## VULNERABILITĂȚI LA INJECȚII DE COD

Vulnerabilitățile din cod la atacuri prin injecție pot avea efecte devastatoare asupra aplicațiilor web. Lipsa validării și sanitizării intrărilor permite exploatarea acestora, fapt ce poate implica executarea de comenzi arbitrare ale sistemului de operare, evitarea autentificării sau dezvăluirea de informații sensibile într-o breșă de date.

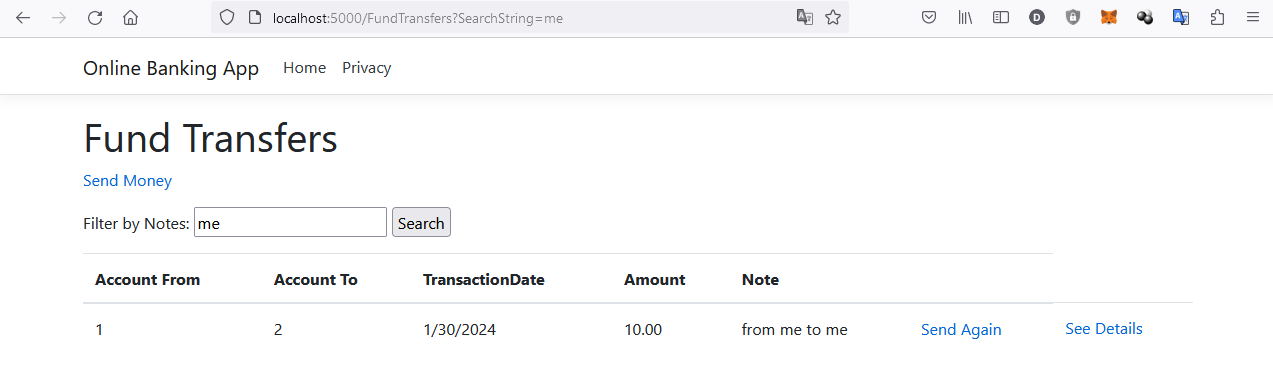
### ATAC SQL INJECTION

Aplicațiile web interacționează cu bazele de date pentru a stoca date și înregistrări. Pentru a comunica cu un DBMS în scopul accesării și gestionării datelor, este folosit limbajul de interogare SQL. Aceste interogări pot fi compuse utilizând doar limbajul de programare sau și o bibliotecă, însă generarea acestora poate fi nesigură atunci când nu sunt urmate bunele practici de securitate.

Un dezvoltator poate scrie cod care produce o intergare SQL în mod dinamic prin concatenarea șirurilor de caractere cu un șir de caractere introdus de utilizator. Fără contramăsuri adecvate, un actor rău intenționat poate injecta comenzi suspecte în șirul de intrare, schimbând astfel scopul final al interogării.

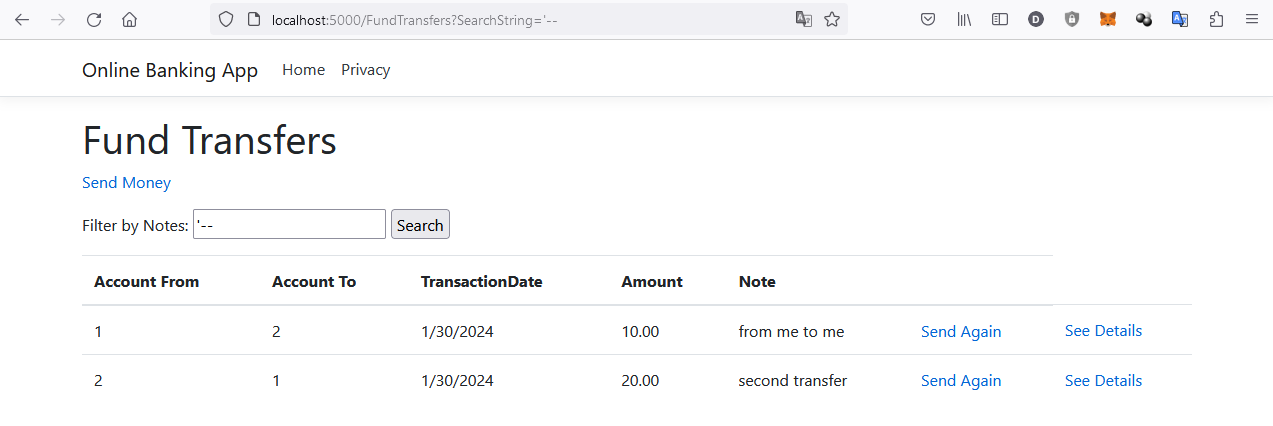
EF este un framework ORM popular ca alegere pentru dezvoltatorii de aplicații web ASP.NET, care prin ușurința sa de utilizare le permite acestora să modeleze datele în obiecte și să le interogheze într-o manieră agnostică în ceea ce privește sursa de date. Cu toate acestea, nu poate fi garantat faptul că API-ul expus este utilizat în mod corect.

Pentru exemplificarea atacului va fi folosită o reprezentare minimalistă a unei aplicații de Online Banking realizată folosind framework-ul ASP.NET, conectată la o bază de date SQLite. Pagina web afișată în browser permite filtrarea transferurilor de fonduri după notele asociate. Căutarea după secvența „me”, returnează o intrare asociată transferului a cărui notă este „from me to me”.

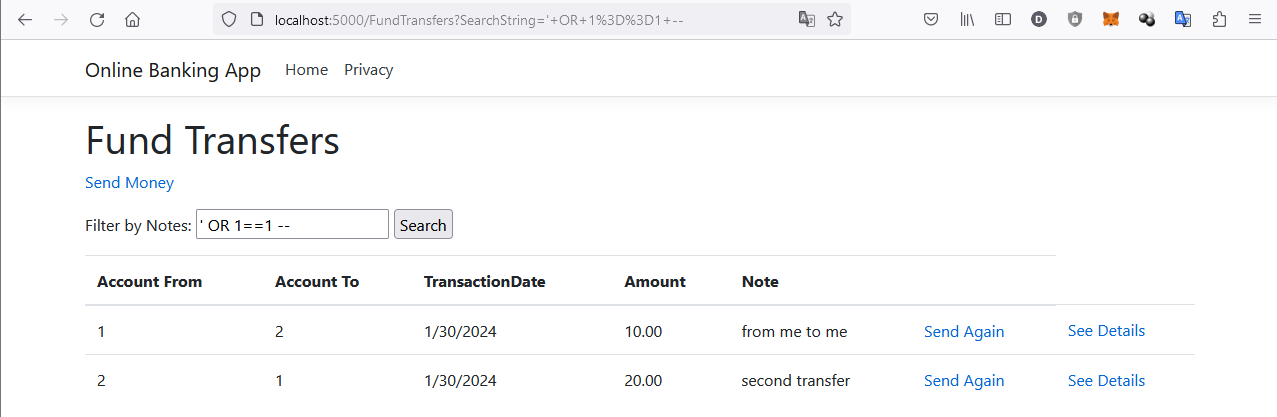


Pentru determinarea vulnerabilității la injecții SQL a aplicației se va încerca alcătuirea unei intrări care să exploateze modul în care limbajul SQL interpretează anumite caractere și observarea comportamentului aplicației care poate fi neașteptat, poate duce la erori sau poate revela date care nu ar trebui să fie accesibile.

