammelns dieser Art von Informationen wird als „Banner-Grabbing“ oder „Fingerprinting“ bezeichnet.
    - Fehlermeldungen können programminterne Informationen preisgeben, beispielsweise über die Struktur, die verwendete Programmiersprache oder Konfigurationen, die ein Angreifer zur Vorbereitung künftiger Angriffe nutzen kann.

- Integrität, z. B.:
    - Werden voreingestellte Passwörter nicht geändert, könnte das einem Angreifer einen einfachen Weg bieten, die Konfiguration eines Routers zu ändern oder manipulierte Firmware auf ein IoT-Gerät zu laden.
    - Das Löschen von Serverprotokollen oder das Löschen von Festplatten, um einen Angriff zu verschleiern.
    - Eine unzureichende Eingabevalidierung könnte es einem Angreifer ermöglichen, schädlichen Skriptcode im Gästebuch einer Website einzubetten[1].
- Verfügbarkeit, z. B.:
    - Denial-of-Service-Angriffe[2] (DoS) oder Distributed Denial-of-Service-Angriffe (DDoS) können einen Server mit Anfragen überlasten, sodass seine Dienste nicht mehr verfügbar sind.
    - Fehlerhafte Programmlogik oder Datenbankabfragen können es einem Angreifer erlauben, Eingaben zu erzeugen, die dazu führen, dass das System eine große Menge an Rechenressourcen verbraucht. Dies kann der Angreifer ausnutzen, um das System lahmzulegen. Auch das ist eine Art von Denial-of-Service-Angriff (DoS).
    - Ein Angreifer benutzt Erpressungssoftware (Ransomware), um eine Festplatte zu verschlüsseln. Das beeinträchtigt auch die Integrität des Systems, da der Angreifer die Speichermedien manipuliert.

Hinweis: Ein Angreifer kann sich sozialer Manipulationstechniken bedienen, um sich Zugang zu einem verwundbaren Teil eines Systems zu verschaffen (Details dazu in Kapitel 2).

Der Aufbau und die Wartung sicherer Systeme erfordert ein kontinuierliches Risikomanagement, z. B. nach [ISO 31000:2018], das mindestens die folgenden Aktivitäten umfasst:

1. Identifikation von Schutzzielen, z. B. Assets, die geschützt werden müssen, Datenschutzregeln, die befolgt werden müssen, usw.
   Identifikation typischer Bedrohungen für diese Assets. Dazu muss eine Organisation ihren Geschäftskontext verstehen und wissen, welche Informationssysteme tatsächlich eingesetzt werden.
2. Festlegung des für jedes System und jede Komponente erforderlichen Schutzniveaus.

[1] Cross Site Scripting oder XSS – siehe Kapitel 2.
[2] Im Deutschen zuweilen auch als Dienstblockade bezeichnet

3. Analyse der genutzten Architektur, Systeme, Komponenten, Bibliotheken usw. und Identifikation konkreter Bedrohungen und Schwachstellen.
4. Entsprechend den Befunden handeln.
5. Prüfen, ob die implementierten Maßnahmen die identifizierten Bedrohungen erfolgreich beseitigen oder die bestehenden Risiken auf ein akzeptables Niveau abmildern.

Die [ISO 31000:2018] betont die zyklische Natur des Risikomanagements. Dementsprechend müssen die Prozessaktivitäten bei Bedarf wiederholt werden.

Risikomanagementprozesse für die Sicherheit müssen auf verschiedenen Ebenen innerhalb einer Organisation implementiert werden. [NIST-SP-800-53] gibt ein Beispiel für einen organisationsweiten Sicherheitsrisikomanagementprozess. Ein Beispiel für einen Bedrohungsanalyseprozess, welcher zur Anwendung in einem Security Software Development Lifecycle geeignet ist, findet sich bei [Howard 06].

Ein wichtiger Faktor in einer Sicherheitsrisikoanalyse ist das Verständnis der potenziellen Bedrohungsverursacher, d. h. wer oder was eine Bedrohung für das System darstellt. [CC-Part-1] nennt als Beispiele:

- Hacker
- Böswilliger Anwender
- Nicht böswilliger Anwender
- Computerprozesse
- Missgeschicke

Die menschlichen Bedrohungsverursacher können kategorisiert werden nach

- Fähigkeiten,
- Motivation,
- Relativer Position zum IT-System.

Eine Kategorisierung nach Fähigkeiten kann Folgende Unterscheidungen treffen

- Scriptkiddies: haben höchstens grundlegende Programmierfähigkeiten, wissen aber, wie man ein Computersystem bedient und vorhandene Programme oder Skripte (daher der Name) verwendet, die bekannte Sicherheitslücken ausnutzen.
- Erfahrene Hacker: haben gute Programmierfähigkeiten, Kenntnisse von Netzwerk- und Betriebssystemtechnologien und gutes Hintergrundwissen über IT-Sicherheitskonzepte und -werkzeuge.
- Professionelle Hacker: verfügen über exzellente Programmierfähigkeiten, Kenntnisse in hardwarenahen Programmiersprachen (Assembler) und können ihre eigenen Sicherheitswerkzeuge, Exploits oder – im Falle böswilliger Absichten – Schadsoftware schreiben.

Professionelle Hacker können auch von großen Organisationen wie Konzernen, Geheimdiensten oder dem Militär eines Landes angeheuert werden. In einem solchen Fall ist die Bedrohung in der Regel viel größer als bei der Bedrohung durch Einzelne oder durch eine Gruppe erfahrener Hacker. Diese Organisationen verfügen über enorme Ressourcen und können damit jede erdenkliche Art von Angriff auf Informationssysteme unterstützen, auch durch geheimdienstliche Operationen.

Eine Kategorisierung nach Motivation kann unterscheiden zwischen:

- Neugier – im Allgemeinen eine Eigenschaft nicht böswilliger Anwender (z. B. Software-Tester)
- Angeberei
- „White-Hat“ oder ethischer Hacker – findet Schwachstellen, erkennt laufende Angriffe und informiert Anbieter, Organisationen und die Öffentlichkeit
- Hacktivist – benutzt Cyberangriffe als Mittel des Protests, zur Bloßstellung politischer Widersacher oder um geheime Informationen zu stehlen und zu veröffentlichen
- Cracker, Black-Hat-Hacker oder Cyberkrimineller – führt illegale Angriffe auf IT-Systeme mit böswilligen Absichten (finanzielle Interessen, Spionage, Terrorismus oder Rache) durch oder unterstützt diese
- Terrorismus, asymmetrische Kriegführung und Spionage – typischerweise durch professionelle Hackergruppen[1] ausgeführt

Eine Kategorisierung nach Position sollte mindestens Folgendes abdecken:

- Externer Angreifer – eine Instanz, die ihren Angriff von außerhalb des lokalen Netzwerks startet, nicht als legitimer Anwender und ohne vorher Insider-Informationen zu besitzen
- Innentäter – ein legitimer Anwender innerhalb der Organisation, der böswillige[2] oder nicht böswillige Absichten (z. B. Neugier) haben kann
- Man-In-The-Middle – kontrolliert oder belauscht den Netzwerkverkehr über ein infiziertes Gerät, z. B. einen Computer, Router, Server oder ein Smartphone, das sich innerhalb oder außerhalb des lokalen Netzwerks befinden kann.

1.3 Prinzipien der Entwicklung sicherer Software (15 Minuten)

Im vorangegangenen Abschnitt ging es darum, was ein sicheres System ausmacht. Der vorliegende Abschnitt befasst sich damit, welche Prinzipien in einer Software verwirklicht werden müssen, damit sie sicher ist:

- Authentifizierung: Der Prozess, mit dem eine Identität (und somit Authentizität) festgestellt werden kann, heißt Authentifizierung. Ein sicheres System muss zuverlässige Authentifizierungsmethoden haben. Diese können beispielsweise wissensbasiert (Passwörter) oder Token-basiert sein, Challenge-Responses nutzen (z.B. Captchas, per SMS oder E-Mail versendete Token), biometrische Daten verwenden oder die Hilfe von vertrauenswürdigen Dritten oder Identitätsanbietern (z.B. eine Organisation, die einen elektronischen Pass mit einem Chip und einer PIN ausstellt) in Anspruch nehmen. Bei einer Multi-Faktor-Authentifizierung kommt mehr als eine dieser Methoden zum Einsatz.
- Autorisierung: Ein sicheres System muss sicherstellen, dass alle Aktionen von Prozessen und Entitäten zu jeder Zeit in den Grenzen der vorher definierten Rechte ausgeführt werden. Typischen Prinzipien, die dabei zur Anwendung kommen sollen, sind unter anderem [Saltzer 75]:

[1] Von solchen Absichten geleitet sind häufig sogenannte Advanced Persistent Threats (APT) – eine Gruppe getarnter und kontinuierlicher Computerhacking-Vorgänge, die sich gegen spezifische Ziele richten, z. B. kritische Infrastruktur.

[2] Hinweis: Ein interner Angreifer kann durch Erpressung oder andere Form der sozialen Manipulation zu seinen Handlungen gezwungen worden sein.

    - Umfassende Zugriffsüberwachung (Complete Mediation): Jeder Zugriff auf Datenobjekte, Prozesse, Dienste usw. wird jedes Mal auf die erforderliche Berechtigung geprüft.
    - Geringste Rechte (least privilege): Ein Programm, Anwender, System usw. kann nur auf genau die Ressourcen zugreifen, die für die Erfüllung seiner legitimen Aufgabe benötigt werden.
- Auditierbarkeit: Die Auditierbarkeit eines Systems ist eine Voraussetzung für die Erfüllung der Merkmale Zurechenbarkeit und die Nachweisbarkeit. Sichere Systeme müssen über Mechanismen zur Aufzeichnung von Ereignissen verfügen, die möglicherweise Teil eines Angriffs sind, sowie über Schutzmaßnahmen gegen Manipulationen der Aufzeichnungen und der Aufzeichnungsmechanismen. [Lamport 78] erklärt, dass die Audit-Funktionen zuverlässige Fehlerbehandlungsverfahren beinhalten sollten. Ein sicheres und geschütztes System muss zu einem gewissen Grad in der Lage sein, seinen eigenen Zustand zu überprüfen.

Sicherheitsfachleute beschreiben die Triade Authentifizierung, Autorisierung und Auditierbarkeit oft als „Goldstandard“, da alle drei Wörter mit der Silbe „Au“ beginnen – dem chemischen Symbol für Gold (Aurum). Diese Au-Eigenschaften wurden zum ersten Mal in [Lamport 78] diskutiert. Jede Implementierung von Sicherheitsfunktionen muss so erfolgen, dass Manipulationen verhindert werden. Daher wird dringend empfohlen, für Sicherheitsfunktionen eine gut gekapselte vertrauenswürdige Datenverarbeitungsbasis (sog. Trusted Computing Base) zu verwenden. Eine Trusted Computing Base sollte unabhängig nach dem Common Criteria Standard [CC-Part-1] bis [CC-Part-3] zertifiziert sein.

1.4 Was IT-Sicherheit auszeichnet (15 Minuten)

Fehlerwirkungen sind typischerweise Abweichungen eines Systems von einem erwarteten Verhalten, die von einem Anwender oder einem Softwaretester leicht beobachtet werden können. Obwohl auch Sicherheitsfunktionen diese Art von Fehlern aufzeigen können, wird der Großteil der sicherheitsrelevanten Fehlerzustände bei der normalen Nutzung eines Systems möglicherweise gar nicht bemerkt. Scheinbar korrektes funktionales Verhalten kann Nebeneffekte haben, die von der Mehrheit der Anwender und sogar der Tester unbemerkt bleiben. Beispielsweise könnte eine Funktion zusätzliche Operationen erlauben, die nicht in ihrer Spezifikation beschrieben sind. Solche Schwachstellen in der Software-Implementierung oder Mängel im Systemdesign und der Architektur können von einem Angreifer ausgenutzt werden. Auch fehlerhafte Compiler, Compiler-Einstellungen, veraltete Bibliotheken oder Hardware von Dritten können zu Sicherheitslücken führen. Andere Sicherheitslücken entstehen durch Schwachstellen in Protokollimplementierungen, Fehlerzuständen in der Speicherverwaltung einer Anwendung oder gar des Betriebssystems. Die Erkennung oder Vermeidung solcher Fehlerzustände erfordert ein tiefes technisches Verständnis.

Typische Gründe für Sicherheitslücken und den diesen zu Grunde liegenden Fehlerzuständen sind:

- Mangel an Sicherheitsbewusstsein, Naivität oder Sorglosigkeit

- Verletzung von Sorgfaltspflichten, z. B. bei der Einhaltung von Vorschriften und bei der Kontrolle und regelmäßigen Aktualisierung von Systemen und Konfigurationen
- Fehlende, unvollständige oder unzureichende Sicherheitsrichtlinien und Kontrollen, ob die Richtlinien tatsächlich eingehalten werden
- Fehlende Sicherheitsanforderungen und -konzepte
- Mängel in Systemarchitektur oder -design
- Mangelnde Benutzerfreundlichkeit bei Sicherheitsfunktionen, z. B. unsichere Standardeinstellungen
- Falsche Annahmen, z. B. die, dass die getroffenen Sicherheitsmaßnahmen bereits ausreichend sind, nur bestimmte Eingaben gemacht würden oder ein Kommunikationskanal sicher sei.

IT-Sicherheit muss deshalb sowohl in jeder Phase der Softwareentwicklung als auch in regelmäßigen Abständen bei laufenden Systemen betrachtet werden.

