Hacker Handbook

Chapter 1 – Web Application (In)security

- Keine statischen Webseiten, sondern interaktive Web Anwendungen
- Nutzen von Web Anwendungen
    1. HTTP ist leichtgewichtig, zustandenlos, kann proxied und getunnelt werden
    2. Jeder Benutzer hat schon einen Browser installiert (Änderungen müssen nur 1x auf dem Server durchgeführt werden)
    3. Browser bieten viele Features (Plugins), User ist vertraut mit Web Interfaces, Clientseitige Skripts
    4. Einfache Technologien und Sprachen zur Entwicklung von WA, viel Open Source Code verfügbar
- WA Security: Neue Technologie, neue Schwachstellen, keine Lösung in Sicht, bedeutenste Kampffeld zwischen Hacker und Verteidiger
- SSL schützt nicht vor unsicheren WA
    - Defekte Autentifizierung (LogIn) 67%
    - Defekte Zugriffskontrolle (kein Schutz vor unberechtigten Zugriffen) 78%
    - SQL Injection 36%
    - XSS 91%
    - Informationsleck (Sensitive Informationen werden ausgegeben) 81%
- SSL sichert die Vertraulichkeit und Integrität zwischen Browser und Server (Eavesdropping)
- Standardisierte Security-Lösungen einsetzen und kein Eigenbau, da teurer und weniger effektiv
- Das Kernproblem der Sicherheit: User können beliebigen Input an die WA versenden
- WA muss „untrusted“ Daten akzeptieren und verarbeiten
- Kein entwickeltes Bewußtsein von Sicherheitsproblemen
- V.a. In-House Entwicklung und third-party Komponenten: Jede Anwendung ist einzigartig mit eigenen Fehlern (keine Standardisierung)
- Betrügerische Einfachheit der Entwicklung: Anfänger können schnell funktionierenden Code schreiben, dieser ist i.d.R. Aber nicht sicher
- Schnell entwickelnde Bedrohungen und Untersuchungen der Bedrohungen, folglich kaum möglich alle Bedrohungen zu kennen und entsprechend zu programmieren
- Auch WA unterliegen Einschränkungen der Ressourcen und Zeit: Mehr Wert auf stabile und funktionierende Anwendungen als auf sichere Anwendungen
- WWW und dessen Technologien haben sich verändert: Alte Technologie für neue Anforderungen erweitert (AJAX) → Seiteneffekte und Security-Schwachstellen
- WA erfordern, dass User darauf zugreifen: Netzwerk-level Verteidiung muss Zugriff auf Webserver gewährleisten, Webserver muss sich auf weitere Systeme (DB, Mainframes) des Netzwerks verbinden können! WA muss sicher sein, sonst Tor zum Netzwerk!

Zudem können die Browser (Clients) anderer Besucher einer verletzbaren Anwendungen zum Angriff auf des client-Netzwerk genutzt werden

- Bewußtsein für serverseitige Schwächen reifen, clientseitige Schwächen werden zunehmend zum Kampfgebiet

Chapter 2 – Core Defense Mechanisms

- Zugriff auf Anwendungsdaten und Funktionalitäten
  - Authentifikation
  - Session Management
  - Zugriffskontrolle
- Benutzer Input kontrollieren
  - Weise schlechten Input ab
  - Akzeptiere guten Input
  - Entfernen des schlechten Inputs (Sanitization)
  - Daten sicher behandeln (sicher Programmmethoden verwenden (gegen SQL Inj))
  - An allen Trust/Vertrauensgrenzen (Input von User, SQL an Db, XML Abfrage) wird der Input validiert
  - Vorsicht bei mulitstep Validierung und anders formartierten Input (Canonicaliation)
- Passende Verteidigungs- und Angriffsmaßnahmen gegen Angreifer vornehmen
  - Fehlerbehandlung/ausgabe
  - Geeignete Logs schreiben, die protokollieren welche Schwäche ausgenutzt wurde und was passiert ist
  - Admins alarmieren bei Anomalien der Nutzung, Anfragen mit Angriffsmustern, Anfragen wo versteckter Inhalt geändert wurde
  - Reaktion auf Attacken durch die Anwendung (immer langsamere Antworten der Anwendung, Sitzung beenden) → Mehr Zeit für Admins zur Reaktion auf Attacke
- Anwendungsmanagement durch Monitoring und Konfiguration durch Admins
  - Muss besonders geschützt werden, da sensible Daten angezeigt werden un der Angreifer vollen Zugriff auf Anwendung bekäme

Chapter 4 – Mapping the Application

- Sammeln der Anwendungsinhalte und -funktionalitäten zur näheren Untersuchung des Verhaltens, der Sicherheitsmechanismen und der genutzten Technologie
- In einer typischen WA ist der meiste Inhalt übers manuelle Browsen zugänglich (Sitemap)
- Automatisches Web Spidering (HTML Formulare können geparst und übermittelt werden, dadurch können Multi-Step Funktionen entdeckt werden, auch JS wird geparst)
  - Probleme: Ungewöhnliche JS Navigation, Multistage Funktionen mit enger Input Validation, unendlich oftes Besuchen einer Seite, da URL sich unterscheidet

(Zufallszahlen in URLS) bzw gleiche URL mit unterschiedlichem Inhalt, nicht alle Spiders können mit Authentifizierung umgehen (Requesten der Logout Funktion, WA beendet aufgrund invalider Inputs Session, Per-Page Tokens oft nicht händelbar, wenn Spider Zugriff auf Admin-Funktionen hat kann das sehr gefährlich werden (Löschen von Usern, DB löschen, Server neu starten))

- robots.txt untersuchen
- Durch Benutzer gelenktes Spidern
  - Navigation durch normales Browser (kein Problem bei JS oder Form Navigation)
  - Benutzer kontrolliert alle Daten und kann sicherstellen, dass der Input valide ist
  - Login-Bereiche problemlos zu spidern
  - Gefährliche Funktionen (lösche User) werden gemappt, da Links innerhalb der Anwendung ebenfalls aufgezeichnet werden
- Entdeckung von versteckten Inhalt (nicht verlinkt)
  - Funktionen die zum Testen oder Debuggen implementiert sind, aber nicht entfernt wurden
  - Verschiedene Benutzergruppen haben verschiedene Rechte
  - Backup Kopien von live files (geänderte Dateiendung mit Möglichkeit auf Zugriff des Quellcodes: txt, bak, src, inc oder old), temporäre Dateien (.DS_Store oder file.php~1)
  - Backup Archive (Zip, rar) mit kompletten Schnappschuss aller Dateien (gesamte WA kann gemappt werden)
  - Neue Funktionen, die auf den Server verfügbar sind, aber noch nicht verlinkt sind
  - Alte Versionen von Dateien, die nicht vom Server entfernt wurden (mit vl. Schwachstellen)
  - Konfigurations- und Includedateien mit DB Zugangsdaten
  - Quelldateien aus denen die WA kompiliert wurde
  - Log-Dateien mit Usernamen, Tokens, besuchten URLs, ausgeführten Aktionen
  - Bruteforce zur Erkennung ob typische Dateien vorhanden sind (login.php, logout.php, register.php, admin.php) Dabei gilt, dass schon aus dem Antwortcode vieles erkannt werden kann
    - 302 – Weiterleitung – keine Berechtigung auf Datei zuzugreifen? Weiterleitung auf Fehlerseite?
    - 400 – WA benutzt eigenes Naming Scheme, die durch die Anfrage nicht berücksichtigt
    - 401/403 – Datei exzistiert, aber keine Berechtigung darauf zuzugreifen
    - 500 – Parameter werden nicht akzeptiert
  - Burp Intruder als Tool
  - Auch die Lenth der Antwort kann ein Anhaltspunkt sein
  - Schließen von veröffentlichten Inhalt (AddDocument.jsp und ViewDocument.jsp → EditDocument.jsp; Report2004.pdf → Report2005.pdf, JS Code untersuchen, Kommentare mit Hinweisen)

- Nutzung von öffentlichen Inhalt (Suchmaschinen (site:www.wa.com), Web Archive, Foren/Blogbeiträge der Entwickler eine WA)
    - Ausnutzen des Web Server (Bugs im Webserver, Default Content (nikto Perl Script), Schwachstellen von third-party Anwendungen)
- Zur Zuordnung: WA Pages (URL) vs. Funktionaler Pfad (nach Funktion)
- Versteckte Parameter entdecken ((debug, test, hide, source usw) auf (true, yes, on, 1 usw) setzen)
- Analysiere die Anwendung (Funktionalität, Verhalten, eingesetzte Technologien)
    - Kernfunktionen der WA
    - Periphäres Verhalten der WA (offsite Links, Fehlermeldungen, Admin/Logging Funktionen, Redirects etc.)
    - Sicherheitsmechanismen (Sessionmanagement, Zugriffskontrolle, Authentifizierungsmechanismen, User registrierung, Passwordwechsel, Account Wiederherstellung)
    - Alle Lokations (URL, Query String, POST, Cookies etc.)
    - Clientseitige Technologien (Skripte, Java Applets, ActiveX, Flash, Cookies)
    - Technologien auf dem Server (SSL, Web Server Software, Interaktion mit DB, eMail und andere Back-End Systeme)
- Eingangspunkte für User Input (URL String, Parameter in Query String, Body eines Post-Requests, Cookies, HTTP Header (User-Agent, Referer, Accept, etc.))
- Einige WA nutzen eigene nicht-standard-konforme Query Strings, die erstmal verstanden werden müssen
- Out-of-Band Channels als Eingangspunkte: SMTP, Content per HTTP, XML, JSON
- Identifizieren von serverseitigen Technologien (Banner Grabbing (über HTTP Server-Angabe, Templates, Custom HTTP headers), sonst HTTP Fingerprinting mit Httpprint
    - Dateierweiterungen (installierte Dateierweiterungen lassen sich manchmal über verschiedene Fehlernachrichten entdecken)
    - Typische Verzeichnisse für Technologien (servlet, pls, rails etc,)
    - Session Tokens (PHPSESSID, JSESSIONID, CFID)
- Untersuchung der third-party Komponenten (bekannte Schwachstellen, download und installieren der Komponenten zur Untersuchung, Source Code anschauen)
- Serverseitige Funktionen erkennen (Untersuchung von URLs und Query Strings)
- Nutzen von WA-eigenen Funktionen zum Angriff (Fehlermeldungen, die einen übergebenen obfuscating String in Klartext übertragen, Sanitazing Funktionen)
- Angriffsfläche
    - Clientseitige Validation (vielleicht nicht auf dem Server umgesetzt)
    - DB (SQL Injection)
    - File upload/dowload (Path Traversal Schwachstelle)
    - Anzeige von Daten, die der Benutzer eingegeben hat (XSS)

- Dynamische Redirects (Redirect, Header Injection Angriffe)
- Login (Sammeln der User, schwache PW, Brute Force)
- Multistage Login (Logic Fehler)
- Sessions (Vorhersehbare Tokens, unsicherer Umgang mit Tokens)
- Zugriffskontrollen (horizontale und verticale Previlgien können ausgenutzt werden)
- Funktionen, wo man sich als anderer User ausgeben kann (Previlegien erweitern)
- Klarnachrichten werden kommuniziert (Session Hijacking, Zugriff auf sensitive Daten)
- Off-site Links (Leck des Query Strings im Referer)
- Interface zu externen Systemen ()Session, Zugriffskontrolle)
- Fehlermeldung (Informationsleck)
- eMail Interaktion (eMail/Command Injection)
- Native Code Komponenten (Buffer Overflows)
- Third-party Anwendungen (Bekannte Schwachstellen)
- Bekannte Server-Software (generelle Konfigurationsschwächen, Softwarebugs)

Chapter 5 – Bypassing Client-Side Controls

- Client-seitige Kontrolle der zu übertragenen Daten kann grundsätzlich immer umgangen werden
- Übertragung von Daten durch den Client
  - Der Inhalt versteckter Formular Felder kann durch den User geändert werden (Content Length muss auch geändert werden!)
  - HTTP Cookies sind auch änderbar!
  - URL Paramter können auch modifiziert werden (auch von eingebetteten ImageURLs mit Parametern, Frames, Formular, das an ein URL mit Parametern geschickt wird, PopUp ohne Locationbar)
  - Referer Header wird ebenfalls durch den Benutzer kontrolliert und kann auf jeden beliebigen Wert geändert werden
  - Unklare Daten (durch Verschlüsselung oder einfach nur Verwirrung: Vielleicht bietet die weitere Verarbeitung der Daten Schwachstellen; vielleicht dazugehörender Klartext bekannt, andere WA-Funktion, die entschlüsselung ermöglicht?, Replay des Wertes in anderem Kontext (Verschl. Preis von 1 EUR bei 1200 EUR einsetzen...))
  - ASP.NET ViewState (ist Base64-encoded String (Startposition muss gefunden werden) und muss mit Hex-Editor geöffnet werden (Lenth beachten), Allerdings gibt es eine Option (EnableViewStateMac (Auswirkung per Page)), die bewirkt, dass im ViewState eine keyed hash genutzt wird, um diesen Wert korrekt zu übertragen muss der Schlüssel bekannt sein (Änderungen der Daten können somit nicht übermittelt werden, die Daten können aber weiterhin betrachtet werden!))
- User-Daten in Formularen abfangen (kann ebenfalls umgangen werden)
  - Length Limit

- Script-basiertes Validierung (zur Usability ok, Server-seitig unablässig!)
- Disabled Elemente

- User-Daten in Thick-Client Komponenten abfangen
  - Technologien: Java applets, ActiveX, Shockwave Flsahs
  - Diese Technologien können Dateninput verarbeiten und Daten an den Server senden
  - Validierung dieser Komponenten kann umgangen werden! (Wenn Übermittlung der Daten an den Server transparent sind, können die Daten abgehört und verändert werden)
  - Wenn übertragene Daten verschlüsselt oder obfuscted sind, muss man die Thick-Client-Komponente analysieren:
    - Java Applets (übertragene obfuscated Parameter analysieren; Dekompilieren des Java Bytecodes (z.B. mit jad) und so das „Durcheinander“ der Daten verstehen; Achtung: Auch der Bytecode kann durcheinander gebracht werden – dann gibt es drei Möglichkeiten (1. Man muss nur bestimmte Methoden/Teile des Codes verstehen, 2. Analysieren des Codes und Methoden und Variablen Namen geben, Kommentare verwenden, 3. Erneut einen Obfuscator verwenden, der entsprechend eingestellt redudanten Code löscht und global verschiedene Namen vergibt))
    - ActiveX Kontrollen (Win32 executables mit vollen Previlegien (auch aufs OS): somit kann das User-OS durch ActiveX Schwachstellen kompromettiert werden; da in C oder C++ geschrieben kann der Quelltext nicht dekompiliert werden, da ActiveX aber auf dem PC des Users läuft kann dieser den ActiveX-Prozess untersuchen und kontrollieren; OllyDbg kann kompilierte Software zur Laufzeit in vielfältiger Weise angreifen (Reverse Engineering!): Identifiziere Methoden mit Breakpoints, Untersuche den Call Stack welche Daten an die Methoden übergeben wurden;; Mit ComRaider können alle Kontrolmethoden und Signaturen (auch solche die nicht benutzt werden) angezeigt werden; Mit dem Filemon und Regmon Tool kann man die Interaktion von ActiveX mit dem Filesystem und der Registry beobachten (zum Umgehen von Kontrollen kann man also die benötigten Dateien oder Registryeinträge einfach erstellen);; Komponenten in C# können dekompiliert werden und man erhält so Zugriff auf den Quelltext)
    - Shockwave Flash (swf Dateien können dekompiliert werden; ein effektiverer Ansatz ist das Ändern des Bytecodes und Disassemblen (Flasm))