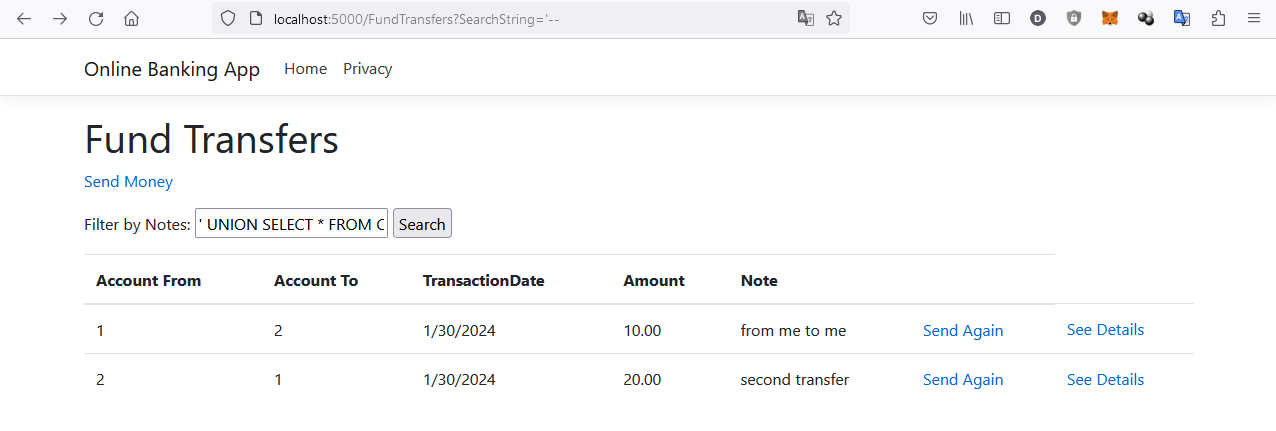
Analizând interfața grafică și presupunând că secvența de cod vulnerabilă concatenează o parte din interogarea SQL cu intrarea, aceasta ar putea avea formatul SELECT smth FROM unknownTable WHERE Note LIKE + ’ + input + ’. Cel mai simplu test este furnizarea unei intrări precum „’--”, care în cazul curent rezultă în interogarea „SELECT \* FROM FundTransfer WHERE Note LIKE '%' --%”. Clauza LIKE prin wildcard-ul „%” va potrivi note cu orice secvențe de caractere.

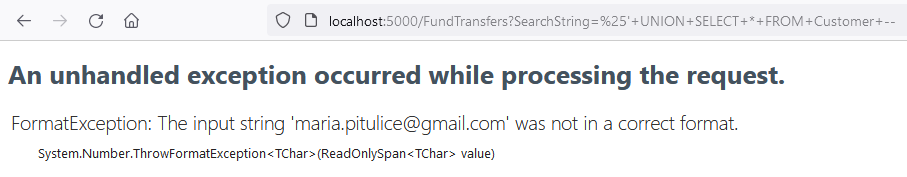


O altă posibilitate este executarea unei interogări ce folosește o tautologie, o expresie care mereu va fi evaluată ca fiind adevărată. Prin urmare, este dezirabil ca intrarea să înceapă cu caracterul „’” pentru a încheia prima condiție a clauzei WHERE și continuarea cu condiția mereu adevărată, urmată de comentarea restului interogării. În cazul curent, intrarea „’ OR 1==1 --” va rezulta în interogarea „SELECT \* FROM FundTransfer WHERE Note LIKE '%' OR 1==1 --%”.

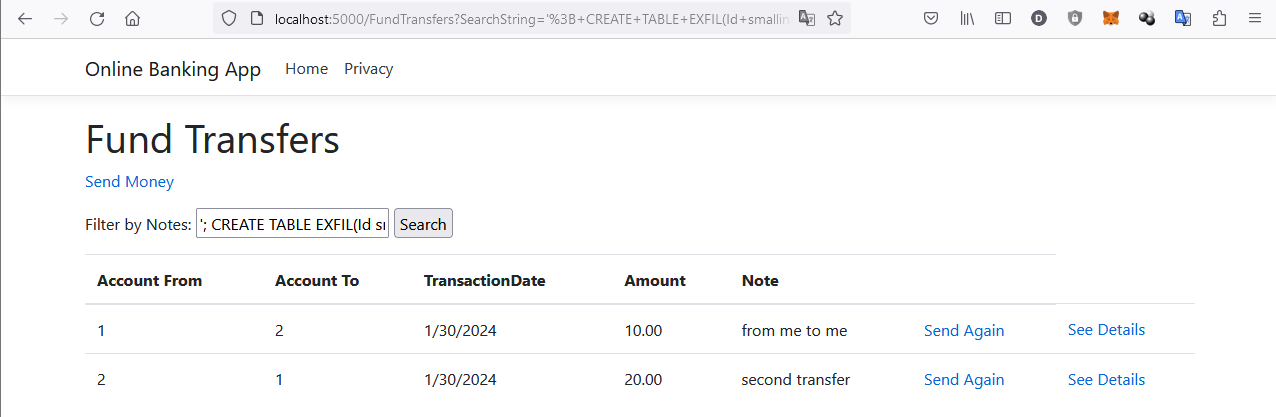


Returnarea mai multor intrări în urma introducerii șirurilor de caractere anteriore confirmă prezența vulnerabilității, fapt ce îndeamnă la mai multe posibilități de exploatare ce pot consta în interogarea sau manipularea datelor din tabelele existente sau chiar crearea și ștergerea altora. Se va încerca adăugarea unor operațiuni suplimentare prin caracterul „;”. Cuvântul cheie UNION va fi folosit pentru a combina rezultatele a două interogări de tip SELECT într-un singur set de rezultate „SELECT \* FROM FundTransfer WHERE Note LIKE ‘%' UNION SELECT \* FROM Customer --’”.

