Securitate Software

IX Vulnerabilități web

SQL Injection, Session Hijacking

Obiective

- prezentarea aspectelor teoretice din spatele vulnerabilităților Web comune
- prezentarea vulnerabilităților
  - SQL injection
  - session hijaking

Continut

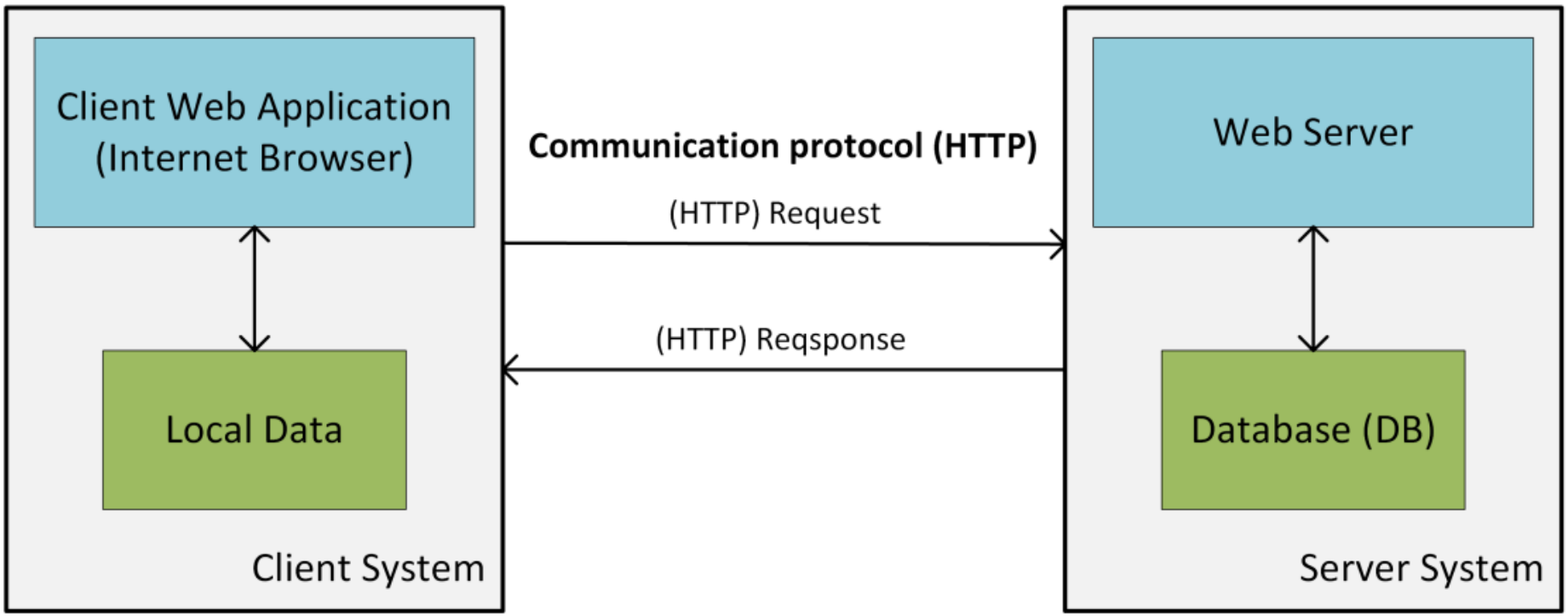
Context

- Aplicațiile web sunt implementate (de obicei) în limbaje sigure
  - $\Longrightarrow$ în general imune la vulnerabilități legate de coruperea memoriei
- vulnerabilități specifice web
  - SQL injection
  - XSS (Cross-Site Scripting)
  - XSFR/CSFR (Cross-Site Forgery Request)
- totuși cauze comune pentru vulnerabilități
  - nevalidarea corespunzătoare a datelor primite de la utilizator
  - $\Longrightarrow$ confuzie între cod și date

