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

Internet Architektur und E-Commerce

Angriffspunkte

- Server hacken
- Nachrichten fälschen
- Device vom User hackenn

Ziele der Security für E-Commerce

- 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)

Programmierfehler

Software Qualität

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

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)

Serialisierung

Serialisierung in E-Commerce

- Links auf Bereiche der Anwendung:
- Beim Einloggen geprüft
- Nacher nicht mehr
- Versteckte Felder
- Links mit vorbelegten Parametern
- Fehlannahmen (HTML-Seite nicht manipulierbar (loool), Versteckte Felder unsichtbar (firebug usw)

=> Folgen :

- Manipulation an Transaktionen (Preis in Url)
- Unerwünschter Zugriff auf beschränkte Funktionen/ Daten anderer Kunden

Einschleusen von Kommandos

SQL Injection

Gegenmaßnahmen - Daten säubern

- Anführungszeichen verbieten oder Anführungszeichen verdoppeln
- PostgeSQL und MySQL: Backslash
- Zahlen durch Parser der Programmiersprache jagen, danach als Zahl behandeln
- Stored Procedures helfen nicht !!

Gegenmaßnahmen - Prepared Statements

- Anweisung ist bereits teilwese verarbeitet
- Daten haben keine Metazeichen mehr
- Typ-Prüfung durch Programmiersprache umwandeln

Shell Injection

- Viele Metazeichen
- Eingabe: test; rm -rf /

Gegenmaßnahmen - Daten säubern

- Kommandos in Quellcode selbst verankern, keine Übergabe von Kommandos
- C/C++ (Exec meist nach fork, nicht system oder popen)
- Benutzerdaten über Standardeingabe, aber Zielprogramm interpretiert Daten
- Metazeichen und Eingabevalidierung
- Escapen in shell schwierig
- Shell nicht verwenden
- Externes Programm nicht verwenden(Wiederspricht re-use,)
- Codefragmente wiederverwenden aber durchsehen und HInweise lesen

interaktion mit c/c++

- Zeichenkette endet mit null-Zeichen (Terminiert Zeichenkette in c)
- Kunden laden Bilder auf Server (Prüfe auf Endung .jpg) Eingabe : Crack.phpNULL.jpg

Evil Eval

- Werte den Inhalt einer Variable aus
- Eval in den meisten Skript Sprachen
- Java Reflection API
- Enthält Variable Benutzereingaben (Kommando Injection)

Metazeichenbehandlung

- Identifiziere Metazeichen im System und allen Subsystemen
- Typen Prüfung
- Injection zweiter Ordnung, O'Conner aus DB lesen, In neue SQL Abfrage eingebaut
- Like Klausel in SQL (Metazeichen %_)

Architektur

- soll beim Fehler vermeiden helfen
- Kommunikation Kapseln (Metazeichen an einer Stelle behandeln)
- Datenbanken persistents Schicht (Hibernate, Java Beans)

Gestaffelte Abwehr

- Defense in the Depth
- Mehrere Sicherheitsmechanismen
- Berechtigung im Subsystem (DB)
- Eingabe Validierung (Format E-Mail, Zahlen, Telefonnummer, Postleitzahlen, Bereichsprüfung, Längenprüfung)
- Unnötige Dienste in der Infrastruktur abschalten
- Redundanz bei Sicherheit wichtig (Validierung in jedem System)

Arten von Eingaben

- URL-Parameter
- Benutzer erzeugte Eingaben (Textfelder/ Textarea)
- Server erzeugte Eingaben (Auswahlfeldern, Checkboxen, Radiobutton, Verborgene Felder)

Parameter Manipulation

- URL selbst erzeugen (Get vs Post)
- Http Header Felder (Cookies)
- Dateisystem (Weboberfläche NAS)
- zusätzliche Parameter
- Server Variablen überschreiben

Folgen: Bei Benutzer erzeugten ungültigen Eingaben (Tippfehler) => höflicher Hinweis, neuer Versuch. Bei Server erzeugten ungültigen Eingaben (absichtlich manipuliert) => Vorgang abbrechen, Protokollierung.

Filtern

- Gut, Schlecht oder unbekannt
- Blacklisting (Filtere schlechte Eingaben)
- Whitelisting (Filtere gute Eingaben)

User Authentication

Anforderungen

- Benutzer eindeutig identifizieren
- Überall verfügbar
- Einfach anwendbar

Passwort

- Benutzer besitzt Passwort (PIN), entweder selbst gewählt oder vom Anbieter zugeteilt
- Authentisierung : Benutzer gibt seine ID und Passwort ein , Server vergleicht mit Datenbank

Loose-Lipped System

- Authentication mit ID und Passwort, erst ID abfragen und ungültige ID abweisen => Angreifer erhält Informationen über gültige ID
- daher immer (ID, Passwort) Paare prüfen => Angreifer weiß nicht ob Passwort oder ID falsch