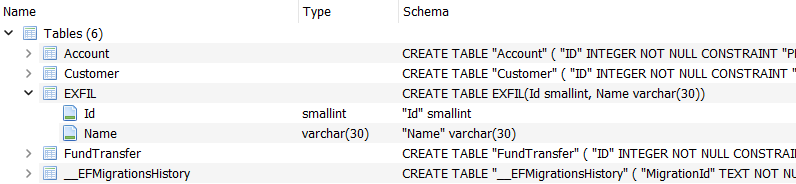




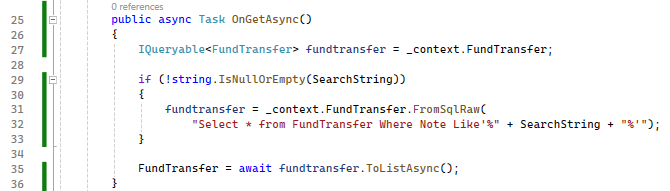
Încercarea de exfiltrare a datelor nu a reușit. Excepția sugerează faptul că este o nepotrivire între formatul de date așteptat și formatul datelor returnate în urma interogării. Se va încerca crearea unei noi tabele: „SELECT \* FROM FundTransfer WHERE Note LIKE '%'; CREATE TABLE EXFIL(Id smallint, Name varchar(30))’”



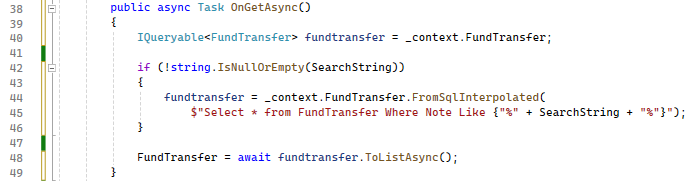
Pentru a confirma faptul că acest conținut creează într-adevăr o tabelă va fi investigată baza de date folosind aplicația DB Browser for SQLite.

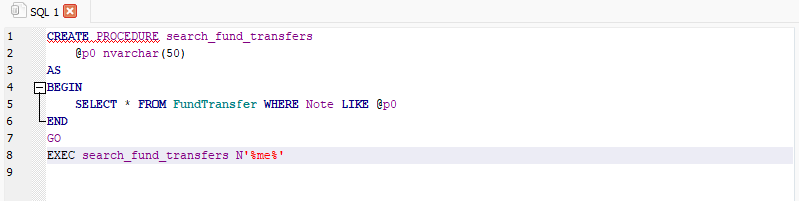


Secvența de cod vulnerabilă este localizată la nivelul metodei OnGetAsync, unde interogarea este compusă în mod dinamic. Șirul de intrare nu este validat sau sanitizat, permițând adversarului să insereze o comandă SQL arbitrară sau să modifice scopul final al interogării. Deși EF permite execuția de interogări SQL folosind metoda FromSQLRaw, există pericolul de a-i transmite ca argument un șir necorespunzător.

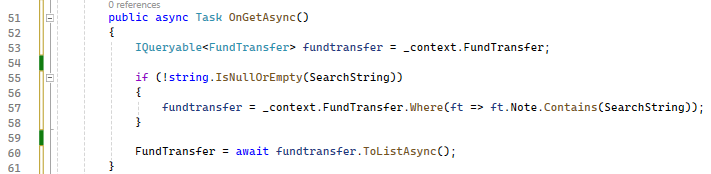


Luând în considerare ultimul șir de intrare folosit în exploatarea vulnerabilității, și concatenarea acestuia cu restul comenzii, în cadrul metodei FromSqlRaw, interogarea SQL rezultată reprezintă o comandă malițioasă alcătuită din secvențe SQL valide care are ca scop crearea unei noi tabele la nivelul bazei de date. Într-o altă circumstanță, situația poate escalada prin furnizarea unei intrări ce permite execuția codului de la distanță prin deschiderea unei noi sesiuni shell. De aceea, această modalitate de a forma interogări SQL nu este văzută ca o bună practică.

Pentru a scrie cod mai bun și mai sigur este recomandată folosirea facilităților puse la dispoziție de framework, precum metoda FromSqlInterpolated sau metodele LINQ. Metoda FromSqlInterpolated ajută la prevenirea injecției SQL prin convertirea automată a valorilor interpolate în parametrii interogării SQL. Un atacator nu poate injecta cu succes cod malițios deoarece acesta nu va fi executat ca și cod SQL, ci va fi doar o valoare, parte a interogării.

Când interogarea este executată, motorul SQL înlocuiește @p0 cu valoarea transmisă, asigurându-se că aceasta este tratată ca dată de intrare și nu drept comandă SQL. Chiar dacă parametrul conține cuvinte cheie SQL sau caractere speciale, acestea nu vor fi interpretate ca parte din comandă, ci doar ca o valoare de comparație în clauza LIKE. EF preia string-ul interpolat, identifică variabilele și le convertește în parametrii, executând operațiuni echivalente din punct de vedere logic cu cele din figura următoare.

Aceeași funcționalitate poate fi obținută și prin metodele LINQ, respectiv prin folosirea metodei Where pentru a filtra entitățile FundTransfer pe baza coloanei Note ce conține string-ul SearchString și apelarea metodei ToList() pentru execuția interogării alcătuite anterior.