Arhitectura aplicațiilor web

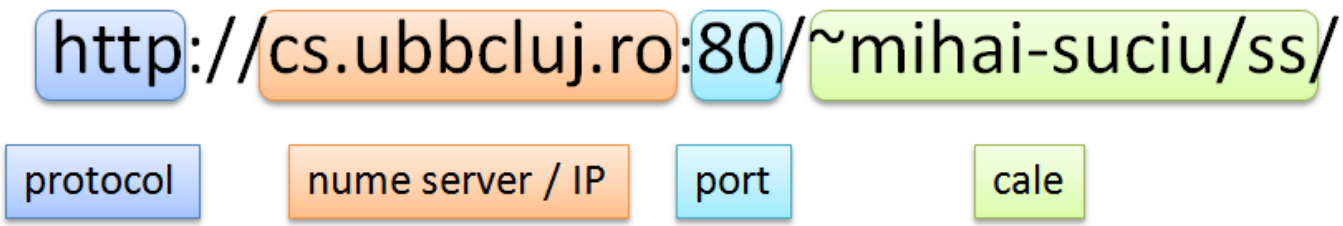
- bazate pe modelul **client-server**
- serverul
  - Apache, MS IIS
  - de obicei se **conectează la o bază de date**
    - locală
    - la distanță
- clientul
  - **client web** (browser), ex. Chrome, Firefox, MS IE, etc.
  - poate **menține local date private**
    - intern (ex. cookies)
    - extern (fișiere)
- **comunicarea** bazată pe un protocol (ex **HTTP**)

Arhitectura client-server pentru o aplicație web



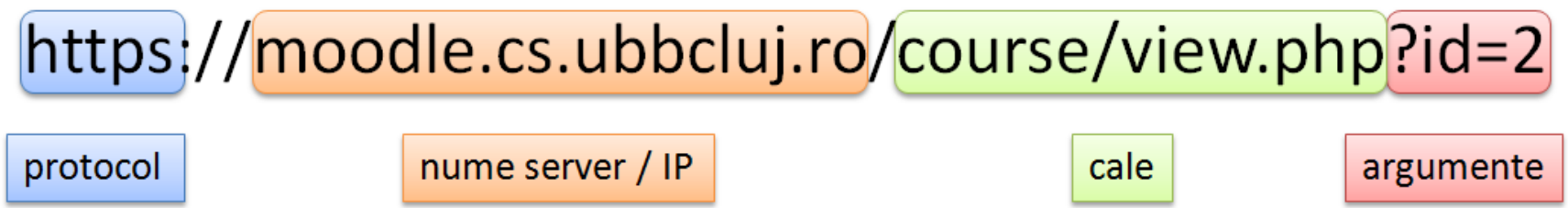
Resurse web (pagini web) - identificare

- Universal Resources Locator (URL)
- **protocol**: HTTP, HTTPS, FTP
- **nume server** (IP)
- **port** (implicit 80 / 443)
- **cale resursa** (pagină)



Pagini web - tipuri

- conținut **static**
  - același conținut pentru fiecare acces
  - extensii uzuale: html, htm
  - URL: *http://www.cs.ubbcluj.ro/~mihai-suciu/index.htm*
- conținut **dinamic**
  - depinde de context
  - **generat prin rularea de cod pe server**
  - de obicei se obțin date suplimentare dintr-o **baza de date (DB)**
  - URL: *https://moodle.cs.ubbcluj.ro/course/view.php?id=2*
  - extensii uzuale: php, jsp, asp
  - componente suplimentare în URL



Protocolul HTTP

- HTTP = HyperText Transfer Protocol
- protocol pe nivelul aplicație (OSI)
- construit peste TCP/IP $\Longrightarrow$ de încredere
- bazat pe schimb de cereri-răspunsuri

Cerere HTTP (HTTP request)

- conținut
  - URL
  - header
- tip
  - GET
    - nu există date suplimentare decât adresa URL
    - nici un efect secundar asupra serverului
  - POST
    - câmpuri suplimentare
    - poate avea efecte secundare

Headere cerere HTTP GET

- URL: http://cs.ubbcluj.ro/~mihai-suciu/index.htm

```
GET / HTTP/1.1
Host: www.cs.ubbcluj.ro
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
```

- urmarește un link URL

```
GET /~mihai-suciu/ss/index.html HTTP/1.1
Host: www.cs.ubbcluj.ro
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 F
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.cs.ubbcluj.ro/~mihai-suciu/index.htm
Connection: close
Upgrade-Insecure-Requests: 1
```