- Client-seitige Daten sicher behandeln
  - Clientseitige Daten vermeiden (z.B. kann der Preis auf dem Server in DB hinterlegt sein) Falls es dennoch sein muss:
    - Daten verschlüsseln und gegen Replay-Attacken schützen (ProduktId + Preis gemeinsam verschlüsseln)
    - Verhindern, dass User Klartext und chiffrierten Text kennen (Salt hinzufügen)
    - Keine sensitiven, cutomized Daten im ViewState speicher UND ViewState Mac aktivieren
  - Daten die auf dem Client validiert wurden, wurden nicht validiert und müssen Serverseitig validiert werden
  - Wenn Daten clientseitig überprüft werden und serverseitig festgestellt wurde, dass die clientseitige Überprüfung umgangen wurde, sollten Gegenmaßnahmen eingeleitet werden (Alarmieren, Loggen der Aktivitäten)

Chapter 6 – Attacking Authentication

- Authentifizierung ist am einfachsten umzusetzen und ist in der Frontline der Verteidigung gegen Angreifer
- Dennoch in WA sehr oft schwächstes Glied
- Technologien:
  - HTML formularbasierte Auth (90%)
  - Muli-Factor (PW und Tokens)
  - Client SSL Zertifikat
  - HTTP basic und disgest auth
  - Mittels NTLM oder Kerberos
  - Auth Services
- Designfehler im Mechanismus
  - Schlechte Passwörter (kurz oder leer, allgemeine Namen aus Wörterbüchern, gleich wie Benutzername, Standardwert)
  - Logins, die Brute Force ermöglichen (automatisierte Anfragen mit häufig verwendeten Usernamen/PW (möglicherweise werden failedLogins clientseitig also per Cookie, Query String oder ähnlichem gezählt und kann umgangen werden))
  - Aussagekräftige Fehlermeldungen (User nicht gefunden, Passwort falsch → gefundene Username können speziell attackiert werden (Passwort erraten, Sessions stehlen, Social Engineering), schlecht auch, wenn Usernamen vorhersehbar sind (user123), um eine Liste von Usern zu bekommen, wenn die Fehlermeldungen gleich sind, kann man die Antwortzeit des Servers nehmen (genaue Zeitmessung natürlich mit Tool))
  - Ungeschützte Übertragung von User/PW (Abhören; auch möglich wenn HTTPS (wenn über query string, oder redirect mit query string, Cookies (muss natürlich kompromettiert werden)), HTTP auf Login-Seite und nach Login HTTPS ist unsicher)
  - Ändern des Passworts (Wichtig, wenn User gezwungen werden das Passwort zu ändern, bedeutet das mehr Sicherheit, da Zeitfenster verkürzt wird;User die mutmaßen, dass PW kompromettiert wurde, müssen neues PW selbst setzen können
    Schwachstellen: Aussagekräftige Fehlermeldung zum Auffinden von Usernamen, Brute-Force Möglichkeit, nur überprüft ob neues Passwort und neues Passwort2 übereinstimmen, aber altes PW wird nicht überprüft)
  - Passwort vergessen (Schwachstelle: Zweite Aufgabe (Geburtsname der Mutter), die für Angreifer einfacher zu erraten ist, da öffentlich bekannt, Frage der Aufgabe kann manchmal durch User festgelegt werden → einfache Aufgabe, Angreifer kann für bekannte User alle Passwort Wiederherstellungsaufgaben ausgeben lassen und einfachste beantworten;; auch hier ist Bruteforce ein Problem,
    Ausgabe des vergessenen Passworts nach erfolgreicher Challenge schlecht, da Angreifer problemlos Account übernehmen kann, ebenfalls ist das Einloggen nach erfolgreicher Recovery nicht ratsam, auch wenn während der PW Recovery die Mail-Adresse zum Zusenden des PW angegeben (auch Hidden oder im Cookie) werden kannst, wird dadurch keine zusätzliche Sicherheit erreicht, Ändern des Passworts direkt nach der Aufgabe sollte auch immer eine Mail an den User nach sich ziehen)
  - Remember Me-Funktion (Mit Cookie, in dem der Username steht ermöglicht es dem

Angreifer ohne LogIn Zugriff als User zu erlangen, statt Username im Cookie auch Tooken möglich (wenn Token verhersehbar → schlecht), auch wenn Inhalt des Cookies verschlüsselt, kann der Inhalt des Cookies mit XSS von anderen Usern ausgelesen werden)

- User „Personifikation“ (als versteckte Funktion, die außerhalb der Zugriffskontrolle liegt; bei Benutzern dieser Funktion werden Eingaben nicht mehr kontrolliert; Personifikation auch als Admin, ein Backdoor-PW (fällt bei Bruteforce auf, da alle User sich damit einloggen))
- Inkomplette Validierung von User/PW (kürzen von PW, case-insensitive, ungewöhnliche Zeichen entfernt)
- Uneindeutige Usernamen (Designfehler: Es wird bekannt welches PW anderer User haben, 2 gleiche User/PW loggen sich evtl. als falsche User ein, ermöglicht Bruteforce durch registrieren gleicher User mit versch. PW, wenn eine WA eindeutige Usernamen verlangt, kann so per Bruteforce ermittelt werden welche typischen Namen vergeben sind)
- Vorhersagbare Usernamen
- Vorhersagbare Anfangspasswörter (werden evtl. nicht geändert, sind vorhersagbar)
- Unsichere Verteilung der User/PW-Kombination (kein Zeitlimit wie lange Kombination gültig ist, keine Anforderung wann Kombination geändert werden muss, bei Account-Activation URLs ist es bei Kenntnis mehrerer URLs evtl mögl. vorherzusagen)

- Implementationsfehler in der Authentifizierung
  - Fail-Open Mechanismen (Exception führt zu Login)
  - Fehler in multistage Login Mechanismen (1. user/pw, 2. PIN, 3. eingabe von Zeichen eines physischen Tokens – zusätzliche Sicherheit, wenn richtig implementiert
    Fehlerhafte Annahmen: Ein User, der sich auf Stufe 3 authentifiziert hat, hat sich auf Stufe 1 und 2 schon eingeloggt
    Anwendung vertraut Daten auf Stufe 2, weil sie auf Stufe 1 validiert wurden
    keine Überprüfung, ob der gleiche User sich auf unterschiedlichen Stufen identifiziert hat)
    Wenn auf Stufe 2 versch. Frage und Antwort gestellt wird: Unsicher wenn User Frage und Antwort übermitteln kann, Unsicher wenn nicht gespeichert wird welche Frage der User schon beantwortet hat und immer wieder gleiche Frage gestellt wird, dessen Antwort der Angreifer kennt
  - Unsicheres Speichern des PW (verschlüsselt in DB speichern, da sonst SQL Inj oder Zugriffskontroll-Fehler zum Bekanntwerden aller Credentials wird)
- Sicheres Authentifizieren (muss natürlich im Verhältnis zur userfreundlichkeit stehen)
  - Starke Credentials (Passwörter mit Sonderzeichen, Groß und klein, min. Länge etc, Benutzernamen sollen eindeutig sein, Systemgenerierte User und PW mit ausreichender Entropie, auch lange PW mit Sonderzeichen abspeichern und vergleichen (nicht kürzen))
  - Credentials geeignet händeln (Erzeugen, speichern, übertragen der Credentials darf nicht von unauthorisierte beeinflusst werden, SSL nutzen (auch LogIn-Bereich!), nur in POST-Request übertragen, crypted Speichern auf dem Server, „Remember Me“ sollte nur nicht-geheime Daten merken, wenn in weniger sicherheitsrelevanten WA auch PW gemerkt werden sollen, müssen die verschlüsselt abgespeichert werden (dann sollten XSS-Attacken auf diese User nicht möglich sein), Ändern des Passwortes sollte implementiert werden und Regelmäßiges PW ändern sollte vorgeschrieben werden,

wenn Credential out-of-band übermittelt, so sicher wie möglich , zeitbegrenzt und ändern beim ersten Login, Dropdown-Menüs einsetzen, da Keylogger damit nichts anfangen können)

- Verhindere Informationslecks (möglichst wenig Info in fehlermeldung, eine Code-Komponente für alle Antworten auf Fehllogins, auch nach Ausloggen wegen Bruteforce keine verbose Fehlermeldung, sinnvoll Mail als Loginname (Bruteforce verhindern, Mail schicken mit Code, ob Mail auch erhalten wurde))
- Verhindere Bruteforce (auf login, Passwort ändern, vergessenes Passwort, Passwort Recovery, unvorhersagbare Usernamen!, deaktivieren von Account, wenn zu viele Fehlerhafte Logins (auf Informationslecks achten!), nicht trotzdem überprüfen, ob Credentials richtig sind, einfach pro Account Maßnahmen verhindern keine Bruteforce Attacken, CAPTCHAs nutzen)
- Password ändern (nur wenn man eingeloggt ist, kein Username übergeben, altes Password eingeben, bei Fehlermeldung auf Infoleck achten, per Mail darauf hinweisen, dass PW geändert wurde)
- Recovery (hohe Sicherheit durch out-of-band: Anrufe, keine PW „Hinweise“, eMail mit einzigartigen, zeitbeschränkten, unverhersagbaren, einzigartiger URL, danach erst Änderung des Passworts und Mail mit PW an User)
- Log Monitor und Notify dieser Aktivitäten!

Chapter 7 – Attacking Session Management

- Session notwendig, da HTTP zustandslos
- Jeder User erhält eindeutigen Token (meist über Cookies)
- Zwei Kategorien von Schwächen: 1. Schwächen beim Generieren von Tokens 2. Schwächen beim Händeln während des Lebens von Tokens
- Alternativen zu Sessions: HTTP authentication (auf jeder Seite werden die Credentials übermittelt), Sessionless state mechanismen (ähnlich wie ViewState in ASP (binäre Blobs zur Speicherung))
- Schwächen in der Session Generierung (SessionIds können problemlos vom Angreifer gesammelt werden)
  - Aussagekräftige Tokens (auch keine kodierte (hexdezimal, XOR, Base64) zusammengesetzte Tokens (Username, AccountID, Vor-Nachname, mail, Gruppe oder Rolle innerhalb der WA, Datum, Zähler, IP-Adresse); wenn Zusammensetzung dem Angreifer klar ist und er eine Userliste hat, kann Angreifer nachvollziehen, welche User eingeloggt ist und die Session klauen)
  - Vorhersagbare Tokens (selbst wenn man nur 1 von 1000 Tokens richtig vorhersagt; verhersagbar aus 3 Gründen: 1. verschleierte Sequenzen (versch. Kodierungen, Subtraktion/Addition zum nächsten Token), 2. Zeitabhängigkeit (Zeit und Zufallsnummer (bei großen WA sehr wahrscheinlich, dass man mit Bruteforce aktive Tokens findet)), 3. Schwache Genierierung von Zufallszahlen (wenn man den Algorithmus kennt und eine Zahl, kann man die nächste Zahl berechnen); Full Blown Test für Zufülligkeit)
- Schwächen beim Händeln von Tokens
  - (SSL ist kein genereller Schutz vor Zugriff auf Tokens!)

    - Offenlegung von Tokens im Netzwerk (bei HTTP können Tokens abgehört werden, HTTPS wird nicht konsequent bei allen Seiten innerhalb einer Domain eingesetzt, Session Token wird während HTTP gesetzt und beim Login bzw HTTPS nicht geändert, Tokens müssen geschlossen werden, Auf HTTPS-Seiten werden per HTTP Bilder etc. geladen)
    - Offenlegungs von Tokens im Log (Tokens können in Adminbereichen, die schlecht geschützt sind, angezeigt werden; in Logs wenn Query String Parameter genutzt wird (Referer), in Browser Log, Weblogs, ISP Proxy, Logs von Off-Site Links Server;; Webmail-Applications können mit einfachen Mails mit eingebetteten Bildern vom Server des Angreifers Tokens erfahren)
    - Schwachstelle beim Mapping von Token zu Session (nicht mehrere gleichzeitige Tokens pro User, keine statischen (immer gleichen) Tokens; keine bedeutungsvollen, kodierten Token (user + token), wo man den User ändern kann, um sich als anderer User einzuloggen)
    - Schwachstellen bei der Session Beendigung (Lebensspanne der Session soll möglichst kurz sein, um Angriffe darauf nahezu unmöglich zu machen & Sitzung durch User beim Logout serverseitig! beenden (nicht nur Cookie clientseitig löschen!))
    - Clientseitig Aufdeckung durch Token Hijacking (per XSS s. Chapter 12, Session Fixation. XSRF)
    - Zu liberale Cookie Scope/Bereich (Tritt bei Subdomains (und falsch gesetzten Pfaden) auf, wo der Cookie auf die gesamte Domain gesetzt wird, es können dann Schwachstellen in den WA aller Subdomains genutzt werden, um die Session Tokens zu bekommen)
- Sicheres Session-Managament
    - Generiere starke Tokens (Große Menge an möglichen Werten, starke Quelle für pseudo-Randomness (etwa eine Konkatenation mit IP, User-Agent, Millisekunde des Requests und Server-eigener Salt (hashen mit SHA-256)))
    - Schützen von Tokens während des Lifecycles (nur über HTTPS übermitteln, wenn Cookies verwendet werden sollten er als „secure“ geflaggt verwenden, wenn HTTP-Bereich muss der Cookie Scope entsprechend gesetzt sein
      Niemals über Query String (wenn keine Cookies verwendet werden können über POST-Requests)
      Richtig implementierte Logout-Funktion
      Session Ablauf nach adequater Zeit (10 Min)
      Keine zeitgleichen Logins pro User (bei jedem Login alten Token verwerfen und neuen ausstellen)
      Wenn administrativer oder diagnostischer Bereich mit Zugriff auf aktive Tokens (müssen die angezeigt werden?), müssen diese Bereiche gesichert werden)
    - Domän und Pfad-Bereich von Cookies müssen möglichst restriktiv gesetzt werden
    - Tokens, die nicht durch die Anwendung vergeben wurden, dürfen nicht akzeptiert werden
    - XSRF und andere Session-Attacken können erschwert werden, wenn kritische Aktionen eine erneute Authentifikation erfordern
      XSRF: Oder ein verstecktes Feld mit geheimen Token (anderer?) im Formular mitübertragen wird, der dem Angreifer nicht bekannt
    - Um Session Fixation zu verhindern, muss eine frische Session nach dem Login erstellt

werden

- Pro-Page Token (Page Token wird erstellt, wenn User eine neue Seite anfragt; übermittelt per Cookie oder versteckten Formularfeld; wenn zurückgesendeter Page Token ungleich releaster ist, wird Session beendet; Session Fixation und Hijacking wird aufgedeckt (gleichzeitiges Surfen führt dazu, dass page token unterschiedlich wird))
- Log, Monitor und Alarmieren (monitore invalide tokens, loggen von Bruteforce-Attacken, alarmiere User über Anomalien (gleichzeitige Logins, Hijacking))
- Sessionbeendigung bei modifizierten verstecken Formularfeldern, Input mit SQL Injection oder XSS Attacken

