

UNIVERSITATEA DE STAT DIN MOLDOVA

**FACULTATEA MATEMATICĂ ȘI INFORMATICĂ
DEPARTAMENTUL INFORMATICĂ**

Lucrare alternativă NR1

la disciplina ”*Securitatea aplicațiilor WEB* “,

Verificat: Pleșca Natalia, lector univ.

Efectuat: Sîrbu Daniel, grupa IA2101

Chișinău, 2023

PROBLEMELE VERSIUNII NESECURIZATE ALE APLICAȚIEI

Întâi de toate, vorbim despre pagina în care utilizatorul își lasă email-ul pentru a primi ulterior deferite informații. Analizăm codul care se ocupă de acest fapt :

```
$dbConnection = connectMeToDB();

if(isset($_POST["submit"])){
    if(isset($dbConnection)){
        $email = $_POST["email"];
        $date = date("Y-m-d H:i:s");
        $sql = "INSERT INTO emails (email,data) VALUES ('$email','$date')";
        $result = $dbConnection->query($sql);
        if($result){
            $_SESSION["message"] = "Ati fost abonat cu succes la newsletter!";
            header("Location: ./index.php");
        }
    }
}
```

Putem observa că este cel mai simplu cod care introduce în baza de date câmpul email, preluat din formular și data la care a fost submitat formularul, care se generează automat. Putem observa că nu se validează câmpul email deloc. Faptul că se face inserare în DB și faptul că se face inserare a unei intrări nevalidate, aduce cu sine o vulnerabilitate foarte mare și anume XSS Injection. Ce presupune asta?

Cross-site Scripting (XSS) este un atac de injectare de cod pe partea clientului. Atacatorul urmărește să execute scripturi malițioase în browserul web al victimei prin includerea de cod malițios într-o pagină web sau aplicație web legitimă. Atacul propriu-zis are loc atunci când victima vizitează pagina web sau aplicația web care execută codul malițios. Pagina web sau aplicația web devine un “vehicul” pentru a livra scriptul malițios în browserul utilizatorului. Locurile vulnerabile pe o pagină web care sunt utilizate în mod obișnuit pentru atacurile Cross-site Scripting sunt forumurile, panourile de mesaje și paginile web care permit comentarii.

O pagină web sau o aplicație web este vulnerabilă la XSS dacă utilizează date de intrare nesanitizate ale utilizatorului în rezultatele pe care le generează. Această intrare a utilizatorului trebuie apoi analizată de către browserul victimei. Atacurile XSS sunt posibile în VBScript, ActiveX, Flash și chiar CSS. Cu toate acestea, ele sunt cel mai des întâlnite în JavaScript, în primul rând pentru că JavaScript este fundamental pentru majoritatea experiențelor de navigare.

PREVENIREA CROSS-SITE SCRIPTING (XSS)

Prevenirea Cross-site Scripting (XSS) nu este ușoară. Tehnicile specifice de prevenire depind de subtipul de vulnerabilitate XSS, de contextul de utilizare a datelor de intrare ale utilizatorului și de cadrul de programare. Cu toate acestea, există anumite principii strategice generale pe care ar trebui să le urmați pentru a vă menține aplicația web în siguranță.

- Formarea și menținerea conștientizării

Pentru ca aplicația web să rămână în siguranță, toți cei implicați în crearea aplicației web trebuie să fie conștienți de riscurile asociate cu vulnerabilitățile XSS. Personalul implicat în dezvoltare ar trebui să fie instruit în domeniul .

- Nu trebuie de avut încredere în nici o intrare a utilizatorului

Toate intrările utilizatorului trebuie tratate ca fiind de neîncredere. Orice intrare a utilizatorului care este utilizată ca parte a ieșirii HTML introduce riscul unui XSS.