Headere cerere HTTP POST

```
POST /login/index.php HTTP/1.1
Host: moodle.cs.ubbcluj.ro
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://moodle.cs.ubbcluj.ro/
Cookie: _ga=GA1.2.202198079.1513077342; _gid=GA1.2.2137786236.1513077342; _gat=1; MoodleSess
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 38

username=moodle_user&password=1234
```

Răspuns HTTP, conținut

- status code
- headers
- data
- cookies

Răspuns HTTP - exemple

```
HTTP/1.1 200 OK
Date: Tue, 12 Dec 2017 11:22:44 GMT
Server: Apache/2.4.18 (Ubuntu)
Expires: Mon, 20 Aug 1969 09:23:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Language: en
Content-Script-Type: text/javascript
Content-Style-Type: text/css
X-UA-Compatible: IE=edge
Cache-Control: post-check=0, pre-check=0, no-transform
Last-Modified: Tue, 12 Dec 2017 11:22:44 GMT
Accept-Ranges: none
X-Frame-Options: sameorigin
Vary: Accept-Encoding
Content-Length: 64764
Connection: close
Content-Type: text/html; charset=utf-8

<!DOCTYPE html>
<html  dir="ltr" lang="en" xml:lang="en">
....
```

Context

- date stocate pe server
- de obicei stocate într-o bază de date
  - date gestionate de sistemele de gestionare a bazelor de date (DBMS)
  - tranzacții ACID
- nevoie: protejați datele de accesul sau manipularea ilicită

SQL - review

- SQL = Standard Query Language
- lucrează cu tabele
  - linii și coloane
- operații de bază
  - `SELECT * FROM Users WHERE Name='Alin';`
  - `UPDATE Users SET notified='true'WHERE Age='42';`
  - `INSERT INTO Users Values('Mihai', ...);`
  - `DROP TABLE Users; --this is a comment`

Cod ce rulează pe server

- serverul interacționează cu DB
- de obicei folosind SQL
- limbaj comun: PHP (PHP Hypertext Preprocessor)
- integrat cu codul HTML
- produce cod HTML bazat pe interogarea DB

Cod ce rulează pe server - exemplu de autentificare

- *$username* și *$password* sunt parametrii pentru o cerere POST
- utilizatorul este autentificat dacă se găsește o înregistrare în baza de date

```
$username = $_POST["username"]
$password = $_POST["password"]
$result = mysql_query("SELECT * FROM Users
    WHERE (Name='$username' AND Password='$password');");
```