Chapter 8 – Attacking Access Controls

- Zwei Kategorien: veritkal (versch. User haben versch Rechte (Admins + normale User)) und horizonalte Zugriffskontrolle (User erhält auf eine bestimmte Untermenge Zugriff (nicht alle Mail, sondern nur Mails, die für einen bestimmt sind) die untereinander verbunden sind)
- Ungeschützte Funktionen (URL wird geheim gehalten, nur verlinkt wenn als Admin eingelogt (teilweise über JS(!)): Diese URLs können aber bekannt werden und sind nicht geheim! (Browser History, werden auf dem Bildschirm dargestellt, erscheinen in Logs des Webservers, vlt werden Mails mit den URLs verfasst etc))
- ID-basierte Funktionen (viewDoc.jsp?docid=1217892163) (auch wenn Ids nicht vorhersagbar sind und somit die URL durch einen Angreifer erraten werden muss, muss es eine Zugriffskontrolle geben, um sicherzustellen, dass nur autorisierte User Zugriff bekommen)
- Multistage Funktionen (bei späteren Stufen wird ggf. nicht mehr überprüft, ob der User das Recht hat die Funktion auszuführen- also alle Stufen müssen Rechte überprüfen! Ebenfalls müssen erneut übertragen Daten vom Client validiert werden)
- Statische Dateien (können nicht überprüfen, ob User Zugriff hat oder nicht, desweiteren kann die Offenbarung einer Datei auf weitere Dateien schließen lassen; stattdessen entweder PHP-Datei der die Datei übergeben wird und AC durchführt ODER per HTTP Auth Zugriff auf Dateien)
- Unsichere Zugriffskontrollen (Query String, Referer, clientseitige Daten zur Entscheidung, ob jemand Zugriff bekommt (admin=true))
- Alle Berechtigungen durch die Session ableiten
- Eine zentrale Komponente für Zugriffskontrolle
- Loggen aller Zugriffe auf sensitive Daten oder Aktionen
- Programmatic Access Control (Matrix von individuellen Previligion wird in einer Tabelle gespeichert)
- Discretionary Access control (Admins können ihre Prevelegien delegieren): closed – wenn Zugriff so lange abgelehnt wird bis er zugelassen wird; open – wenn Zugriff so lange zugelassen wird, bis er verboten wird
- Role-based access control (User können versch. Rollen mit untersch. Prevelegien angehören)
- declarative control (versch Accounts, Account mit wenigsten Previlegien wird genutzt für DB Zugriff etc.,)

Chapter 9 – Injecting Code

- Code in interpretierte Sprachen einspeisen (Sprachen, die zur Laufzeit interpretiert werden (PHP, bash))
- Einspeisen in SQL
  - Verschiedene DB (Oracle, MS-SQL, MySQL)
  - Ausnutzen einfacher Schwachstellen (wenn Eingabe von ' zu Fehler führt, ist Schwachstelle vorhanden; Eingabe von ' OR 1=1-- gibt alle Einträge aus)
    - Login umgehen durch Eingabe von *admin'--* führt dazu, dass das PW überhaupt nicht überprüft wird, alternativ kann auch *' or 1=1--* eingegeben werden, das oft dazu führt, dass man sich als 1. User in der DB (idR admin) einloggt.
  - Finden von SQL Injection Bugs
    - String: Ausbrechen aus den Anführungsstrichen (Einfaches Anführungsstrichen, zwei Anführungsstriche hintereinander als maskiertes Anführungsstrich, Setze Injection mit Konkatenation (Oracle ||, MS-SQL +, MySQL blank) zusammen)
    - Numerische Daten: Versuche mathematische Ausdrücke anstelle des eigentlichen Werte zu übertragen (2+1), wenn Eingabe wie 3 behandelt wird, kann es sich um eine Schwachstelle handeln, Versuche Eingaben wie 68-ASCII('A') (= 65)
    - Achtung: Daten müssen URL-kodiert werden
  - Einspeisen in versch. Statement-Typen
    - Bei SELECT meistens in WHERE-Klausel
    - Bei INSERT hinter VALUES (Damit Statement korrekt ist, muss die richtige Anzahl an Feldern im INSERT erraten werden; für zusätzliche Felder 1 verwenden, da int in String konvertiert wird und 1 bei jedem Typ akzeptiert wird)
    - UPDATE Statements (Mit Eingabe von admin' or 1=1-- wird eventuell alle Datensätze auf einen Wert geändert)
  - UNION (nützlich, wenn man eine Schwachstelle im SELECT-Statement hat und andere Daten mit auslesen will z.B. mit der Eingabe von *test' UNION SELECT username, password, uid FROM users--*)
    - Um mit Union zu arbeiten muss 1. die Anzahl des ersten SELECT mit zweiten SELECT übereinstimmen und 2. man muss die Tabelle und Felder für zweite Query kennen (mit ausgegebenen DB-Errors generell problemlos möglich)
    - Wenn Errors nicht ausgegeben werden:
      Mit SELECT NULL kann entspr. Feld ausgefüllt werden (mit ORDER BY 1 oder 2 .. kann der Index so lange erhöht werden bis ein Fehler auftritt)
      Wenn zusätzliche Zeile durch die WA ausgegeben wird, kann man mit der Anzeige dieser zusätzlichen Zeile arbeiten
      Wenn man ein String-Feld identifiziert hat, können darin weitere Daten von der DB angezeigt werden (NULL systematisch durch 'a' ersetzen)
  - Um DBMS zu erfahren, eignet sich die Konkatenation von Strings (Oracle ||, MS-SQL +, MySQL blank), danach kann man mit den entspr. Systemabhängigen Variablen weitere Infos erhalten (Tabelle, Feldernamen etc.)
  - Nutzung von ODBC Fehlermeldungen (erlaubt Aufzählung von Tabellen, Feldnamen, Warte (PW von User Admin))

    - Filterabwehr bei SQL Injection umgehen
        - Vermeiden von verbotenen Zeichen (einfaches Anführungszeichen ist in numerischen Daten nicht notwendig, Query kann so formuliert werden, dass Kommentarzeichen nicht gebraucht wird)
        - Wenn Keywörter verboten sind versuche andere Kodierung, groß/kleinschreibung oder „rekursive“ Formulierung SELSELECTECT oder Kommentare SEL/*dfu/*ECT, Konkatenation mit entspr. Befehlen, CHR(), Konkatenation in exec, hex-kodierte Befehle an exec übergeben
        - Wenn es eine Begrenzung der Länge (20) gibt und ' maskiert wird, übertrage 19 x a und '. Dadurch kann man aus den Quotes ausbrechen und einen zweiten Input nutzen, um seine SQL Anweisungen einzufügen Bsp:

          WHERE username = 'aaaaaaaaaaaaaa'' AND passwort = ' or 1=1--'
          '' ist falsch maskiertes Anführungszeichen, username ist also *aaaaaaaaaaaaa'' AND passwort =*
          *Testen der Schwachstelle durch ü*bertragen von folgenden langen Strings. Wenn eine der String Fehler produziert, gibt es die Schwachstelle (S 270):
          ''
          a''

- Second Order SQL-Injection (' wurde beim Insert maskiert, aber bei nachfolgenden Querys nicht mehr, so dass erst dann die die SQL Injection auftritt)
- Fortgeschrittenes Ausnutzen von Schwachstellen (Anzeige von Ergebnissen ist nicht mehr geradeaus)
    - Neben dem Auslesen von Daten kann auch die DB heruntergefahren oder Daten gelöscht werden
    - Daten als Nummern erhalten (wenn kein String vorhanden ist: arbeiten mit ASCII und SUBSTRING, um Zeichen für Zeichen auszulesen)
    - Benutzen von Out-of-Band Channels (MS-SQL: OpenRowSet, Oracle: URL_HTTP, UTL_INADDR (DNS Lookup), MySQL: SELECT INTO OUTFILE, OS-Befehle über DB aufrufen (mail, telnet))
    - Inferenz-basiert (SQL-Injection wird ausgenutzt, um über WA-Logic Zeichen für Zeichen das Ergebnis auszulesen (Ausgabe funktioniert ja nicht), z.B. bei Login: Wenn Login klappt, landet man im Mitgliedsbereich, wenn nicht bleibt man auf Login-Seite; Mit Absinthe kann man diese Abfragen automatisisieren, mit ASCII und Substr kann man für jedes Zeichen alle mögliche Zeichen durchgehen und dann das nächste Zeichen herausbekommen
      Alternativ kann auch HTTP 500 Code genutzt werden: SELECT 1/0 FROM … WHERE (SELECT-Anweisung, die stimmt (dann fehler) oder nicht))
      Alternativ misst man die Zeit und nutzt Verzögerungen (waitfor delay, wenn's stimmt)
- Ausweiten der DB Attacke
    - wenn weitere WA auf der DB laufen, kann man auf diese Daten vielleicht auch zugreifen
    - vlt. Erhält man auch Zugriff aufs OS
    - Zugriff aufs Netzwerk (evtl. auch im geschützt Bereich, da DB dort liegt)
    - Verbindungsaufbau zum eigenen PC, um sensible Daten zu übertragen
    - Existierende Funktionen der DB durch user-defined Funktionen „erweitern“

    - MS-SQL: xp_cmdshell Zugriff auf Kommandozeile, xp_regread, xp_regwrite, OpenRowSet für Portscan im gesicherten Netzwerk
    - Oracle: Package, um sich selbst DBA Rechte zu geben
    - MySQL: Wenn man FILE_PRIV hat, kann man mit SELECT LOAD_FILE(datei) die Datei auslesen
- Verhindern von SQL Injections (Prepared Statements für jedes Query; Defense in Depth (möglichst wenig Privilegien des DB-Users, (Standard)Funktionen deaktivieren, Sicherheitspatches installieren))
- Betriebssystem-Befehle einspeisen (per exec oder wscript.shell haben WA Zugriff auf OS-Befehle, um mit diesem zu interagieren, per Formularfelder oder Query String Schwachstelle ausnutzbar)
    - 2 versch. Typen von Metazeichen, um Befehle voneinander zu trennen:
        - ; | & um daraus ein Batch von Befehlen zu machen
        - ` zur Kapselung (Befehl in ` wird als erstes ausgeführt und die Ergebnisse dann einem anderen Befehl aus Input übergeben)
    - Um zu verifizieren, dass es eine Lücke gibt, übergebe ein Delay-Befehl und messe die Zeit, die zur Antwort gebraucht wird
    - Von eine direkte Ausgabe des Ergebnisses eines Befehls nicht möglich ist, benutzte tftp, mail oder schreibe mit *dir* > *c:\test.txt* das Ergebnis in eine Datei, die man sich ausgeben lassen kann
      Viele Befehle haben die Möglichkeit den Output in einer Datei zu schreiben
    - Verhindern des Einspeisens von Betriebssystem-Befehlen
        - Wenn möglich keine Befehle mit Userdaten ans OS schicken
        - Wenn möglich mit einer White List an erlaubten Zeichen bzw. Befehlen arbeiten, wenn Eingabe andere Zeichen beinhaltet → Befehl nicht ausführen
        - Nutzen von Command APIs (Runtime.exe Process.start unterstützen keine Shell Metazeichen) und nicht direkt den Shell-Interpreter
- Einspeisen in Web Script Srachen (PHP, Perl). Zwei verschiedene Schwachstellen
    - Dynamisches Ausführen von Code, das Userdaten einbezieht (PHP: eval, ASP: Execute;; Als Attack String: ;echo%2011111)
    - Dynamisches Einbinden von Code-Dateien, die durch Userdaten spezifiziert werden (PHP: include, require_once können auch entfernten Code einbinden (furl_open); oder Code lokal einbinden, auf den man normalerweise keinen Zugriff hat (Adminbereich, geschützter Bereich mit statischen Dateien))
    - Übertrage an jeden Parameter eine URL zu einer Datei, die man kontrolliert; übermittele eine URL zu eine nicht-existierenden IP falls Timeout gibt es die Schwäche
      Für Testen, ob lokale inclusion-Schwäche verhanden ist: einbinden eines bekannten Skripts, wenn sich Verhalten ändert → Schwachstelle, übermittle bekannte statische Datei, wenn Quelltext dieser Datei in der Antwort vorhanden ->schwachstelle
    - Verhindern, indem man keine Userdaten verwendet; ansonsten White List verwenden
- Einspeisen in SOAP (Antwort von SOAP ist XML, die interpretiert wird, wenn Angreifer <>/ einfügen kann, kann er aus die Struktur ausbrechen und das Verhalten der WA

beeinflussen)