Acum, o să încerc câteva metode de injectare, prin cod HTML, CSS și JS :

- 1) Ce-a mai banală injecție XSS și anume voi insera simbolurile ”<!--” astfel, voi comenta tot codul HTML de după înregistrarea în care a fost injectate aceste simboluri :

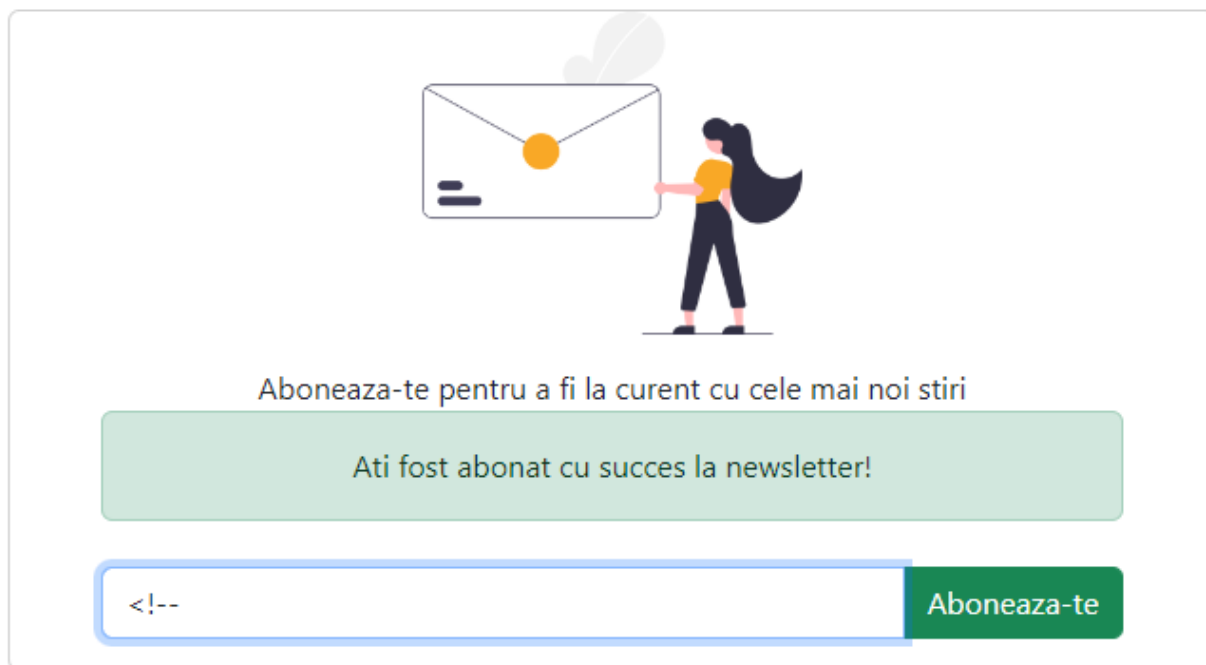


Figura 1 : Încercare de injectare prin cod HTML

Acum, voi mai adăga câteva înregistrări :




























<input type="checkbox"/>	 Изменить	 Копировать	 Удалить	1	sirbu.daniel.2018@gmail.com	2023-10-12
<input type="checkbox"/>	 Изменить	 Копировать	 Удалить	2	sirbu.daniel.2018@gmail.com	2023-10-12
<input type="checkbox"/>	 Изменить	 Копировать	 Удалить	3	sirbu.daniel.2018@gmail.com	2023-10-12
<input type="checkbox"/>	 Изменить	 Копировать	 Удалить	4	ionutemilian@mail.ru	2023-10-12
<input type="checkbox"/>	 Изменить	 Копировать	 Удалить	5	dumitru@gmail.com	2023-10-12
<input type="checkbox"/>	 Изменить	 Копировать	 Удалить	6	<!--	2023-10-13
<input type="checkbox"/>	 Изменить	 Копировать	 Удалить	7	fyjj@gmail.com	2023-10-13
<input type="checkbox"/>	 Изменить	 Копировать	 Удалить	8	sumaschi_artur@gmail.com	2023-10-13
<input type="checkbox"/>	 Изменить	 Копировать	 Удалить	9	sdblog2022@gmail.com	2023-10-13

Figura 2 : Structura tabelului emails după injectare

Verificăm, acum, ce se afișează pe pagina de "dashboard" :

Email	Data
sirbu.daniel.2018@gmail.com	2023-10-12
sirbu.daniel.2018@gmail.com	2023-10-12
sirbu.daniel.2018@gmail.com	2023-10-12
ionutemilian@mail.ru	2023-10-12
dumitru@gmail.com	2023-10-12

Figura 3 : Ce se afișează defapt pe pagină

Putem observa că înregistrările de după injecție nu se mai afișează. Acest lucru are loc datorită faptului că nu am inserat altceva decât codul care browserul îl interpretează ca început de comentariu în HTML, astfel tot ce urmează după acest simbol, nu se mai interpretează de către browser. Ne putem convinge de asta și făcând o simplă inspecție de cod(CTRL + U) :