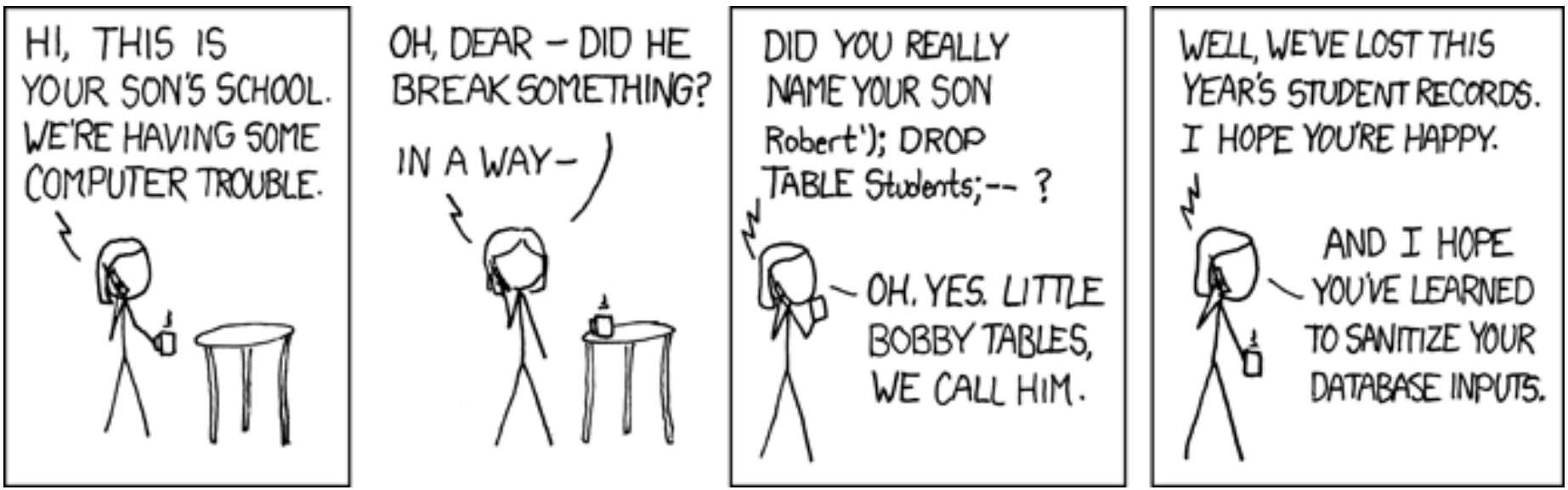
SQL injection

- o **vulnerabilitate** care duce la o posibilitate de exploatare
- bazat pe furnizarea de date de utilizator rău intenționate
- datorat
  - amestec de cod și date
  - încredere în datele introduse de utilizator
  - $\Longrightarrow$ confuzie date cu cod
- problemă similară cu *buffer overflow*
- condițiile generale și greșelile
  1. limite între datele și cod
  2. încredere nefondată
  3. intrări de utilizator neverificate / nesanitizate

Ideea de bază

- greșeala: încrederea în datele utilizatorului (user input)
  - se construiesc dinamic șiruri ce reprezintă interogări SQL
  - prin concatenarea / inserarea parametrilor primiți de la utilizator
  - parametrii nu sunt verificați
- atac
  - schimbă semantica interogării SQL
  - prin datele introduse de utilizator
  - $\Longrightarrow$ se elimină anumite condiții
  - $\Longrightarrow$ se adaugă funcționalități noi în interogarea SQL
- efecte posibile
  - modifică rezultatul interogării si funcționalitatea aplicației
  - scurgeri de informații din baza de date
  - compromite integritatea bazei de date (ex. sterge tabele, inserează înregistrări)

SQL Injection - exemple



https://imgs.xkcd.com/comics/exploits_of_a_mom.png

SQL injection - exemple (I)

```
$username = $_POST["username"]
$password = $_POST["password"]
$result = mysql_query("SELECT * FROM Users
    WHERE (Name='$username' AND Password='$password');");
```

- evită clauza Where din interogare prin
  - $username = "john' OR 1=1); -- "
  - $password = "anything"
- interogarea SQL inițială

```
SELECT * FROM Users
WHERE (Name='$username' AND Password='$password');
```

- devine

```
SELECT * FROM Users
WHERE (Name='john' OR 1=1); -- ' AND Password='anything');
```

SQL injection - exemple (II)

- inserează comenzi prin parametrii dați de utilizator
  - `$username = "john' OR 1=1); DROP TABLE Users; -- "`
  - `$password = "anything"`
- interogarea SQL inițială

```
SELECT * FROM Users
WHERE (Name='$username' AND Password='$password');
```

- devine

```
SELECT * FROM Users WHERE (Name='john' OR 1=1);
DROP TABLE Users; -- ' AND Password='anything');
```

SQL injection - exemple (III)

- pentru DBMS care nu acceptă comenzi
  - `$username = "alin' OR 1=1"`
  - `"anything' OR 1=1); DROP TABLE Users;"`
- interogarea SQL inițială

```
SELECT * FROM Users
WHERE (Name='$username' AND Password='$password');
```

- devine

```
SELECT * FROM Users WHERE(Name='alin' OR 1=1
AND Password='anything' OR 1=1); DROP TABLE Users;
```

SQL injection - exemple (IV)

- substituie un identificator
  - `$id = "1234' OR '1'='1"`
- interogarea SQL inițială (cod PHP)

```
$id = $_COOKIE["id"];
mysql_query("SELECT MessageID, Subject FROM messages
WHERE MessageID = '$id';");
```

- devine după substituție

```
SELECT MessageID, Subject FROM messages
 WHERE MessageID = '1234' OR '1'='1';
```

- problema
  - nu se face validarea lui *"$id"* deoarece se presupune că atacatorii nu pot modifica valoarea cookie
- soluție

```
$id = intval($_COOKIE["id"]);
```

Referințe CWE

- CWE-20: "Improper Input Validation"
  - no validation or incorrectly validation of input that can affect the control flow or data flow of a program
- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
  - software constructs all or part of an SQL command
  - using externally-influenced input from an upstream component,
  - but it does not neutralize or incorrectly neutralizes special elements
  - that could modify the intended SQL command when it is sent to a downstream component
- 1st place in the 2011 CWE/SANS Top 25 Most Dangerous Software Errors!

