Laboratorul 10

# Exercitiul 1

1. A
2. F
3. A
4. A
5. F
6. F
7. A
8. F

# Exercitiul 2

Localhost/login (hostat in apache prin xampp, aplicatia e in C:\Program Files\XAMPP\htdocs\login)

# Exercitiul 3

La fel ca la 2

# Exercitiul 4

Exemplul 1. Una din cele mai comune tipuri de exploatari pe care o poate folosi un atacator impotriva unei aplicatii web este un atac de tip buffer overflow. Acesta se intampla cand o zona din stack care trebuie sa pastreze o bucata de cod sau un string, primeste o bucata de cod sau un string prea mare, neverificand dimensiunea acestuia si astfel suprascriind o parte din stack cu bucata de cod sau de string in plus. Aceasta deschide posibilitatea rularii codului extern, creat de atacator, prin plasarea codului atacatorului in stack. Sunt greu de depistat si exploatat aceste vulnerabilitati insa sunt comune pentru ca exista foarte multe metode de a produce accidental o asemenea vulnerabilitate, precum utilizarea de librarii, framework-uri sau cod extern livrat dintr-o sursa ce nu se afla in controlul direct al aplicatiei; toate aceste surse de cod avand posibilitatea de a contine o vulnerabilitate de tip buffer overflow ce nu va fi cunoscuta programatorului aplicatiei. Un atacator poate crash-ui aplicatia in acest fel, sau ii poate lua controlul prin rularea propriului sau cod in cadrul aplicatiei.

Exemplul 2. Scurgeri de memorie. O scurgere de memorie se intampla atunci cand un program aloca o adresa de memorie pentru a stoca date, dar nu o elibereaza cand aceasta nu mai e necesara. Rezultatul este ca, in timp, toate resursele de memorie vor fi epuizate ceea ce va rezulta in esuarea programului sau comportamente imprevizibile cauzate de lipsa de memorie. Un atacator care descopera scurgeri de memorie intr-un program poate bloca programul prin umplerea memoriei sau profita de alte comportamente neasteptate a unui program in asemenea conditii.