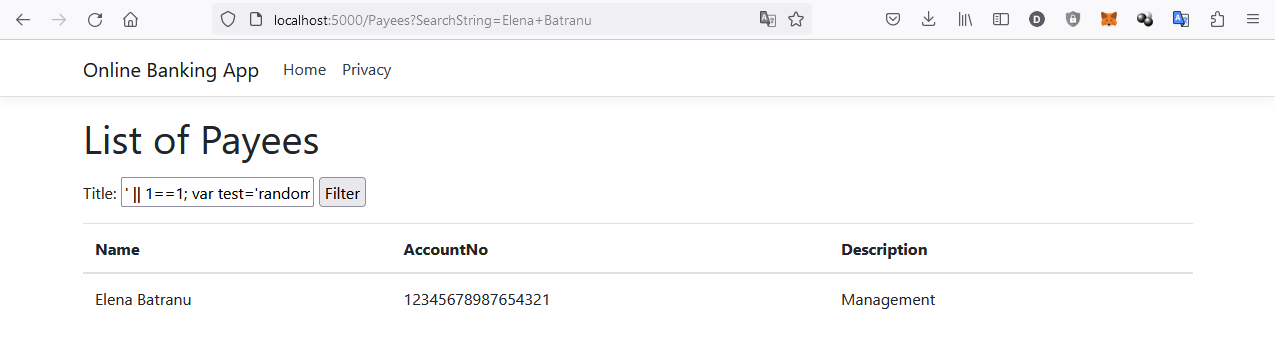


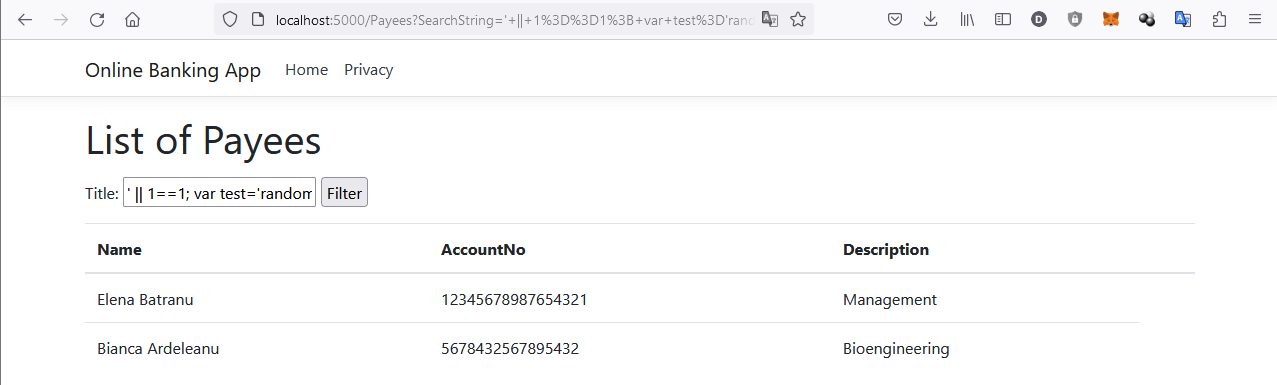
Deși această variantă este mai facilă de utilizat, uneori metoda FromSqlInterpolated este necesară în situații unde expresiile LINQ nu sunt suficiente sau eficiente precum atunci când sunt efectuate join-uri complexe sau subinterogări.

### ATAC NOSQL INJECTION

Bazele de date NoSQL sunt un tip diferit de baze de date în care sunt stocate date non-relaționale și semi-structurate. Există multe tipuri de baze de date NoSQL precum Cassandra, Redis, DynamoDB și MongoDB, fiecare cu propriul limbaj de interogare. Deși diferite unele de altele, aceste interogări sunt, de asemenea, predispuse la atacuri de injecție.

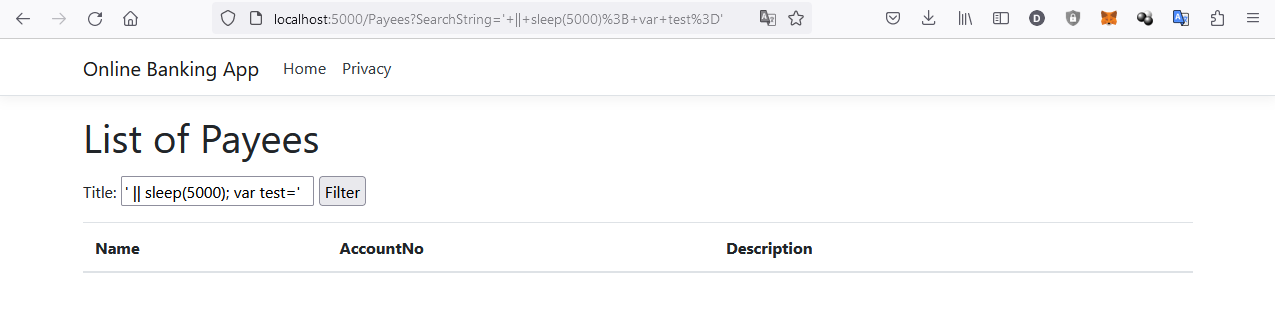
La fel ca în cazul unei injecții SQL o modalitate de testare este executarea unei interogări ce folosește o tautologie, o expresie care mereu va fi evaluată ca fiind adevărată „' || 1==1; var test='random string to be closed by quotation mark” ce rezultă în filtrul „{$where: "function() {return this.Name == '' || 1==1; var test='random string to be closed by quotation mark'}"}”.



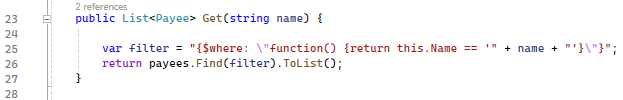


În cazul unei injecții SQL normale, un atacator poate să execute comenzi SQL arbitrare, expunând sau manipulând datele după bunul plac. Din cauză că JavaScript este un limbaj cu mai multe capabilități, acest lucru nu numai că permite unui atacator să manipuleze datele, ci și să ruleze cod arbitrar. Se va încerca furnizarea unei intrări care să întârzie execuția interogării, fapt ce poate rezulta într-un atac de tip Denial of Service „' || sleep(5000); var test='”, „{$where: "function() {return this.Name == '' || sleep(5000); var test=''}"}”. Execuția repetată sau concomitentă a unor astfel de interogări poate bloca resursele serverului, ceea ce duce la degradarea performanței.

Apostroful de la început este folosit pentru a termina contextul șirului inițiat de interogarea originală (this.Name == '). Operatorul logic OR în JavaScript, se asigură că următoarea expresie (sleep(5000)) este evaluată indiferent de evaluarea anterioară (this.Name == ''). sleep(5000) este o comandă destinată să întrerupă execuția timp de 5 secunde. După funcția sleep(), este declarată și definită o variabilă „var test='” pentru ca următorul apostrof din filtru să închidă definiția, iar filtrul să fie corect din punct de vedere sintactic.

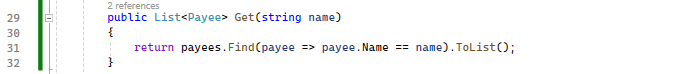


Bazele de date NoSQL permit de obicei executarea de interogări printr-un limbaj procedural precum JavaScript. Această secvență de cod execută metoda MongoDB Find pe colecția payees pentru a găsi documentele care se potrivesc cu filtrul dat de expresia Javascript cu operatorul $where, care verifică dacă numele documentului se potrivește cu parametrul dat.



În cazul necesității utilizării unui filtru JavaScript, este importantă asigurarea faptului că toate intrările utilizatorului (cum ar fi variabila nume) sunt validate și sanitizate.

În exemplul curent, pentru a mitiga riscul de securitate, ar trebui înlocuită utilizarea operatorului $where cu cod JavaScript construit dinamic, bazat pe intrarea utilizatorului prin utilizarea unei interogări parametrizate excutate prin metoda Find(), unde parametrul nume este utilizat în siguranță printr-o expresie lambda.



Filtrele JavaScript pot fi semnificativ mai lente decât operațiunile standard de interogare MongoDB. Acest lucru se datorează faptului că fiecare document trebuie evaluat de motorul JavaScript, ceea ce poate consuma multe resurse, în special pentru colecțiile mari. De asemenea, folosirea filtrelor JavaScript poate introduce riscuri de securitate deoarece poate fi executat cod JavaScript arbitrar prin atacuri de injecție dacă nu sunt sanitizate intrările în mod corespunzător.