CWE-89

- C#

```
...
string userName = ctx.getAuthenticatedUserName();
string query = "SELECT * FROM items WHERE owner = '" + userName +
    "' AND itemname = '" + ItemName.Text + "'";
sda = new SqlDataAdapter(query, conn);
DataTable dt = new DataTable();
sda.Fill(dt);
...
```

- interogarea SQL inițială

```
SELECT * FROM items WHERE owner = <userName> AND itemname = <itemName>;
```

- atacatorul introduce pentru *itemName*

```
name' OR 'a'='a
```

- interogarea devine

```
SELECT * FROM items WHERE owner = 'wiley' AND itemname = 'name' OR 'a'='a';
```

- echivalent cu

```
SELECT * FROM items;
```

Limbaje afectate

- C#
- PHP
- Perl/CGI
- Python
- Ruby on Rail
- Java and JDBC
- C/C++
- SQL (*stored procedures* care nu validează parametrii primiți de la utilizator)

Second order SQL injection

- combină mai multe interogări SQL
  - unele inserează date primite de la utilizator în câmpurile bazei de date
  - altele folosesc aceste date
- problema
  - dacă se pot insera metacaractere în câmpurile bazei de date
  - $\Longrightarrow$ aplicația este vulnerabilă la atacuri de tipul *SQL injection*
- datele controlate de utilizator pot fi
  - câmpuri din baza de date
  - numele pentru anumite elemente din baza de date (ex. tabele, câmpuri)
  - obiecte (ex. funcție apelată ca și parte a unui API)

Second order SQL injection - exemple

- pentru un nume de utilizator creat anterior

  "abc'OR username='JANE"

  - următoarele interogări SQL (Oracle)

    ```
    EXECUTE IMMEDIATE 'SELECT username FROM sessiontable
    WHERE session=''' || sessionid || '''' into username;
    ```

  - conduc la interogarea

    ```
    SELECT ssn FROM users WHERE username='XXX' OR username='JANE'
    ```

- crearea unui tabel "users; –"
  - următoarea interogare

    ```
    mysql_query("SELECT * FROM $table WHERE userid = 100;");
    ```

  - conduce la interogarea

    ```
    SELECT * FROM users; -- WHERE userid = 100;
    ```

Validare input de la utilizator

- nevoie: date de la utilizator
- regula: **nu aveți încredere in datele primite de la utilizator!**
- datele primite de la utilizator trebuie validate inainte de folosire
  - verificare și refuzare
  - eliminare caractere nedorite (*"sanitize user input*)

Sanitize user input - metoda blacklist

- eliminare caractere nedorite / interzise
  - ', ;, -, #, "
- limitări
  - unele metacaractere, uneori sunt necesare
  - ex. nume "Peter O'Connor"

Sanitize user input - escaping

- evita semnificația specială a metacaracterelor
- ex: "\'", "\;", "\-", "\\"
- metode
  - manual (nerecomandat)
  - folosind funcții dedicate
    - *mysql_real_escape_string()* din PHP/MySQL
    - *$dbh→quote($value)* din Perl/DBD
- limitări
  - unele metacaractere, uneori sunt necesare
  - protejează doar un subset din interogările afectate
    - interogările ce folosesc șiruri ca și date de intrare
    - nu elimină vulnerabilitatea pentru interogări ce folosesc întregi

Sanitize user input - escaping (II)

- exemplu: CVE-2008-2790 (https://www.exploit-db.com/exploits/5840/)

```
SQL Injection:
http://[target]/[path]/detail.php?id=[SQL]
PoC:
http://acme.org/detail.php?id=-1%20union%20select%20USER()
        ,2,3,4,5,@@VERSION,7,8,9,10,11,12,13,database(),15,16
```

Moduri de a evita mecanismul de *escaping*

- second order injection: metacaracterele ajung în baza de date și sunt folosite apoi
- utilizatorul dă ca și date de intrare șirul "*myname ' drop users*", care va deveni

```
INSERT INTO mytable id, item VALUES ( 10,
    'myname '' drop users --');
```