1.5 Sicherheitsnormen (10 Minuten)

Normen bieten eine Auflistung von Anforderungen, die von Fachexperten definiert werden, sowie Prozess- und Dokumentvorlagen. Sie können auch als Checklisten genutzt werden und helfen so, häufige Fehler zu vermeiden.
Es gibt Hunderte von Normen zur Informationssicherheit und damit verwandten Themen – viel mehr, als man sinnvollerweise anwenden kann. Um einen umfassenden Überblick zu geben, unterscheidet dieser Lehrplan vier Kategorien:

- Vorschriften, z. B. Gesetze, Richtlinien oder Verordnungen wie die europäische Datenschutzgrundverordnung (DSGVO).
- Normen für die sichere Softwareentwicklung wie SDLC (Software Development Lifecycle)-Modelle, die in [ISO/IEC 27034] behandelt und durch den Microsoft Security Development Lifecycle umgesetzt werden. Ein weiteres Beispiel ist der Common-Criteria-Catalogue (siehe [CC-Part-1], [CC-Part-2], [CC-Part-3]), der zum Beispiel bewährte Verfahren für Sicherheits-anforderungen sowie Verfahren zur Verifizierung und Validierung von Sicherheitsfunktionen enthält.
- Normen für Managementsysteme zur Informationssicherheit, beispielsweise [ISO/IEC 27001] oder [NIST-SP-800-39]. Diese Normen beschreiben Risikomanagementprozesse und wie diese auf Personen, Prozesse und IT-Systeme angewendet werden können, um die Sicherheit von Informations-Assets zu bewahren.
- Fachspezifische Standards. Diese sind entweder branchenspezifisch, wie die Standards der Kreditkartenbranche oder technologiespezifisch, wie die vom Open Web Application Security Project (OWASP) veröffentlichten Standards und Empfehlungen.

2. Angriffe verstehen (255 Minuten)

Begriffe

Angriffsfläche, Angriffsvektor, CAPEC, CSRF, DDOS, DOS, Exploit-Kit, Man-in-the-middle, Phishing, Replay-Angriff, Root-Kit, Scareware, Smishing, Sniffing, Trojaner, Virus, Vishing, Wurm, XSS, Zero-day-Exploit, Zugriffskontrolle

Lernziele

(2.1.1) Sich typische Muster ins Gedächtnis rufen können, die zur Klassifizierung von Angriffen verwendet werden. (K1)

(2.2.1) Die Eigenschaften kennen, die einen Virus, Wurm, Trojaner, Scareware, ein Root-Kit und ein Exploit-Kit definieren. (K1)

(2.3.1) Angriffsflächen und Angriffsvektoren erklären können. (K2)

(2.3.2) Die Prinzipien eines speicherbasierten Angriffs erklären können. (K2)

(2.4.1) Die vier Schichten der Internetprotokollfamilie und ihren Zweck nennen. (K1)

(2.4.2) Typische Angriffsflächen und -vektoren einer Client-Server-Sitzung beschreiben können. (K2)

(2.4.3) Verstehen, dass Angriffe auf allen Ebenen der Internetprotokollfamilie stattfinden können. (K2)

(2.4.4) Das Konzept eines Injektionsangriffs und mögliche Verteidigungsmaßnahmen verstehen. (K2)

(2.4.5) Unterscheiden können, wie DoS und DDoS arbeiten. (K2)

(2.4.6) Unterscheiden können, wie verschiedene XSS-Typen funktionieren. (K2)

(2.4.7) Erklären können, wie ein CSRF funktioniert. (K2)

(2.4.8) Mögliche Ursachen für Sicherheitsrisiken in der Authentifizierung, Sitzungsverwaltung und Zugriffskontrolle in Webanwendungen unterscheiden können. (K2)

(2.4.9) Die Risiken schwacher oder fehlender Verschlüsselung kennen. (K2)

(2.4.10) Die Prinzipien symmetrischer und asymmetrischer Verschlüsselung und die Notwendigkeit der Authentifizierung in der Webkommunikation erklären können. (K2)

(2.5.1) Die Schritte im Rahmen eines Social-Engineering-Angriffs beschreiben können. (K2)

(2.5.2) Die Merkmale von Phishing, Vishing und Smishing nennen. (K1)

(2.6.1) Typische Bedrohungen für die drahtlose Kommunikation beschreiben können. (K2)

(2.6.2) Prinzipien und Beispiele sicherer drahtloser Kommunikation beschreiben. (K2)

2.1 Überblick über die Taxonomie von Angriffen (5 Minuten)

Angriffe oder – spezifischer – Angriffsvektoren können kategorisiert werden nach:

- Angreifertyp – siehe Abschnitt 1.2
- Art der verwendeten Schadsoftware – Beispiele in Abschnitt 2.2.
- Angriffsfläche, die die Schnittstelle zur angreifbaren Anwendung darstellt. Details in Kapitel 2.3.
- Anfällige Stelle oder Eigenschaft in einer Software, z. B. eine defekte Authentifizierung, Skriptausführung usw. Eine ausführlichere Beschreibung sowie eine kurze Diskussion des CAPEC-Klassifizierungsschemas finden sich in Kapitel 2.4.
- Beim Angriff verwendete Methode, beispielsweise Denial-of-Service, Injektion oder soziale Manipulation – siehe Kapitel 2.4. und 2.5.

Es gibt auch Klassifizierungsschemata für Sicherheitslücken, beispielsweise STRIDE, CVSS oder die „Common Vulnerabilities and Exposures“ (CVE). Diese werden in Kapitel 3 dieses Lehrplans behandelt.
Ein Sonderfall ist der Zero-Day-Exploit. Er nutzt eine Schwachstelle als Angriffsvektor, die vor dem Angriff unbekannt[1] war – es gibt also noch keinen Fix. Aktuelle Beispiele aus der Praxis für die in diesem Kapitel besprochenen Angriffstypen findet man z. B. unter [URL:CISA alerts] oder [URL:NVD].

2.2 Typen von Schadsoftware (15 Minuten)

Als schädliche Software – engl. Malware[2] – wird jede Art von Software bezeichnet, die in böswilliger Absicht geschrieben, installiert oder ausgeführt wird.
Die folgende Auflistung beschreibt häufige Malware-Arten. In der Praxis kann Schadsoftware auch eine Kombination verschiedener Typen sein:

- Virus: ein Schadprogramm, das zur Ausführung ein Wirtsprogramm benötigt. Es kann nicht ohne ein anderes Programm laufen, sich jedoch selbst replizieren – d. h. weitere Dateien, Programme, Computer usw. infizieren.
- Wurm: ein Schadprogramm, das eigenständig laufen und sich replizieren kann und typischerweise bei Infektion Systeme oder Computer markiert und bereits infizierte Systeme als solche erkennt und nicht erneut infiziert. Besteht in der Regel aus mehreren Teilen oder Segmenten, wie ein echter Wurm. Würmer werden verwendet, um Schadprogramme innerhalb eines Netzwerks von Rechner zu Rechner zu verbreiten.

[1] typischerweise dem Hersteller des Produkts und den Organisationen, die das Produkt nutzen.
[2] Kurzform für Malicious Software

- Trojaner: eine Art Schadprogramm, das versteckt auf einem Rechner installiert wird, oft zusammen mit anderer, nicht bösartiger Software, und das in der Lage ist, seine Operationen zu verbergen. Trojaner verfügen normalerweise über Installsationsfuntionen zum Nachladen weiterer Schadprogramme wie Keylogger, Rootkits, Viren usw.
- Exploit-Kits: werden von Angreifern typischerweise auf Servern installiert und sind in der Lage, automatisch Software-Schwachstellen in Client-Rechnern, die mit dem Server kommunizieren, zu erkennen und auszunutzen. Dabei laden sie Schadcode auf die Clients und führen diesen aus.
- Rootkit: dazu gedacht, unberechtigten Zugriff – normalerweise auf Administratorebene – auf einem Zielrechner zu erlangen, ohne dabei von Überwachungsprogrammen wie etwa einem normalen Systemmonitor bemerkt zu werden.
- Scareware: eine Art Schadprogramm, das Social-Enginieering-Techniken anwendet. Es manipuliert Anwender, sodass sie unnötige und meist nutzlose Software kaufen, indem es Ängste oder das Gefühl einer Bedrohung weckt.

2.3 Angriffsfläche (30 Minuten)

[Howard 06] definiert die Angriffsfläche als „die Summe des gesamten Codes und aller Funktionen, auf die Anwender und potenzielle Angreifer Zugriff haben."
Aus der Perspektive des Testers unterscheidet [Whit03] verschiedene Arten von Angriffsflächen:

- die Anwenderoberfläche (User Interface – UI)
- die Anwendungsprogrammierschnittstelle (Application Programming Interface – API)
- das Dateisystem
- der Speicher des Systems.

Angriffe über eine beliebige Angriffsfläche erlauben potenziell

- verschiedene Arten von Injektionen
- Kontrolle von Verzeichnispfaden, URLs usw. mit Lese- und Schreibrechten
- unbefugten Zugriff auf Daten oder Dienste
- direkten Zugriff auf Betriebssystemfunktionen.

Angriffe über die Anwenderoberfläche können mittels jeder Eingabe erfolgen, die von einem Anwender direkt getätigt wird. Dazu gehören Angriffe gegen schwache Authentifizierungsmechanismen, z. B. die Verwendung von Standardpasswörtern, gegen unzureichende Zugriffskontrollen, bei denen auf vertrauliche Informationen auf mehreren Wegen mit unterschiedlichen Rechten zugegriffen werden kann (weitere Diskussion und Beispiele dazu in Abschnitt 2.4), Einschleusung von Befehlen wie z.B. bei einer SQL-Injection (siehe auch Abschnitt 2.4) oder Pufferüberläufe (siehe speicherbasierte Angriffe weiter unten).

Angriffe, die ein API nutzen, sind spezifisch für das jeweilige API. Oft werden dabei bekannte Sicherheitslücken älterer, ungepatchter API-Versionen oder typische API-Programmierfehler ausgenutzt. Häufig werden Nebeneffekte von Funktionsaufrufen oder von Funktionsüberladungen im API ausgenutzt. Dies kann bei Design und Implementierung schwer zu entdecken sein, da der Code für sich betrachtet dem Anschein nach nur zulässige Funktionen und Konstrukte verwendet. Ein bekanntes

Beispiel ist die Verwendung des open-Befehls von Perl mit zwei Parametern (statt drei), was eine Shellcode-Injektion oder DoS ermöglicht.

Typische Angriffe auf das Dateisystem beinhalten das Auslesen von Informationen aus unverschlüsselten Dateien, die Manipulation von gespeicherten Daten, Konfigurationsdateien oder ausführbaren Dateien, die Disassemblierung von Binärdateien oder die Entschlüsselung schwach verschlüsselter Dateien, z. B. MD5-Hashtabellen mit Passwörtern.

Speicherbasierte Angriffe nutzen das Layout und die Operationen des Speichers eines Computers aus. Um zu verstehen, wie speicherbasierte Angriffe funktionieren, benötigt man ein Grundverständnis des Layouts eines Computerspeichers. Typische fundamentale Speicherstrukturen sind Stack und Heap. Beide enthalten Daten, Anweisungen und Jump- oder Return-Adressen. Return-Adressen geben an, wo sich Anweisungen befinden, die als nächstes ausgeführt werden sollen. Zu den Daten, die im Speicher des Computers abgelegt sind, gehören typischerweise anwenderkontrollierte Eingaben, die über die Anwenderoberfläche, eine Datei oder eine API eingegeben werden. Daten, die von einem Anwender geschrieben oder abgefragt werden, haben in der Regel eine vorgegebene Maximallänge. Zwei Grundmuster, die bei speicherbasierten Angriffen zur Anwendung kommen, nutzen diese simple Tatsache aus:

- Pufferüberlauf (buffer overflow): Der Angreifer schreibt eine größere Datenmenge in den Speicher, als das Programm erwartet. Auf diese Weise kann er den Inhalt nachfolgender Datenfelder verändern. Möglicherweise kann er Return-Adressen oder Steuervariablen manipulieren und so Programmausführung beeinflussen. Außerdem könnte er seinen eigenen Schadcode in seine Eingaben einbetten und zur Ausführung bringen.[1]
- Puffer-Überlesen (buffer overread): Ein Angreifer kann Speicherpositionen auslesen, auf die er keinen Zugriff haben sollte. Dies kann der Fall sein, wenn der Angreifer (z. B. in einer Abfrage) einen längeren Datensatz anfordert, als das Programm erwartet. Durch so einen Buffer-Over-Read kann ein Angreifer unbefugten Zugriff auf Daten jeglicher Art erhalten, darunter zusätzliche Informationen über den auf einem Server laufenden Quellcode, den Inhalt von Kontrollvariablen und Return-Adressen sowie deren Position im Speicher, um künftige Angriffe vorzubereiten.

Beide Angriffsarten sind möglich, wenn das Vollständigkeits-Prinzip für Zugriffe (complete mediation) auf der Implementierungsebene nicht konsequent angewendet wird.

Hinweis:
Ein Angreifer wird oft mehr als eine Angriffsfläche für einen Angriff nutzen. So könnte er etwa eine manipulierte Zeichenfolge über die Anwenderoberfläche eingeben, die eine Return-Adresse im Speicher des Computers korrumpiert und dann maschinenlesbaren Code ausführt, der in die Zeichenfolge eingebettet ist.

[1] Fortgeschrittene Formen rufen Bibliotheksfunktionen (z. B. Return-to-libc-Angriffe) in böswilliger Weise auf oder verwenden Schnipsel des programmeigenen Codes (Return-Oriented Programming: ROP). Diese werden hier nicht behandelt.