Filtrele JavaScript din MongoDB, de obicei prin operatorul $where, ar trebui utilizate cu moderație și numai în cazuri specifice în care există o nevoie imperioasă de o logică personalizată care nu poate fi exprimată folosind operatori standard de interogare MongoDB și unde impactul performanței este acceptabil. Un exemplu concret poate fi găsirea comenzilor în care un calcul complex personalizat bazat pe diferite câmpuri determină dacă se potrivesc cu interogarea. Acest calcul poate implica mai mulți pași și condiții care depășesc capacitățile operatorilor standard.

db. orders.find({ $or: [{ quantity: { $gt: 50 } }, $or: [{ quantity: { $lt: 100 } }, });

db.orders.find({ $where: function() {

return (this.quantity > 50) && (new Date(this.date) > someReferenceDate) && someComplexCalculation(this);

}});

Un alt exemplu poate consta în filtrarea unor evenimente prin comparația cu o dată transformată printr-o funcție Javascript care nu este suportată în mod nativ în limbajul de interogare MongoDB.

db.events.find({ $where: function() {

return new Date(this.startDate).getDay() === 5; // select events that start on a Friday

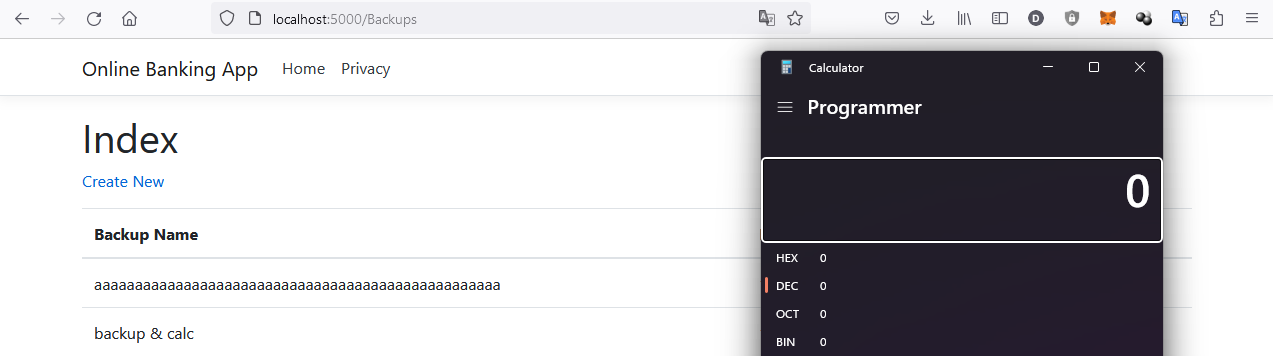
}});

### ATAC COMMAND INJECTION

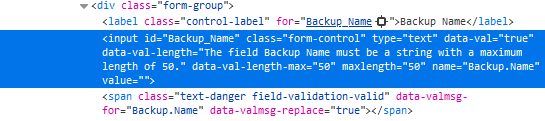
Aplicațiile web precum cele dezvoltate cu ASP.NET Core au o multitudine de componente și biblioteci care le permit să execute comenzi ale sistemului de operare în gazdă. Dacă nu este scris în siguranță, codul care compune și rulează aceste comenzi poate expune aplicația web ASP.NET Core la exploatare prin injecția de comenzi shell.

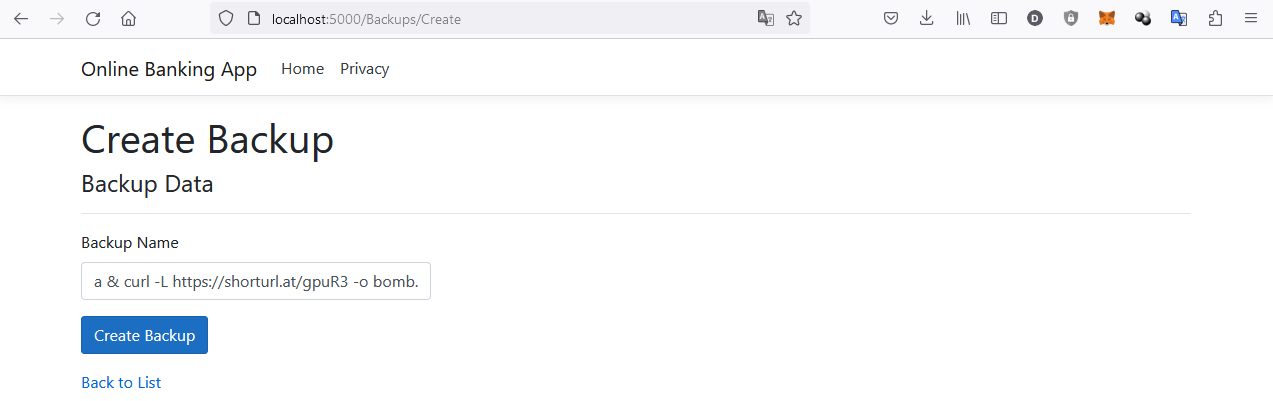
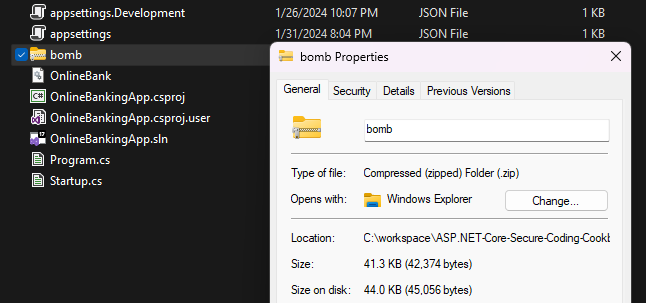
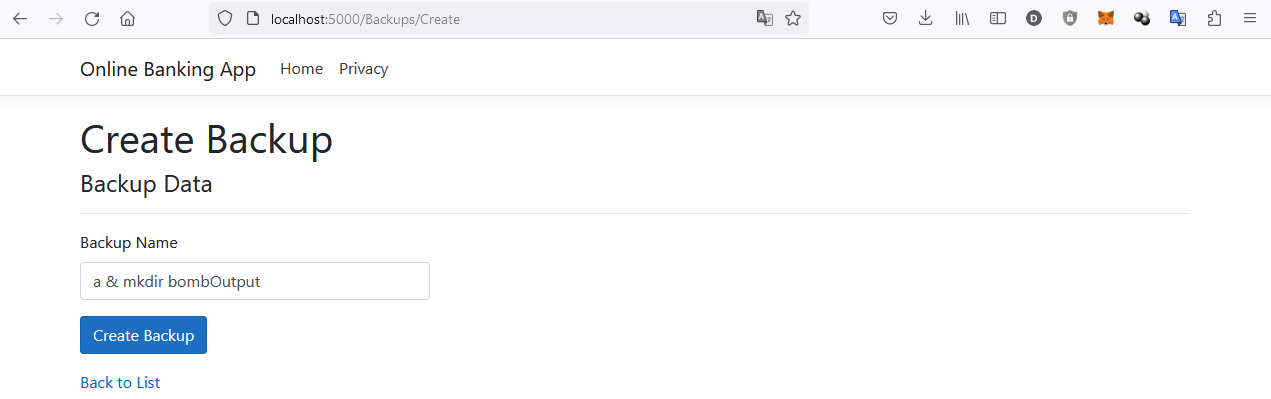
Pentru exemplificarea atacului va fi folosită o pagină web ce permite crearea unui backup al bazei de date de către un administrator. După introducerea numelui backup-ului și apăsarea pe butonul de creare, utilizatorul va fi redirecționat către lista de backup-uri create.