  - Entdecken von SOAP Injection Schwachstellen: Übermittle ein schließendes Tag </foo> (evtl. wird es bereinigt oder nicht eingefügt; wenn ein Fehler auftaucht, übermittele <foo></foo>: wenn dann der Fehler verschwindet, ist die Schwachstelle vorhanden;)
  - Durch kommentieren kann das XML trotz einfügen von Tags, valide bleiben
  - Verhindern: HTML-kodieren der User-daten
- Einspeisen in Xpath (zur Abfrage von XML-Dateien)
  - ähnlich wie SQL Injection
  - es kann auch hier mit substring Zeichen für Zeichen der Wert eines Feldes ausgelesen werden und über die WA Logic das Ergebnis detektiert werden (Login erfolgreich(/los))
  - da Xpath mit relativen Pfadangaben arbeitet kann der komplette Aufbau der XML-Datei (Name der Felder, Feldwerte) über vorherige Attacke ausgelesen werden. Hilfreiche XPath-Funktionen: count und position (Navigieren durch die Kinderknoten) und strin-length und substring (Navigieren durch den Wert,Name)
  - Verhindern: Keine Userdaten nutzen, ansonsten blockieren von ()='[]:,*/ und Leerzeichen
- Einspeisen in SMTP (zum Spam-Versand Mail-Header erweitern oder SMTP command Injection (über spw. Querystring oder Formularfelder))
  - Verhindern: Übergebene Mail-Adressen mit Regexp überprüfen, Message sollte keine Newsline Zeichen enthalten und eine max. Länge aufweisen,
- LDAP Injection
  - Verhindern mit einer Whitelist an erlaubten Zeichem

Chapter 10 - Exploiting Path Traversal

- Path Traversal bedeutet, dass man auf Dateien und Verzeichnisse zugreifen kann, die der Entwickler nicht beabsichtigt hat
- Im schlimmsten Fall kann der Angreifer dadurch die WA und das Betriebssystem kompromettieren
- Allgemeine Schwachstellen
  - mit ..\ oder ../ kann auf das Über-Verzeichnis zugegriffen werden
- Seiten, wo man Dateien hoch oder runterladen kann, können diese Schwachstelle enthalten
- Auffällige Parameterwerte im Query-String, wo Dateien oder Verzeichnisse angegeben werden
- Übergeben (../etc/passwd
  oder ../boot.ini)
- Es gibt auch Path Traversal Schwachstellen, wo man Dateien beschreiben kann
- Schranken für Path Traversal umgehen: (/ und \ versuchen, kodieren von dot %2e slash %2f und backslash %5c oder 16-but unicode oder double URL-encoding oder UTF-8, wenn sie Daten bereinigt werden, ist es vielleicht möglich die Userdaten so zu übergeben, dass sie nach der Bereinigung schadhaft sind; überpüfung der Dateiendung umgehen durch boot.ini %00.jpg)

- Ausnutzen um Zugangsdaten zu OS und DB zu erhalten, Conf-Dateien, DB und XML-Dateien, Quelltext von WA-Dateien, Logdateien
- Verhindern von Path Traversal
    - statische URLs statt PHP-Dateien mit Übergabe der Datei
    - Statt übergabe des Dateinamens, eine Indexnummer für diese Datei übergeben
    - APIs benutzen, die schon einen Schutz vor Path Traversal nutzen
    - Nach Dekodieren und Canocalization der Userdaten: Wenn auffällige Zeichen vorhanden sind, abbrechen (nicht bereinigen)
    - eine hardkodierte Liste mit Dateiendungen, die erlaubt sind
    - eine chrooted Umgebung nutzen, in der sich nur die Dateien befinden, auf die man zugreifen können soll
    - Logen und Alarmieren von auffälligen Inputs

Chapter 11 – Attacking Application Logic

- Sind schwer zu charakterisieren und können mit Scannern nicht entdeckt werden
- Bestimmte Annahmen, die gemacht wurden, aber nicht der Realität entsprechen
- Real-World Logik Fehler
    - Password ändern-Funktion: Wenn kein existierendes PW übergeben wurde, ist es ein Admin, der das PW so ändern darf
    - Multi-Stages werden nacheinander durchgeführt: Aber User kontrollieren alle Inputdaten und konnten so eine Bestellung abschicken, ohne dass sie vorher bezahl hatten
    - Multi-Stage-Fehler: Versicherung kann man sich als Versicherungsnehmer selbst unterschreiben, wenn man die entspr. Parameter kennt
    - Objekt falsch eingesetzt: In einer Bank-WA wurde das Objekt Ccustomer bei der Registierung falsch eingesetzt, dies ermöglichte einen bereits Registrierten Kunden durch Ausfüllen der Registrierung sich als dieser User der Registrierung zu imitieren
    - usw.
- Verhindern: Dokumentation! Kommentieren des Quellcodes (auch Annahmen)! Annahmen im Quelltext überprüfen!

Chapter 12 – Attacking other Users

- Erlaubt Session Hijacking, unauthorisierte Aktionen und die Enthüllung persönlicher Informationen
- Zunehmender Fokus auf clientseitige Sicherheit
- Warum in eine Bank einbrechen, wenn man 10 Mio Bankkunden hat, die man auch angreifen kann?
- Cross-Site Scripting (weit verbreitet, eignen sich aber nur indirekt dazu die WA zu übernehmen)
    - Reflected XSS (Parameter wird übergeben und ungeändert auf der Seite dargestellt: error.php?msg=Sorry+an+error

Über diese Schwachstelle kann auch Javascript eingespeist werden
wird auch first-order XSS genannt)
Warum erstellt ein Angreifer nicht einfach eine Seite, mit dem schadhaften JS-Code?
1. Es geht meist um das Session Token (Cookie) und dieser ist nur auf der WA-Seite zugänglich
2. Sicherheitskontext der WA wird verwendet und folglich das eingeschleuste Scripts problemlos ausgeführt
3. User vertraut der WA und es ist wahrscheinlicher, dass er die URL der WA besucht
4. Würde der Angreifere Phishing betreiben, würden viele Mailclients diesen Versuch erkennen und davor warnen

- Stored XSS (wenn Userdaten gespeichert werden (in DB) und angezeigt werden, ohne sie zu bereinigen oder zu filtern)
  Kommt v.a. Bei interaktiven WA vor, wo versch. User miteinander kommunizieren
  JS wird beim Aufrufen dieser Daten ausgegeben und vom Browser ausgeführt
  Aus 2 Gründen ernster als reflektives XSS:
  1. Angreifer braucht nur darauf zu warten, dass Opfer die Seite aufrufen
  2. Opfer benutzt auch die Anwendung, wenn er auf der Seite mit dem schadhaften Code landet
  Es auch möglich in hochgeladenen Dateien XSS zu speichern:HTML-Dateien, bei IE auch andere Dateitypen wie JPEG mit HTML-Sourcecode, da IE sich auf Dateiendung verlässt und defaultmäßig als HTML rendert
- DOM-basierte XSS Schwachstellen (die Schwachstelle wird durch JavaScript möglich (schreiben der Message der URL durch JS auf die Seite))
- Real-World Attacken MySpace Wurm, Webmail-WA mit XSS-schadhaften Mails
- XSS mit anderen Schwachstellen kann zu erheblichen Problemen führen (z.B. XSS bei Benutzername + Schwachstelle den Benutzername von jedem anderen User zu verändern = Katasrophe!)
- XSS kann neben SESSION Hijacking auch dazu verwendet werden, die Seite zu verändern und neue (gefährliche) Elemente einzufügen
  XSS kann trojanische Funktion einspeisen (Login-Maske, die die Daten zum Angreifer übermittelt)
  XSS kann zu ungewollten Aktionen des Users führen (Privelegien des Attackers erweitern (was natürlich nur dann erfolgreich ist, wenn ein Admin Opfer ist))
  XSS kann Autocomplete und gespeicherte PW an Angreifer übermitteln
  XSS ist auf einer Trusted Site des IE möglich per ActiveX aufrufe zu tätigen und Daten abzufragen
  Auch Key-Logging und Abfragen des Zwischenspeichers (bei IE), History-Daten, merken von Suchabfragen, PortScan des lokalen Netzwerks und genutzte Anwendungen können aufgezählt werden. Es können durch den Browser auch andere Netzwerk-Hosts angegriffen werden
- Verteilen der Seite mit XSS-Attacke an das Opfer
  - Bei Reflected und DOM-basiert
    - Bei einem gerichteten Angriff die Mail mit URL an Opfer(gruppe) senden
    - URL per Instant-Message
    - Third-party Anwendungen nutzen, um Anfragen zu generieren, die XSS Schwächen triggern
    - Banner und Ads zu Seiten mit XSS-Attacke oder XSS-Attacke wird direkt auf

der Seite mit Werbebanner ausgeführt (zum Session Hijacking)
    - „Tell a friend“ oder Feedback-Funktion nutzen mit URL zur XSS-Attacke
  - Stored XSS
    - in-Band (innerhalb der WA über Felder die anderen Benutzern angezeigt werden wie Name, eMail etc.)
    - out-of-band (XSS wird über Schnittstellen zu anderen Daten (Mail, XML etc.) eingespeist)
- Auffinden von XSS-Schwachstellen (übermitteln von <script>alert('hello');</script>)
  - Es gibt einfache Filter, die überprüfen, ob <script> vorkommt
    - Umgehen mit <ScRipt> oder <script > oder kodieren der Klammern, versuche auch <scri<script>pt>
  - Übermittele einen einzigartigen String und überprüfe, ob er in dem Quellcode wieder auftaucht
  - weitere Stellen zum Einschleusen von JS (mit style=“x:expression(alert())“) oder onerror oder onfocus braucht ebenfalls kein <script>-Tag
  - umgehen mit Null Byte %00, url-encoden, double-encoding oder HTML-encoded (auch UTF-7, US-ASCII, UTF-16)
  - Bereinigungen umgehen: <scr<script>ipt>, null byte, teilweise schleust man auch direkt in JS-Code ein, so dass kein Tag notwendig ist#
  - Längenbeschränkung kann durch verweis eines externen Skripts oder indem man mehrere Eingabefelder nutzt umgangen werden
  - POST und GET können ausgetauscht werden und so die Filter umgangen werden
- HTTPOnly Cookies und Cross-site Tracing
  - HTTPOnly Cookies werden nicht von allen Browser unterstützt und daher kann JS weiterhin auf die Cookies zureifen
  - Wenn JS HTTP Trace aufrufen kann mit Ajax, kann das Cookie ebenfalls ausgelesen werden
- Verhindern von Reflected und stored XSS
  - Identifizieren aller Instanzen, wo Userdaten in der WA angezeigt werden
  - Validieren des Inputs: Längenbeschränkung, nur erlaubte Zeichen, entspricht einem RegExp
  - Validieren des Outputs: HTML-kodieren aller Zeichen
  - Gefährliche Einfügepunkte eliminieren: kein Einfügen direkt in JS, innerhalb von <img> oder <input> (HTML-kodieren bringt oft nichts, da viele Browser das automatische dekodieren, auch Vorsicht wenn Attacker Zugriff auf Kodierung hat und diese verändern kann)
- Verhindern von DOM-basiertes XSS
  - Validieren des Inputs: clientseitiges Überprüfen per RegExp, aber auch serverseitiges Überprüfen, ob Parametername vorkommt und welchen Wert er hat (falls nicht, seite nicht anzeigen)

        - Validieren des Outputs: clientseitiges HTML-kodieren
    - XST verhindern, indem TRACE auf dem Webserver deaktiviert wird
- Redirection Angriff (Tritt auf, wenn Userdaten zum berechnen des Redirects verwendet werden; können zum Phishing verwendet werden)
    - Redirects per HTTP 3xx Status
    - Per HTTP Refresh
    - Per HTML-Meta Refresh
    - Per JS
    - Filter/Reinigung (absolute URLs verhindern) umgehen: hTtp:// oder %00http:// oder Leerzeichen vor http oder Kodierungen, bei reinigung htthttp://p:// oder http://http:// Wenn nur eigene Domain erlaubt ist http://attack.com/?http://wa.com Auch ist es möglich auf Subdomains des Angreifers weiterzuleiten (wa.com/redir.php?target=admin.php → wa.com/admin.php, wa.com/redir.php?target=.attack.com → ca.com.attack.com)
    - Verhindern: Keine Userdaten in Berechnung der Redirection, sonst direkte Links oder Indexnummer mit Hinweis zur URL, sonst
        - relativer Pfad, also /[a-z]
        - vor allen Redirects wird http://wa.com gestellt, nur Redirects mit diesem Prefix werden erlaubt
- HTTP Header Injection (etwa bei Location oder Cookie: Angreifer kann neue Zeilen in Header einfügen: etwa weitere Cookies, Redirection, Angriffe gegen ActiveX Controls)
    - Es können auch Proxys mit schadhaften Code verseucht werden (bei HTTPS nicht möglich)
    - Verhindern: Wenn es notwenid ist, entsprechende Input Validierung und Output Validierung (unterhalb ASCII 0x20 ist verdächtig)
- Frame Injection (möglich, wenn Frames unvorhersagbar benannt sind, dann ist es möglich den Frameinhalt zu überschreiben (dazu muss der Besucher auch auf der Frameseite sein) Verhindern: keine Framenamen oder unvorhersagbare, eindeutige Framenamen)
- Request Forgery (auch bekannt als Session Riding: Der Angreifer muss die SessionId nicht kennen, sondern nutzt das normale Verhalten von Browsern, um Anfragen durchzuführen, die der User nicht beabsichtigt)
    - On-Site Request Forgery (ist ähnlich dem stored XSS; allerdings auch dann möglich, wenn XSS nicht möglich ist: Input wird HTML-kodiert, Input wird innerhalb des img src ausgegeben: Eingabe von *../deleteUser.php?id=124#* könnte beim Admin dazu führen, dass der User gelöscht wird)
      Verhindern durch Validieren des Userinputs (Zeichen verbieten (./?&=)), HTML-kodieren des Output bringt nichts, da der Browser diesen Schritt rückgängig macht
    - Cross-Site Request Forgery (Angreifer richtet eine Seite ein, die ein Request auf eine XSRF-offene WA loslässt, ist one-way, OSRF kann auch wie ein Wurm sein, eBay war 2004 betroffen, man konnte ein Request loslassen, der das Opfer auf eine Auktion bieten ließ)
        - XSRF ist dann wahrscheinlich, wenn Cookies genutzt werden, da die Cookiedaten bei jedem nachfolgenden Aufruf übertragen werden