Figura 4 : Ce vede browserul defapt

P.S(nu știu de ce browserul interpretează așa tabelul, SORRY))))

- 2) Acum, încercăm o injecție prin cod CSS. Deoarece stilurile interne au prioritate asupra celor externe, putem ușor profita de această lacună, inserând anumite stiluri în input, drept exemplu, voi modifica culoarea paginii, inserând în input următorul cod : `<style> body{ display : none;}</style>`

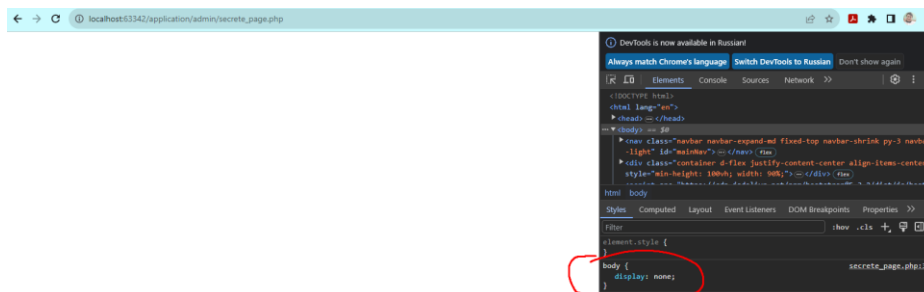


Figura 5 : Aplicarea stilului în browser

După cum putem vedea, stilul s-a aplicat, iar consecințele sunt destul de grave

- 3) Cel mai mare risc, îl prezintă totuși, injecțiile cu cod JS. Deoarece este limbaj client side, putem manipula o varietate de elemente cu ajutorul lui. Spre exemplu, voi injecta un mini-script care va afișa o casetă de tip alert, la fiecare o secundă. Trebuie să înțelegem, că fără a edita înregistrarea cu respectivul script sau fără a opri JS din browser, navigarea în pagină va fi imposibilă. Voi încerca să inserez un cod pentru a rula în fiecare secundă o casetă alert. Pe lângă faptul că aceasta v-a rula la infinit(dacă înregistrarea nu se modifică sau șterge), acesta este periculoasă pentru că blochează accesul la pagină atât timp cât este activă. Pentru asta, folosesc următorul script :

```
<script>
  setInterval(function () {
    alert("Esti neputincios ((((((((")
  }, 1000);
</script>
```

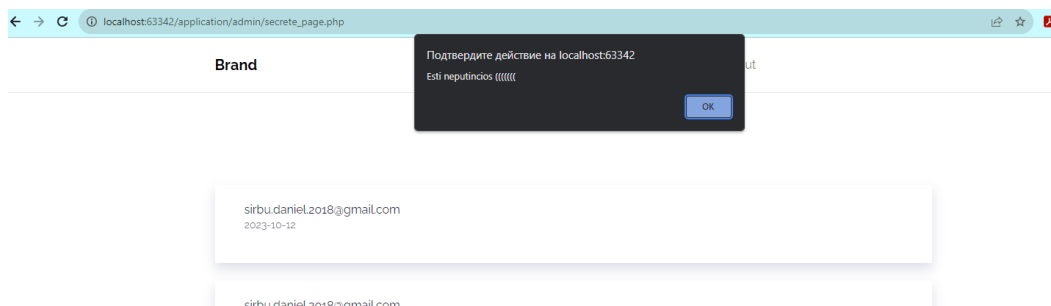


Figura 6 : Execitarea scriptului în browser

!Remarcă : Pentru a vedea aceste injecții, trebuie să ștergem sau să edităm injecțiile anterioare, pentru că acestea strică logica HTML și CSS.

- 4) Mi-a venit încă o idee de injecție în timp ce realizam chestia cu alertul, și anume la intervale identice de timp, să construiesc în body blocuri(div-uri), folosesc următorul cod :

```
<script>
  setInterval(function() {
    let body = document.querySelector('body');
    let random = Math.floor(Math.random() * 100);
    for (let i = 0; i < random; i++) {
      let div = document.createElement('div');
      div.style.width = '100px';
      div.style.height = '100px';
      div.style.backgroundColor = 'red';
      div.style.position = 'absolute';
      div.style.top = Math.floor(Math.random() * 100) + 'vh';
      div.style.left = Math.floor(Math.random() * 100) + 'vw';
      body.appendChild(div);
    }
  }, 3000);
</script>
```

!Remarcă : Pentru ca scriptul să funcționeze, apostrofele trebuie ecranate !