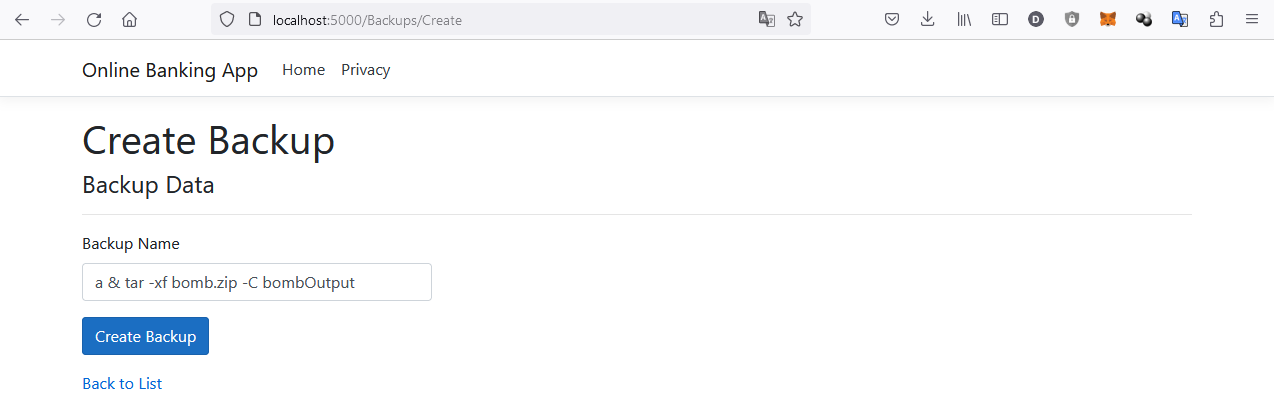
Pentru determinarea existenței vulnerabilității de tip Command Injection, numele ce va fi introdus va fi de forma backup & calc. Comportamentul aplicației a fost similar, utilizatorul fiind redirecționat către lista de backup-uri, însă de această dată a fost lansată în execuție și aplicația Calculator.

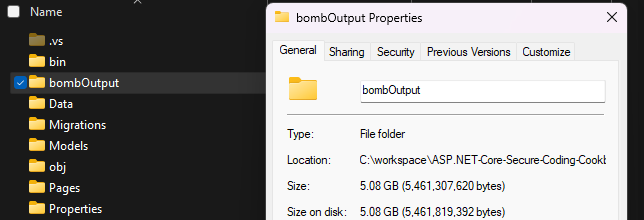


Dacă acest bug nu este gestionat corect, această problemă expune gazda la execuția de cod de la distanță cu consecințe mai grave decât în cazul precedent. Un exemplu ar fi descărcarea unei bombe ZIP și dezarhivarea acesteia pentru a declanșa un atac de tip Denial of Service care să pună monopol pe procesorul gazdei și să umple spațiul de pe disc cu informație inutilă. Analizând codul HTML al paginii se poate observa că există o limită de 50 de caractere a câmpului dedicat numelui backup-ului, ceea ce nu permite scrierea unei comenzi elaborate. Însă, folosind un URLShortener, aceasta a putut fi compactată. Ulterior descărcării arhivei, a fost creat un director pentru dezarhivare și a avut loc dezarhivarea.

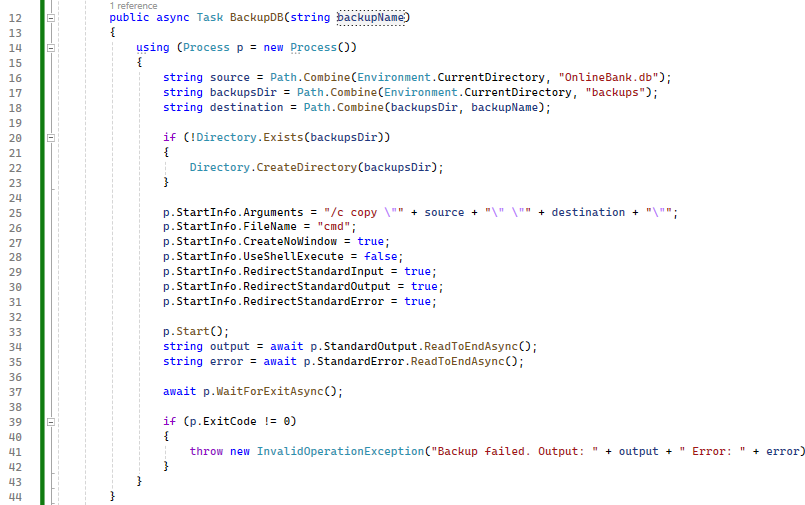


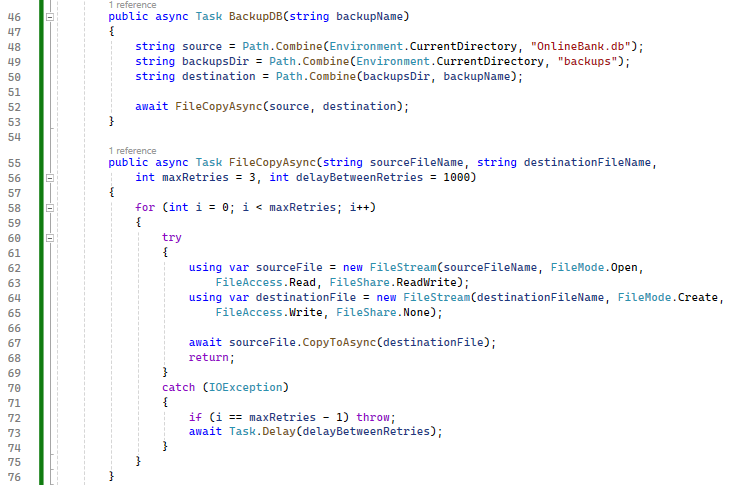




În interiorul metodei BackupDb este creat un nou proces folosit pentru a executa comanda copy în prompt-ul de comandă Windows (cmd.exe) pentru copierea fișierului de backup din locația sursă la destinație. Șirul de intrare dat ca parametru metodei BackupDb este folosit în construcția comenzii. Se așteaptă ca șirul de intrare să fie numele fișierului destinație, însă acesta poate fi manipulat astfel încât să includă și comenzi ce nu reprezintă în mod normal o valoare justă pentru un parametru. Lipsa validării și sanitizării ar putea cauza execuția de comenzi shell nedorite sub identitatea și autorizarea aplicației web.

