Allgemein

- Confidentiality (Vertraulichkeit),Integrity (Integrität),Availability (Verfügbarkeit)
- Komplexität von SOAP/REST-Webservice lässt sich anhand von WSDL/WADL Files erkennen.
- Wenn die Anwendung dazugebracht wird, mein XML-File mit Entitys an den XML-Parser weitergeschickt/abgearbeitet wird => XXE(XML external Entity Injection.
- Billion laughs Attake ist ein DDos Angriff , man schickt verschachtelte Entitys mit LOL's die der Parser auspackt =>exponentielles Wachstum => BÄÄM!
- Bei Clickjacking wird eine Webseite so präperiert, das der Nutzer nicht sieht worauf er tatsächlich klickt. (Bsp.: 2 Frames übereinander, man klickt auf email senden, es wird aber code ausgeführt der eine Zahlung veranlasst.)
- SQL-Injection (UNION, Mächtigkeit der SELECTs) - -'; ((- -) OR # in MYSQL)
- Bool'sche Muster
- http-only und secure flag in Session-Id Cookie setzen!
- neue Session-ID bei login (Zustandsänderung = Änderung in der Session ID)
- indexing auf Sererseite Abschalten
- Alogrithmus zum ver/entschlüsseln muss auch in Hardware langsam sein (FPGA´s)

OWASP Top 10

A1 Injection

Hier geht es um einschleusen von Code über Eingabefelder. Meist wird ein zusätzliches Kommando dazu genutzt, um Daten vom Server zu lesen, schreiben oder zu verändern ohne das dies von der Anwendung kontrolliert wird. Unterarten von Injection

- SQL
- XML
- shell

Kann mit Escapen/ prepared Statements bekämpft werden

- "Not so blind": Sprechend, d.h. mit Fehlermeldungen der DB
- Blind-Injectinon: Allgemeine, leicht unterschiedliche Fehlermeldungen
- Totally-Blind-Injection: Immer identische Meldung => Messung von Laufzeitunterschieden, erzwingen von Fehlern (division-by-zero)
- Advanced: alles komplexere, hier umgehung Boundary Filtering, z.B. durch encoding der Daten;

A2 Fehler in Authentifizierungs und Session-Management

- Session-Management und ID sind falsch implementiert,
- Session hijacking,
- ID berechenbar,
- Passwörter nicht gehasht,
- SessionID läuft nicht ab,
- keine Transportverschlüsselung

A3 Cross-Site Scripting

- JavaScript Injection: von Benutzer/Angreifer eingegebener JS-Code wird unescaped an Browser weitergeleitet
- Die vom User in den Browser eingegebenen Daten werde nicht validiert bzw. die Daten die an den Server Server geschickt werden.
- Reflection(ist nur einmalig, bleibt nicht in der DB)
- Persistent (wird gespeichert und jeder der die Seite aufruft wird injected)
- Hier hilft meist escapen und testen der Anwendung (manuelle pentest, reviews usw.) und indirekte Objektreferenz: z.B. Index auf Liste der Konten des Kunden

A4 Unsichere direkte Objektreferenzen

- ID 1 im Browser = ID 1 in der DB => erratbar
- Zugriff muss auch auf Ressourcen Ebene vom Server überprüft werden (id 1 darf nur daten von id 1 sehen)

A5 Sicherheitsrelevante Fehlkonfiguration

- veraltete Softwarekomponenten
- nicht benötigte Komponenten aktiv oder installiert
- Standardkonten mit initial PW's aktiv
- Fehlermeldungen, Stack Traces geben zuviel Informationen über das System raus
- Framework Einstellung sind nicht sicher,

A6 Verlust der Vertraulichkeit sensibler Daten

Data in Motion(Daten im Arbeitsspeicher), Data at Rest (im Backup)

- Daten werden in Klartext gespeichert
- Daten in Klartext übertragen
- schwache/alte Krypto Verfahren
- schwache Schlüssel oder falsches Verwalten der Schlüssel
- Sicherheitsdirektiven und Header werden nicht genutzt