Angriffe gegen Passwort

- Probiere alle Passwörter (Brute-Force)
- Probiere viele wahrscheinliche Passwörter (Dictionarys)
- Probiere Passwörter, die für den Benutzer wahrscheinlich sind
- Suche die Liste der Passwörter
- Frage den Benutzer
- Passwörter werden wiederverwendet
- Passwort Muster ... (OHM-geheim. Mail-geheim ist doof => Verrät Passwort für andere ID's / Seiten)

Einmal-Passwort

- Benutzer besitzt Passwortliste
- Streicht verwendete PW
- Streichen vergessen
- Frage nach Passwort oder neues PW nach Login

Chipkarte

- Karte speichert (Symmetrischen Schlüssel,Privaten Schlüssel)
- Karte führt Verschlüsselung durch
- Nachteil: Verfügbarkeit in Standard Client nicht gegeben

Biometrische Verfahren

- Messe ein Körpermerkmal
- Gelten als sehr sicher
- False acceptance rate, False rejection rate
- Sensor täuschen, umgehen
- Im Internet ungeeignet weil Scanner bei Client

Klassifikation

- Was man weiß: Passwort
- Was man hat: Passwortliste, Chipkarte
- Was man ist: Fingerabdruck, Ohr, Retina

Kombinationen der Klassen

- Wissen und Haben (Online Banking) => Passwort und Tan Oder Passwort und SecurID
- Haben und Sein => Chipkarte und Fingerabdruck

Authentication in HTTP

- Basic Authentication (Passwort im Klartext, Ok wenn über TLS)
- Digest Authentication (Passwort plus Challenge mit Hash)a

Viren, Würmer und Trojanische Pferde

Virus (Definition)

- Programm (-Stück)
- ÏnfiziertP̈rogramm
- Verbreitung über infizierte Programme
- Makroviren betreffen auch "Daten"
- Meist Schadensfunktion (Dateien löschen, Platten formatieren/verschlüsseln, Konfiguration manipulieren)

Infektionsmechanismen

- Wirtsprogramm (Virus hängt/umfasst oder integriert sich an/in das Programm)
- Andere Infektionsorte (Boot-Sector, Speicher (RAM), Programm Bibliotheken)

Viren Scanner

- Allgemeines Virenproblem unentscheidbar
- Erkennen anhand von Mustern
- Wirkt nur gegen bekannte Viren
- Ständige Aktualisierung (mehrmals täglich, automatisiert)

Würmer (Definition)

- Verbreitung über Netze (Internet)
- Ziel: alle Rechner im Netz infizieren
- Meist Schadensfunktion

Wurm Schadensfunktion

- Wie bei Viren
- Zombies für DDOS
- Zugang zu Rechnern öffnen und offen halten
- Nutzt Schwachstellen in Diensten (Programmierfehler, Fehlkonfiguration)

Code Red und Varianten

- Programmierfehler in Microsoft IIS (Beliebiger Code ausführbar)
- Rasche Verbreitung via HTTP
- Schäden(Startseite ersetzt(Defacement), Hohe Netzlast, DDOS gegen Whitehouse, Nimda (Hintertür mit Admin-Rechte)

Schutz

- Schwachstelle beseitigt (Patch) (schwierig auf Produktivsystemen)
- Konfiguration regelmäßig prüfen
- Nur Notwendige Dienste aktiv
- Firewall als Unterstützung

Trojanisches Pferd (Defenition)

- Programm mit zweiter Funktion (versteckt)
- Benutzer verwendet Programm freiwillig
- Schäden (gefälschter Login Dialog, Keylogger)

Weitere Formen

- Logische Bombe(wird bei bestimmter Bedingung aktiviert)
- Zeitbombe
- Hintertür
- Fork Bombe

Im Web

- Phising (gefälschte Seite, vorbereitete Transaktion im Link, Formular automatisch Absenden)
- Cross-Site-Request Forgery (XSRF), Sitzung in anderem Fenster, Kommandos an andere Programme

Lösung

- Ticket für Aktionen(Zufallszahlen, Unterscheidung durch ID, Hidden Fields)

Grenzen

- GET, Cross-Side-Scripting

Weitere Fortschritte

- Web Service
- AJAX
- Web Sockets

Public Key Infrastructure (PKI)

Ziel

Ziel => sichere Verteilung der Public Keys mit Zertifikaten

Allgemeines zu PKI

- Es gibt unterscheidliche Vertrauensmodelle, alle haben Ihre Stärken und Schwächen
- Grundsatzfrage : wer vertraut wem und wie vertrauenswürdig ist derjenige dem ich vertraue?
- Zertifikate sind Bestätigungen wer wer ist
- Verfallsdatum von Zertifikaten sinnvoll
- Ziel ist es, eine stabile, kaum angreifbare Vertrauenskette zu erschaffen!