2.4 Gängige Angriffe und Web-Sicherheit (160 Minuten)

Angriffe auf die Sicherheit von Informationssystemen wurden schon lange vor dem Aufkommen des Internets durchgeführt. Die Boten des Feindes foltern, seine geheimen Nachrichten lesen, seine Funksignale abhören oder eine Telefonleitung anzapfen sind nur einige Beispiele. Tatsächlich waren viele der ersten Internet-Hacker sogenannte „Phone Phreaks“ – Personen, die aktiv nach Schwachstellen in Telefonnetzen suchten und diese aus verschiedenen Beweggründen ausnutzten, z. B. um kostenlos telefonieren zu können (Betrug), um anzugeben oder auch einfach aus technischer Neugier [MIT03].
Stand 2020 unterscheidet die Common Attack Pattern Enumeration and Classification (CAPEC™) von MITRE zwischen 9 Hauptmechanismen und 6 Hauptangriffsdomänen. Insgesamt werden mehr als 500 einzigartige Angriffsmuster [URL:CAPEC] aufgelistet.
Es versteht sich von selbst, dass diese nicht vollumfänglich in einem Einstiegskurs zur IT-Sicherheit diskutiert werden können. Aus diesem Grund beschränkt sich dieses Kapitel auf repräsentative Angriffstypen, die typische Beispiele für die Vorgehensweise von Angreifern bieten.

2.4.1 Die Internetprotokollfamilie

Das Internet bietet eine weltweite Vernetzungsstruktur, die die Reichweite potenzieller Angreifer enorm steigert. Um IT-Sicherheit zu verstehen, muss man wissen, wie die Kommunikation in diesem riesigen Netzwerk prinzipiell funktioniert.

Kommunikation innerhalb des Internets erfordert Hardwarekomponenten und Software, die verschiedene Aspekte der Informationsübertragung steuern.
Die Internetprotokollfamilie (kurz TCP/IP) besteht aus einer Reihe von Spezifikationen ([RFC 1122], [RFC 1123]), die beschreiben, wie Kommunikation mittels einer vierschichtigen Architektur erfolgt.

Die vier Schichten::

- Die Netzzugangsschicht oder ‚link layer‘ stellt die Kommunikation zwischen einem Host (z. B. einem Desktop-Rechner, einem lokalen Server, einem Tablet usw.) und seiner direkten Umgebung her. Diese Umgebung ist das direkt angeschlossene oder „lokale Netzwerk“[1]. Unterhalb der Netzzugangsschicht befinden sich noch weitere Softwareschichten. Sie stellen physikalische Verbindungen her – z. B. zwischen Netzwerkkarten oder verschiedenen Hardwarekomponenten in einem Computer.
- Die Protokolle der Internetschicht (engl. ‚internet layer‘) dienen dem Transport von Datenpaketen (Datagrammen) zwischen Hosts in verschiedenen lokalen Netzen. Die Protokolle sind „verbindungslos“, d. h. es gibt keine Zustellgarantie für ein Datagramm.
- Die Transportschicht (engl. ‚transport layer‘) stellt die durchgängige Kommunikation zwischen Anwendungen sicher, die auf unterschiedlichen Hosts laufen. Auf dieser Schicht werden zwei Hauptprotokolle verwendet. Das erste ist das zustands- bzw. verbindungslose User Datagram Protocol

[1] Nicht zu verwechseln mit dem Local Area Network, womit eher der geografische Bereich wie ein Haus, eine Schule oder ein Firmengelände gemeint ist.

(UDP). Das zweite ist das Transmission Control Protocol (TCP). Letzteres ist ein zuverlässiges Protokoll, das eine bestätigte Verbindung über eine Handshake-Transaktion herstellt und alle gesendeten Datenpakete durch Sequenzierung verfolgt.

- Die Anwendungsschicht umfasst zwei Protokollkategorien. Die erste beinhaltet Protokolle, die Dienste direkt für Anwender bereitstellen, beispielsweise Telnet, FTP und SMTP.
  Die zweite besteht aus Unterstützungsprotokolle, die den von den Anwendern verwendeten Protokollen assistieren. Ein Beispiel für ein Unterstützungsprotokoll ist das Domain Name System (DNS). Es dient der Lokalisierung und Identifizierung von Computerdiensten und Geräten durch von Menschen lesbare Namen anstelle von abstrakten Adressen.

2.4.2 Client-Server-Sessions im Web

Eine Sitzung ist ein Austausch von Informationen zwischen zwei Geräten, beispielsweise zwei Computern. Web-Sitzungen zwischen einem Client und einem Server werden typischerweise von Protokollen der Anwendungsschicht abgewickelt. Der Aufbau der Kommunikation sowie der eigentliche Austausch von Datenpaketen wird von den Protokollen der unteren Ebene übernommen.
Das Verschlüsselungsprotokoll TLS (Transport Layer Security) wird verwendet, um Daten der Anwendungsschicht zu verschlüsseln, bevor sie über ein Transportprotokoll wie TCP verschickt werden. Um anzuzeigen, dass ein Protokoll TLS verwendet, wird der Buchstabe „s“ für secure dem Protokollnamen hinzugefügt, wie in https oder sftp.
Eine typische Web-Sitzung wird von einem Client initiiert, der eine Anfrage an einen Webserver sendet, z. B. über einen Browser. Die Anfrage wird vom Webserver verarbeitet – beispielsweise mittels Skripten für den Datenbankzugriff. Anschließend wird eine Antwort, wie ein Status der Anfrage oder angefragte Daten, an den Client zurückgeschickt.

2.4.3 Die Internetprotokollfamilie als Angriffsvektor

Der ursprüngliche Zweck der Internetprotokollfamilie war die Spezifikation zuverlässiger, effizienter und einfach zu implementierender Kommunikationswege. Die Erfinder hatten nicht die Sicherheit als Hauptanforderung im Sinn. Aus diesem Grund sind viele der ursprünglichen Protokollspezifikationen mit Sicherheitsmängeln behaftet. Viele Protokolle besitzen keine sicheren Verfahren zur Authentifizierung und Autorisierung. Darüber können die Spezifikationen sicherer Protokolle Mängel im Design oder Sicherheitslücken in ihrer Implementierung aufweisen. Die Behebung dieser Probleme ist ein fortlaufender Prozess.

Sicherheitslücken können in jeder der vier Schichten auftreten. Auf der Netzzugangsschicht könnte ein Angreifer beispielsweise versuchen, die Media-Access-Control-Adresse (MAC) eines von ihm kontrollierten Geräts mit der Internetprotokoll-Adresse (IP) eines anderen, echten Hosts im Netzwerk zu verknüpfen. Da dabei das Adressauflösungsprotokoll (adress resolution protocol – ARP) verwendet wird, heißt diese Art von Angriff ARP-Spoofing oder ARP-Cache-Poisoning.

Das Fälschen (Spoofing) einer IP-Adresse zum Senden von Paketen an einen Zielhost von einer vermeintlich vertrauenswürdigen Quelle erfolgt auf der Internetschicht.
Die meisten webbasierten Angriffe nutzen die Transport- und Anwendungsschicht als Angriffsvektor. Angriffe können z. B. auf einen Webserver abzielen, wobei u.a.Sicherheitslücken in der Konfiguration eines Servers, in Skripten, Datenbanken, Protokollversionen, Diensten oder im Betriebssystem ausgenutzt werden.
Von einem Server aus kann ein Client über Sicherheitslücken im Browser oder anderen Diensten, die mit dem Server in Kontakt treten, angegriffen werden. Ein Angriff kann auch von einem „Man-in-the-middle“ initiiert werden, der die Kommunikation zwischen Server und Client kontrolliert.

2.4.4 Injections

Das Ziel einer Injection[1] ist es, das Zielsystem dazu zu bringen, eine Dateneingabe auf eine unzulässige Weise zu behandeln. Eine in arglistiger Absicht erstellte Benutzereingabe, Protokollnachricht oder Datei kann diese Art von Sicherheitslücke ausnutzen.
Beispiele:

- SQL-Injections: Die Eingabe für eine Datenbankabfrage kann so manipuliert werden, dass ein anderer SQL-Code ausgeführt wird, als von der Anwendung vorgesehen.
- Lightweight Directory Access Protocol (LDAP)-Injection: Eingaben werden verwendet, um Dateisystemoperationen wie Suchen, Hinzufügen, Ändern oder Löschen auf unbefugte oder unbeabsichtigte Weise zu manipulieren.
- Betriebssystem-Injektion: Eine Anwendung übergibt Benutzereingaben direkt an einen Aufruf einer Systemfunktion, die mit den gleichen Berechtigungen ausgeführt wird, mit denen die Anwendung ausgeführt wird.
- Code injection mittels eines Pufferüberlaufs, wie in Abschnitt 2.3 beschrieben.

Injektionsangriffe können immer dann erfolgen, wenn Eingabedaten einer potenziell nicht vertrauenswürdigen Quelle durch ein Computerprogramm verarbeitet werden. Eine Einschleusung kann also überall stattfinden, wo die Eingabe eines Programms nicht sorgfältig „bereinigt“[2] wird. Daher sollte eine ordnungsgemäße Eingabebereinigung immer gründlich getestet werden.
Typische Methoden der Eingabebereinigung sind:

- Whitelisting: Erlaubt nur zuvor definierte Eingabemuster. Alles, was nicht auf der Whitelist steht, ist verboten. Die Datenverarbeitung wird abgebrochen, wenn ein von der Whitelist abweichendes Muster erkannt wird.
- Blacklisting: Zurückweisen von vordefinierten Eingabemustern oder Verbieten einer Reihe von bestimmten Eingaben. Alles, was nicht auf der schwarzen Liste steht, ist erlaubt. Z. B. können Daten rekursiv gefiltert und nicht erlaubte Sequenzen entfernt oder die Datenverarbeitung wird sofort abgebrochen, wenn ein verbotenes Muster erkannt wird.
- Alle Ein- und Ausgaben so kodieren, dass jede unerwünschte Interpretation einer Eingabe unmöglich wird, z. B. indem ausführbare Befehle als Eingabe in einen einfachen String oder alle Zeichen in Großbuchstaben umgewandelt werden.

[1] Manchmal mit Injektions- oder auch Einschleusungsangriff übersetzt.
[2] engl. sanitized

Eine spezifische Methode SQL-Injections zu verhindern, ist die Verwendung von ‚Prepared Statements', bei denen die Eingaben des Anwenders enkodiert und an String-Variablen gebunden werden. Diese String-Variablen werden in festen (vorbereiteten) Anweisungen verwendet. Die Anwendung benutzt dann nur noch diese Art von Anweisungen für Datenbankabfragen.

2.4.5 DoS und DDoS

„In a denial-of-service (DoS) attack, an attacker attempts to prevent legitimate users from accessing information or services."[1] [URL:US-CERT-DOS]
Denial of Service kann entweder durch Herbeiführen des Absturzes einer Anwendung mit Hilfe eines bekannten Fehlers oder durch Überlastung der Verarbeitungsfunktionen des Zielsystems erreicht werden. Letzteres kann durch Auslösen einer ressourcenfressenden Rechenoperation (z.B. Speicherressourcen, CPU-Kapazität) oder durch Überlastung der Kommunikationsschnittstellen des Zielsystems erreicht werden.
Bei einem Distributed-Denial-of-Service-Angriff (DDoS) überlasten Einzelanfragen einer großen Anzahl von Computern oder anderen digitalen Geräten mit Internetanschluss die Ressourcen eines Ziels oder seine Anbindung ans Netz.
DoS- und DDoS-Angriffe können auf jedes Element abzielen, das einen anvisierten Dienst mit dem Internet verbindet.

2.4.6 Cross Site Scripting (XSS)

Cross Site Scripting (XSS) ist eine Art von Injektionsangriff, bei dem ein Skript vom Browser eines Anwenders ausgeführt wird, das nicht von der tatsächlich besuchten Website stammt.
Es gibt drei von Angreifern eingesetzte XSS-Hauptvarianten.

- Bei einem persistenten oder ‚stored XSS'-Angriff speichert ein Angreifer bösartige Skripte direkt auf einem Webserver. Solche Skripte können überall dort versteckt sein, wo ein Anwender Daten eingeben kann, die anderen Anwendern angezeigt werden.
- Reflektiertes oder nicht-persistentes XSS schleust ausführbaren Code in eine vom Anwender gesendete Anfrage ein. Der ausführbare Code wird vom Server an den Anwender zurückgespiegelt, z. B. als Teil einer Fehlermeldung, und dann vom Browser ausgeführt. Eine Abfrage, die die Injektion enthält, kann von einem Anwender unfreiwillig gesendet werden, wenn er auf einen manipulierten Link in einer E-Mail oder auf einer anderen Webseite klickt.
- DOM-basiertes XSS nutzt Sicherheitslücken in der Implementierung der DOM-API oder in deren Nutzung auf dem Client aus. DOM bezeichnet den Document Object Model-Standard [URL:W3C-DOM]. Er ist in verschiedenen APIs implementiert, die die dynamische Website-Erstellung in Browsern unterstützen. Wenn ein Skript zur dynamischen Website-Erstellung externe Eingaben (z. B. vom Anwender) verwendet, um die Darstellung einer Website kontextabhängig im Browser zu erzeugen, kann es für diese Art von Angriff anfällig sein. Ein Angreifer geht dabei ähnlich wie beim reflektierten XSS vor. Der Hauptunterschied besteht darin, dass der Angriff lokal im Browser des

[1] Übersetzung: „Bei einem Denial-of-Service-Angriff (DoS) versucht ein Angreifer, rechtmäßige Benutzer vom Zugang zu Informationen oder Diensten abzuschneiden."

Anwenders stattfindet und der Webserver möglicherweise nicht einmal involviert ist (außer in der Bereitstellung eines potenziell anfälligen Skripts).

Ein guter Schutz gegen jede Form von XSS ist eine sorgfältige Bereinigung aller benutzergesteuerten Eingaben auf einer Website. Weitere Informationen bietet [URL:OWASP-Categories].