        - XSRF verhindern: nicht nur auf Cookies verlassen (sicherheitskritische Anwendungen: Session Tokens, die mit Hidden Fields übertragen werden; verlassen auf den Referer macht keinen Sinn, da er nicht zuverlässig ist, multi-stage, um Aktion auszuführen und danach weiter mit Token oder Nonce)
          XSRF-Verteidigung kann durch XSS-Schwachstellen umgangen werden (da XSS kann 2-way mit WA interagieren, also die Nonce erhalten und die diese für folgende XSRF-Anfragen verwenden)
- JSON Hijacking (Erhalten von Daten, verarbeiten bzw. ändern; man kann damit die Same Origin Policy umgehen (JS und damit JSON können von einer anderen Domain stammen))
    - 1. Array Konstruktor überschreiben, um die Daten zu bearbeiten
    - 2. Implementieren der angegebenen Callback-Funktion
    - Finden der Schwachstelle (AJAX, wo sensible Daten übertragen werden, nur per GET möglich, da include nur über <script> möglich ist)
    - Verhindern durch Anti-XSRF-Maßnahmen (sensible Daten nur, wenn ein vorher genierter, zufälliger Parameter richtig zurück übergeben wurde;
      da die WA mit Ajax statt mit <script> die Daten abrufen kann, kann problematischer Code (etwa while(1);) eingefügt werden, beim Abruf mit Scripts wird dieser Code ausgeführt, während mit Ajax der Code vorher noch verarbeitet werden kann;
      Benutze ausschließlich die POST-HTTP Methode, um die JSON-Daten zu erhalten)
- Session Fixation (Schwachstelle entsteht, wenn anonyme Sessions beim ersten Zugriff vergeben werden, also ein Upgrade der Session von anonym zu authentifiziert; bei der Session Fixation bekommt der Angreifer eine Session und übergibt sie dem Opfer, so kann der Angreifer mit der vorgegebenen Session, wenn das Opfer sich eingeloggt hat, die Session mitbenutzen)
    - Übertragen der SessionId zum Opfer
        - Per URL, wenn über Query String
        - per XSS-Schwachstelle setzen des entsprechenden Wertes, wenn per Cookie oder versteckten Formularfeld
        - teilweise ist es auch möglich eine SesssionID einfach vorzugeben, die vorher nicht angefordert werden musste
        - Finden von SessionFixation: Vor dem einloggen wird eine Session verwendet und beim Einloggen wird keine neue vergeben, sondern die bestehende nur aufgewertet
        - Vor dem Login wird kein Token vergeben. Wenn man eingeloggt sich als anderer User einloggt, wird kein neuer Token verwendet, sondern der alte weiterhin
        - Verhindern: Sobald sich ein Benutzer einloggt oder wenn ein User zuerst persönliche oder sensible Daten überträgt → neue Session
          Per Page Tokens
          Nicht beliebige Tokens akzeptieren
- ActiveX Kontrollen angreifen
    - ActiveX Kontrollen können klassische Schwachstellen enthalten wie Buffer Overflows, Integer Bugs etc.
    - Methoden, die gefährlich und schwachstellen in der Benutzung beinhalten wie (LaunchExe, SaveFile, Loadlibrary oder ExecuteComander)

- Es gibt Methoden, die nicht aufgerufen werden (zum testen, erweitern etc.), aber schadhaft sein können (COMRaider zum Auflisten aller Methoden einer ActiveX Control)
- Die Entwickler einer ActiveX Control müssen darauf achten, dass ihre Methoden keine schadhaften Aktionen erlauben
  Überprüfen der Domain, welche die Methode aufruft/übergebene Parameter müssen kryptografisch signiert sein
  Solche Maßnahmen können durch XSS-Schwachstellen auf der Seite mit ActiveX umgangen werden

- Local Privacy Attacken
  - Persistente Cookies (speichern sensibler Daten in Cookies mit Expire Angabe, Zugriff übers Dateisystem des Opfers möglich)
  - Cached Web Inhalt (mit sensiblen Inhalt (wenn folgende Angaben fehlen wird gecacht: Expires:0, Cache-control: no-cache, Pragma: no-cache))
  - Browser History
  - Autocomplete (wenn autocomplete=off nicht gesetzt ist, werden die Daten gespeichert)
  - Verhindern: Keine sensiblen Daten in persistenten Cookies, keine Cache, niemals Query String zum übertragen von sensiblen Daten verwenden (nur POST mit Formularen), bei sensiblen Daten autocomplete=off

- Fortgeschrittene Techniken
  - Ajax (mit Ajax kann innerhalb einer XSS-Schwachstelle jede Art von Anfrage an eine WA geschickt werden, z.B. Ändern des Passworts; der MySpace-Wurm hat Ajax verwendet)
    - Per Ajax kann zwar keine andere Domain aufrufen, aber mit JS können dennoch Off-Site Requests abgegeben werden (<img src="anderedomain.de + passw")
  - Anti-DNS Pinning (brechen der Same Origin Policy, indem die Namensauflösung sich ändert (Zugriff auf bisherigen Server muss per Firewall unterbunden werden) während das Script ausgeführt wird (um auf Seiten im LAN des Opfers zuzugreifen))

Chapter 13 – Automating Bespoke Attacks

- Ist uninteressant, da keine neuen Schwachstellen beschrieben werden, sondern nur Methoden und Tools, mit denen man Angriff automatisieren kann

Chapter 14 – Exploiting Information Disclosure

- Fehlermeldungen verwenden
  - Script Fehlermeldung (bessereres Verständnis der Anwendung, Code kann indirekt untersucht werden)
  - Stack Traces (enthält: Grund des Fehlers, verwendete Librarys und 3rd-party Anwendungen, Zeilennummer, Info zur Anwendung und Umgebung)
  - Informative Debug Nachrichten (mit Ausgabe vieler Angaben (Verzeichnisse, Sessions, Back-end Systeme, Schlüssel))
    - Server und DB Nachrichten (SQL-Fehler mit Angabe von Tabellen und Felder)

- Öffentliche Informationen verwenden
    - Suchen nach ausgegebenen Meldungen (→ Diskussionen, Referenzen, 3rd-party Anwendungen mit Open Source und bekannten Schwachstellen, Bugs etc.)
    - Informationen aufgrund des Designs (als Teil des WA Kerns), Seiteneffekte durch anderen Funktionen, Debugging-Infos, durchSchwachstellen (z.B. Zugriffskontrolle), sensible Infos, die die WA veröffentlicht (usernamen, nnerhalb von Profilen (Rollen), Password wird zwar maskiert angezeigt, ist im Sourcecode aber veorhanden, Logfiles, Kommentare im Quelltext)

Chapter 15 – Attacking Compiled Applications

- Wenn WA in nativer Sprache geschrieben, ist es für Buffer Overflows etc. Schwachstellen empfänglich
    - Buffer Overflow (Wenn Userdaten in einen zu kleinen Speicher-Buffer kopiert wird, dadurch wird angrenzender Speicher mit den Userdaten überschrieben, dadurch kann ein Angreifer bewirken, dass beliebiger Code auf dem Server ausgeführt wird)
        - Stack Overflow (z.B. strcpy, wo ein Buffer mit variabler Größe in einen Buffer mit fester Größe kopiert wird; wenn Funktion aufgerufen wird, wird eine Adresse auf dem Stack gespeichert, wohin zurückgesprungen wird, wenn die Funktion ausgeführt wurde, wenn die Funktion (bzw. Variablen innerhalb der Funktion) mehr Speicher in Anspruch nehmen als geplant, wird die Rückkehr-Adresse mit Daten überschrieben)
        - Heap Overflow (gleich wie bei Stack, nur dass der Buffer sich im Heap befindet; nicht Return Adresse betroffen, sondern Heap-Kontrol-Strukturen, die überschrieben werden, beim Reparieren des Heaps werden Userdaten verlinkt)
        - Off-by-One Schwachstelle (erscheint, wenn ein Angreifer ein Byte hinter den allokierten Buffer schreiben kann)
        - Kann dazu führen, dass Daten anderer Benutzer ausgegeben werden, wenn Überlange Daten eingegeben werden, weil der Null Byte-Begrenzer verschwindet
    - Integer Overflows (erhöhung eines Wertes über seinen maximalen oder erniedrigung unter minumum)
    - Vorzeichen-Fehler (Vorzeichen und vorzeichenlose Zahlen werden verglichen oder miteinander verrechnet → Zahl mit Vorzeichen wird als vorzeichenlos behandelt, so dass eine negative Zahl eine große positive Zahl wird)
    - Format String Schwachstelle (%n gibt die Zahl der ausgegebenen Bytes und sichert sie in der angegebenen Variable, wenn der Format String mehr Parameter enthält als angegeben, wird weiter vom Call Stack abgearbeitet → Anwendung stürzt ab)

Chapter 16 – Attacking Application Architecture

- Gestufte Architekturen (Grobe Abstufung: Präsentation, Anwendung, Daten-Layer)
    - Feine Abstufung (Anwendungsserver (Tomcat), Präsenation (WebWork), Authorization & Authentifizierung (JAAS), Kern Anwendung (Struts), Business Logic (Java Beans), DB objekt relational mapping (Hibernate), JDBC, DB)
    - Wenn es Defekte auf einer Schicht gibt, kann dies zu Schwachstellen führen
        - Ausnutzen von Trust-Relations zwischen den Schichten (oft ist alleine die

Anwendungsschicht dafür zuständig, dass die Zugriffskontrolle implementiert wird; verhindern indem man z.B. für die DB versch. Benutzer einsetzt, die z.B. nur lesen dürfen; WA läuft im OS unter einem Benutzer mit zu vielen Rechten)

- Wenn verschiedene Schichten nicht adäquat voneinander getrennt sind, kann ein Defekt auf einer Schicht es ermöglichen die Sicherheitsbemühungen einer anderen Schicht zu untergraben (kommt oft vor, wenn die Schichten auf der selben Maschine laufen, z.B. mit Path Traversal-Schwachstelle möglich auf DB-Dateien von MySQL zuzugreifen und Zugangsdaten auszulesen)
- Das Kompromettieren einer Schicht kann es ermöglichen eine andere Schicht anzugreifen und so weitere Schichten zu kompromettieren

- Verhindern: Bei LAMP führt das kompromettieren einer Schicht in der Regel zum Kompromettieren aller Schichten
  - Jede Schicht sollte eigene Kontrollen implementieren, um sich gegen unauthorisierten Zugriff zu schützen
  - Verschiedene Hosts, um versch. Schichten laufen zu lassen; keine unbeabsichtigte Interaktion der Schichten miteinander
  - Sicherheitspatches der verschiedenen Software installieren
  - Konfiguration härten (ungenutzte Funktionen löschen, Sicherheit beachten)
  - Sensible Daten in den verschiedenen Schichten verschlüsseln
- Shared Hosting und ASP
  - Virtual Hosting
  - Apllication Service Provider (gleicher Service an versch. Kunden (z.B. Hosting))
  - Angriff auf Zugriffsmechanismen (Remote Access ist unsicher (nicht verschlüsselt beim Übertragen von Zugangsdaten)), Zugriffsmechanismen versch. Kunden sind nicht vollständig voneinander getrennt, gleiches gilt für DB, Administrative Programme oder Dateisysteme
  - Backdoor, um Zugriff auf Anwendungen anderer Kunden zu bekommen (Apache versch. Kunden läuft als gleicher Benutzer)

Chapter 17 – Attacking the Web Server

- Schwachstellen in der Server-Konfiguration
  - Standard-Zugangsdaten
  - Standard Inhalt (zum Debuggen (wie phpinfo()), Beispiele zu Funktionen (die Schwachstellen haben oder einem Angreifer nutzen), Mächtige Funktionen (die zwar eigentlich geschützt sein sollten, aber dennoch durch einen Angreifer genutzt werden können))
  - Verzeichnis-Listen (um versteckte Inhalte zu finden)
  - Gefährliche HTTP-Methoden, die nicht deaktiviert wurden (PUT: Upload einer Datei, DELETE, COPY, MOVE, SEACH, TRACE (Ajax))
  - Manchmal ist ein Webserver als Proxy konfiguriert (→ Webserver kann zum Angriff auf 3rd-party WA genutzt werden, innerhalb des WA-Netzwerk auf Hosts verbinden, kann sich auf andere Services des Proxys verbinden und so Firewall umgehen)

- Falsch konfigurierte Virutal Hostings (defaulthost muss ebenfalls konfiguriert sein, sonst wird Security-Konfigs darauf nicht angewendet)
- Verhindern: Ändern von Zugangsdaten, Default Content und Funktionen deaktivieren, alle Directory-Listing disablen, HTTP-Methoden deaktivieren, Überprüfen, dass der Webserver nicht als Proxy verwendet werden kann
- Fehler in der Webserver Software (↔ Updates installieren)

Chapter 18 – Finding Vulnerabilities in Source Code

- Ansätze für Code Reviews
  - Black-Box (bisherige Methoden) vs. White-Box Tests (effektiv, da die Struktur untersucht werden kann, Backdoor PW entdeckt werden)
  - Methoden zum effektiven Code Review
    - Verfolge Userdaten vom Eingangspunkt aus durch die WA
    - Finde Signaturen, die allgemeine Indizien für Schwachstellen sind
    - Führe ein zeilenweises Code Review durch, wenn es eine „riskante“ Stelle ist, um die Funktionsweise zu verstehen
  - Signaturen für allgemeine Schwachstellen
    - XSS (Userdaten werden ungefiltert in der HTML-Ausgabe verwendet (subtiler, wenn Userdaten erst in Variablen zwischengespeichert werden))
    - SQL Injection (Userdaten werden verwendet, um SQL-Befehl zu erzeugen)
    - Path Traversal (Userdaten werden für Zugriffe aufs Dateisystem verwendet)
    - Beliebige Weiterleitungen (Userdaten werden verwendet, um Location:redirects durchzuführen)
    - OS-Befehle Einspeisen (Userdaten werden innerhalb von Betriebssystem APIs verwendet)
    - Backdoor PW
  - Quelltext Kommentare können auch Hinweise auf Fehler oder unsichere Passagen hindeuten

Risikomanagement in IT-Projekten

Einführung in das Risikomanagement

- Allgemeines
  - Regel: Agieren und nicht reagieren!
  - Risikomanagement (RM) ist vorausschauendes Handeln
  - Die Behebung eines eingetretenen Risikos ist teurer als das vorausschauende RM
  - RM ist wichtig, weil
    - Gesetze es erfordern
    - Budgets in der Projektabwicklung/Produktentwicklung werden enger