Vertrauensmodelle

- Monopol (Vorteil : Theoretisch einfach, Nachteil: Wem vertrauen alle, Schlüssel verlust)
- Registrierungsstellen (Vorteil: Zertifizierungstelle muss nicht Identität prüfen , Nachteil: Ist sicherheitskritisch) VS. Delegation (Vorteil: Unterschreiben die Untergeordneten Stellen, unter Zertifikate, Nachteil: Sicherheitskritisch)
- Oligarchie (Vorteil: mehrere CAs , Nachteil: eine CA kompromittiert dann sind alle Komprimittiert (meltdown))
- Anarchie (Vorteil: Dezentrale Lösung , Nachteil: transitivität des Vertrauens)
- Name Constraints (Vorteil: Zertifikate zweckgebunden, Nachteil: keine bekannt)
- Top-Down (wie Monopol)
- Button-Up (Vorteil: , Nachteil:)

Firewalls

Definition

- Internes, geschütztes Netz
- Externes, weniger vertrauenswürdiges Netz
- Firewall(FW) filtert Kommunikation zwischen internem und externen Netz
- Ziele : Schutz internen Netzes vor externen Angriffen und Kontrolle Kommunikation zu externem Netz

Kernidee

Referenzmonitor:

-
- Immer aufgerufen (einzige Verbindung zu externe via FWs)
- Manipulationssicher
- Klein und einfach genug für Verifikation

Paketfilter

- Filtert Quell und Zieladresse (auch auf Port)
- Filtert manipulierte interne Adresse
- Einfache Regeln
- Aufwand für komplexe Policies (Viele Regeln, SSH für Admins)

Stateful inspection

- Betrachte Vorgeschichte zu Paket
- Merke Zustand (State, context)
- Erkennt fragmentierte Angriffe (TCP Paketreihenfolge)

Applikation Proxies

- Bastion Host
- Simuliert Anwendung
- Filtert auch auf Paketinhalt (zb nur Http POST / GET)
- Schutz für Standard Anwendungen
- für Web Anwendungen begrenzt geeignet

Guards

- Application Proxy
- Und Anfragen ändern
- Regeln beliebig flexibel
- Virenscan auf eingehenden Dateien
- Web Application Firewall

Applications-Firewall

- Selbstlernend (Parsing HTML-Seiten, Identifkation möglicher Antworten, Filter unmöglicher Antworten)
- Implementation: Reverse Proxy
- Grenzen: Valierung der Benutzereingabe, Performance

Personal Firewalls

- Schutz einzelner Rechner
- Programm auf dem PC
- Definition der Policy (Anwender?)
- Angreifer umgeht PFW
- U-turn Attack bei mobilen Benutzer

Möglichkeiten und Grenzen

- FW als einziger Netzübergang
- Kein Schutz nach Verlassen der FW
- FW als Ziel der Angreifer
- FW Konfiguration periodisch prüfen
- Geringe Kontrolle über Inhalte
- Firewalls alleine machen eine Infrastruktur nicht sicher!

Fortschritte

- VPN, IDS etc. auf der FW
- Firewall-Tunneling (HTTP, SSH,. . .) , SOAP, Websocktes
- Webservices
- Moderne Firewalls != Referenzmonitor
- Komplex nicht einfach
- Getunnelt nicht immer aufgerufen
- Internetsteuerbar nicht Manipulationssicher

Chipkarten und digitale Unterschriften

Arten

- Speicherkarten
- Mikroprozessorkarten
- Kontaktlose Chipkarten (RF-ID)
- Formate (Scheckkartenformat, SIM-Karte, USB-Token)

Sicherheit

Gefährdung: (Herausgabe/Veränderung von Informationen) Schutz :

- Mechanisch (Kartenkörper)
- Elektrisch (Halbleiter),
- Logisch (Betriebssystem)

Passive Hardware => Alles auf einem Chip, Anordnung der Leitungen auf dem Chip Aktive Hardware:

- Deaktivierung bei Beschädigung
- Spannungsüberwachung
- Untertaktung verhindern

Betriebssystem

- Kontrolle I/O
- Verwirft falsche Protokollnachrichten
- K̈onstanter S̈tromverbrauch
- Zugriffskontrolle auf Daten (File System)
- Karte permanent deaktivierbar

Anwendung: Digitale Unterschrift

- Digitale Unterschrift: Hash des Dokumentes, Verschlüsselung mit PK
- Karte enthält Private Key, führt Verschlüsselung aus, Hash auf dem PC, Schlüsselpaar auf Karte generieren
- "Konstanter SStromverbrauch
- Zugriffskontrolle auf Daten (File System)
- Karte permanent deaktivierbar

Willenserklärung

- Kunde muss Dokument verstehen
- Rechner soll Dokument verarbeiten
- Kunde muss alles sehen können(Kleingedrucktes, Farben)

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