2.4.7 Cross Site Request Forgery (CSRF)

„ Cross Site Request Forgery (CSRF) attacks occur when a malicious web site causes a user's web browser to perform an unwanted action on a trusted site."[1] [Zeller 08] Wie beim reflektierten XSS verleitet ein Angreifer den Anwender dazu, eine verlinkte Adresse aufzurufen.
Die technische Herangehensweise ist jedoch anders. Bei einer CSRF kommt eine Anfrage oder ein Aufruf an einer Website von einer scheinbar vertrauenswürdigen und authentifizierten Quelle und wird von der Webseite verarbeitet. Der Aufruf der Webseite kann ohne Wissen des Anwenders erfolgen, z. B. wenn er eine E-Mail in einer Anwendung öffnet, bei der die Option „Automatische Anzeige von Remote-Inhalten" aktiviert ist. Ein CSRF-Angriff auf einen Homerouter [URL:router-hack], könnte auf diese Weise über einer E-Mail erfolgen, in der folgende HTTP-Anfrage an den Router versteckt ist:
„http://admin:12345@192.168.1.1/start_apply.htm?dnsserver=66.66.66.66".
Wenn das Router-Passwort auf das Standard-Passwort (im Beispiel 12345) gesetzt ist, kann der Angreifer nun jede Anfrage an eine Website dorthin umleiten, wo er sie haben möchte.

2.4.8 Schwachstellen im Web bei Authentifizierung und Autorisierung

Das Ziel der Ausnutzung von Sicherheitslücken in der Authentifizierung und Autorisierung ist in der Regel der Diebstahl von Informationen wie Dokumenten, Zugangsdaten oder Finanzdaten. Typische Schwachstellen sind:

- Lecks und Design-Mängel in der benutzerdefinierten Authentifizierung und Sitzungsverwaltung.[2] Die Authentifizierung kann z.B. durch Injektionen umgangen werden, die Schwachstellen im Skriptcode der Website ausnutzen.
- Unzureichende Zugriffsprüfungen auf Objekte können umgangen werden, z. B. wenn die Anwendung externe Eingaben verwendet, um einen Pfad zur angeforderten Ressource zu konstruieren.
- Eine Anwendung prüft nur, ob sich ein Anwender authentifiziert hat, aber nicht, auf welche Ressourcen er zugreifen darf.
- Vorhersehbare Cookies: Cookies sind Schlüssel-Wert-Paare, die zwischen Client und Server hin- und hergeschickt werden. HTML ist ein zustandsloses Protokoll. Das heißt, wenn ein bereits authentifizierter Anwender mehr als eine Anfrage über http oder https sendet, kann der Server nicht allein mittels HTML

[1] Übersetzung: „Cross Site Request Forgery (CSRF)-Angriffe liegen vor, wenn eine betrügerische Website den Webbrowser eines Benutzers dazu bringt, eine nicht wünschenswerte Aktion auf einer vertrauenswürdigen Website durchzuführen."

[2] Ein Regelwerk zur Verwaltung von Interaktionen während einer Sitzung– z. B. zwischen einem Client und einem Server oder einem Anwender und einer Webanwendung. Eine Sitzung beginnt mit dem Login und wird durch ein Ausloggen beendet.

entscheiden, ob der Anwender dazu berechtigt ist. Ein eindeutiges Cookie, das mit der Anfrage gesendet wird, kann dem Server jedoch mitteilen, dass der Anwender sich authentifiziert hat und eine Ressource anfordern darf. Wenn ein Cookie für einen Angreifer vorhersehbar ist (z. B. weil es sich nur um den enkodierten Namen des Anwenders handelt), kann ein Angreifer die Identität des Anwenders sehr leicht fälschen. Wenn ein Cookie gestohlen wird, kann es auch zu diesem Zweck verwendet werden – insbesondere, wenn das Cookie nie abläuft.

2.4.9 Verschlüsselte Kommunikation und Website-Authentizität

Ein gutes modernes Verschlüsselungssystem implementiert die folgenden Prinzipien:

- Kerckhoffs'sches Prinzip: Die Stärke der Verschlüsselung ist unabhängig von der Geheimhaltungsstufe der Methode, die zur Verschlüsselung einer Nachricht verwendet wird. Sie sollte nur von der Anzahl der verfügbaren Schlüssel und der Zeit abhängen, die ein Angreifer benötigt, um den richtigen Schlüssel zu finden.
- Der verschlüsselte Text muss die Eigenschaften einer zufälligen Zeichenfolge besitzen , sodass eine statistische Analyse die Verschlüsselung nicht brechen kann.
- Wenn sich nur ein Teil der Nachricht ändert, muss sich der gesamte verschlüsselte Text ändern – nicht nur ein Teil. Andernfalls könnte eine differentielle Kryptoanalyse[1] die Verschlüsselung knacken.
- Perfect Forward Secrecy [MIT17]: Jeder Nachrichtenaustausch verwendet einen anderen Schlüssel. Selbst wenn die Verschlüsselung eines Informationsaustauschs in der Zukunft geknackt wird, bleiben andere abgefangene verschlüsselte Texte geheim.

Symmetrische Verschlüsselung

Bei der klassischen Verschlüsselung einer Nachricht wird ein Schlüssel zusammen mit einem Algorithmus verwendet, um einen Klartext in einen verschlüsselten Text zu verwandeln. Die Rückwärtstransformation verwendet denselben Schlüssel und einen inversen Algorithmus zur Entschlüsselung der verschlüsselten Nachricht. Mit dem Schlüssel und dem Algorithmus ist die Ver- und Entschlüsselung relativ einfach. Bekannte historische Beispiele sind die Caesar- und die Vigenère-Verschlüsselung. Ein weiteres bekanntes Beispiel ist das One-Time-Pad. Bei dieser Methode wird für jede Nachricht ein zufälliger, individueller Schlüssel verwendet, der mindestens so lang wie die Nachricht selbst ist und niemals wiederholt werden darf. Wenn ein Angreifer nicht in den Besitz des Schlüssels gelangen oder ihn erraten kann, ist das One-Time-Pad nachweislich nicht zu brechen. Die Schwäche jedes symmetrischen Verfahrens ist die Notwendigkeit, die Schlüssel für die Ver- und Entschlüsselung im Vorfeld der verschlüsselten Kommunikation sicher auszutauschen.

Asymmetrische Verschlüsselung

Die Bezeichnung asymmetrische Verschlüsselung rührt von der Nutzung zweier unterschiedlicher Schlüsselsätze her. Jeder Anwender erzeugt ein Schlüsselpaar: Der erste Schlüssel des Paares wird veröffentlicht und daher „öffentlicher Schlüssel“

[1] Wenn ein Angreifer zwei einander sehr ähnliche Nachrichten abfängt, die sich nur durch eine kleine Änderung unterscheiden, z. B. dem Datum, kann er die Unterschiede im verschlüsselten Text vergleichen.

genannt. Er kann von jedem verwendet werden, der eine verschlüsselte Nachricht an den Besitzer des Schlüssels schicken möchte. Der zweite Schlüssel ist der „private Schlüssel“ und wird für die Entschlüsselung durch den Besitzer des Schlüssels verwendet. Daher muss dieser Schlüssel unter allen Umständen geheim gehalten werden. Die verwendeten Verfahren beruhen auf komplizierten mathematischen Konzepten.
Bei der asymmetrischen Verschlüsselung werden mathematische Funktionen verwendet, die schnell und einfach den Klartext aus einer Verschlüsselung berechnen können, wenn der richtige private Schlüssel bekannt ist, aber eine enorme Rechenleistung benötigen (möglichst mehrere Hunderttausend Jahre), wenn dies nicht der Fall ist. Das Verfahren erfordert keinen sicheren Schlüsselaustausch. Die Geheimhaltung hängt von der Zeit ab, die benötigt wird, um den privaten Schlüssel zu erraten oder zu berechnen und von der Annahme, dass kein algorithmisches Verfahren gefunden und kein Computer gebaut wird, der diesen Prozess um mehrere Größenordnungen beschleunigen kann.

Website-Authentifizierung
Asymmetrische Verschlüsselung kommt auch bei elektronischen Signaturen zum Einsatz. Vertrauenswürdige Zertifizierungsstellen (Trusted Certificate Authorities) stellen digitale Zertifikate aus. Ein digitales Zertifikat bescheinigt, dass das darin genannte Subjekt im Besitz eines öffentlichen Schlüssels ist. Die Zertifizierungsstelle ist eine Organisation, die die Authentizität des Zertifikatinhabers sicherstellt und das ausgestellte Zertifikat elektronisch signiert. Hersteller von Browsern und Betriebssystemen verlassen sich auf die Zertifikate, die von solchen Organisationen ausgestellt werden. Viele Zertifikate sind sogenannte domainvalidierte Zertifikate, die nicht sicherstellen, dass eine bestimmte juristische Person mit dem Zertifikat einer Website verbunden ist. Ein Extended-Validation-Zertifikat wird ausgestellt, wenn die rechtliche Identität des Website-Besitzers festgestellt und bestätigt wurde. Diese Art von Zertifikat sollte z. B. für einen Webshop verwendet werden.
Ein selbstsigniertes Zertifikat wurde mit dem geheimen Schlüssel seines Ausstellers signiert und kann mit dessen öffentlichem Schlüssel verifiziert werden. Dies bedeutet, dass keine dritte Partei, wie z. B. eine vertrauenswürdige Zertifizierungsstelle, an der Feststellung der Vertrauenswürdigkeit des Zertifikats beteiligt ist. Selbstsignierte Zertifikate sollten als nicht vertrauenswürdig angesehen werden.

2.5 Social Engineering (20 Minuten)

Unter Social Engineering versteht man im Kontext der Informationssicherheit jede psychologische Beeinflussung von Menschen, die eingesetzt wird, um Informationen über oder Zugriff auf Computersysteme zu erlangen.

Ein typischer Social-Engineering-Angriffszyklus [URL:SEF] besteht aus 4 Phasen:
1. Informationssammlung: Sammeln von Informationen über das Ziel, z. B. Mitarbeiter, Anwender der Plattform, potenzielle Opfer usw. Häufig werden soziale Medien, Firmen- oder private Webseiten usw. als Informationsquellen genutzt.
2. Aufbau einer Beziehung und Schaffung von Vertrauen: Je nach Ziel kann dies durch wiederholte Telefonate, E-Mails, Kontakte über soziale Netzwerke usw. erfolgen.

3. Ausnutzen: Der Angreifer verwendet gesammelte Informationen und aufgebaute Beziehungen, um das Ziel zu infiltrieren.
4. Ausführung: Das eigentliche Ziel des Angriffs wird erreicht und Spuren werden nach Möglichkeit verwischt, sodass das Opfer möglicherweise nicht einmal bemerkt, dass ein Angriff stattgefunden hat.

Gängige Erscheinungsformen des Social Engineering sind:

- Phishing: Eine E-Mail, die scheinbar von einer seriösen Quelle stammt, verleitet den Anwender zum Besuch einer betrügerischen Webseite, persönliche Daten preiszugeben oder Schadsoftware herunterzuladen. Eine Sonderform ist das Spear-Phishing, bei dem die Phishing-Mail spezifisch auf das Opfer oder einen kleineren Personenkreis zugeschnitten ist. Eine weitere Sonderform ist das „Whaling“, eine Sonderform des Spear-Phishings, die auf die oberste Führungsebene eines Unternehmens abzielt.
- SMiShing: Genau wie Phishing, aber unter Verwendung von SMS, MMS oder anderen Messenger-Diensten als Medium.
- Vishing: Die Praxis, einer Person bei einem Telefonat Informationen zu entlocken oder Einfluss auf sie auszuüben. Angreifer geben sich oft als Kunden, technischer Support von Telefongesellschaften oder Internetdienstanbietern oder als Mitarbeiter von Finanzinstituten aus.

2.6 Sicherheit bei drahtloser Kommunikation (25 Minuten)

Bei einer drahtlosen Kommunikation werden Informationen von einem Sender zu einem Empfänger übertragen, wobei beide nicht durch ein informationstragendes Gerät (z. B. ein leitendes Kabel oder Glasfaser) verbunden sind. Folglich ist die Informationsübertragung für jeden zugänglich, der über geeignete technische Mittel verfügt.

Bedrohungen für die drahtlose Kommunikation

Ein Angreifer könnte einen Empfänger einrichten und Übertragungen aufzeichnen. Dies wird als Sniffing bezeichnet und ist ein passiver Angriff. Bei drahtlosen Übertragungen kann Sniffing in der Regel nicht erkannt werden. Sniffing wird auch in kabelgebundenen Netzwerken wie Computernetzwerken angewendet. Der Angreifer kann einen Sender einrichten und versuchen, eine Übertragung zu stören, indem er Störsignale sendet (DoS) oder er kann sich als legitime Informationsquelle ausgeben (Spoofing).
Wenn Informationen über Relaisstationen wie Funk-Repeater, Basisstationen in Mobilfunknetzen oder drahtlose Router in einem Computernetzwerk ausgetauscht werden, kann der Angreifer einen Man-in-the-middle-Angriff durchführen, indem er mittels einer vorgetäuschten Relaisstation den Informationsaustauch steuert. Insbesondere Relaisstationen mit Zugang zum Internet, werden bei diesem Angriff gerne missbraucht.

Drahtlose Fernbedienungen und andere Geräte, die Befehle über einen drahtlosen Kanal empfangen, können anfällig für einen „Replay-Angriff“ sein: Ein Angreifer zeichnet das Signal eines Geräts auf und spielt es später mit einer geeigneten Funkeinheit wieder ab. Ein Einbrecher, der diese Technik bei einem unsicheren System anwendet, könnte so ein Garagentor oder ein Auto öffnen.

Bitte beachten Sie, dass potenziellen Bedrohungen einer drahtlosen Kommunikation auch für kabelgebundene Systeme bedeutsam sind.

Bedrohungsabwehr in der drahtlosen Kommunikation

Zur Sicherung der Kommunikation über einen drahtlosen Kanal sind mindestens sämtliche der folgenden Maßnahmen erforderlich:

- Jegliche Kommunikation muss ausreichend verschlüsselt sein.
- Alle Sender und Empfänger müssen manipulationssichere Kennungen haben.
- Es muss einen sicheren Authentifizierungsprozess für Sender und Empfänger geben.