Pe lângă faptul că la un moment dat suprafața paginii devine imposibil de văzut, la un moment dat, blocurile vor fi într-atât de multe, în cât vor bloca browserul

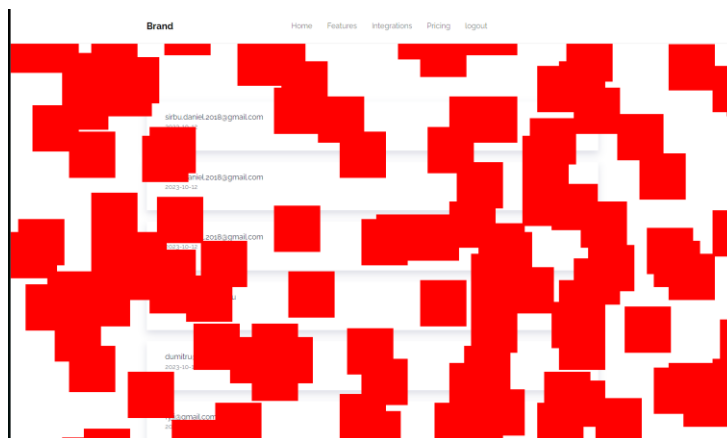




Figura 7 : Rezultatele execuției în browser

- 5) Tot prin XSS injecție putem accesa cookie-uri. Acest lucru se poate face foarte simplu, de exemplu prin urmatorul cod:

```
<script>
    console.log(document.cookie);
</script>
```

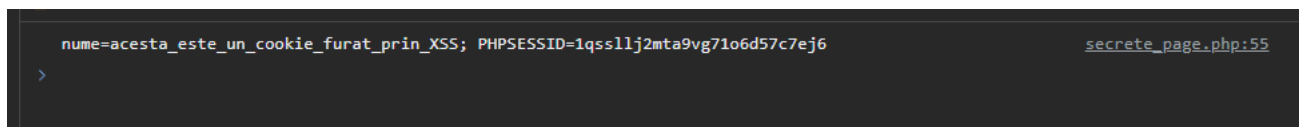


Figura 8 : Accesul la cookies prin intermediul XSS injection

Putem observa că obținem nu doar cookieurile, ci și ID-ul sesiunii. Practic, avem acces la sesiunea utilizatorului, fără a fi utilizat careva credențiale pentru asta.

Prima și cea mai importantă metodă de protecție este validarea intrărilor utilizatorilor. Pentru o siguranță mai mare, voi realiza această validare atât pe partea client cât și pe partea server. De ce nu mă limitez doar la validările pe partea client ? Pentru că JS-ul poate fi ușor dezactivat din browser, printr-un click .

Cum arată codul ?

```
<script>
    window.addEventListener('load', function () {
        const form = document.querySelector("form.needs-validation");

        form.addEventListener("submit", function (event) {
            const emailInput = document.getElementById("email");
            const emailError = document.getElementById("email-error");

            const emailRegex = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/;
            if (!emailRegex.test(emailInput.value)) {
                emailError.textContent = "Loginul invalid";
                emailInput.classList.add("is-invalid");
                event.preventDefault();
            } else {
                emailError.textContent = "";
                emailInput.classList.remove("is-invalid");
            }
        });
    });
</script>
```

Figura 9 : Validarea pe partea client

Codul, este extrem de simplu, dar foarte important. Ideea principală este că la submitarea formularului se verifică textul introdus de utilizator în baza unui pattern.

[a-zA-Z0-9._-]+: Această parte a expresiei validează partea locală a adresei de email (partea dinaintea simbolului '@'). Ea permite litere majuscule (A-Z), litere mici (a-z), cifre (0-9) și câteva caractere speciale, cum ar fi puncte (.), underscore (_) și cratime (-).

[a-zA-Z0-9.-]+: Această parte a expresiei validează domeniul (partea din spatele simbolului '@') și subdomeniile adresei de email. Ea permite litere majuscule (A-Z), litere mici (a-z), cifre (0-9), puncte (.) și cratime (-). În dependență de validitatea intrării, se afișează un mesaj de eroare.