A7 Fehlerhafte Autorisierung auf Anwendungsebene

- Links zu Funktionen werden nur ausgeblendet und dann werden die Rechte nicht vom Server überprüft (Security by obscurity)
- serverseitige Prüfung von Authentisierung und Autorisierung wird nicht durchgeführt
- serverseitige Prüfung nur mit Daten vom Anwender

A8 Cross-Site Request Forgery

- geheimer Token bei jeder Anfrage/Link/Formular wird nicht mitgeschickt
- Dem User wird meistens ein Request untergeschoben womit ohne Benutzereingabe was gemacht wird
- Bsp.: Request wird in einem HTML-Objekt versteckt (z.B. IMG) , User geht auf die Seite, während er noch die Seite offen hat die den Request entgegen nimmt => Request wird abgeschickt ohne das der Nutzer es weiß

A9 Nutzung von Komponenten mit bekannten Schwachstellen

Ein oder mehrere kleine oder auch große Lücken (auch hintereinander in unterschiedlichen Programmen) können ausgenutzt werden um an die Server/Daten zu kommen

Verzeichnisse: cve, nvd

A 10 Ungeprüfte Um- und Weiterleitungen

- Umleiten sollte vermieden werden
- Benutzer kann auf Angreiferwebseite weiter geleitet werden (Phising)
- Benutzer informieren wenn er umgeleitet wird

Allgemein

- Regel 1: Dont underestimate the power of the dark Side
- Regel 2: Benutze Post anfragen wenn Seiteneffekte auftreten.
- Regel 3: in einem serverseitigen Kontext gibt es keine clientseitige Sicherheit!
- Regel 4: Benutze nie den Referer Header zur Authentifizierung oder Autorisierung
- Regel 5: Generiere immer eine neue Session ID, wenn sich der Benutzer anmeldet.
- Regel 6: Gebe nie ausführliche Fehlermeldungen an den Client weiter.
- Regel 7: Identifiziere jedes Zeichen, das in einem Subsystem als Metazeichen gilt.
- Regel 8: Behandel jedes Mal die Metazeichen, wenn Daten an Subsysteme weitergegeben werden.
- Regel 8: Übergebe, soweit möglich, Daten getrennt von Steuerinformationen.
- Regel 10: Achte auf mehrschichtige Interpretation.
- Regel 11: Strebe gestaffelte Abwehr an.
- Regel 12: Vertraue nie blind einer API Dokumentation .
- Regel 13: Identifiziere alle Quellen, aus denen Eingaben in die Anwendung gelangen.
- Regel 14: Achte auf die unsichtbare Sicherheitsbarriere.
- Regel 15: Wihtelisten anstatt Blacklisting
- Regel 23: Erfinde keine eigenen kryptographische Algorithmen, sondern halte dich an die existierenden.
- Regel 24: Speichere Passwörter nie als Klartext.
- Regel 25:
- Regel (Trommler): Implementiere keine kryptographische Algorithmen, benutze existierende Bibliotheken.

0.1 sonstiges

OWASP: Open Web Application Security Project

§202 a-d: (Vorbereiten von) Ausspähen/Abfangen von Daten + Datenhehlerei

Cookie stehlen: document.cookie

Terminal - Einfache, wirtschaftlich sichere Unterschriften

Motivation: PC ist kompromittiert

Stand der Technik: TruPoSign: zeigt Komplexe Datenformate

ZTIC: USB-Stick mit SSL/Proxy, Bankspezifisch

Secoder: Class 3 Terminal

Wirtschaftlich sicher: Angreifer gewinnorientiert, nicht Vandale

Schutz finanziell relevanter Daten: extrahieren, Kunde verifiziert PlainText

Erweitertes FinTS: User Defined Signature: String + Signatur

Verifikation: Signatur vs Text, Text vs Transaktion

=> Keine Versionsupdates, string kann aus Umgebung kommen, wird verifiziert => Vereinfachte Signaturumgebung: string lesen, anzeigen, bestätigen und auf Karte signieren, zurückgeben