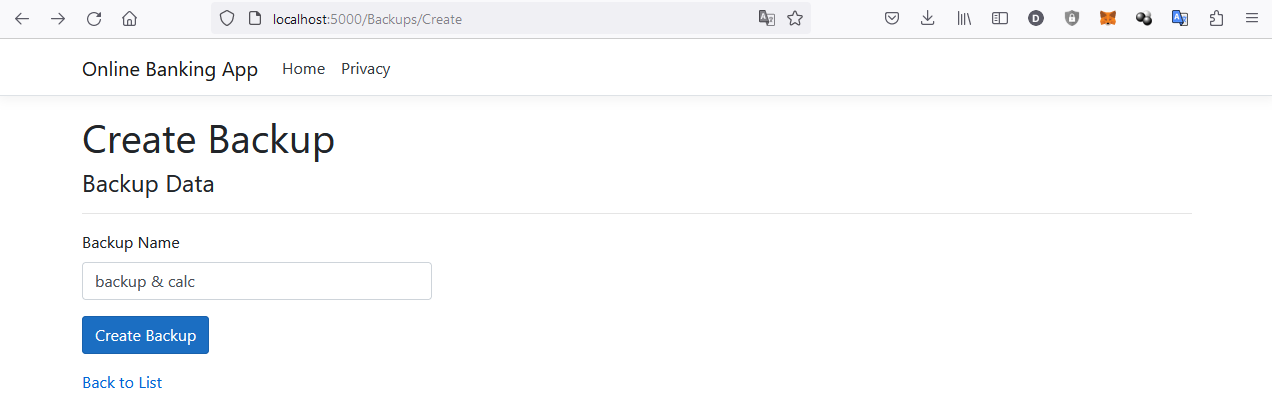


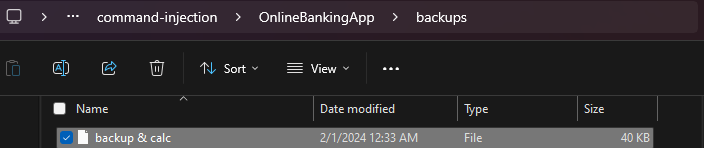
Pentru a remedia vulnerabilitatea și a evita execuția comenzilor shell nedorite poate fi adăugată o nouă metodă care realizează funcția de copiere a fișierelor, iar metod BackupDb să folosească metoda nou creată sau poate fi implementată o validare adecvată a șirului de intrare care să îl limiteze doar la caractere alfanumerice.



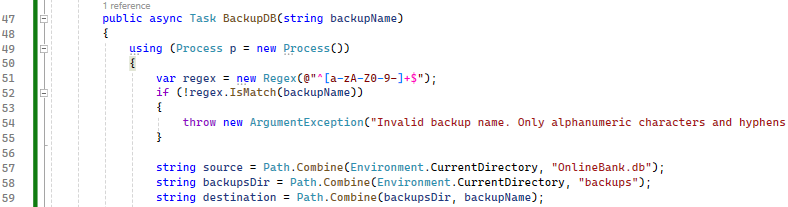
O funcție de copiere a fișierelor utilizează direct API-ul sistemului de operare pentru a gestiona fișierele. Această abordare este mai eficientă, deoarece evită consumul de resurse legat de crearea și gestionarea unui proces separat. Scrierea funcției de copiere a fișierelor poate face codul mai independent de platformă. Comenzile externe, cum ar fi copy în Windows Command Prompt sau cp în Unix/Linux, pot avea sintaxe sau comportamente diferite pe platforme. Rularea unui proces extern poate prezenta riscuri de securitate, mai ales dacă comanda sau parametrii sunt construiți din intrarea utilizatorului, ceea ce duce la potențiale vulnerabilități de injectare a comenzii. În schimb, o funcție de copiere a fișierelor care utilizează API-uri specifice limbii este mult mai sigură în acest sens

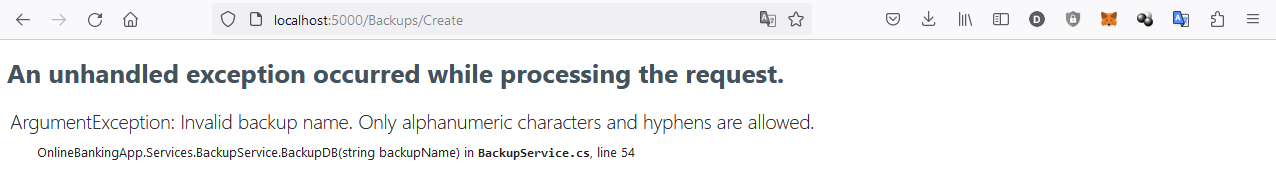
În cazul creării funcției FileCopyAsync() este necesar ca fișierul sursă să fie deschis cu flag-ul FileShare.Read pentru a permite altor procese să citească din fișier atunci când e deschis de aplicația web. În caz contrar, va fi aruncată o excepție de tip IOException care indică faptul că procesul nu poate accesa fișierul OnlineBank.db deoarece acesta este folosit de alt proces, respectiv de motorul SQL, care blochează accesul pentru o perioadă de timp. Metoda va încerca să efectueze o copie de câteva ori.





Folosind o expresie regulată pentru a crea un whitelist este o abordare bună pentru a reduce riscul de injectare a comenzii. Expresia regulată ^[a-zA-Z0-9]+$ asigură faptul că numele backup-ului este format numai din caractere alfanumerice, ceea ce reduce semnificativ riscul procesării intrărilor rău intenționate.