Codul pe partea de backend este similar, doar că există mai multe condiții de verificare :

```
private function validateEmail($email):void {
    if (empty($email)) {
        throw new Exception("Emailul este obligatoriu");
    }

    $emailPattern = "/^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/";
    if (!preg_match($emailPattern, $email)) {
        throw new Exception("Emailul nu corespunde sablonului");
    }
}

private function isEmailUnique($email):bool {
    $sql = "SELECT * FROM emails WHERE email = ?";
    $stmt = $this->dbConnection->prepare($sql);
    $stmt->bind_param("s", $email);
    $stmt->execute();
    $result = $stmt->get_result();
    return $result->num_rows == 0;
}
```

Figura 10 : Validarea pe partea server

Pe lângă faptul că validez după pattern, verific dacă intrarea nu este nulă și tot odată verific dacă nu mai există un asemenea email în baza de date. Nu există logică în a abona unul și același utilizator de câteva ori, corect? Eu cred că da. Acum câteva teste :



Figura 11 : Încercare de XSS respinsă de client

Observăm că a funcționat validarea pe partea de client. Iar dacă comentez scriptul, deja funcționează validarea pe partea server :

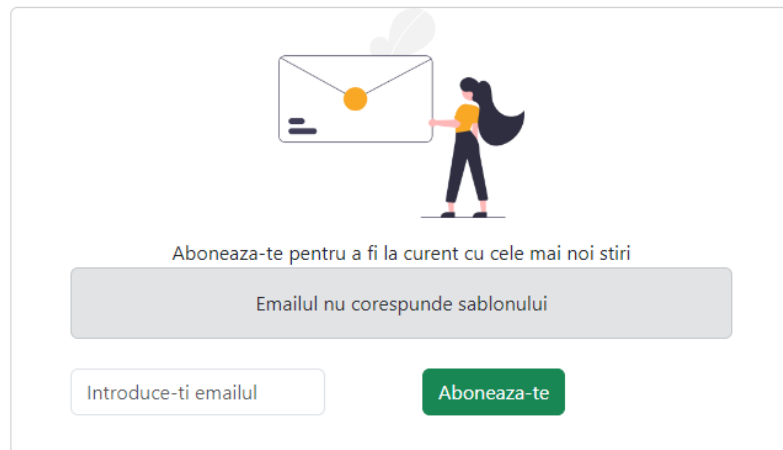


Figura 12 : Încercare de XSS respinsă de server

Încă un moment important, dacă cumva rău făcătorul a pătruns în baza de date și a editat unele înregistrări, la afișare filtrează datele provenite din baza de date prin funcția ***filter_var(\$email, FILTER_VALIDATE_EMAIL)***, astfel încât în loc de input invalid, se afișează doar un mesaj text.

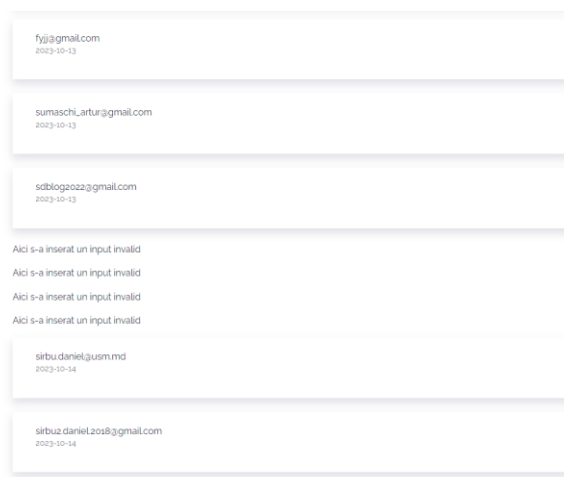


Figura 13 : Cum se afișează datele filtrate

Următoarea vulnerabilitate este riscul de SQL injecție. Aici, trebuie să fac o paranteză. Vulnerabil la SQL injecție este formularul de login, și anume în clauza WHERE. Ideea este că instrucțiunea SQL poate fi manipulată extern, în așa fel în cât instrucțiunea să returneze TRUE, chiar dacă în DB nu există asemenea înregistrări. Dar, ce reprezintă SQL injecție și cum ne putem proteja ?