- ulterior, în altă interogare se folosește câmpul ce contine metacaracterul dorit

```
$username = mysqlquery("SELECT name FROM mytable
    WHERE id = 10");
$newquery = "SELECT * FROM mydetails
    WHERE id = '".$username."'";
```

- interogarea finală

```
SELECT * FROM mydetails WHERE id = 'myname ' drop users --'
```

- codarea metacaracterele
  - funcții speciale: char(0x20)
  - secvențe hexazecimale: "0x73687574646f776e" ("shutdown")

Metoda whitelist

- se verifică dacă datele primite de la utilizator conțin strict caractere valide
  - un numar conține doar cifre și este într-un anumit interval
  - doar caractere sigure
- principiul "fail-safe defaults" - e mai ușor de refuzat decât de reparat
- limitări
  - dificil de a stabili reguli pentru date complexe

Pseudo remediu - Stored Procedures

- la fel de vulnerabile
- problema: folosesc date primite de la utilizator fara validare când se construiesc dinamic interogările SQL
- problema: modul în care sunt apelate
  - exemplu, cod vulnerabil

```
"SELECT xp_myquery('" + userData + "')"
```

- problema: alte declarații SQL după apelul procedurii

```
exec sp_GetName 'Blake'; insert into client
    values(1005,'Mike'); -- '
```

Prepared statements

- metoda recomandată
- impune separarea codului de date
- se definesc template-uri de interogări, de care se "leagă" ulterior datele prin corespondență
- declarațiile SQL sunt compilate înainte de binding

Prepared statements - exemplu

- interogarea SQL inițială

```
$result = mysql_query("SELECT * FROM Users
    WHERE(Name='$username' AND Password='$password');");
```

- devine

```
$db = new mysql("localhost", "user", "pass", "DB");
$statement = $db->prepare("SELECT * FROM Users
    WHERE(Name=? AND Password=?);");
$statement->bind_param("ss", $username, $password);
$statement->execute();
```

- dacă "*$username*" primește "*alin*'OR 1-1); –"
- interogarea va lua datele ca un șir de caractere

Prepared statements - exemplu II

- interogarea SQL inițială

```
$result = mysql_query("SELECT * FROM Users
    WHERE(Name='$username' AND Password='$password');");
```

- folosind *prepared statements* devine

```
$statement = $db->prepare("SELECT * FROM
    users WHERE (user=? AND pass=?);");
$statement->bind_param("ss", $user, $pass);
$statement->execute();
```

- variabilele "$user" și "$pass" nu mai pot modifica semantica interogării

Reguli generale de evitare SQL injection

- principiul "*least privilege*", nu rulați aplicația care se conecteză la baza de date cu drepturi prea mari (inutile)
- utilizator dedicat cu drepturi limitate
  - SQL injection + drepturi de administrator $\Longrightarrow$ atacatorul poate rula comenzi sistem daca baza de date suportă acest lucru
  - ex. "*xp_cmdshell*" stoder procedure pentru SQL Server
  - ex. "*\! cmd*" pentru client mysql
- nu concatenați sau înlocuiți șiruri
- verificați datele ce ajung în interogarea SQL
- încercați sa aplicați ***whitelisting*** "șterge tot înafară de date bune" în loc de ***blacklisting*** "șterge datele care se știe ca sunt greșite"

Reguli pentru a coda fără vulnerabilități SQL injection

1. nu permiteți conexiuni la baza de date nesigure (fara parola)
2. pentru acces creați un utilizator cu privilegile minime necesare
3. refuzați în mod explicit permisiuni de scriere, acolo unde nu este necesar
4. validați datele furnizate de utilizator
5. dacă este posibil, folosiți *stored procedures*
   - $\Longleftarrow$ logica aplicației este ascunsă
   - numele bazei de date, tabelelor nu sunt vizibile aplicației
6. folosiți parametrii pentru interogări, nu concatenați șiruri pentru a construi interogarea

Reguli pentru a coda fără vulnerabilități SQL injection (II)

7. ascundeți șirurile care pot dezvălui informații despre conexiunile bazei de date, numele utilizatorului și parola
8. în cazul execuției eronate a interogării SQL
   - nu afișați informații legat de motivul erorii
   - doar reportați eroarea
9. închideți întotdeauna conexiunea la baza de date, indiferent dacă interogarea reușește sau nu
   - $\Longrightarrow$ mitigați un posibil atac DoS