        - Versuchsprojekte mit überhöhten Budget werden aufgrund der IT-Krise kaum noch durchgeführt
        - Dadurch: Weniger QS-Maßnahmen (→ Fehleranfälligkeit steigt)
          Sparen an Werkzeugen (Tolls wie Projektmanagement sind zwingend notwendig)
          Sparen von Personal
        - Einsparungen haben großen Einfluss auf RM (Anzahl der Risiken steigt, Auswirkungen der Risiken steigt)
        - → Einsparungen nur in Einklang mit RM
    - Enger werdende Releasezyklen
        - Es entstehen neue Risiken:
            - Keine Koordination der verschiedenen Releases auf dem Markt zu erscheinen (Kunde muss dafür Sorge tragen, dass zwei Releases miteinander per Schnittstelle kommunizieren können)
            - → Welche Auswirkung, wenn Schnittstelle nicht einwandfrei funktioniert
            - → Schnittstelle wird nicht mehr weitergepflegt
            - → Schnittstellen-Hersteller wird von Wettbewerber übernommen
            - → Auswirkung, wenn aus Zeitgründen nicht auf aktuelle Version geupdatet werden kann
    - Reduzierung der personellen Ressourcen innerhalb des Entwicklungsteams
        - In Maßnahmen des RM werden oft zusätzliche Personelle Ressourcen als Maßnahme aufgeführt, dies ist dann nicht mehr möglich
        - Personalreduzierung lässt Mitarbeitermotivation (ebenfalls ein Risiko) sinken
        - Know-How-Verlust kann zu technischen Risiko werden
    - Übertragung von Risiken auf Unterauftragnehmer
        - V-Modell als Instrument zur Risikoübertragung
        - RM wurde Auftragnehmer aufgezwungen (durch Branchen, wo RM sich durchgesetzt hat)
        - Auf Hochschulen wird RM gelehrt, so dass diese Schüler RM in ihrem Unternehmen einsetzen
- RM beschäftigt sich mit Risiken bevor sie eintreten
- Folgende Aktivitäten
    - Identifizieren von Risiken
    - Analyse & Bewertung von Risiken (inkl. Erstellung des Maßnahmenkatalog)
    - Festlegung von Risikostrategien
    - Überwachung der Risiken
- Risikoanalyse und Bewertung am schwierigsten
    - 2 Hilfsmittel: Risikoliste & Risikomatrix
- RM wird verstärkt eingeführt, dies aber nicht aus Einsicht, sondern aus Druck

        - Vom Militär und Luftfahrt in die IT-Branche
- Einstellung zu Risiken
    - Starke wechselseitige Abhängigkeiten im Umgang mit Risiken in beruflichen wie privaten Bereich
    - Einflussfaktoren zur Einstellung gegenüber Risiken
        - Wirtschaftliche Situation des gesamten Marktes
            - siehe Anlager vor und nach Börsencrash des neuen Marktes (aus Fehlern lernt man)
        - Wirtschaftliche Situation des Arbeitgebers
        - Umfeld/Branche des Arbeitgebers/Unternehmens
            - Unternehmen im sicherheitskritischen Bereich haben RM als Unternehmenskultur
            - New Economy sind extrem risikobereit (immer neue Technologien, deren Risiken nicht einschätzbar sind)
            - Unernehmen, die sich auf neue Märkte begeben, müssen gewisse Risiken eingehen
            - Etablierte Unternehmen haben meist einen gesunden Mix an risikobereiten und -bewussten Mitarbeitern – bei Start-Ups ist dies meist anders
        - Individuelle Erfahrungen des Einzelnen
            - Je größer Erfahrungsschatz, deso professioneller der Umgang mit Risiken (→ nicht, dass er vorsichtiger wird, sondern Risikowahrscheinlichkeit und Auswirkungen einschätzen kann)
            - Junge/neue Mitarbeiter tendieren zu extremen (Übervorsichtig/Risikobereit)
        - Private Situation des Einzelnen
            - Risikobereitschaft eines mehrfachen Familienvaters geringer als bei einem ledigen Mann
            - Erfahrung der letzten Projekte
- Risikotypen in der IT
    - steckt allgemein noch in den Kinderschuhen
    - Risikotypen:
        - Geschäftliche/käufmännische Risiken
        - Technische/technologische Risiken
        - Terminliche Risiken
        - Ressourcenrisiken
        - Politische Risiken (schwerwiegendsten Risiken)
    - Arten von Projekten
        - Interne Projekte
        - Externe Projekte (Auftrag durch externen Kunden)

        - Produktentwicklung
    - Kaufmännische Risiken in externen Projekten (finanzielle Absicherung)
        - Budget wird durch Auftraggeber gekürzt
        - Auftraggeber stoppt Projtekt bzw. verschwindet vom Markt
        - Auftragnehmer muss neue Technologie setzen, die er vorher nicht einkalkuliert hat
        - Auftraggeber benötigt zusätzliche Ressourcen, die nicht eingeplant waren
    - Kaufmännische Risiken bei internen Projekten
        - haben nicht solche Auswirkungen auf die finanzielle Lage des Unternehmens wie bei externen Projekten
        - Unterscheidung zwischen Forderungsprojekt und Projekt mit Einfluss auf interne Geschäftprozesse (SAP): im letzten Fall deutlicher Einfluss, da man in SAP investiert hat
        - Beispiele
            - Fehlerhafte Entwicklung zur Auftragabwicklung
            - Budgetkürzung (v.a. Wenn externe Projekte in Schieflage geraten)
            - Mitarbeiter verlassen unternehmen bzw. personelle Ressourcen für Projekt werden gekürzt
    - Kaufmännische Risiken in der Produktentwicklung
        - Wettbewerber mit vergleichbaren Produkt kommt zuerst auf den Markt
        - Schlüsselkunde entscheidet sich für anderes Produkt
        - Preisverfall für eigenes Produkt
    - Technische Risiken bei externen Projekten
        - Welche Technologie? Haben Projektmitarbeit Erfahrung mit der Technologie? Haen zusätzliche Mitarbeiter Erfahrung für den Fall eines Engpass?
        - Technologie wird abgelöst
        - Setzt Auftragnehmer Technologie schon ein oder nur ein Versuchsballon
        - Erfahrung der Unterauftragnehmer mit Technologie
    - Technische Risiken bei internen Projekten (vergleichbar mit externen)
        - Wird Technologie auch extern eingesetzt (oder ist interne Technologie bereits überholt)
        - gleiches gilt für Werkzeuge
    - Technische Risiken bei der Produktentwicklung
        - wesentlich mehr zukunftsorientiert
        - Plattform zukunftssicher
        - Datenbank
        - Architektur plattformunabhängig
        - Wie verhalten sich die Partner des Herstellers

- Zusammenhang zwischen technischen und kaufmännischen Risiken
  - Veraltete Technologie im Einsatz → Investition in neue Technologie
  - Schnittstelle wird nicht merh unterstützt → Investition in eigenleistung
  - Kaufmännische Risiken haben nahezu keinen Einfluss auf technische
- Terminliche Risiken bei externen Projekten
  - Durch Abzug von Ressourcen gerät das Projekt in Verzug
  - Technologien werden abgezogen, dadurch leidet die Produktivität des Teams
  - Interne Projekte werden nicht so ernst genommen
- Terminliche Risiken bei Produktentwicklung
  - Anderer Hersteller mit vergleichbaren Produkt kommt früher auf den Markt
  - Zu entwickelnde Schnittstellen können zu terminliche Risiken führen, da technische Probleme auftreten können
  - Änderung von gesetzesgrundlagen
  - Releasepläne können nicht eingehalten werden, da Großkunde Wünsche hat
- Zusammenhang zwischen terminlichen und kaufmännischen Risiken
  - Kein Zusammenhang im umgekehrten Fall
- Ressourcenrisiken bei externen Projekten
  - Krankheit einiger Mitarbeiter
  - Entlassungwelle oder Kündigungen einiger Mitarbeiter
  - Mitarbeiter aus anderen Projekten können nicht eingesetzt werden, da anderes Projekt sich ebenfalls verzögert hat
  - Keine ausgebildete Resourcen für neue Technologie (Ausbildungsmaßnahmen kosten Zeit und binden Personal!)
  - Auftraggeber nimmt Option wahr, weitere Funktionalitäten umzusetzen, mit denen der Auftragnehmer nicht gerechnet hat
- Ressourcenrisiken bei internen Projekten
  - Kurzfristiger Abzug von Mitarbeitern
  - Neue Technologien einsetzen (Risiko höher, da eigenfinanziert)
  - Interne Projekte als Ausbildungsmaßnahme für Mitarbeiter
  - Wenn Unternehmen sich von freien Mitarbeitern trennt
- Ressourcenrisiken bei Produktentwicklung
  - sind Gewohnheit (s. Personalabbau)
  - eher Krisenmanagement
    - Release kann nicht termingemäß erstellt werden
    - kaum Tests von neuen Releases
    - Integration von zugesagten Funktionalitäten wird verzichtet

            - Vernachlässigung der internen Weiterbildung zu neuen Releases
            - Fehlende Stabilität des Releases hat zur Folge, dass es nicht eingesetzt wird
            - ->Wartungseinnahmen sinken drastisch
        - Typische Vorgehensweise bei Personalabbau
            - Heilige Kuh „Vertrieb“ wird geschont
            - Marketing + Entwicklungsbereich (dort v.a. Testabteilungen)
            - Presales & Consultants in Landesniederlassung
        - Politische Risiken
            - sind rational nicht nachvollziehbar und treten aufgrund von persönlicher Motivation einzelner Entscheidungsträger auf
            - → höchst komplex und kritisch!
                - Entscheidung von Ausschuss wird durch Entscheidungsträger ignoriert
                - Trotz besserer Qualifikation wird ein anderer Unterauftragnehmer beauftragt
                - Besser geeignete Technologie wird nicht eingesetzt
                - Personal wird von Projekt abgezogen, obwohl es wichtig ist
            - Sind gefährlich, weil nicht berechenbar.
            - Starke Auswirkungen hinsichtlich des Auftretens anderer Risikotypen
            - Entscheidung ist schwer revidierbar, da sie vom Management stammt
            - Ursachen
                - Technische Unerfahrenheit des Entscheidungsträgers
                - Persönliche Vorteile oder Beziehungen des Entscheidungsträgers
                - Wettbewerbssituation
                - Persönliche Aversion, Machtsicherung gegenüber anderen Managern
                - Persönlich Aversion gegenüber Produkten oder Herstellern von Produkten
- Auswirkungen nicht erkannter Risiken
    - Auswirkungen nicht erkannter kaufmännischer Risiken ist, dass das Projekt defizität wird
        - Bei externen Projekten wird dann versucht nachzuverhandeln
        - Bei internen Projekten wird nachbudgetiert oder gestoppt
        - Bei Produktentwicklung wird Release mit weniger Funktionen veröffentlicht
            - Wenn Produktentwicklung gestoppt, kann Produkt verkauft oder in eigener Firma ausgelagert werden (Spin Off)
        - Es gilt einige Besonderheiten bei der Produktentwicklung:
            - Produkt wird über die Wartungseinnahmen finanziert (15% - 18% der jährlichen Lizenz)
            - Wenn Käufer keine neuen Releases und keine neuen Lizenzen brauchen, ist die

Deckung der Kosten nicht möglich

- Auswirkungen nicht erkannter technischer Risiken
  - Zusätzliches Investition in ein alternatives Werkzeug
  - Erhöhter Schulungsaufwand
  - Nutzung von veralteten Werkzeugen
  - Zugriff auf externe Berater
  - Erhöhte Eigenleistung oder länger Dauernde Suche nach alternativen Unterauftragnehmer

- Dem RM angelehnte Managementtechniken
  - Managementtechniken, die zum Tragen kommen, wenn Risiko eingetreten ist:
    - Krisenmanagement
      - Wenn Risiko eingetreten ist
      - Eine Krise ist die erste Eskalationsstufe eines unterschätzten oder nicht erkannten Risikos, das eingetreten ist
      - RM hat also versagt. Warum?
        - Risiko war nicht vorhersehbar
        - Eintrittswahrscheinlichkeit des Risikos wurde unterschätzt
        - Auswirkungen wurden unterschätzt bzw. waren in der Form nicht vorhersehbar
      - Krisenmanagement erst dann, wenn
        - Reguläre Fortschritt des Projekts fraglich oder ausgeschlossen UND
        - Gravierende Auswirkungen des eingetretenen Risikos, die innerhalb des Projektmanagements nicht gelöst werden konnten UND
        - Auswirkungen des Risikos bekannt
      - Zuständig ist der Krisenstab (bestehend aus Risikomanager, Projektleiter, Vertreter des Auftraggebers, kaufmännischer Vertreter, Vertreter des Managements)
    - Notfallmanagement
      - nächste Eskalationsstufe, wenn Krise nicht bewältigt werden konnte

Risikoidentifizierung

- Allgemeines
  - Bestandsaufnahme der für das Projekt betreffenden Risiken
  - Risiko, das nicht erkannt wurde, kann beim Eintreten auch nicht mit entsprechenden Maßnahmen in den Griff bekommen werden
  - Risikoidentifizierung über das gesamte Projekte!
  - Identifizierung beginnt bereits vor Erstellung des Angebots (viele unbekannte Parameter, daher viele Risiken)

- Erstellung ist ressourcenintensiv und kostet daher Geld
  - Budget bereits vorgesehen und freigegeben
  - Keyplayer beim Kunden identifiziert und bekannt
    - Project owner (Unterzeichner des Projekts)
    - Budget owner (Freigeber des Budgets)
    - Technischer Owner (Verantwortlich für die technischen und fachlichen Inhalte)
  - Exisitiert ein Champion beim Kunde (Person, die sich beim Kunden für die Vergabe des Projekts an eigenes Unternehmen einsetzt)
  - Stellenwert des Projektes beim Kunden (wichtig oder nice to have)
  - Ausschreibung so gehalten, dass man eine seriöse Angebotskalkulation durchführen kann
  - Ausschreibung durch dritten, der sich auch für die Vergabe des Projekts bewirbt?
  - Auftragnehmer schon im Auge und nur zweites Angebot nötig
- Risikoidentifizierung während der Angebotserstellung
  - Im Mittelpunkt steht die Analyse des Auftraggebers und des Umfelds des Projekts
    - Da Projektbeginn und Laufzeit bekannt sind, kann schon eingeschätzt werden, ob die Ressourcen im eigenen Haus vorhanden sind (einbeziehen der geplanten Projekte, sind diese Projekte wichtig? Sind Projekte dann in kritischer Phase, so dass mehr Personal dort notwendig ist?)
    - Sind gewünschte Technologien bekannt? Wie ist das Know-How?
    - Wie ist die wirtschaftliche Situation des Auftraggebers? Wie sind die Referenzen des Auftraggebers?
    - Ist das Projektende fest? Unumstößlich?
- Risikoidentifizierung bei den Vertragsverhandlungen
  - Besonders kaufmännische Risiken
  - Risiken werden innerhalb der Vertragsverhandlungen besprochen
    - Zahlungsziele (wenn Bonität des Auftraggebers nicht gewährleistet ist)
    - Einzusetzende Technologien & Werkzeuge (Alternativen auch ok)
    - Unterauftragnehmer, die vorgegeben werden (wer trägt dann die Risiken?)
    - Meilensteine
    - Ressourcenfestlegung (bestimmte Mitarbeiter gewünscht?)
  - Bestandskunden weniger riskant als Neukunden
  - Standardprojekt (Einführung von SAP) oder Neuprojekt
  - Risiken an Autraggeber übertragen bzw. reduzieren
- Risikoidentifizierung bei Erstellung eines detaillierten Projektplans
  - Risiken, die bereits identifiziert wurden, werden nochmal unter die Lupe genommen
  - Da Proposalmanager nicht identisch mit Projektmanager ist