Eine adäquate Verschlüsselung erschwert Sniffing und die Vorbereitung von Spoofing- und Man-in-the-middle-Angriffen. Manipulationssichere Kennungen in Verbindung mit einer sachgerechten Authentifizierung mitigieren Spoofing- und Man-in-the-middle-Angriffe.

WiFi ist eine Abkürzung für Wireless Fidelity und beschreibt Technologien zum Aufbau von Verbindungen innerhalb eines WLAN (Wireless Large Area Network), die dem Standard IEEE 802.11 entsprechen (Vijay K. Varma im IEEE Emerging Technology portal, 2006 – 2012) und damit Zugangspunkte zu einem lokalen Netz oder sogar zum Internet bieten.
Die Kommunikation erfolgt über elektromagnetische Wellen in den Funkfrequenzbereichen UHF und SHF[1].
Offene WLAN-Stationen erfüllen keine der oben genannten Anforderungen an eine sichere Kommunikation. Kennungen wie eine MAC-Adresse (Media-Access-Control-Adresse) eines Geräts oder ein SSID (Service Set Identifier, d. h. der „Name“ des Funknetzwerks) können gefälscht werden. Die Kommunikation zwischen einem Gerät und der WLAN-Anlage ist nicht verschlüsselt und es findet keine Authentifizierung statt. Zur Sicherung der WLAN-Kommunikation müssen die WiFi Protected Access-Protokolle WPA2 und WPA3 verwendet werden. Diese Protokolle unterstützen zusätzliche Identifizierungs- und Authentifizierungsmethoden wie die Verwendung eines Passworts, einer Benutzername/Passwort-Kombination und eines digitalen Zertifikats. Die Kommunikation zwischen einem Gerät und einem Zugriffspunkt wird adäquat verschlüsselt.
Bluetooth ist ein von der [Bluetooth SIG] gepflegter Standard für drahtlose Technologien zum direkten Datenaustausch zwischen Geräten über kurze Distanzen.[2]
Ein benutzerfreundlicher Identifikations- und Autorisierungsmechanismus (Gerätepaarung) ist vorhanden und die Kommunikation ist verschlüsselt.
Schwachstellen im Protokoll oder bei dessen Verwendung oder eine schlechte Protokollimplementierung können sowohl bei der Bluetooth- als auch bei der WLAN-Kommunikation zu Sicherheitslücken führen.

---

[1] UHF (Ultra High Frequency) = Frequenzband zwischen 300 MHz und 3 GHz, SHF (Super High Frequency) zwischen 3 GHz und 40 GHz. Diese Frequenzbänder werden auch von einigen Fernbedienungen genutzt, z. B. von Presentern, Autoschlüsseln oder Garagentoren. Auch Bluetooth nutzt das UHF-Frequenzband. Im UHF-Spektrum nutzen WLAN und Bluetooth normalerweise das offene ISM-Band zwischen 2,4 und 2,48 GHz, das für industriellen, wissenschaftlichen und medizinischen Funkverkehr reserviert ist.

[2] Je nach Bluetooth-Klasse eines Geräts kann diese zwischen einem halben Meter und 100 Metern betragen.

3. Sicherheit im Softwarelebenszyklus (350 Minuten)

Begriffe

Application Security Development Lifecycle Process[1] (ASDL), CVE, CVSS, CWE, Datenflussdiagramm (DFD), Design Pattern (Designmuster), Design Principle (Designprinzip), Ethical Hacking, fuzz testing, Incident Response Plan (Notfallplan), Misuse Case, Penetrationtest, security framework (Sicherheitsrahmenwerk), security plan (Sicherheitsplan), STRIDE, Threat Modeling (Bedrohungsmodellierung), Vulnerability Assessment

Lernziele

(3.1.1) Die wichtigsten Aktivitäten im Lebenszyklus für eine sichere Anwendungsentwicklung (ASDL) beschreiben (K2)

(3.2.1.1) Die Zusammenhänge zwischen Bedrohung, Anforderung und Risikominderung kennen (K1)

(3.2.1.2) Potenzielle Bedrohungen einem gegebenen Datenflussdiagramm eines Systems zuordnen und geeignete Maßnahmen zur Risikominderung vorschlagen (K3)

(3.2.1.3) Bedrohungen und Bedrohungsabwehrmaßnahmen in einem vorgegebenen Misuse case Diagramm deuten (K3)

(3.2.2.1) STRIDE-Kategorien mit vorgegebenen Sicherheitslücken verknüpfen (K3)

(3.2.2.2) Den Zweck von CVSS und seinen Basismetriken erklären (K2)

(3.3.1.1) Zweck und Ziele sicherer Design Principleien beschreiben (K2)

(3.3.1.2) Zweck und Ziele sicherer Design Pattern kennen (K1)

(3.3.2.1) Zweck und Ziele des sicheren Programmierens erläutern (K2)

(3.4.1) Typische Ziele von Tests der IT-Sicherheit wiedergeben (K1)

(3.4.2) Typische Eingangs- und Endekriterien für IT-Sicherheitstests benennen (K1)

(3.4.3) Zwecke und Ziele verschiedener Arten von Tests der IT-Sicherheit unterscheiden (K2)

(3.5.1) Verstehen, warum sicherheitsrelevante Fehlerzustände anders als andere Fehlerarten behandelt werden sollten (K1)

(3.5.2) Erklären, wie die MITRE-Initiative „Common Vulnerabilities and Exposures“ (CVE) funktioniert (K1)

[1] Entwicklungsprozessmodell für (Informations-)sichere Anwendungen

3.1 Der Security-Development-Lifecycle-Prozess (30 Minuten)

Howard und Lipner konstatieren in [Howard 06]: *„Den gegenwärtigen Softwareentwicklungsmethoden mangelt es an Sicherheitsbewusstsein, Disziplin, bewährten Verfahren und Sorgfalt (…)“.* Sie beschreiben einen von Microsoft gewählten Ansatz, der Systemsicherheit in jeder Phase des Softwarelebenszyklus berücksichtigt.
Ein Application Security Development Lifecycle Process (ASDL) bezieht Sicherheitsaktivitäten in alle Schritte ein, die zur Entwicklung, Management, Betrieb und Wartung einer Anwendung erforderlich sind. Er berücksichtigt auch das Infrastrukturmanagement und Audit-Maßnahmen. Ein allgemeiner Prozess wird in [ISO/IEC 27034] diskutiert.
Der ASDL ist kein sequenzielles Modell. Es beschreibt, welche Prozessphasen und -disziplinen in einem bestehenden Lebenszyklus angepasst werden müssen. Obwohl der Schwerpunkt der zitierten Norm auf der sicheren Entwicklung von Anwendungen liegt, lässt sich das Phasenmodell auch generell auf die Entwicklung von Systemen übertragen[1]. Ein Lebenszyklusmodell, mit dem ein angemessenes Sicherheitsniveau gewährleistet werden kann, erfordert mindestens (vgl. [ISO/IEC 27034] und [Howard 06]):

1. Schulung: Alle Mitarbeiter, die an der Erstellung einer Anwendung beteiligt sind (z. B. Entwickler, Tester und Programmmanager), absolvieren eine Grundlagenschulung in IT-Sicherheit. Themen der Grundlagenschulung sind Datenschutz, Threat Modeling, sicheres Design, Programmierung und Sicherheitstests. Weiterführende Schulungen zu diesen Themen werden je nach Bedarf angeboten.
2. Anforderungen: Dazu gehören sowohl Anforderungen an die Anwendungssicherheit als auch Prozessanforderungen – z.B. die Klärung der Fragen, wann und wie oft Bedrohungsanalysen durchzuführen sind oder welche Arten von Tests der IT-Sicherheit in den verschiedenen Lebenszyklusaktivitäten erforderlich sind. In dieser Phase muss das Projektteam auch Freigabe-Kriterien (item pass/fail criteria) mit Bezug zu sicherheitsrelevanten Fehlerzustände definieren. Die Einhaltung dieser Kriterien muss während des gesamten Projektverlaufs überwacht werden[2].
3. Design: In der Designphase der Anwendung müssen Angriffsflächen identifiziert und eine Bedrohungsmodellierung erfolgen. (siehe Abschnitt 3.2 dieses Lehrplans). Der Zweck dieser Aktivitäten besteht darin, Sicherheitsmängel in der Architektur der Anwendung aufzudecken und zu ermitteln, wo Sicherheitsmechanismen implementiert werden müssen.
4. Im Rahmen der Aktivitäten zur Implementierung wird der Einsatz geeigneter Werkzeuge zur statischen Analyse gefordert. Unsichere Funktionen werden entdeckt und ausgeschlossen. Dies stellt sicher, dass keine bekannte Schwachstellen im Code verbleiben und der Einbringung neuer Schwachstellen in den Code vorgebeugt wird.
5. Verifizierung: Alle Verifizierungsphasen beinhalten spezifische Sicherheitstestaktivitäten, beispielsweise dynamische Analyse und Fuzz

[1] Zum Vergleich siehe [NIST-SP-800-160].
[2] Siehe “Bug Bars” (Fehlerbalken) im Microsoft Security Development Process [Howard 06].

testing (siehe Abschnitt 3.4.2). Die Verifizierung muss ein Risikomanagement beinhalten, zumindest eine erneute Durchsicht aller Angriffsflächen. Ein solches Review stellt sicher, dass das Produkt angemessen geschützt ist und keine weiteren Angriffsvektoren durch Tests aufgedeckt wurden.

6. Release: Release-Aktivitäten sind die Aktionen, die zur Vorbereitung einer Freigabe erforderlich sind. Sie beinhalten ein abschließendes Review aller Sicherheitsaktivitäten und deren Erfolg. So wird sichergestellt, dass der ASDL-Prozess in allen Entwicklungsphasen korrekt umgesetzt wurde. In einer agilen Umgebung kann dies inkrementell in jeder Iteration durchgeführt werden. Bevor eine Anwendung freigegeben wird, muss das verantwortliche Projektteam einen Notfallplan (Incident Response Plan) für die Zeit nach der Freigabe definieren. Der Plan beschreibt erforderliche Maßnahmen, wer informiert werden muss, wie Vorfälle analysiert und wie zugehörige Bedrohungen gemindert oder ganz beseitigt werden. Außerdem sollte die Release-Konfiguration so archiviert werden, dass Änderungen durch Dritte während oder nach der Auslieferung (z. B. durch Download) erkannt werden können. Eine gängige Methode hierfür ist das „Signieren“ einer Version mit einem kryptografischen Hash [URL:Signing].
7. Response: Die wichtigsten Response-Aktivitäten umfassen die Einrichtung eines Response-Teams und die Implementierung bzw. Vorbereitung der im Notfallplan für das freigegebene Produkt vorgesehenen Maßnahmen.

Ein Sicherheitsplan nach [NIST 800-39] kann beschreiben, wie die hier gelisteten Aktivitäten bei einem bestimmten Projekt durchgeführt werden. Der Sicherheitsplan kann bzgl. Testaktivitäten auf ein Testkonzept verweisen. Bei agilen Ansätzen müssen sich die Aktivitäten z. B. in Team-Chartas, Definitionen von „ready“ (bereit) und „done“ (erledigt) sowie Abnahmekriterien widerspiegeln.
Ein Sicherheitsplan sollte mit dem Sicherheitsrahmenwerk der Organisation kompatibel sein.
Ein Sicherheitsrahmenwerk besteht aus einer Reihe von Vorgaben, erwünschten Ergebnissen, Richtlinien, Beschreibung von Aktivitäten und Rollen zur Identifizierung, Erkennung, zum Schutz vor und zur Reaktion auf Bedrohungen für die Vermögenswerte einer Organisation. Es sollte auch Wiederherstellungsmaßnahmen beschreiben [NIST 2018].

3.2 Bedrohungsmodellierung und Anforderungsentwicklung (145 Minuten)

Bedrohungsmodelle dienen der Ermittlung von Sicherheitsanforderungen, Anforderungen für Maßnahmen zur Bedrohungsminderung und ‚non-requirements', für Situationen in denen eine Bedrohung durch das System nicht abgeschwächt werden kann (siehe Kapitel 12 in [Shostack 14]). Sie werden auch zur Bewertung von Schutzmaßnahmen sowie zur Ermittlung und Priorisierung von Testbedingungen für Sicherheitstests verwendet. Es besteht eine enge Wechselwirkung zwischen der Bewertung von Bedrohungen, der Ermittlung möglicher Abhilfemaßnahmen und der Festlegung von Anforderungen. [Shostack 14]. Daher müssen diese Aufgaben iterativ durchgeführt werden.

Als Bedrohungsmodellierung (engl. Threat modeling) bezeichnet man jeden strukturierten Ansatz, der die mit einem System oder einer Anwendung verbundenen Sicherheitsrisiken identifiziert, bewertet und behandelt. Sie ist ein streng risikobasierter Ansatz, der folgende Schritte umfasst:

- Aufspalten des Systems in Bestandteile und Identifizierung von Angriffsflächen sowie zugehörigen Bedrohungen
- Erfassen/analysieren und bewerten der Bedrohungen
- Maßnahmen zur Abhilfe implementieren

Die in diesem Kapitel vorgestellten Methoden kommen typischerweise in der ASDL-Anforderungs- und Designphase zur Anwendung.
Orientierungshilfe zur Bedrohungsidentifikation und -analyse sowie zur Entwicklung geeigneter Gegenmaßnahmen auf Organisationsebene findet man in [ISTQB-AL-SEC] und in [NIST-SP-800-37]. [Shostack 14] beschreibt einen umfassenden Ansatz zur Bedrohungsmodellierung, Definition von Sicherheitsanforderungen und Implementierung von präventiven und defensiven Maßnahmen.
Außerdem können Ansätze wie CAPEC™ oder CWE™ bei der Bedrohungsmodellierung zum Einsatz kommen.