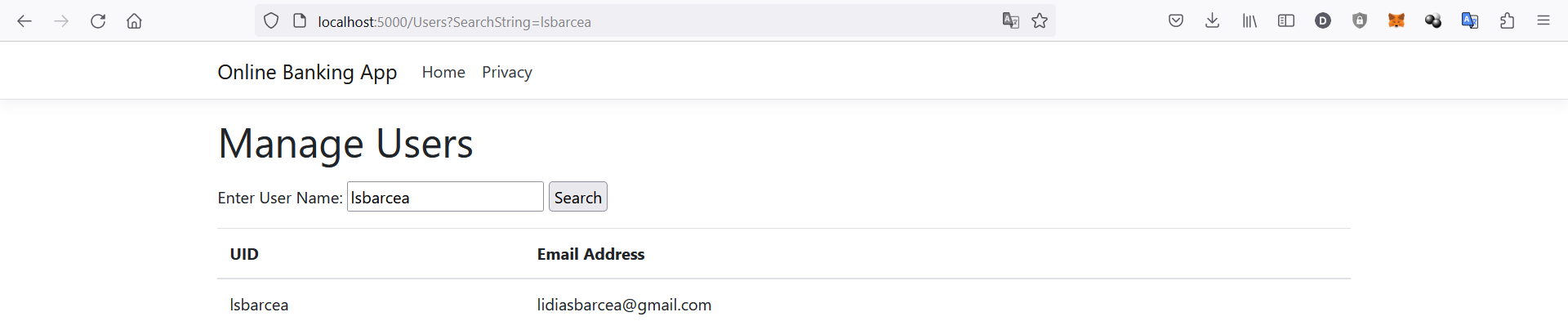


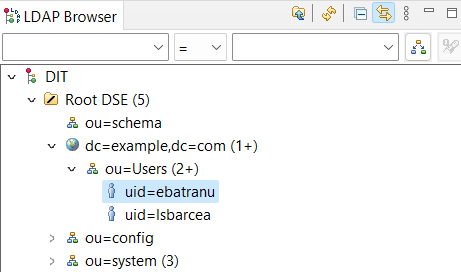
Injecția LDAP

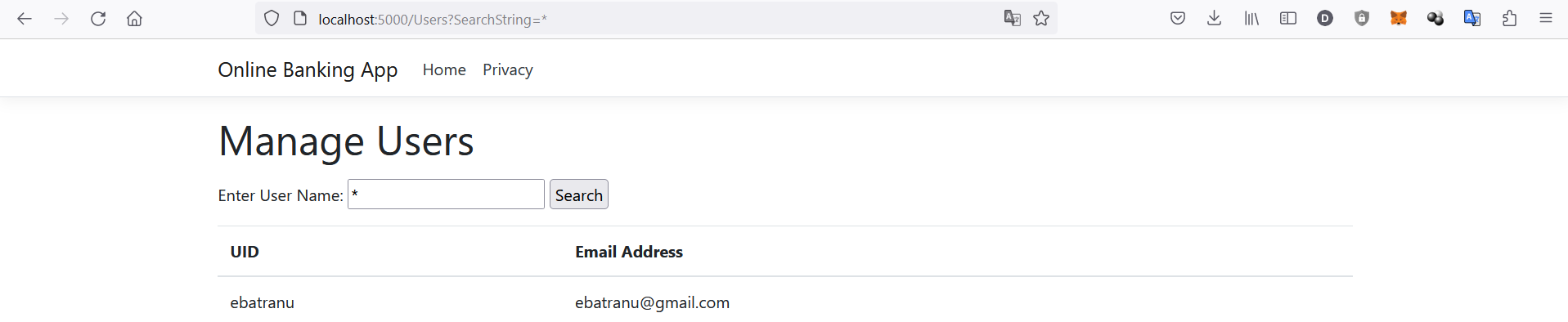
Descrierea vulnerabilității

LDAP este un protocol standard utilizat pentru a accesa servicii de directoare cum ar fi Microsoft Active Directory sau Apache Directory. Aplicațiile web folosesc LDAP pentru a obține informații despre utilizatori și grupuri din serverul de directoare, care servește și ca mijloc de autentificare. Această preluare a datelor de pe serverul de directoare LDAP este posibilă datorită limbajului de interogare LDAP și a filtrelor acestuia. Dezvoltatorii scriu cod pentru a compune aceste interogări. Construcția dinamică a interogării poate face codul vulnerabil la atacurile de tip injecție LDAP, atunci când intrarea concatenată furnizată utilizator nu este validată și sanitizată.

Pentru exemplificarea atacului va fi folosită o pagină web care permite unui utilizator cu rolul de administrator să caute mai multe detalii despre un anumit utilizator. Introducerea ID-ului utilizatorului vizat și apăsarea butonului Search va trimite o interogare LDAP către serviciul de directoare de unde vor fi preluate datele asociate utilizatorului care are ID-ul respectiv.



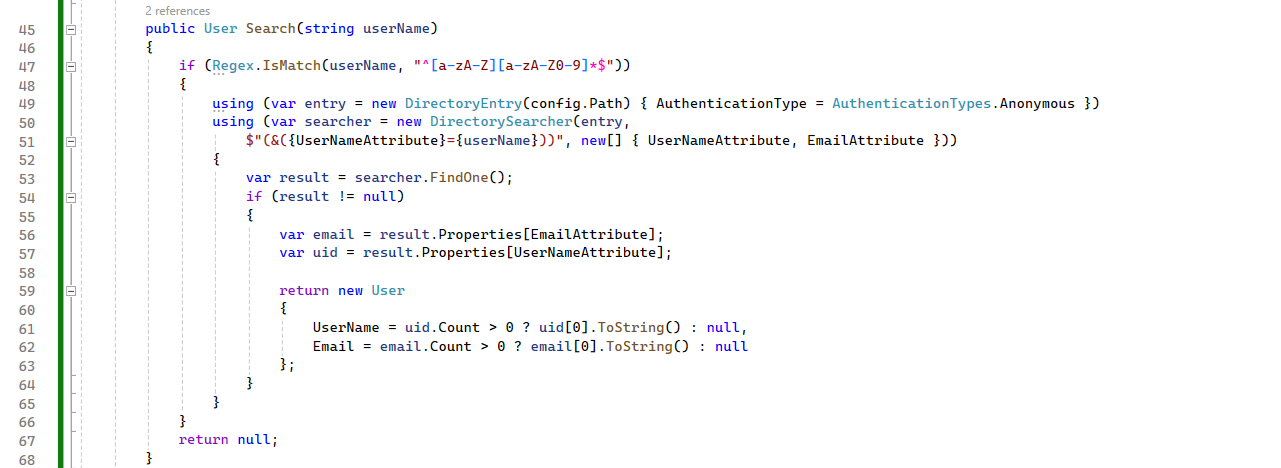




Interogarea (&(uid=\*)) este un filtru de căutare simplu care se potrivește cu orice obiect cu un atribut uid prezent, ceea ce înseamnă că ar putea returna toți utilizatorii din director. În cazul de față este folosită metoda FindOne() și implicit este returnat doar un rezultat: primul obiect cu atributul uid prezent.

O interogare mai complexă ar putea fi (&(uid=\*)(|(uid=\*))) ce reprezintă o tehnică de injecție mai sofisticată. Când este furnizată ca intrare necorespunzătoare poate închide efectiv condiția de interogare inițială și începe o nouă condiție care este garantată a fi adevărată. Acest lucru poate fi deosebit de util pentru ocolirea verificărilor de autentificare de tipul (&(uid=[userInput])(userPassword=[passwordInput])) ce rezultă în (&(uid=\*)(|(uid=\*)))(userPassword=[passwordInput])).

În cadrul metodei Search, interogarea LDAP este compusă dinamic și un filtru este concatenat cu valoarea introdusă de administrator. Parametrul userName nu este validat sau sanitizat, iar un actor rău intenționat poate exploata acest lucru prin injectarea de filtre suspecte care ar putea prelua informații sensibile de pe serverul de directoare.



Pentru a mitiga acest atac este folosită validarea parametrului userName ce constă în verificarea potrivirii acestuia cu o expresie regulată ce permite doar caractere alfanumerice. Implementarea validărilor trebuie inclusă ca parte a strategiei generale de programare sigură, pentru a verifica corectitudinea intrărilor furnizate de utilizator și pentru a proteja aplicația de atacurile prin injecție.