Code review

- verifică dacă aplicația interoghează o bază de date
- limbaje / cuvinte cheie
  - VB.NET : Sql, SqlClient, OracleClient, SqlDataAdapter
  - C#: Sql, SqlClient, OracleClient, SqlDataAdapter
  - PHP: mysql_connect
  - Perl: DBI, Oracle, SQL
  - Python: MySQLdb, DCOracle, pymssql
  - Java: java.sql, sql
  - ASP: ADODB
  - C++ (MFC): CDatabase
  - C/C++: #include <mysql++.h>, #include <mysql.h>, #include <sql.h>, ADODB, #import "msado15.dll"
  - SQL: exec, execute, sp_executesql

Code review (II)

- caută interogări SQL si se verifică dacă
  - se concatenează șiruri, se înlocuiesc șiruri
  - pe date nesigure
- testare
  - câmpuri text: se încearcă ' sau " ca și input
  - câmpuri numerice: se încearcă adâugarea de clauze la interogare (*id=10 AND 1=1*)
  - concatenarea interogărilor (*id=10; INSERT INTO ...*)
  - se aplică tuturor datelor externe aplicației
- `https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005)`

Stateless HTTP

- sesiune HTTP
  - mai multe perechi de cerere-raspuns între client și serverul web
- cererile nu sunt automat mapate pe client
  - nu se face asocierea între cerere și client
- cum se poate evita autentificarea pentru fiecare cerere trimisă de un client la același site web?

Web server - *Maintained State*

- serverul web menține starea sesiunii
- se trimite și clientului
- clientul o atașează fiecărei cereri
- metode
  - câmpuri ascunse
  - cookies

Session State - câmpuri ascunse

- încorporată în pagina trimisă clientului
  - câmpuri care conțin informații necesare pentru conectarea paginilor web
  - astfel de câmpuri sunt ascunse
  - în mod automat și transparent trimis înapoi la server de către client cu următoarea solicitare
  - funcționează pe pagini bazate pe formulare
- exemplu - pagina pay.php trimisă utilizatorului

```
<html>
<head> <title> Confirm Payment </title> </head>
<body>
<form action="submit_order" method=GET>
The total cost is $10. Confirm order?
<input type="hidden" name="price" value="10">
<input type="submit" name="pay" value="yes">
<input type="submit" name="pay" value="no">
</body>
</html>
```

Session State - câmpuri ascunse (II)

- exemplu: pe partea serverului (backend)

```
if (pay == "yes" && price != NULL) {
   bill_credit_card($price);
   deliver_products();
} else
cancel_transaction();
```

- problema
  - câmpurile ascunse vin de la client
  - atacatorii pot controla aceste câmpuri, chiar dacă sunt ascunse

Capabilities

- serverul menține starea de încredere
- client primește simbol de acces - *token* (ex. un identificator)
- clientul trimite jetonul de acces ca un câmp ascuns
- jetonul oferă clientului dreptul
  - de a accesa starea corespunzătoare a serverului
- pentru a preveni falsificarea jetonului
  - ales numere aleatorii mari
  - dificil (imposibil) de ghicit
- exemplu: pagina pay.php trimisă utilizatorului

```
<html>
<head> <title> Confirm Payment </title> </head>
<body>
<form action="submit_order" method=GET>
The total cost is $10. Confirm order?
<input type="hidden" name="sid" value="4685993747091">
<input type="submit" name="pay" value="yes">
<input type="submit" name="pay" value="no">
</body>
</html>
```

Capabilities (II)

- exemplu: pe partea serverului (backend)

```
price = search(sid);
if (pay == "yes" && price != NULL) {
   bill_credit_card($price);
   deliver_products();
} else
cancel_transaction();
```

- limitări
  - dificil de a menține relații complexe între pagini
  - închiderea unei pagini $\Longrightarrow$ se pierde identificarea clientului $\rightarrow$ procesul trebuie repornit