Nürnberger Sicherheitstrichter: Angriffsoberfläche verringern

Defence in Depth

Feingranulare Zugriffskontrolle in der Datenbank anhand Projekt

Zugriffskontrolle anhand Kategorien (Teilnehmer, Pflegedienst, etc)

Problem: Kontrolle über ganze Anwendung verteilt. => schwierig(Vollständig, Korrekt, Verständnis) => von anderem Code abhängig

Ansatz

trennen von Code

Anforderungen: Zentrale Datenbank, keine Zusatzprogramme(WAF), geringe Änderungen im Programm

Zugriffsmodell anhand: eigener Daten, vergangene Aktionen, Regeln

Spalten u. Zeilen beschränken

Vom Datenbankinhalt abhängig

Befehlsaufbau

Anchor TabA.TabAId = Ext

TabA -> TabB VIA TabB.TabAID

TabB: Keyword (readonly ...)

Implementierung

Parametrized Views => Abhängig von z.B. SessionID

(User Defined Functions, Query Rewriting, Oracle Virtual Private Database)

=> API um Session Parameter erweitern => Applikation ändern

=> Alternative: vor und nach der Appl. Nachrichten abfangen, zusammenhang herstellen => Query Rewriting

Internet Architektur und E-Commerce

Sicherheit nicht im Design des Internet

- Fehlen der Theorie
- Geschlossenes Militärnetz
- Schwerpunkt: Ausfallsicherheit

Architektur

- Austausch von Forschungsergebnissen
- Vernetzung von Wissen
- Webserver liefert statische Dokumente, die vom Browser dargestellt werden

Angriffspunkte

- Server hacken
- Nachrichten fälschen
- Device vom User hackenn

Ziele der Security für E-Commerce

- Daten schützen
- Fehler vermeiden
- Fehler erkennen und beseitigen
- Daten schützen
- Fehler vermeiden
- Auswirkung der Fehler begrenzen

Funktionen für E-Commerce

- Dynamische Inhalte
- Zustand (mehere Anfragen pro kaufvorgang, suchen, auswählen, bezahlen)
- Erweiterung des Web Servers (CGI)
- nach Appliationserver (Java Servlets, Java Server Pages, Active Server Pages)

Programmierfehler

Software Qualität

- Definition SSicher"?
- Hoher Zeitaufwand System zu brechen
- Erfüllt die Sicherheitsspezifikation
- Flaw (Unerwartetes Verhalten), Vulnerability (Klasse von Flaws, zb. Buffer Overflow)

Fehlerklassen

- Pufferüberlauf
- Unvollständige Entkopplung (30.2 eingeben können)
- Serialisierungsfehler (time-of-check to time-of-use)

Ort des Fehlers

Standardsoftware

- Gefundene Fehler publiziert
- Anbieter bietet Lösung an
- Aber evtl. auch Exploit verfügbar

Eigene Webanwendung

- Legitime Benutzer finden selten Fehler
- Anzahl Hacker geringer
- Exploits unwahrscheinlich

Pufferüberlauf

- 2L Wasser in 1L-Eimer
- Example in C : Char sample[3]; Sample[3] = 'a';
- Nicht entscheidbar

Folgen des Überlaufs, Überschreiben von:

- Programmdaten
- Betriebssytemdaten
- Betriebssytemcode
- Programmcode
- Rechnen mit falschen Daten
- Ausführen unerwünschter Instruktionen

Identifikation eines Überlaufs

- Vermutung der Existenz eines Puffers (Konsistenzprüfung, Signaturen)
- Vermutung über die Länge des Puffers (Telefonnummer, Postleitzahl)
- Zufall (Absturz bei bestimmten Eingabedaten)

Überlauf und Sicherheit

- Absturz (System oder Programm)
- Code mit erweiterten Rechten (Priviligen im OS oder Server Anwendung)

Unvollständige Entkopplung

- Benutzereingaben nicht validiert (30.2, Auswahlmenüs, Format und Wertebereichsprüfung)

=> Folgen

- Absturz (Ausfall eines Handelsystems)
- Manipulation an Systemen (SQL oder Shell Injection)