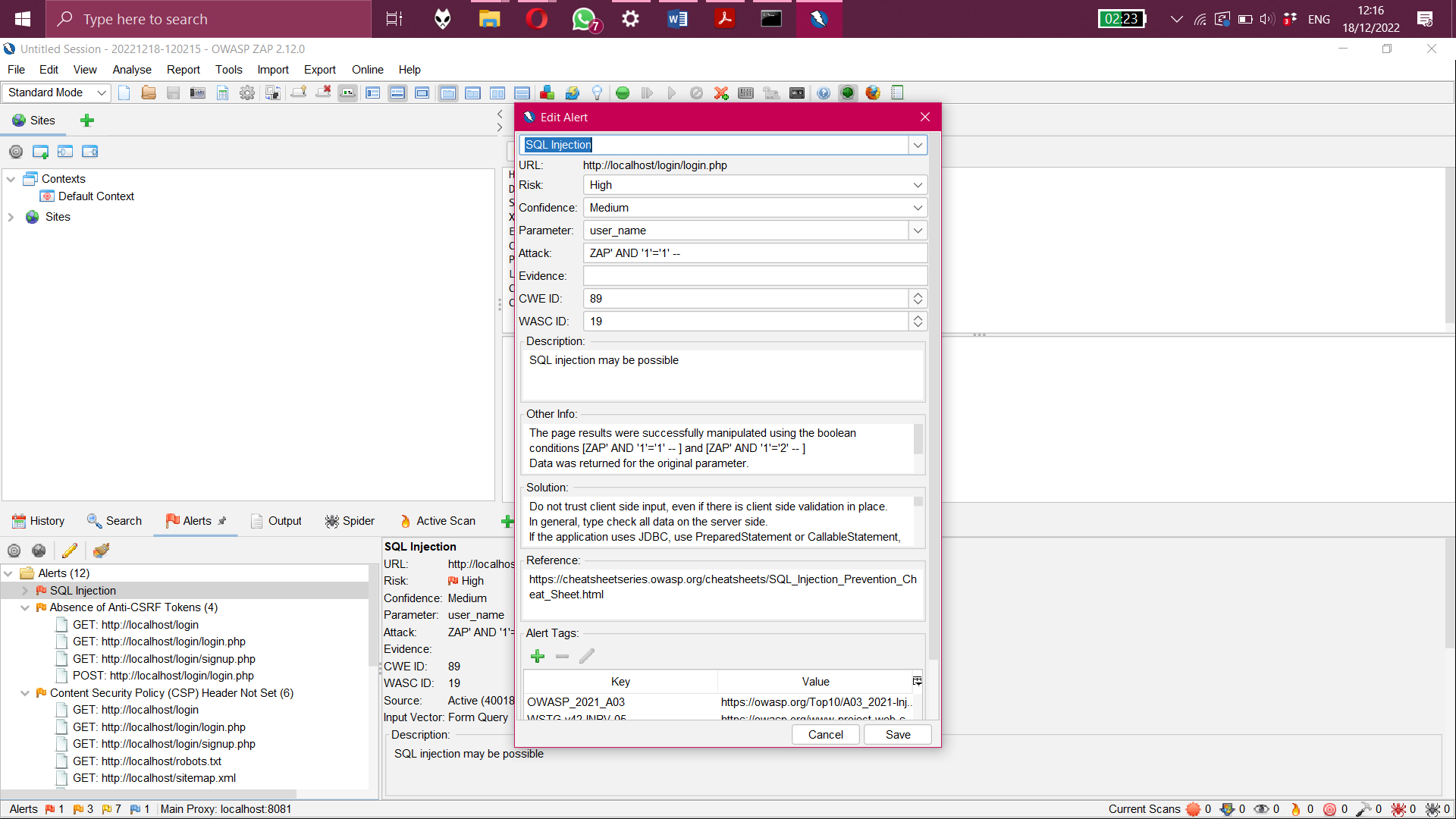
Exemplul 3. Absenta tratarii erorilor. Tratarea erorilor uzuale, de tip 404, 500, sau erori cauzate de javascript este esentiala, precum si o tratare default pentru oricare alta eroare neasteptata. Acest aspect este important pentru ca in situatia in care un adversar cauta erori in aplicatia noastra, si descopera o eroare netratata ce ii ofera informatii de tip stack trace, sau alte informatii despre baza de date folosita sau back end, informatii despre limbajul de programare, functiile sau librariile folosite, toate acestea pot fi indicii pentru adversar si sa ii spuna exact ce sa caute, ce tip de baza de date e folosit, ce tabele avem (poate vedea asta intr-un query ce poate fi inclus intr-un mesaj de eroare), ce librarii folosim care ar putea la randul lor contine unele vulnerabilitati. Nu trebuie sa punem asemenea informatii ce pot fi cruciale pentru securitatea aplicatiei la dispozitia unui adversar.

Exemplul 4. Utilizarea unor metode nepotrivite de criptografie. Fie ca e vorba de algoritmi creati de programator in mod specific pentru acest program, ce se pot dovedi a fi ineficienti, sau ca este folosit un algoritm ce nu mai este sigur datorita evolutiei naturale a tehnologiei si capacitatii calculatoarelor moderne de a descifra un asemenea algoritm criptografic invechit; utilizarea unor algoritmi criptografici nepotriviti rezulta in date importante ce ar trebui sa fie pastrate in conditii de siguranta dar nu sunt. Utilizarea algoritmilor precum MD5 sau SHA-256 poate rezulta in impresia ca unele date sunt sigure, cand de fapt nu sunt deloc, aproape ca si cand nu ar fi securizate deloc. Bineinteles, rezultatul este ca un atacator ce incearca asemenea algoritmi cunoscuti si nesiguri pe date dintr-o aplicatie este ca acesta poate obtine acces la informatii confidentiale, precum parole.