3.2.1 Identifikation von Bedrohungen

Anwendungsfalldiagramme helfen, Bedrohungsverursacher und Angriffsvektoren bereits in frühen Designphasen zu identifizieren. Dazu werden bestehende Anwendungsfälle um die Beschreibung eines Verhaltens erweitert, das der System-/Entitätsinhaber nicht wünscht [Sindre 05]. Letzteres wird als Misuse case bezeichnet. Zudem wird die Menge der Akteure aus den ursprünglichen Anwendungsfällen um die Kategorie ‚Misactor' erweitert. Ein Misactor charakterisiert einen Angreifertyp, genauso wie der Akteur einen Benutzertyp repräsentiert.
Misuse cases werden aus Anwendungsfällen in 5 Schritten abgeleitet werden [Sindre 05]:

1) Beschreibung der normalen Akteure und Anwendungsfälle
2) Identifikation der wesentlichen Angreifertypen (als misactor) und Misuse cases.
3) Untersuchung und Modellierung der Beziehungen zwischen Anwendungs-fällen und Misuse cases.
4) Einführung zusätzlicher Anwendungsfälle, die dem Erkennen, Vermeiden oder dem Eindämmen von Misuse cases dienen.

5) Ableitung und Dokumentation von detaillierten Sicherheitsanforderungen aus den überarbeiteten Anwendungsfallmodellen.

Aus der Anforderungs- und Entwicklungsperspektive unterstützen Missbrauchsfälle die Identifikation erforderlicher Sicherheitsfunktionen.
Aus der Testperspektive gesehen stellen die Missbrauchsfälle aus Schritt 2 und die Anwendungsfälle aus Schritt 4 Testbedingungen dar. Diese können dazu verwendet werden, Szenarien für Penetrationtests zu entwickeln.

Datenflussdiagramme (DFD) unterstützen die Bedrohungsmodellierung auf Systemarchitektur-Ebene.
Ein Datenflussdiagramm beinhaltet [DeMarco 78], [Howard 06]:

- Datenspeicher – dargestellt durch zwei horizontale, parallele Linien
- Elemente, die Systemteile darstellen die Eingaben oder Ausgaben liefern oder empfangen – dargestellt durch Rechtecke
- Funktionen oder (Sub-)Systeme, die Daten und Datenflüsse verarbeiten – in einem DFD als Kreise dargestellt
- Datenflüsse zwischen den oben genannten Komponenten – abgebildet als durchgehende Linien mit einer Pfeilspitze
- Systemgrenzen – dargestellt als gestrichelte Linien [Pohl 11]

Mit Hilfe von DFDs kann lässt sich feststellen, welche Datenflüsse kompromittiert sein könnten und wo dies Schaden anrichten könnte. Softwarearchitekten und Entwickler können aus diesen Diagrammen herleiten, wo Sicherheitsmechanismen eingerichtet werden müssen. Ein Tester kann festlegen, welche externen und internen Systemschnittstellen einem Sicherheitstest unterzogen werden müssen.

In späteren (z. B. detaillierten) Design- und Implementierungsphasen erfordert die Bedrohungsmodellierung ein tiefes technisches Verständnis der relevanten Datenstrukturen, Speicherlayouts, Programmiersprachen und Kommunikationsprotokolle. Dies erfordert die Fähigkeiten eines erfahrenen Hackers und wird hier nicht behandelt.

3.2.2 Bedrohungsermittlung und -bewertung

STRIDE

Sobald Angriffsflächen und damit in Zusammenhang stehende Bedrohungen identifiziert wurden, sollte ihre Beziehung weiter analysiert, klassifiziert und bewertet werden.
Zur Unterstützung dieses Vorhabens hat Microsoft die STRIDE-Methodik ([URL:STRIDE], [Howard 06], [Howard 02]) eingeführt. Für jede Angriffsfläche werden Bedrohungen aus den folgenden sechs Kategorien identifiziert:

- S – Spoofing Identity (Identität vortäuschen): Jede Bedrohungsaktion, die darauf abzielt, die Authentifizierungsinformationen einer anderen Entität zu erlangen und diese zu verwenden. Ein Beispiel für eine Bedrohung dieser Kategorie wäre eine Anwenderoberfläche mit unsicheren Zertifizierungsmechanismen.
- T – Tampering (Daten-Manipulation): Daten-Manipulationen können auftreten, wenn ein Angreifer bestehende Daten modifizieren oder verändern kann. Stored XSS und die Installation von Schadprogrammen sind Beispiele, die in diese Kategorie fallen.

- R – Repudiation (Nicht-Nachweisbarkeit): Böswillige Operationen oder Systemschwächen, die eine spätere Analyse verhindern. Insbesondere fehlende Funktionen zur Nachverfolgung von legalen sowie illegalen Transaktionen ermöglichen eine Nicht-Nachweisbarkeit.
- I – Information Disclosure (Offenlegung von Informationen): Ein Angreifer kann Informationen ohne die erforderliche Zugriffsberechtigung lesen. Ein konkretes Beispiel ist die bereits erwähnte Heartbleed-Sicherheitslücke.
- D – Denial of Service: Siehe Kapitel 1.
- E – Elevation of Privilege (Erhöhung der Zugriffsrechte): Diese Kategorie trifft zu, wenn ein System oder eine Anwendung Aktionen zulässt, die darauf abzielen, die autorisierte Zugriffsebene so anzuheben, dass sie Zugriff auf zugriffsbeschränkte Informationen oder Dienste erlaubt – beispielsweise Administratorenrechte.

CVSS

„Das Common Vulnerability Scoring System (CVSS) bietet eine Möglichkeit, die wichtigsten Merkmale einer Schwachstelle zu erfassen und einen numerischen Wert zu ermitteln, der ihren Schweregrad widerspiegelt.“ [URL:CVSS].
Die wichtigsten Merkmale einer Sicherheitslücke werden in drei Hauptkategorien erfasst

- Die Basismetriken beinhalten eine Bewertung der Ausnutzbarkeit, Größe und Auswirkungen der Sicherheitslücke.
- Zeitliche Metriken evaluieren einerseits den aktuellen Stand der Technik in Bezug auf die Verfügbarkeit von automatisierten Exploits und andererseits die Verfügbarkeit von Abhilfemaßnahmen für die Sicherheitslücke.
- Umgebungsmetriken ermöglichen eine Anpassung der Risikobewertung in Bezug auf die Organisation des CVSS-Anwenders.

Die CVSS-Basismetriken ermöglichen ein erstes umfassendes Assessment des Schweregrads einer Sicherheitslücke. Im Rahmen dieses Lehrplans werden nur diese Basismetriken besprochen[1].

Die CVSS-Basismetriken ab Version 3.1 im Detail:
<u>Ausnutzbarkeit</u>

1. Der ‚Attack Vector‘ (Angriffsvektor) bewertet die erforderliche Nähe eines Angreifers zu seinem Angriffsziel. Vier Werte sind möglich: „P – Physical“ erfordert physischen Zugang zum Ziel, z. B. einer Festplatte oder Speicherkomponente. „L – local“ erfordert lokalen Zugriff, z. B. über die Tastatur eines verwundbaren Rechners. „A – adjacent“ gibt vereinfacht gesagt an, dass ein Zugriff auf eine lokale Netzwerkprotokollebene erforderlich ist. Schließlich besagt „N – network“, dass die Schwachstelle von buchstäblich jedem Ort im Internet aus angreifbar ist.
2. Die ‚Attack Complexity‘ (Angriffskomplexität) wird mit ‚high‘ (H) oder ‚low‘ (L) bewertet. Hoch gilt für Angriffe, deren erfolgreiche Durchführung Voraussetzungen erfordert, die außerhalb der Kontrolle des Angreifers liegen, also wenn z. B. bestimmte Konfigurationen vorhanden oder ein anderer Angriffstyp erfolgreich sein muss, um Zugang zu der fraglichen Sicherheitslücke

[1] Umfassendes Schulungsmaterial zu allen Aspekten von CVSS ist direkt bei FIRST [URL:CVSS] erhältlich.

zu erhalten. Liegen keine solchen Bedingungen vor, wird die Komplexität als niedrig (low) eingestuft.

3. ‚Privileges Required' (Erforderliche Rechte) beschreibt, welche Zugriffsrechte der Angreifer benötigt, um zum Erfolg kommen zu können. Die Metrik erhält den Wert ‚high' (H), wenn z. B. Administratorrechte erforderlich sind, ‚low' (L), wenn Basiszugriffsrechte, z. B. die eines „normalen" Anwenders, ausreichen und ‚none' (N), wenn keinerlei Autorisierung erforderlich ist.
4. ‚User Interaction' (Anwenderinteraktion) gibt an, ob der Angreifer die Mitwirkung des Anwenders benötigt, z. B. eine Bestätigung. (N) für 'none' wird vergeben, wenn keine unterstützende Aktion des Anwenders erforderlich ist und (R) (required) andernfalls.

Scope

Der Grundgedanke der Scope-Metrik ist die Bewertung, ob die Ausnutzung einer Sicherheitslücke eines Systems, Ressourcen in einem weiteren System außerhalb der Kontrolle des für das angegriffene System Verantwortlichen beeinflusst (veränderter (changed) Umfang – C) oder nicht (unveränderter Umfang – U). Ein Angriff auf die virtuelle Maschine eines Cloud-Nutzers mit dem Potential, in die umgebende Hardware des Cloud-Anbieters einzubrechen, ist ein Beispiel für einen veränderten Umfang.

Impact

Die Auswirkungsmetrik (Impact) bewertet jedes der drei Attribute der Sicherheitstriade CIA mit H (high bzw. hoch) für einen vollständigen Verlust, L (low bzw. niedrig) für einen begrenzten Verlust oder N (none) für keinen Verlust der Sicherheit im jeweiligen Attribut.

Auf Basis dieser Metriken kann ein Risiko-Score für jede Sicherheitslücke berechnet werden. Für die Berechnung steht ein Online-Tool zur Verfügung.

3.3 Prinzipien für sicheres Design und sichere Programmierung (95 Minuten)

3.3.1 Sicheres Design

Die Identifizierung und Implementierung geeigneter sicherer Designtechniken gehört zur dritten Gruppe der ASDL-Aktivitäten. Hier sollten sichere Designprinzipien und sichere Designmuster (Design Pattern) angewendet werden.

Designprinzipien
Designprinzipien sind Leitlinien, die helfen, häufige Designschwächen zu vermeiden, z. B. die Möglichkeit ein bestehendes Sicherheitskonzept zu unterlaufen.
Mindestens die folgenden Prinzipien sollten berücksichtigt werden[1]:

- Complete Mediation (Vollständige Zugriffsprüfung): Jeder Zugriff auf jedes Objekt muss auf seine Rechtmäßigkeit geprüft werden.
- Least Privilege (Geringste Rechte): Berechtigungen werden nur in dem Maß und für den Zeitraum erteilt, der für die Ausführung einer Aufgabe benötigt wird. Niemals mehr. Niemals länger.
- Need-to-know: Der Zugriff auf Daten sollte auf die Daten beschränkt sein, die für rechtmäßige Zwecke benötigt werden.
- Auditierbarkeit und Wiederherstellung: Ereignisse müssen protokolliert werden, damit Angriffe analysiert werden können. Daten und Dienste müssen gegen Verlust und Denial-of-Service geschützt werden.
- Secure the weakest link: Absicherung des schwächsten Gliedes dient der effizienten Reduzierung der Angriffsfläche. Man sollte den Aufbau von Schutzmaßnahmen für unverteidigte Zugangspunkte durchführen, bevor man bereits bestehende Schutzmaßnahmen verbessert.
- Defend-in-depth-and-diversity – vielschichtige und vielfältige Verteidigungsmechanismen nutzen: Die Verteidigung eines Systems sollte aus mehreren Schichten physischer, technischer und administrativer Kontrollen bestehen.
- No security by obscurity – Keine Sicherheit durch Verstecken: Verlassen Sie sich nicht auf die Geheimhaltung Ihres Designs oder Ihrer Implementierung als einzige Methode zum Schutz der IT-Sicherheit. Das wird letztendlich scheitern.
- Secure defaults – sichere Voreinstellungen: Die Standardkonfiguration einer Software oder eines Systems sollte den maximalen Sicherheitseinstellungen des Systems entsprechen.
- Fail in a secure manner – Sicheres Versagen: Programme besitzen Fehlerzustände und sind anfällig für Bedienfehler. Man sollte damit im Zusammenhang stehende Sicherheitsrisiken identifizieren und entschärfen, damit auch bei einem Ausfall oder Teil-Ausfall eines Systems ein definiertes Sicherheitsniveau aufrechterhalten werden kann.
- Eingabevalidierung: Alle Eingaben sollten von einem System validiert werden, bevor sie verwendet werden.

[1] Siehe auch Abschnitt 1.3.

- Output encoding – klar definierte Ausgabeformate: Es muss verhindert werden, dass die Ausgaben eines Systems für einen Injektionsangriff auf ein anderes System verwendet werden können.
- Code und Daten trennen: Daten dürfen nie als ausführbarer Code fehlinterpretiert werden[1].
- Usable Security – Anwenderfreundliche Sicherheit: Unterstützen Sie die Anwender dabei, bei der Arbeit mit dem System IT-sichere Entscheidungen zu treffen. Versuchen Sie, das System so zu gestalten, dass seine Hauptnutzungszwecke und die zugehörigen Sicherheitserfodernisse nicht miteinander kollidieren.