Cookies Based Session State

- serverul menține starea
- starea indexata de cookie
- inclus în protocolul HTTP
- la prima cerere a clientului
  - serverul generează starea
  - trimite clientului indexul (cookie)
  - exemplu

```
HTTP/1.1 200 OK
Date: Sun, 06 Dec 2015 12:50:41 GMT
Server: Apache/2.4.7 (Ubuntu)
x-powered-by: PHP/5.5.9-1ubuntu4.14
Set-Cookie: MoodleSession=fn0rpckv4sevk4i7b6f566tia1; path=/
Expires: Mon, 20 Aug 1969 09:23:00 GMT
Content-Script-Type: text/javascript
Last-Modified: Sun, 06 Dec 2015 12:50:41 GMT
X-Frame-Options: sameorigin
Content-Length: 7609
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
```

Cookies Based Session State (II)

- form: *Set-Cookie: key=value; options; ...*
- exemplu:
  Set-Cookie: edition=us; expires=Wed, 18-Feb-2015 8:20:34 GMT;
  path=/; domain=.zdnet.com

Cookies Based Session State -III

- clientul
  - stochează local fișierul cookie
  - trimite cookie la server la fiecare cerere
  - exemplu

```
GET / HTTP/1.1
Host: moodle.os.obs.utcluj.ro
User-Agent: Mozilla/5.0 (X11; Ubuntu;
Linux x86_64; rv:42.0) Gecko/20100101 Firefox/42.0
Accept: text/html,application/xhtml+xml,application/xml;
Cookie: _ga=GA1.2.604356790.1425374046; MoodleSession=fn
Connection: keep-alive
```

- cookie-based state
  - nu se pierde la închiderea paginii
  - independent de conținutul paginii → identifică clientul, nu cererea

Folosirea cookie

- utilizatorului autentificat i se asignează un cookie
- serverul trimite cookie-ul creat utilizatorului
- clientul web stocheaza cookie-ul local (ca și un fișier)
- clientul web trimite cookie-ul în următoarele cereri
  - pentru a identifica utilizatorul
- $\Longrightarrow$ utilizatorul nu trebuie sa se re-autntifice
- transparent pentru utilizator

Utilizarea cookie-urilor

- permite utilizatorilor anonimi să personalizeze o pagină web, ex. font, culori, etc.
- pagina creează un cookie care salvează diferiți parametrii, de posibil interes pentru utilizator
  - pe baza interacțiunilor trecute
- pe baza componentelor cookie pagina web poate fi personalizată
- avantaje: utilizatorul este anonim, chiar dacă preferințele acestuia sunt inregistrate

Autentificare folosinf cookie-uri

- după ce utilizatorul s-a autentificat pe o pagină web
- utilizatorului i se asociază un cookie
- trimis utilizatorului
- cererile ulterioare trimit serverului cookie-ul primit, dovedind că este utilizatorul autentificat

Furt cookie

- cel care deține cookie-ul poate accesa pagina web cu drepturile utilizatorului autentificat
- furt cookie $\Longrightarrow$ se imită un utilizator
  - acțiuni în numele utilizatorului autentificat

Furt cookie (II)

- furt cookie
  - compromite un server (care emite cookie-uri și le menține)
  - compromite clientul (care stochează fișierul cookie)
  - se prezice cookie-ul, dacă algoritmul de generare pentru cookie-uri al serverului este determinist (slab)
  - se ascultă traficul de rețea și se găsește valoarea pentru cookie
  - manipulează rețeaua pentru a trimite pachete atacatorului
- apărare
  - trafic criptat
  - ex. interacțiunile sensibile, după autentificare ar trebui să folosească HTTPS

Cookie-uri nepredictibile

- valori mari aleatorii pentru cookie-uri
- cookie valid doar pentru un anumit interval de timp
- ștergerea cookie-urilor când sesiunea se termină
- exemplu vulnerabilitate: Twitter (2013)
  - un singur cookie, *auth_token*, pentru a valida utilizatorul
  - cookie funcție de username și parolă
  - nu se schimbă între două autentificări
  - nu devine invalid când utilizatorul termină sesiunea (logged out)
  - $\Longrightarrow$ un cookie poate fi filosit până se schimbă parola

Bibliografie

- “24 Deadly Sins of Software Security”, chapter 1, 2, 3, 4, pp. 3 – 88
- SQL Injection Attacks by Example, http://www.unixwiz.net/techtips/sql-injection.html