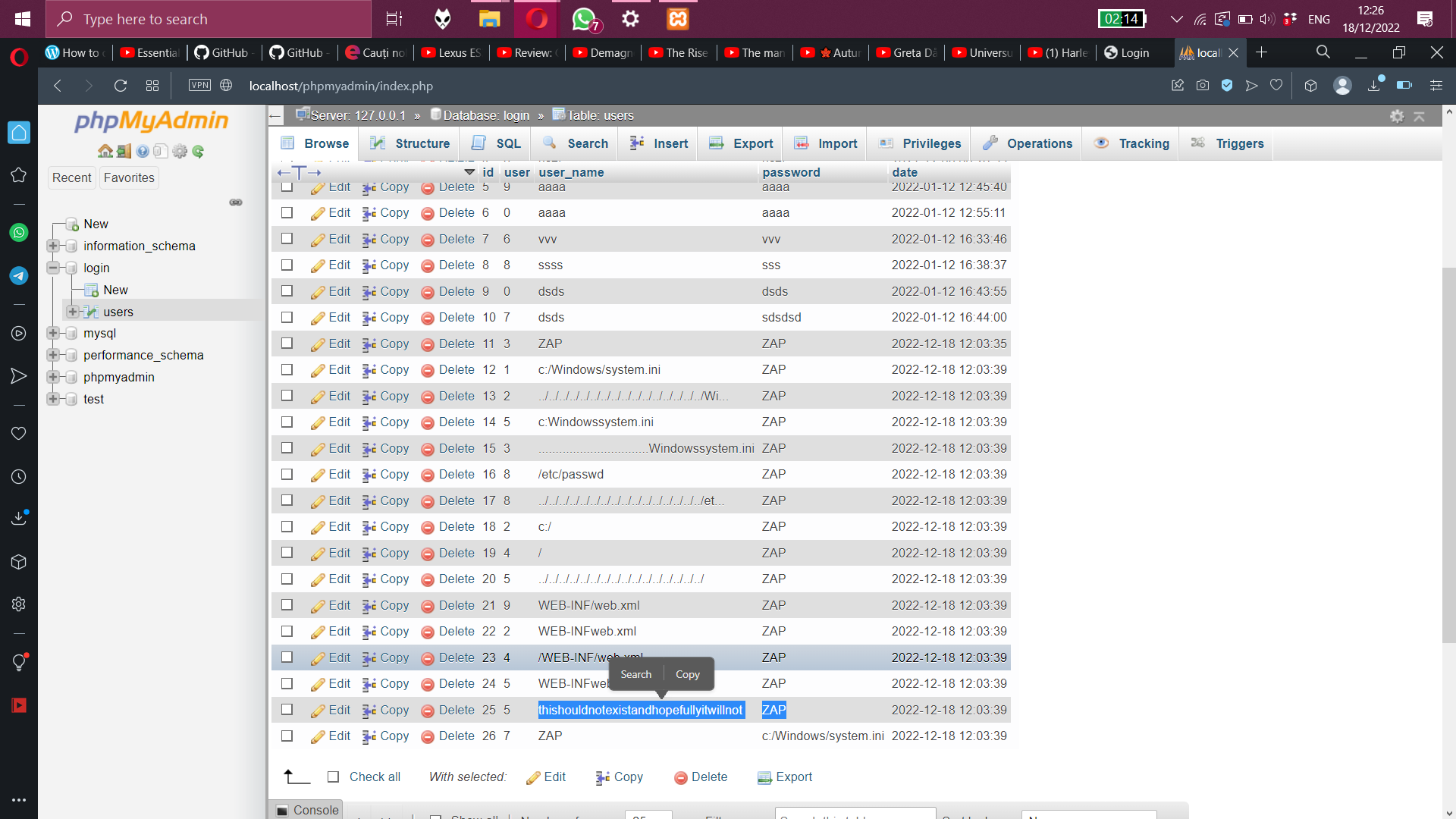
Exemplul 5. Parole hardcodate sau default. Pe multe sisteme, precum routere, diverse echipamente de retea, cartele sim etc exista parole default ce sunt identice intre toate dispozitivele de tipul acelui sistem. Multi utilizatori nu schimba aceste parole niciodata, prin urmare este statistic foarte posibil pentru un atacator sa obtina acces la multe sisteme prin incercarea parolei default pe multe exemplare din acel sistem. O metoda mult mai buna de securizare ar fi necesitatea de a crea o parola custom de catre fiecare utilizator al unui sistem.