CE SUNT SQL INJECTION ȘI CUM POT COMBĂTUTE

Injecția SQL (SQLi) este un tip de atac prin injecție care face posibilă executarea de instrucțiuni SQL malițioase. Atacatorii pot utiliza vulnerabilitățile SQL Injection pentru a ocoli măsurile de securitate ale aplicațiilor. Aceștia pot ocoli autentificarea și autorizarea unei pagini web sau a unei aplicații web și pot prelua conținutul întregii baze de date SQL. De asemenea, aceștia pot utiliza SQL Injection pentru a adăuga, modifica și șterge înregistrări în baza de date.

O vulnerabilitate de tip SQL Injection poate afecta orice site sau aplicație web care utilizează o bază de date SQL, cum ar fi MySQL, Oracle, SQL Server sau altele. Infractorii o pot folosi pentru a obține acces neautorizat la datele dumneavoastră sensibile: informații despre clienți, date personale, secrete comerciale, proprietate intelectuală și multe altele. Atacurile de tip SQL Injection sunt una dintre cele mai vechi, mai răspândite și mai periculoase vulnerabilități ale aplicațiilor web. Organizația OWASP (Open Web Application Security Project) enumeră injecțiile în documentul OWASP Top 10 2017 ca fiind amenințarea numărul unu la adresa securității aplicațiilor web.

Pentru a realiza un atac de tip SQL Injection, un atacator trebuie mai întâi să găsească intrări vulnerabile ale utilizatorului în cadrul paginii web sau al aplicației web. O pagină web sau o aplicație web care prezintă o vulnerabilitate de tip SQL Injection utilizează astfel de intrări ale utilizatorului direct într-o interogare SQL. Atacatorul poate crea conținut de intrare. Un astfel de conținut este adesea numit sarcină utilă

malițioasă și reprezintă partea cheie a atacului. După ce atacatorul trimite acest conținut, comenzile SQL malițioase sunt executate în baza de date.

Atacatorii pot folosi SQL Injections pentru a găsi acreditările altor utilizatori din baza de date. Ei pot apoi să se dea drept acești utilizatori. Utilizatorul care se dă drept utilizator poate fi un administrator de bază de date cu toate privilegiile bazei de date.

Metode de prevenire a SQL injection :

- Folosirea Parametrizării: Una dintre cele mai eficiente metode de prevenire a injectiilor SQL este folosirea parametrizării în interogările SQL. Acest lucru implică folosirea unui mecanism de substituție a parametrilor în loc să construim manual interogările SQL. Majoritatea limbajelor de programare și framework-urile web moderne oferă funcționalități pentru a face acest lucru în mod corect.
- Filtrarea datelor de intrare: Datele introduse de utilizator trebuie filtrate înainte de a le include în interogările SQL. Acest fapt presupune eliminarea oricăror caractere periculoase sau transformarea lor, astfel încât să nu mai fie interpretate ca instrucțiuni SQL.
- Evitarea scrierii codului nativ: În 2023, este bine venită utilizarea frameworks-urilor, care dispun de mecanisme performante de protecție contra SQL injection
- Monitorizarea și păstrarea logurilor pentru a detecta orice activitate suspectă sau tentativă de injectare SQL

TENTATIVE DE SQL INJECTION

1. Introduc '1' or '1'='1' în inputuri. Cu un asemenea input, modificăm semnificativ logica interogării SQL. În momentul introducerii unui astfel de input, instrucțiunea arată în felul următor :

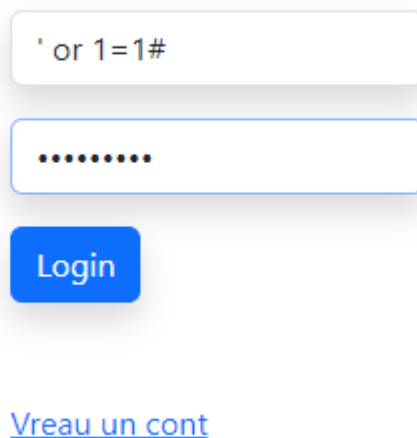
```
"SELECT * FROM user WHERE login = '1' or '1'='1' AND password = '1' or '1'='1'";
```

Figura 14 : Cum se modifică interogarea în timpul inecției SQL

Printr-un calcul simplu de logică, observăm că interogarea mereu returnează TRUE. Iar pentru că avem if-ul unde condiția de logare este să avem un număr de înregistrări > 0, obținem toate înregistrările din baza de date, astfel obținând identitatea tuturor conturilor.

```
object(mysql_result)#2 (5) { ["current_field"]=> int(0) ["field_count"]=> int(3) ["lengths"]=> NULL  
["num_rows"]=> int(6) ["type"]=> int(0) }
```

2. Încă o combinație funcționabilă este aceasta, care se bazează, de asemenea pe logica că $1 = 1$, iar # comentează restul codului.