Design Pattern

Ein Design Pattern (Entwurfsmuster) ist eine allgemeine, wiederverwendbare Lösung für ein häufiges Designproblem. Sichere Entwurfsmuster zielen darauf ab, das Risiko des versehentlichen Einfügens typischer Sicherheitslücken in ein System zu vermeiden oder zu mindern.
Existierende zertifizierte Implementierungen für gängige Design Pattern sollten (falls vorhanden) verwendet werden für:

- Identifizierung und Authentifizierung
- Autorisierung, z. B. bei Sitzungsverwaltung und Zugangskontrolle
- Verschlüsselung

Ohne das erforderliche Fachwissen und gebührende Sorgfalt können selbstkonzipierte Lösungen schwere Designfehler und daraus resultierende Sicherheitslücken in den bereitgestellten Produkten und Dienstleistungen enthalten.

3.3.2 Sichere Programmierung

Regeln für sicheres Programmieren dienen dem Zweck, typische sicherheitsrelevante Programmierfehler zu vermeiden, die zu folgenden Problemen führen können:

- Unsichere Handhabung von Eingaben und Ausgaben
- Unsichere Handhabung von Speichermedien und Arbeitsspeicher, was z. B. zu Pufferüberläufen führen kann
- Unsicherer Umgang mit ‚race conditions' (Wettlaufsituationen), die zu Abstürzen führen können oder zu unzuverlässigen Autorisierungsprüfungen, wenn z. B. Autorisierung und Datenzugriff unter bestimmten Voraussetzungen in der falschen Reihenfolge erfolgen können
- Unsicherer Umgang mit Berechtigungen und Ressourcen, z. B. nicht spezifizierte Standardgenehmigungen.

[1] Dasselbe gilt natürlich auch für eine Interpretation von Anwender-Eingaben als Konfigurationsdaten, wie das beispielsweise bei einigen CSS-Injections vorkommt, siehe z. B. https://www.owasp.org/index.php/Testing_for_CSS_Injection_(OTG-CLIENT-005).

Eine Organisation sollte Regeln für sichere Programmierung implementieren, die folgende Punkte berücksichtigen:

- Allgemeine sichere Designprinzipien und -muster (Beispiele siehe Abschnitt 3.3.1)
- Programmiersprachenspezifische Regeln[1] zur Vermeidung der Verwendung von unsicheren Sprachkonstrukten und sprachspezifischen Fehlern. [URL:CERT SEI] gibt Beispiele für C und C++.
- Regelmäßige und obligatorische Durchführung von statischen Analysen und Code-Reviews (siehe auch Abschnitt 3.4), um die Einhaltung der Regeln durchzusetzen.

Entwickler müssen auch ein gutes Verständnis für typische Programmierfehler haben. Mit dem Verzeichnis Common Weakness Enumeration (CWE) [URL:CWE] stellt MITRE eine umfassende Sammlung typischer Programm- und Hardware-Schwachstellen zur Verfügung.

3.4 Sicherheitstests (60 Minuten)

Dieser Lehrplan konzentriert sich auf die wichtigsten Ziele der IT-Sicherheit und vermittelt ein praktisches Verständnis von Bedrohungen und deren Analysen.
Eine ausführliche Diskussion von Prozessen in Sicherheitstests, ein detaillierter Blick auf die Ziele von Sicherheitstests und ein organisationsweiter Prozess für das Risikomanagement von Sicherheitsrisiken werden in [ISTQB CTAL-SEC] erörtert.

Typische Ziele von Sicherheitstests sind:

- Erhöhen des Sicherheitsbewusstseins und Aufzeigen des betriebswirtschaftlichen Nutzens von Sicherheitsmaßnahmen durch Sichtbarmachen der möglichen Folgen von Sicherheitsmängeln
- Identifizieren von Sicherheitslücken in Systemarchitekturen, Entwürfen, Abläufen und organisationsweiten Richtlinien und Prozessen
- Verbesserungsvorschläge zu den bestehenden Sicherheitsmechanismen
- Erfüllen gesetzlicher Auflagen
- Entdecken neuer Bedrohungen
- Aufdecken von Schwachstellen in Programmen, Systemen und Prozessen.

3.4.1 Ziele von Sicherheitstests, Eingangs- und Endekriterien

Testziele
Drei Hauptfaktoren bestimmen die Spezifikation von Sicherheitstestzielen ([Palmer 01], [Ruef 07]):

- Das zu schützende Asset
- Wogegen das Asset geschützt werden muss
- Das verfügbare Budget, um den Schutz des Assets zu testen und zu verbessern.

Zur Entwicklung überprüfbarer Sicherheitstestziele ist eine detaillierte Beschreibung der Assets und der damit in Verbindung stehenden Systeme und Dienste

[1] Einschließlich Compilereinstellungen

erforderlich. Allgemeine Aussagen wie „die Datenbank“ oder„die Webshop-Dienste“ reichen nicht aus ([Palmer 01], [Ruef 07]).
Zur Identifizierung und Beschreibung, wovor geschützt werden soll, ist eine Erläuterung der Art der Bedrohung und der möglichen Angriffsvektoren erforderlich, z. B. nach CIA (siehe Kapitel 1) oder STRIDE (siehe Abschnitt 3.2). Sicherheitstests erfordern ein angemessenes Budget: Ein zu kleines Budget erlaubt möglicherweise nur eine konzeptionelle Prüfung (siehe unten), obwohl eigentlich ein voller Penetrationtest erforderlich wäre. Ein risikobasierter Ansatz, der z. B. STRIDE und CVSS anwendet, hilft bei der dem Budget angemessenen Schwerpunktsetzung.

Eingangs- und Endekriterien

Bestimmte Kriterien müssen erfüllt sein, bevor ein Sicherheitstest beginnen kann. Die Mindestvoraussetzungen für den Start eines Sicherheitstests sind:

- Eine Genehmigung und ein Auftrag zur Durchführung der Tests gemäß der Beschreibung in einem Sicherheitstestplan. Bei Penetrationtests muss die Genehmigung durch eine dazu autorisierten Stelle in rechtskräftiger Form erfolgen. Auftraggeber und Penetrationtester müssen sich darauf verständigen. Diese Genehmigung wird in Anlehnung an ein bekanntes Brettspiel oft als „Du kommst aus dem Gefängnis frei“-Karte bezeichnet [Palmer 01].
- Testziele, Zielsysteme und Umfang der Tests sind klar definiert.
- Verantwortliche Ansprechpartner für jedes Zielsystem in der Organisation sind benannt, eingebunden und verfügbar.
- Ein angemessenes ‚Vulnerability Management‘ (Sicherheitslückenmanagement) inklusive eines Kategorisierungsschemas der Sicherheitslücken ist definiert.

Endekriterien stellen sicher, dass alle Testziele der Sicherheitstests eingehalten werden. In einem ASDL sollten für jede Phase Endekriterien hinsichtlich Überdeckung der Testelemente und Pass/Fail-Kriterien für diese definiert werden. Mögliche Überdeckungsdeckungskriterien:

- welche Schnittstellen mit Fuzz testing zu prüfen sind
- wo ein Penetrationtest auf dem Systemtestlevel durchgeführt worden sein muss
- welche Regressionstests von früheren Versionen ausgeführt worden sein müssen

Pass/Fail-Kriterien können bestimmen:

- welche Arten von Sicherheitsproblemen vor der Freigabe behoben werden müssen
- welche Arten von Datenschutzproblemen vor der Freigabe behoben werden müssen

Ergebnisse von Penetrationtests beinhalten oft Vorschläge, wie eine entdeckte Sicherheitslücke beseitigt werden kann.

3.4.2 Arten von Sicherheitstests

Sicherheitsaudit (Security audit)
Sicherheitsaudits auf der Organisationsebene ([RUEF 07], [ISTQB CTAL-SEC]) bewerten die Prozesse und Verfahren in einer Organisation im Hinblick auf die Einhaltung einer Reihe von definierten Sicherheitsrichtlinien und -zielen. In der ASDL sollten organisatorische Audits Teil der Anforderungsphase sein. Ein typischer Gegenstand der Überprüfung ist der Sicherheitsplan. Das Audit muss sicherstellen, dass alle Entwicklungsphasen Maßnahmen zur Entwicklung eines sicheren Produktes enthalten. Ein Audit eines Softwareprojektes auf Organisationsebene sollte in der Release-Phase als Teil der finalen Sicherheitsüberprüfung wiederholt werden.
Konzeptionelle Audits [RUEF 07] bewerten, ob eine vorgeschlagene oder bestehende Lösung ein angemessenes Sicherheitsniveau bietet. Sie können unabhängig oder im Rahmen eines Vulnerability Assessment durchgeführt werden. Im ASDL sollten konzeptionelle Audits Teil der Designphase sein und bei der Implementierung, Verifizierung und Freigabe regelmäßig wiederholt werden. Bei einem bereits existierenden Produkt sollten sie in den Änderungsmanagement-prozess integriert werden.
Organisatorische und konzeptionelle Audits sind Reviews, d. h. statische Prüfungen.

Vulnerability Assessment
Ein technisches Vulnerability Assessment ([RUEF 07], [Weidman 14]) zielt darauf ab, Sicherheitslücken so ökonomisch wie möglich zu erkennen. Es kann sowohl konzeptionelle Audits und statische Analysen als auch dynamische Techniken umfassen. Dynamische Tests können Schwachstellenscans, Fuzz testing bis hin zu umfassenden Penetrationtests sein. Die tatsächlich eingesetzten Techniken hängen vom Kontext der Tests ab [ISTQB_CTFL] und konzentrieren sich auf bestimmte Ziele.
Vulnerability Assessments sollten ein integraler Bestandteil eines jeden SDLC sein und spätestens in der Designphase beginnen.
Vulnerability Assessments sind auch eine häufige Aufgabe bei Wartungstests, siehe [ISTQB_CTFL].

Statische Analyse
Eine statische Analyse bewertet ein Softwareentwicklungs-Artefakt, ohne es auszuführen, typischerweise mit Hilfe eines Werkzeugs.
Im Rahmen der Entwicklung sicherer Systeme und Anwendungen kann die statische Analyse unsichere Funktionen aufdecken, die von einem Programmierer verwendet werden. Beispiele für Analysetools, die solche Konstrukte erkennen können, sind der Clang-Analyzer für C/C++ und FindBugs [URL:FindBugs] und PMD [URL:PMD] für Java. Ein gängiger Ansatz ist die Taint-Flow-Analyse: Der statische Analysator wandelt den Quellcode in ein Modell der Software um und lokalisiert Bereiche in der Software, die anfällig für Injektionen von außen sein können. Die Taint-Flow-Analyse verwendet Techniken, die denen der Datenflussanalyse ähneln (siehe [ISTQB CTAL-TTA]). Ein Beispiel für ein Tool, das statische Taint-Flow-Analysen für PHP-Code verwendet, ist [URL:RIPS].
Statische Analysen sollten bei jeder Software- und Systementwicklung routinemäßig durchgeführt werden.

Dynamische Analyse

Bei Sicherheitstests wird in verschiedenen Kontexten eine dynamische Analyse durchgeführt. Sie wird zur Überprüfung von Code während der Ausführung genutzt, z. B. bei der dynamischen Taint-Analyse und der dynamischen symbolischen Ausführung [Schwartz 10]. Die dynamische Analyse hat das gleiche Ziel wie die statische Taint-Analyse. Sie kann gegen Einschleusungen anfällige Sicherheitslücken in einer Anwendung oder einem System aufdecken.
Eine weitere Form der dynamischen Analyse ist das ‚Vulenrability Scanning' – das Scannen auf Sicherheitslücken. Vulnerability-Scans überprüfen Computer, computergestützte Systeme, Netzwerke und Anwendungen auf bekannte oder potenzielle Sicherheitslücken. Ein Vulnerability-Scanner kann ungepatchte und damit unsichere Programm- oder Betriebssystemversionen, unsichere Konfigurationen wie Datei- oder Zugriffsberechtigungseinstellungen, abgeschaltete Authentifizierungsmechanismen und vieles mehr entdecken. Die eingesetzten Tools reichen von Netzwerkscannern bis zu Penetrationtest-Frameworks. Diese verfügen unter Umständen über große Datenbanken mit ausführbaren Exploits für verschiedene Arten von Zielsystemen, die automatisch ausgeführt werden können, um das Vorhandensein bekannter Sicherheitslücken aufzudecken. Eine dynamische Analyse sollte Bestandteil von Verifizierungs-, Freigabe- und Response-Aktivitäten sein, als auch im Rahmen von Wartungstests durchgeführt werden.

Fuzz testing

Fuzz testing prüft die Eingabevalidierungseigenschaften einer Anwendung. Bei einem Fuzztest werden mit Hilfe eines Werkzeugs fehlerhaft formatierte Eingaben erzeugt und in die zu testende Anwendung oder das System eingespeist, um zu beobachten, wie es darauf reagiert. Wenn die Anwendung ausfällt, wurde ein potenziell sicherheitsrelevanter Fehler gefunden. [Howard 06] nennt drei Kategorien von Fuzz testing:

- Dateiformat-Parser, die Dokument-Dateien, ausführbare Dateien und Bilddateien manipulieren.
- Netzwerkprotokoll-Parser, die Datenpaketinhalte und Paketsequenzen manipulieren
- Anwendungsprogrammierschnittstellen (API)-Parser und sonstige Parser für spezielle APIs oder Zwecke.

Beim „Dumb Fuzzing" werden die Eingabedaten zufällig verändert. Beim „Smart Fuzzing" werden bestimmte relevante Teile einer Datenstruktur geändert. Testentwurfsverfahren wie Äquivalenzpartitionierung und Grenzwertanalyse unterstützen Smart-Fuzzing-Ansätze. Eine umfassende Einführung bietet [BSI Fuzz].