            - Sind wirklich ausreichende Ressourcen vorhanden
            - Ressourcen ausgebildet
            - Erforderliche Werkzeuge im Unternehmen vorhanden
            - Sind die Meilensteine realistisch gesetzt
            - Unterauftragnehmer ausgebildet und wie riskant
    - Risikoidentifizierung in der Analysephase
        - Anforderungsmanagement:
            - Anf. Verändern sich
            - Von unterschiedlichen Personen aus dem Kundenumfeld gestellt (ggf. widersprechen sie sich)
            - Handelt es sich um Anforderung, Änderungswunsch oder um Fehlermeldung?
            - 3 verschiedene Typen von Anforderungen:
                - zu Beginn des Projekts
                - Änderungswünsche während des Projekts
                - Fehlermeldung nach Übergabe des fertigen Produkts:
                    - während Gewährleistungszeit
                    - nach Ablauf der Gewährleistungszeit
            - steigendes Risiko je später Anforderung/Änderungswunsch/Fehlermeldung eingeht
                - Wer ist verantwortlich für Anforderungen auf Kundenseite?
                - Eindeutige Formulierung von Anforderungen
                - Sich ändernde Anforderungen
                - Welche Änderungen sind signifikant für das Projekt
                - welche sind nice.to.have
    - Risikoidentifizierung in der Entwicklungsphase
        - Risiken werden kaum noch identifiziert, sondern treten, wenn dann ein
            - Krankheiten/Entlassungen/Kündigungen von Mitarbeitern
            - Insolvenz des Unterauftragnehmers
            - Auswirkungen eines plötzlichen Technologiewechsels
    - Risikoidentifizierung in der Deploymentphase
        - Voraussetzungen bei der Hardware gegeben
        - stehen Admins zur Verfügung
    - Identifizierte Risiken nehmen mit der Zeit ab
- Hilfsmittel der Risikoidentifizierung
    - Risikoliste (meist mit Excel)

    - Risikoabhängigkeitsliste (geht aus der Risikoliste hervor: Welches Risiko ist wie mit einem anderen Risiko abhängig?)
    - Risikorangliste
    - Risikomatrix
- Risikoeinstellungen
    - Risikoaversion (Risiken werden gemieden)
    - Risikoneutralität (gesunde Einstellung zu Risiken)
    - Risikofreude (geht bewusst Risiken ein)
    - Optimale Kopplung von Risikoeinstellung und Risikobewusstsein (hohes Risikobewusstsein & Risikofreude/Risikoneutralität)
- Risikomanager kann im nachhinein feststellen, wie gut seine Einschätzungen waren und so Erfahrungen sammeln

Risikoanalyse und -bewertung

- Analyse ist
    - Feststellung der potentiellen Wahrscheinlichkeit des Eintretens
    - Ermittlung der möglichen Auswirkungen
    - Aufstellung von Maßnahmen zur Vermeidung des Eintretens
    - Aufstellung von Maßnahmen, die durchzuführen sind, wenn Risiko eingetreten ist
- Risikobewertung
    - RM soll vorhandene Wissen über mögliche Probleme bündeln und die Kommunikation darüber fördern
    - Rechtzeitige Schritte für Problemsituationen, um diese
        - zu verhindern
        - Eintrittswahrscheinlichkeit zu minimieren
        - Schadenshöhe zu minimieren
        - Notfallplanung bereits abzuschließen
    - Zur Bündelung und Kommunikation gibt es 2 Ansätze
        - RM-Manager sammelt per Interviewtechnik
        - Teamorientierter Ansatz (jedes Mitglied bewertet die Risiken selbst und die Ergebnisse werden aggregiert)
    - Verschiedene Typen bei der Risikobewertung
        - Der Nörgler
        - Der Entrepreneur
        - Der Besserwisser
    - Aggregation von Risikowissen
        - Qualitative vs. Quantitative Risikobewertung

        - Quantitativ bedeutet in Geldeinheiten (z.B. bei Versicherungen oder Bilanzierungen)
        - Qualitativ, also Risiken relativ zueinander bewerten, reicht meist aus
    - Für Risiken gibt es folgende Planungsmaßnahmen:
        - Verhinderung
        - Verringerung
        - Schadensbegrenzung
        - Notfallplanung
    - Motivation für Risikoanalyse
        - Priorisierung der Risiken bzgl. der Planungsnotwendigkeit
        - Priorisierung der durchzuführenden Maßnahmen bzgl. der Durchführungsreihenfolge
- Bewertungsmaßstäbe und Größen
    - Eintrittswahrscheinlichkeit (Probability) – Skala von 1 bis 5
    - Schadenshöhe (Impact) – Skala von 1 bis 3
    - Die Abstufungen werden für jedes Projekt neu gesetzt (manchmal sogar innerhalb des Projekts)
    - Die relative Risikomaßzahl (Exposure) = Impact * Probability
    - Um die Einschätzung aller Teammitglieder zu aggregieren wird das arithmetische Mittel für Exposure gebildet. Diese Kennzahl für die Risiken wird dann zur
        - QS der Bewertungen
        - Priorisierung der Risiken (Hohes E, hoher Rang)
        - Beobachtung des Gesamtrisikoverlaufes
        - Beobachtung der Wirksamkeit von Maßnahmen verwendet
    - Nur für die Top10 der Risikorangliste Maßnahmen planen, um Aufwand gering zu halten
- Bewertungszyklen
    - Bewertungen werden regelmäßig wiederholt
    - Welche Risiken steigen bzw sinken?
    - Wenn Risiko in den Top10 landet, muss die Wirksamkeit der Gegenmaßnahme überprüft werden
    - Gesamtsumme der Risiken ist ebenfalls wichtig
        - da somit das Gesamtrisiko steigt
        - Exposure von Risiken steigt
        - Vorherige Risikoidentifizierung ist verbesserungswürdig, da neue Risiken identifiziert wurden
    - Qualittätssicherung bei der Bewertung
        - Qualitätsmaßzahl ergibt sich aus min(Exposure)/max(Exposure) der verschiedenen Mitglieder

        - Wenn q gering → Diskussionsbedarf bzgl Verständnis der Definition des Risikos (was wird unter dem Risiko verstanden?)
        - Wenn q nahe 1 → gleiche Bewertung durch das Team
        - Wenn Teammitglied immer in der Nähe des statistische Mittels → Unerfahrenes Mitglied und benötigt Coaching
    - Zeit ist ebenfalls ein wichtiger Punkt:
        - Wann ist der wahrscheinliche Zeitpunkt, wann ein Problem eintreten kann
        - Wann kann eine Gegenmaßnahme frühestens begonnen werden
        - Dauer der Umsetzung der Gegenmaßnahme
        - Wie viel Zeit vergeht zwischen Umsetzung der Gegenmaßnahme und ihrer Wirksamkeit
        - t1 = Zeitspanne zwischen frühestmöglicher Umsetzungsbeginn und Problemzeitpunkt
        - t2 = Umsetzungsdauer + Wirksamkeit
        - delta_t = t2/t1
        - Wenn delta_t >= 1 bleiben Maßnahmen wirkungslos und sind damit überflüssig
        - Maßnahmen haben höchste Priorität, deren delta_t < 1 und am größten sind
        - Formel, welche Gegenmaßnahme umgesetzt werden soll: Up = Exposure * delta_t
    - Risikoverteilung anhand des Eintragens der Risiken in eine 2D-Matrix
    - Ebenfalls ist eine Risikoquellenanalyse möglich:
        - Anhand der Risikoquellen kategorisiert
            - Organisation
            - Finanzen
            - Technik
        - Kennzahlen werden kategorieabhängig addiert und Anteile errechnet
    - Historisierende Auswertung
        - Risikoquellen oder Top10-Analyse über die Zeit
        - Gegenüberstellung des Aufwands für Gegenmaßnahmen und mögliche Risikoschäden über die Zeit

Risikomatrix

- Baut auf Risikoliste und der Risikoanalyse auf
- muss ständig gepflegt werden
- 2 Elemente
    - Risikoklassen
    - Risikowahrscheinlichkeitsklassen
- Vorteil, dass auf einen Blick der Zustand des Projekts ersichtlich ist

- Risikoklassen
  - Verschiedene Typen von Risikoklassen
    - Allgemeine Risikoklassen (externe Projekte)
    - Spezifische Risikoklassen (interne Projekte)
    - Spezifische Risikoklassen (Produktentwicklung)
  - Risikoklassen beschreiben die Auswirkungen des Eintretens eines Risikos
  - Allgemeine Risikoklassen
    - typischerweise fünf
      - A: Sofortiger Abbruch
      - B: Deutliche Überschreitung des Projektbudgets
      - C: Signifikante Überschreitung des Projektbudgets
      - D: Erheblicher Mehraufwand innerhalb einer Iteration des Projekts
      - E: Gerinfügiger Mehraufwand innerhalb einer Iteration des Projekts
    - immer im Verhältnis zu sehen: Ein Projekt, welches keinen Einfluss auf den Unternehmenserfolg hat, muss auch nicht mit Risikomatrix gemonitort werden
  - Spezifische Risikoklassen (interne Projekte)
    - eher fachliches als wirtschaftliches Ergebnis im Vordergrund
  - Spezifische Risikoklassen (Produktentwicklung)
- Risikowahrscheinlichkeitsklassen
  - keine Differenzierung bei verschiedenen Projekttypen
  - anwendbare Klassen
    - 1: Eintreten des Risikos durchaus wahrscheinlich
    - 2: Eintreten des Risikos möglich
    - 3: Eintreten des Risikos nur unter bestimmten Bedingungen
    - 4: Eintreten des Risikos eher unwahrscheinlich
    - 5: Eintreten des Risikos nahezu ausgeschlossen
  - zweite Dimension zur weiteren Differenzierung sinnvoll:
    - A: kann jederzeit eintreten
    - B: nur zu bestimmten Zeitpunkten möglich
    - C: kann nur eintreten, wenn ein anderes Risiko eingetreten ist
  - Einordnung von Risiken in Risikowahrscheinlichkeitsklassen
    - Risiken innerhalb von Risikowahrscheinlichkeitsklasse 4-5 können fallengelassen werden, bei RWK 3 müssen die Bedingungen erfasst werden, RWK 1,2 müssen genau analysiert und bewertet werden
  - Bedeutungsebenen eines Projektes
    - Für Unternehmenserfolg unbedeutend

        - Für Unternehmenserfolg kaum Bedeutung
        - Für Unternehmenserfolg mittlere Auswirkung
        - Für Unternehmenserfolg signifikante Auswirkungen
        - scheitern des Projekts hätte Insolvenz des Unternehmens zur Folge
        - → Kritischer Bereich der Risikomatrix wird dementsprechend erweitert/verkleinert
        - → Risiken im spiegelverkehrten unkritischen Bereich können vernachlässigt werden
    - Beim Eintragen eines Risikos kann ebenfalls ein Trend angegeben werden (Pfeil)
    - Bei >95% der Risiken im unkritischen Bereich ist vom positiven Projektabschluss auszugehen
    - Bei >30% im kritischen Bereich muss über Maßnahmen bzw. Abbruch nachgedacht werden
    - Monitoring der Risikomatrix
        - Folgende zwei Phasen:
            - Bis zum Abschluss der Anforderungsanalyse (v.a. hinzufügen von Risiken)
                - wöchentliche Überarbeitung
            - bis zum Deployment (wenig neue Risiken, stattdessen treten Risiken ein und Maßnahmen werden eingeleitet)
                - monatliches überarbeiten
                - wenn mehr als 10% kritische Risiken sind, verkürzung auf 2 Wochen
            - Bei eintreten eines Risikos wird die Matrix überarbeitet
        - Abhängige Risiken müssen ebenfalls entfernt werden oder nun gemonitort werden
        - Bei eingetretenen Risiken muss festgehalten werden, weshalb Risiko eingetreten ist und ob dieses vorher bereits bekannt war
- Zuständigkeit für Risikomatrix
    - Vier-Augen-Prinzip berücksichtigen, um Projektblindheit zu unterbinden!
    - Projektleiter und Risikomanager legen kritischen und unkritischen Bereich fest
    - Risiken werden in Matrix eingetragen
    - Eventuell externe Hilfen in Betracht ziehen
    - Unternehmensübergreifende Aspekte
        - Auftraggeber
        - Unterauftragnehmer
        - Consultants von Werkzeug-Herstellern, die innerhalb des Projekts zum Einsatz kommen
        - Risikoprotokolle ähnlicher Projekte (und deren Teilnehmer)
        - Externe Risikomanagement-Berater
        - Je mehr Input, desto besser

        - Lieber ein Risiko zu viel in der Matrix als ein vergessenes Risiko
        - Ebenfalls sollte der Kunde integriert werden (zumindest für Risiken in der Kundenumgebung)
            - Projektrisiken sind allen bekannt
            - → Risikotransfer einfacher durchführbar
- Weitere Bereiche in denen die Risikomatrix eingesetzt werden kann
    - Auswahl von Unterauftragnehmern
    - Besetzung von Schlüsselpositionen
    - Auswahl von Werkzeugen

Risikomanagementstrategien

- Es gibt folgende Strategien
    - Risikovermeidung
    - Risikoakzeptierung
    - Risikominimierung
    - Risikotransfer
- Pro Risiko wird eine Strategie angewandt
- Risikovermeidung
    - Ziel: Wahrscheinlichkeit des Eintretens des Risikos gegen 0 & Maßnahmen für den Fall des Eintretens detailliert geplant
    - Vorgehensweise
        - Risikoidentifizierung
        - Nach Ermittlung der Auswirkungen und wenn diese gravierend sind, wird Risikovermeidung angewendet
        - Maßnahmenkatalog zur Vermeidung (auch Alternativen) und Kosten der Maßnahmen aufstellen
            - Wenn Auswirkung des Risikos sich verändern, muss ebenfalls die Strategie überdacht werden (Risikovermeidung)
            - Wenn während der Überwachung das Risiko nicht mehr auftreten kann, kann die Strategie abgebrochen werden und die Ressourcen freigegeben werden
    - Risikoschutz als Ergänzung
        - Firewall einrichten
        - Verstärktes QS
        - Spiegelung der Daten auf anderen Server
    - Politische Risiken lassen sich nicht vermeiden
    - Da sehr kostenintensiv nur in gravierenden Situationen einsetzbar
    - Als letzter Schritt bleibt der Projektrückzug