The image shows a web form with two input fields and a button. The top input field contains the text `' or 1=1#`. The bottom input field contains ten dots. Below the input fields is a blue button labeled "Login". Below the button is a blue hyperlink labeled "Vreau un cont".

Figura 16 : O altă încercare de SQL injection

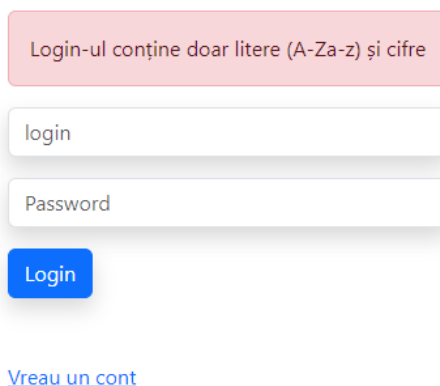
Cred că este clar cum funcționează SQL injection și la ce consecințe poate duce. Acum despre protecție :

Iarăși, validarea este cheia "succesului", de aceea codul nu o să de diferențeze foarte mult de codul pe care l-am folosit pentru XSS, prin urmare nu o să mai descriu fiecare bucată de cod împarte. Într-o altă ordine de idei, o validare calitativă, ne protejează de ambele tipuri de injecții. Pe partea de server, lucrurile sunt puțin mai complexe, pentru că am combinat validarea datelor cu parametrizarea interogării. Acestea implică divizarea instrucțiunii SQL în două etape separate: prima etapă constă în pregătirea instrucțiunii SQL, iar a doua etapă constă în atașarea valorilor la această instrucțiune. În PHP, acestea sunt implementate utilizând funcții precum `prepare`, `bind_param`, și `execute`.

Prepararea Instrucțiunii SQL (`prepare()`): În prima etapă, se definește instrucțiunea SQL cu "sloturi" sau "locuri rezervate" pentru valorile pe care le veți furniza mai târziu. Aceste sloturi sunt reprezentate de semnul întrebării (poate fi înlocuit și de de marcajul `:nume` în instrucțiunea SQL.)

Legarea Parametrilor (`bind_param()`): În a doua etapă, se leagă valorile reale ale parametrilor la sloturile pregătite anterior. Aceasta se face utilizând metoda `bind_param`, care asigură că valorile sunt corect validate și escapate pentru a preveni injectiile SQL.

Executarea Instrucțiunii (execute()): În final, se execută instrucțiunea SQL. Valorile legate prin bind_param sunt utilizate în instrucțiunea SQL, dar acestea sunt tratate în mod sigur pentru a preveni SQL injection.



The screenshot shows a login form with a red error message at the top: "Login-ul conține doar litere (A-Za-z) și cifre". Below the message are two input fields: "login" and "Password". The "login" field contains the text "login". Below the input fields is a blue "Login" button. At the bottom of the form is a blue link that says "Vreau un cont".

Figura 17 : Încercare nereușită de SQL injecțion

```
private function validateLogin($login): void
{
    if (empty($login)) {
        throw new Exception("Loginul este obligatoriu");
    } elseif (!preg_match('/^[A-Za-z0-9]+$/', $login)) {
        throw new Exception("Login-ul conține doar litere (A-Za-z) și cifre");
    }
}

private function validatePassword($password): void
{
    if (empty($password)) {
        throw new Exception("Parola este obligatorie");
    } elseif (strlen($password) < 8 || strlen($password) > 36) {
        throw new Exception("Parola trebuie să aibă cel puțin 8 caractere și maxim 36 caractere");
    }
}

private function isLoginUnique($login): bool
{
    $stmt = $this->dbConnection->prepare("SELECT * FROM users WHERE login = ? LIMIT 1");
    $stmt->bind_param("s", $login);
    $stmt->execute();
    $result = $stmt->get_result();
    return $result->num_rows === 0;
}
```

Figura 18 : Codul de validare a înregistrării unui nou utilizator, pe partea de backend

Încă o vulnerabilitate este accesul neautorizat, este vorba despre sesiuni. Dacă nu se setează corect, pagina secretă poate fi accesată cunoscând adresa ei din url. Nu pot să demonstrez asta prin poze, dar totul este deja implementat în cod.