Penetrationtests und ethisches Hacking

Als Penetrationtest wird jede *„Testmethode, die darauf abzielt, die Sicherheitsfunktionen eines Systems zu umgehen"* bezeichnet [NIST-SP-800-160]. Somit kann er als Oberbegriff für alle oben genannten Testarten verstanden werden. Der Begriff wird oft als Synonym für ‚Ethical Hacking' verwendet. Unter ‚ethical' oder White-Hat-Hacking wird allgemein die Durchführung eines autorisierten Angriffs auf ein bestehendes Computersystem oder Netzwerk mit dem Ziel der Prüfung von dessen Sicherheit verstanden [Palmer 01]. Beim White-Hat-Hacking kann jede der oben genannten Testarten sowie ‚Social Engineering'-Techniken eingesetzt werden, um in ein System einzudringen. Der typische Ansatz des Ethical Hacking besteht darin, die Sichtweise eines echten bösartigen Angreifers einzunehmen und zu demonstrieren, was der Angreifer tun könnte, ohne dass der Angriff bemerkt wird.

Dieser Lehrplan verwendet den Begriff ‚Penetrationtest' so: Als konzeptioneller Nachweis, dass ein bestimmter realer Angriff möglich ist.
Sicherheitstests – in ihren verschiedenen Formen – sollten während des gesamten Softwarelebenszyklus zum Einsatz kommen werden. Im ASDL können Penetrationtests in der Freigabe- und der Response-Phase durchgeführt werden. In der Tat führen viele Unternehmen ein Bug Bounty Programm[1] in Abstimmung mit ihrem Incident Response Plan durch.

3.5 Fehlermanagement und -klassifizierung (20 Minuten)

Das Fehlermanagement unterstützt das Software-Sicherheitsmanagement über den gesamten Lebenszyklus eines Produkts oder einer Dienstleistung. Das Fehlermanagement für sicherheitsrelevante Fehler sollte bereits früh in der Anforderungsphase des ASDL eingerichtet werden. Sicherheitsrelevante Fehlerzustände erfordern besondere Zugriffsrechte. Nur autorisiertes und vertrauenswürdiges Personal sollte einsehen können, wie eine existierende Sicherheitslücke ausgenutzt werden kann. Zugehörige Details sollten auch niemals in einen allgemeinen Testbericht aufgenommen werden [Palmer 01]. Ein Klassifizierungsschema für Sicherheitsmängel sollte ebenfalls frühzeitig eingeführt werden. Beispiele für gängige Klassifizierungen sind CIA, STRIDE, DREAD [Howard 02, URL:DREAD] und verschiedene Versionen des Common Vulnerability Scoring System CVSS [NISTIR 7946]. Sie erleichtern das Risikomanagement und erforderliche Neubewertungen der Sicherheit sowie Audits auf Organisationsebene.

Auch das Fehler- und Abweichungsmanagement spielt in der Response-Phase des ASDL eine entscheidende Rolle. Kunden über Sicherheitslücken zu informieren ist eine echte Herausforderung. Aus diesem Grund veröffentlichen Apple, Microsoft, Red Hat und viele weitere Organisationen Sicherheitslücken und deren Beseitigung über die MITRE CVE-Liste. Die MITRE Corporation ist ein gemeinnütziges Unternehmen, das mehrere Forschungs- und Entwicklungszentren [URL:MITRE] in den Vereinigten Staaten betreibt. CVE ist die Abkürzung für „Common Vulnerabilities and Exposures" (Häufige Schwachstellen und Anfälligkeiten). Die CVE-Liste weist jeder veröffentlichten Sicherheitslücke eine eindeutige Kennung zu.
Der Zuweisungs-Prozess beginnt mit der Entdeckung einer potenziellen Sicherheitslücke [URL:CVE-FAQ]. Der nächste Schritt ist die Beantragung einer CVE-Kennung (CVE-ID) bei einer dafür vorgesehenen Vergabestelle (CVE Numbering Authority –CNA). Eine CNA ist eine Organisation[2], die von MITRE autorisiert ist, einer Sicherheitslücke eine ID zuzuweisen und sie in der CVE-Liste zu veröffentlichen. Eine vollständige Liste der CNAs findet man bei [URL:CNA]. Bei bestimmten Problemen muss sich ein Softwarehersteller an eine bestimmte CNA wenden. Die aktuelle Liste ist unter [URL:CVE-List] zu finden.
Das National Institute of Standards and Technology (NIST) unterhält außerdem eine nationale Sicherheitslückendatenbank [URL:NVD], die auch alle Einträge der CVE-Liste enthält.

[1] Ein Programm, das Einzelpersonen Anerkennung und unter Umständen auch Geld für das Melden von Fehlern verspricht.
[2] Einschließlich MITRE

4. Literatur

4.1 ISTQB-Unterlagen

[ISTQB_CTFL] ISTQB Foundation Level Syllabus, Version 2011

[ISTQB CTAL-TTA] Certified Tester Advanced Level Technical Test Analyst, Version 2012

[ISTQB CTAL-SEC] Certified Tester Advanced Level Syllabus Security Tester, Version 2016

4.2 Normen

[CC-PART-1] Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and general model, Version 3.1, Revision 4, September 2012

[CC-PART-2] Common Criteria for Information Technology Security Evaluation Part 2: Security functional components, Version 3.1, Revision 4, September 2012

[CC-PART-3] Common Criteria for Information Technology Security Evaluation, Part 3: Security assurance components, Version 3.1, Revision 4, September 2012

[ISO 25010] Systems and software engineering –
Systems and software product Quality Requirements and Evaluation (SQuaRE) – System and software quality models

[ISO/IEC 27001] Information technology — Security techniques — Information security management systems — Requirements, 2013.

[ISO/IEC 27034-1] ISO/IEC 27034-1: Information technology – Security techniques – Application security – Part 1: Overview and concepts

[ISO 31000:2018] ISO 31000:2018, Risk management – Guidelines

[NIST 2018] Framework for Improving Critical Infrastructure Cybersecurity. Version 1.1, NIST, 2018

[NIST-SP-800-37] NIST Special Publication 800-37: Guide for Applying the Risk Management Framework to Federal Information Systems, Rev. 1, 2010, überarbeitet 2014.

[NIST-SP-800-39] NIST Special Publication 800-39: Managing Information Security Risk, Organization, Mission, and Information System View, März 2011

[NIST-SP-800-160] NIST Special Publication 800-160 Systems Security Engineering, NIST, November 2016

[RFC 1122] Requirements for Internet Hosts – Communication Layers, url: https://tools.ietf.org/html/rfc1122, zuletzt abgerufen am 14. Juli 2017

[RFC 1123] Requirements for Internet Hosts – Application and Support, url: https://tools.ietf.org/html/rfc1123, zuletzt abgerufen am 14. Juli 2017

4.3 Bücher

[DeMarco 78] Tom DeMarco, Structured Analysis and System Specification, Yourdon Inc., New York, 1978

[Eck 14] Claudia Eckert, IT-Sicherheit – Konzepte – Verfahren – Protokolle, Oldenbourg Verlag, 9th edition, 2014

[Howard 02] Michael Howard und David LeBlanc, Writing Secure Code, Microsoft Press, 2002

[Howard 06] Michael Howard and Steve Lipner, The Security Development Lifecycle, Microsoft Press, 2006

[MIT03] Kevin Mitnick, William L. Simon, The Art of Deception: Controlling the Human Element of Security, Wiley Publishing Inc, 2003

[MIT17] Kevin Mitnick, The Art of Invisibility, Little, Brown and Company, New York, Boston, London, 2017

[Pohl 11] Klaus Pohl & Chris Rupp, Requirements Engineering Fundamentals, Rocky Nook Computing, 2nd edition, April 2015

[Ruef 07] Marc Ruef, Die Kunst des Penetration Testing – Handbuch für professionelle Hacker, C&L Computer und Literatur Verlag, Böblingen, 2007

[Shostack 14] Adam Shostack, Threat Modeling, John Wiley & Sons, 2014.

[Weidman 14] Weidman, G., Penetration Testing. A Hands-On Introduction To Hacking, No Starch Press, 2014.

[Whit03] James A. Whittaker, Howard H. Thompson, How to break Software Security, Addison Wesley, 2003

4.4 Andere Quellen (Artikel und Webseiten)

[Bluetooth SIG] Website der Bluetooth Special Interest Group, https://www.bluetooth.com/, zuletzt abgerufen am 25. Juni 2019.

[BSI Fuzz] Fuzzing Primer: A Fuzz Introduction with CC-specific Guidance. Version 1.6. Bundesamt für Sicherheit in der Informationstechnik. August 2020.

[ENISA] ENISA-Glossar, URL: https://www.enisa.europa.eu/topics/threat-risk-management/risk-management/current-risk/risk-management-inventory/glossary#G51, zuletzt abgerufen am 11. Juli 2017.

[Lamport 78] Leslie Lamport, The implementation of reliable distributed multiprocess systems, Computer Networks, Volume 2, Issue 2, Seiten 95–114, Elsevier, Mai 1978.

[NISTIR-SP-800-53] Security and Privacy Controls for Federal Information Systems and Organizations, NIST Special Publication 800-53, Revision 4, 2013 mit Aktualisierungen von 2015. Siehe: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf

[NISTIR 7946] Joshua Franklin and Harold Booth, CVSS Implementation Guidance, NISTIR 7946, NIST, April 2014

[**Palmer 01]** C.C. Palmer, Ethical hacking, IBM Systems Journal, Vol. 40, No. 3, 2001

[Saltzer 75] Saltzer, Jerome H. & Schroeder, Michael D.: The Protection of Information in Computer Systems, Proceedings of the IEEE 63, September 1975

[Schwartz 10] Schwartz et al., All You Ever Wanted to Know about Dynamic Taint Analysis and Forward Symbolic Execution (but Might Have Been Afraid to Ask), SP '10 Proceedings of the 2010 IEEE Symposium on Security and Privacy, pp. 317-331, Mai 2010

[Sindre 05] Sindre and Opdahl, Eliciting Security Requirements with Misuse Cases, Requirements Engineering, 10: 34 - 44, 2005

[URL:CAPEC] Common Attack Pattern Enumeration and Classification — CAPEC™, https://capec.mitre.org/index.html, abgerufen am 14. Juli 2019

[URL:CVSS] Common Vulnerability Scoring System v3.1 https://www.first.org/cvss/ , abgerufen am 4. Dezember 2020

[URL:CVSS-Calculator] The NVD Common Vulnerability Scoring System Calculator https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator, abgerufen am 4. Dezember 2020

[URL:CWE] Common Weakness Enumeration https://cwe.mitre.org, abgerufen am 3. Dezember 2020

[URL:CERT SEI] SEI CERT Coding Standards, https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards, retrieved on July 11, 2019

[URL:CISA alerts] Cybersecurity and Infrastructure Security Agency (CISA), Alerts, https://www.us-cert.gov/ncas/alerts, abgerufen am 25. Juni 2019

[URL:CNA] https://cve.mitre.org/cve/request_id.html, abgerufen am 3. August 2017

[URL:CVE-FAQ] https://cve.mitre.org/about/#how_cve_works, abgerufen am 3. August 2017

[URL:CVE-List] https://cve.mitre.org/cve/, abgerufen am 3. August 2017

[URL:DREAD] https://msdn.microsoft.com/en-us/library/ff648644.aspx,

zuletzt abgerufen am 3. August 2017

[URL:FindBugs] http://findbugs.sourceforge.net/, zuletzt abgerufen am 31. Juli 2017

[URL:MITRE] https://www.mitre.org/, zuletzt abgerufen am 3. August 2017

[URL:MSDL] https://www.microsoft.com/en-us/sdl, zuletzt abgerufen am 24. Juni 2019

[URL:OWASP-Categories] https://www.owasp.org/index.php/Category:Attack,

[URL:PMD] http://pmd.sourceforge.net/pmd-5.2.3/, zuletzt abgerufen am 31. Juli 2017

[URL:RIPS] http://rips-scanner.sourceforge.net/, zuletzt abgerufen am 31. Juli 2017

[URL:router-hack] http://www.heise.de/security/meldung/Mail-hackt-Router-17**,** zuletzt abgerufen am 18. Juli 2014

[URL:SEF] https://www.social-engineer.org/framework/general-discussion/, zuletzt abgerufen am 14. Juli 2017

[URL:STRIDE] https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx, abgerufen am 2. August 2017

[URL:Signing] Introduction to Code Signing, Microsoft Developer Network, https://msdn.microsoft.com/en-us/library/ms537361(v=vs.85).aspx, zuletzt abgerufen am 31. Juli 2017

[URL:US-CERT-DoS] https://www.us-cert.gov/ncas/tips/ST04-015, zuletzt abgerufen am 14. Juli 2017

[Zeller 08] William Zeller, Edward W. Felten: Cross-Site Request Forgeries: Exploitation and Prevention, Revision vom 15.10.2008, Princeton University, 2008, zuletzt abgerufen am 23. Juli 2017

5. Änderungen von Version 1.5 zu 1.5.1:

- CWE als prüfungsrelevanten Begriff in Kapitel 3 hinzugefügt. Quellenangabe in Kapitel 4 hinzugefügt.
- Fehlenden Link für [URL:W3C-DOM] hinzugefügt.
- CAPEC ist jetzt ein prüfungsrelevanter Begriff. Abschnitt 2.4 über CAPEC verbessert. Ein Teil von Abschnitt 2.4 in Abschnitt 2.4.1 verschoben. In Abschnitt 3.2 wurde ein Hinweis auf die mögliche Verwendung bei der Bedrohungsmodellierung hinzugefügt.
- LZ (3.2.2.2) über DREAD durch ein neues LZ 3.2.2.2 über CVSS ersetzt. Entsprechende auf DREAD bezogene Textpassagen ersetzt oder überarbeitet. Quellenangaben zu CVSS und seiner Berechnungsmethode in Kapitel 4 hinzugefügt.
- Deusche Version:
    - BSI-Quelle zu Fuzz Testing ergänzt.
    - Einzelne Aussagen des englischen Lehrplans klarer gefasst.