- Risikoakzeptierung
  - defensivste Strategie im RM – Mut zur Lücke
  - Ansätze
    - Risikoakzeptierung mit Notfallplanung
    - Risikoakzeptierung ohne Notfallplanung
  - Aufwändige Notfallplanung bei kleineren Projekten nicht gerechtfertigt
  - Auch in extrem ungesicherten Umfeld ist Notfallplanung oft nicht möglich
  - Vorgehensweise
    - Risikoidentifizierung und in Risikoliste eingetragen
    - Auswirkungen ermittelt und als nicht gravierend eingestuft
    - Grober Maßnahmenkatalog für den Fall, dass das Risiko eintritt
  - Auswirkungen sind also entscheidend → komplettes Projekt kaum mit dieser Strategie zu händeln
  - Strategie lässt sich nur durch sehr erfahrene Projektleiter/Risikomanager anwenden
    - wenn Auswirkungen falsch eingeschätzt werden, wird aus dem Risiko schnell eine Krise
    - auf der anderen Seite lässt sich mit der Risikoakzeptierung viel Geld sparen
- Risikominimierung
  - Auswirkung von identifizierte Risiken werden mit geeigneten Maßnahmen beim Eintreten minimiert
  - Goldener Mittelweg: nicht so teuer nicht so riskant
  - Nur unter folgenden Voraussetzungen möglich
    - Auswirkungen müssen bekannt sein
    - Terminliche Risiken sind nur dann zu minimieren, wenn bekannt ist wie die Karenzzeit ist
    - Da v.a. Mit vertraglicher Absicherung gearbeitet wird, sind kooperative Partner maßgeblich
  - Mit Vertragsklauseln
    - wie ist die Karenzzeit, also wie lang dürfen Termine überschritten werden
    - neue Technologie auf Kundenwunsch muss vom Kunden bereit gestellt werden
  - auch hier sind die Erfahrungen des Projektleiters bzw. Risikomanagers entscheidend
- Risikotransfer
  - RM-Kosten können gesenkt werden, aber Reihe an Aufwendungen bei den Vertragsverhandlungen und potentielle Streitigkeiten bei Nachweispflicht
    - Übertragung der Risiken an Auftraggeber
      - kaum erfolgreich – lediglich Kompromiss möglich
        - Wird bestimmtes Produkt durch Auftraggeber gefordert, sind die dadurch

entstehenden Risiken auf Auftraggeber zu übertragen
        - gleiches gilt für geforderten Unterauftragnehmer
        - Software-Komponenten des Auftraggebers
    - Übertragung der Risiken an Unterauftragnehmer
        - kaum tauglich, da Auftraggeber sich nicht darauf einlassen wird, dass der Auftragnehmer Risiken abwälzt
        - Ausnahmen:
            - Wenn Hersteller von Soft- oder Hardware ist
            - Wenn Komponenten des Unterauftragnehmers abgrenzbar sind, ist Risikotransfer möglich
        - Auch wenn Risiko auf Unterauftragnehmer übertragen wurde, müssen die Risiken weiterhin gewissenhaft vom Auftragnehmer überwacht werden
    - Übertragung der Risiken an Versicherungsunternehmen
        - einfach, aber kostspielig
- Weitere Möglichkeit: Interner Risikotransfer
    - auf Mitarbeiter
    - Variable Gehaltsanteil falls Meilensteine eingehalten werden oder nicht

RM in Prozessmodellen

- Prozessmodelle sollen qualitativ hochwertige Software-Entwicklung gewährleisten
    - V-Modell
        - System/Software-Erstellung
        - Projektmanagement
        - Qualitätssicherung
        - Konfigurationsmanagement
    - Hat andere Risikotypen und berücksichtigt politische Risiken nicht
    - RM als periodische Aktivität
- Rational Unified Process
    - Disziplinen
        - Geschäftsprozessmodellierung
        - Anforderungsmanagement
        - Analyse und Design
        - Implementierung
        - Test
        - Deployment
        - Konfigurations- und Änderungsmanagement

        - Projektmanagement
        - Umgebungsmanagement
    - Nur Risikoakzeptierung, Risikoübertragung und Risikovermeidung (Risikominimierung fehlt!)
    - Politische Risiken werden nicht berücksichtigt
    - Zur Risikoidentifizierung wird Brainstorming vorgeschlagen
    - Veraltete Inhalt zu RM
- Microsoft Solution Framework
    - Top5-Ursachen für das Scheitern/Gefährdung von IT-Projekten
        - mangelnde Benuzterakzeptanz, da sie in den Projektphasen nicht einbezogen werden
        - mangelnde Managementunterstützung für das Projekt
        - unerfahrene Projektmanager
        - fehlende Betriebswirtschaftliche Begründung für das Projekt
        - unzuverlässige Aufwandsabschätzung
    - MSF plant für den Erfolg und nicht Misserfolg
    - Projekt ist beschränkt durch: Buget, Zeitplan, Featureliste
      Es gilt, dass diese Beschränkungen entweder fixed, chosen oder adjustable sein können
      Team kann also aus diesen Vorgaben heraus arbeiten
    - Proaktives RM
        - Aufzählung von Risiken aus 2 Richtungen
            - Potentielle Probleme und ihre wahrscheinliche Konsequenzen
            - Potentielle Konsequenzen und ihre wahrscheinlichen Ursachen
        - Einrichtung einer Wissensdatenbank: überwundene Risiken, um
            - Risiken schneller zu identifizieren und zu bewerten
            - Gegenmaßnahmen übernehmen
            - Effektiveres managen der Risiken

Lösungsansätze zum RM

- RM muss gelebt werden und in die bisherigen Prozesse integriert werden
- Erfolgsfaktoren für die Einführung:
    - Etablieren eines formalen standardisierten Prozesses
    - Kontinuierliche Risikobetrachtung während des gesamten Projekts
    - Risikoidentifizierung als positiver Prozess
    - Risikobasierte Entscheidungsfindung
    - Einbindung aller Schlüsselpersonen, Prozesse, Geschäfts- und Technologiefelder

    - Kontinuierliche Risikokommunikation
- Einführungsstrategie
    - Top-Down (Akzeptanz eher gering)
    - Bottom-Up (Akzeptanz höher, allerdings kocht jeder Projektleiter evtl. sein eigenes Süppchen)
    - Kombination – Praxiserprobte Einführungsstrategie
    - RM kann auch outgesourct werden (hohes Know-How, aber Externe müssen sich erstmal in die Unternehmenskultur und Projekte hinein finden)

No Risk – No Fun

- Bewusste Eingehen von Risiken kann einen signifikanten Beitrag zum Erfolg haben
- Risiken ↔ Chancen
- 3 Bereiche in einem Projekt
    - sichere Bereich: auch Chancen-Part genannt
    - gefährlicher Bereich: Aktivitäten mit Risiken behaftet
    - dazwischenliegender Bereich: viele Unbekannte
- Erfahrung: Fehler sind dazu da, dass alle aus ihnen lernen, damit sie nicht wieder begangen werden
- Knowledge-Management zu Erfahrungen innerhalb von Projekten
- Faustregel:
    - Existenzbedrohend: verhindern
    - Schwerwiegend: reduzieren
    - Tragbar: Überwachen
    - Gering: Ignorieren
- Das Chancenpotential muss größer sein als das Gefahrenpotential, ansonsten liegt ein kritisches Projekt vor

IT-Risiko-Management mit System

- Projektrisiken: Folgen der Abweichung:
    - Dauer
    - Budget
    - Qualität
- Abweichungen von System-Zielen:
    - Vertraulichkeit
    - Integrität
    - Verfügbarkeit der Informationen/Systeme
- Bedrohung → Ziel-Abweichung

- Wenn keine Gegenmaßnahme vorhanden ist → Schwachstelle
- Ziel-Abweichung → Schäden (z.B. Reputationsschaden)
- Keine Ziel-Abweichung → keine Schäden → „sicher“!
- Subjektivität der Risiko-Einschätzung
  - Ex Post: Aus Erfahrungen heraus
  - Ex Ante: Prognose
  - Am Besten durch ein Team
- Schadenseinstufung
  - kardinal: rechenbar also EUR
  - ordinal: (klein, mittel, groß)
- Risikokarte
  - 2 Dimensionen: Häufigkeit & Auswirkung
  - Akzeptanz-Linie
- Risikoanalyse
  - Risikoidentifizierung
    - möglichst vollständige Erfassung von Gefahrenquellen und Risikoquellen
  - Risikoeinschätzung
    - Einschätzung von Wahrscheinlichkeiten und Schäden
  - Teilanalyse
    - Impact-Analyse
    - Bedrohungsanalyse: Relevante Bedrohungen für ein Objekt
    - Schwächen-Analyse: Schwachstellen eines Objekts
    - Kombination der bisher genannten
  - 2 Dringlichkeitskategorien bei Schwachstellen
    - Vulnaribility (potentielle Verletzlichkeit)
    - Exploit (wurde in der Praxis bereits ausgenutzt)
    - Schritte in der Risikoanalyse
      - 1.: Bildung von Risiko-Objekten und Abgrenzung des für die Risiko-Analyse relevanten Bereichs
        - bspw. Prozesse, System, Informationen, Personen
      - 2.: Identifikation der für die Objekte massgeblichen Bedrohungen und Schwächen
        - pro Risikoart werden Bedrohungen (Bedrohungslisten) identifiziert und Schwächen den Objekten zugeordnet (Nichteinhaltung von Sicherheitsstandards, Vorschriften, Regulationen und Gesetze)
      - 3.: Analyse der Bedrohungen auf die Objekte und Einschätzung der Haufigkeit

des Eintritts eines Schadens

- 4.: Einschätzung des voraussichtlichen Schäden
  - wenn mehrere Systen-Ziele müssen Schäden pro System-Ziel geschätzt werden
- 5.: Bestimmung der Risiken eines Objekts

- Schätzung des Risiko
  - Max. mögliche Werte für Schäden
  - Mittlere Werte
  - Risiko-Werte anhand statistischer Verteilungsfunktion
- Analyse-Methoden
  - Bottom-Up: Aufgrund von Ursachen werden mögliche Schadenseregnisse und deren Konsequenzen hergeleitet und die Risiken eingeschätzt
  - Top-Down: Schadensereignisse werden als Risiken angesehen
- Such-Methoden (Bottom-Up: Welche Auswirkungen hat ein Fehler?
  Top-Down: Welche Ursachen kann eine beobachtete Auswirkung haben?)
- Szenarien-Analyse (Szenario ist die Beschreibung des Übergangs von einer Bedrohung zu einem Risiko; Also WENN-DANN-Regeln)

SSL & HTTPS

SSL

- Besteht aus zwei Schichten
  1. SSL Handshake, SSL Change Cipher Spec, SSL Alert, SSL Application Data
  2. SSL Record
- SSL Record Protocol
  - Absicherung der Verbindung
  - 2 Dienste
    - Ende zu Ende-Verschlüsselung mit symetrischen Schlüsseln (DES, Triple DES, AES)
    - Bildung einer kryptografischen Prüfsumme zur Sicherstellung der Integrität und Authentizität
- SSL Handshake Protokoll
  - Bietet folgende Funktionen (noch bevor die ersten Bits des Anwendungsstroms ausgetauscht werden)
    - Identifikation und Authentifizierung der Kommunikationspartner auf Basis von asymetrischen Kryptoverfahren
    - Aushandeln der Verschlüsselungsalgorithmen und Schlüssel (TLS kann man auch ohne Schlüssel verwenden)

- Handshake kann in 4 Phasen unterteilt werden
    1. Client schickt an Server ein client_hello
        - Vom client untersützte, höchste SSL-Version
        - 32 Byte Zufallsinformation (Timestamp + Random): wird später zur Bildung des pre-master secrets verwendet (Schützt vor Replay Attacken)
        - Session-ID
        - zu verwendete Cipher Suite
    2. Server identifiziert sich gegenüber Client mit Zertifikat und Server fordert ggf. Client-Zertifikat an.
    3. Client identifiziert sich ggf. gegenüber Server. Zertifikat des Servers wird verifiziert. Server-Zertifikat enthält öffentlichen Schlüssel des Servers.
       Wenn RSA verwendet: Pre-Master Secret wird mit dem öffentlichen Schlüssel an Server übermittelt
       Sonst: Diffie-Hellman zum Generieren des pre-master secrets
    4. Abschluss des Handshakes: Aus pre-master secret wird master secret und daraus wiederum der session key abgeleitet
- SSL Change Cipher Spec Protocol
    - 1 Nachricht, 1 Byte groß, Mitteilung des Senders an Empfänger, dass er zur ausgehandelten Cipher Suite wechselt
- SSL Alert Protocol
    - Fehler/Warnhinweise des Protokolls
- SSL Application Data Protocol
    - Anwendungsdaten werden in Teile zerlegt, komprimiert und verschlüsselt

HTTPS

- Ziel: Verschlüsselung und Authentifizierung (gegen Phishing-Angriffe) im WWW
- Varianten der HTTPS-Anwahl
    - Serverseitig nur HTTPS: Onlinebanking
    - LogIn über HTTPS, danach weiter mit gesetztem Cookie
    - Sowohl http wie auch https
    - Browser-Plugin, welches automatisch auf https-Seite umleitet
- Vorinstallierte Zertifikate sind durch Root-Zertifikate abgeleitet und werden vom Browser problemlos akzeptiert
- Zertifikat wird von Zertifizierungsstelle ausgestellt
- Selbst ausgestellte Zertifikate sind verwundbar gegen Man-In-The-Middle-Attacken, da keine Authentifizierung durchgeführt wurde
- Ausstellung eines Zertifikats, wenn Admin erreichbar: Neues erweitertes Zertifikat, wenn Postadresse stimmt und ob Person zeichnungsberechtigt für die Firma ist
- Schwachstellen

- HTTPS verhindert nicht Man-In-The-Middle Attacken, wenn kein Zertifikat verwendet wird
- Automatisches Bestätigen von HTTPS-Verbindungen wird nicht mehr bewusst durch User eingegangen (→ Phishing)
- HTTPS/HTTP Mischbetrieb führt zu Sicherheitsrisiken (z.B. Session Hijacking)