# Exercitiul 5.

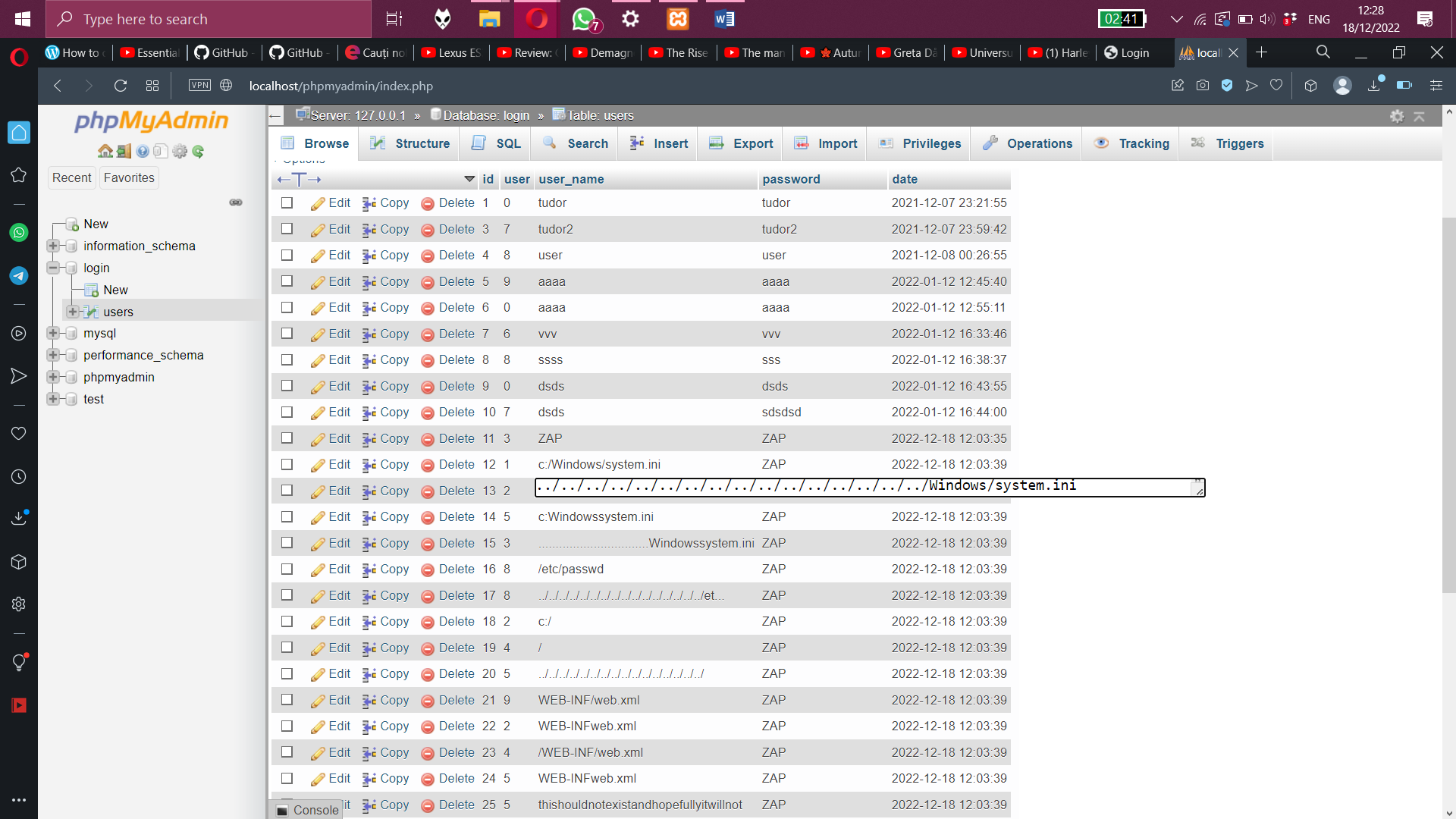
Am rulat OWASP (zap) pe site-ul meu, in mod automat, si am descoperit urmatoarele probleme de securitate:



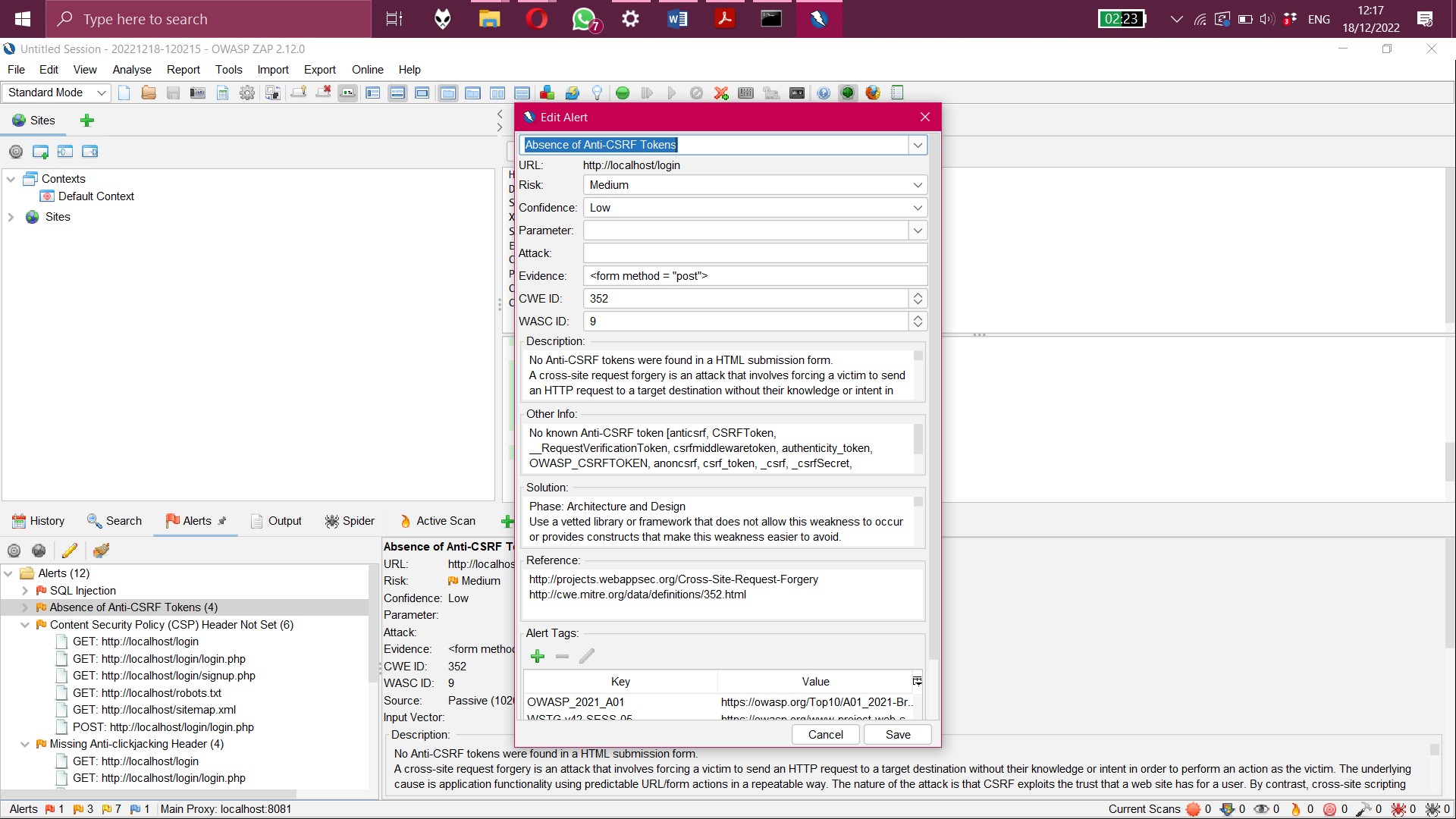
-Posibilitatea de Injectie SQL;



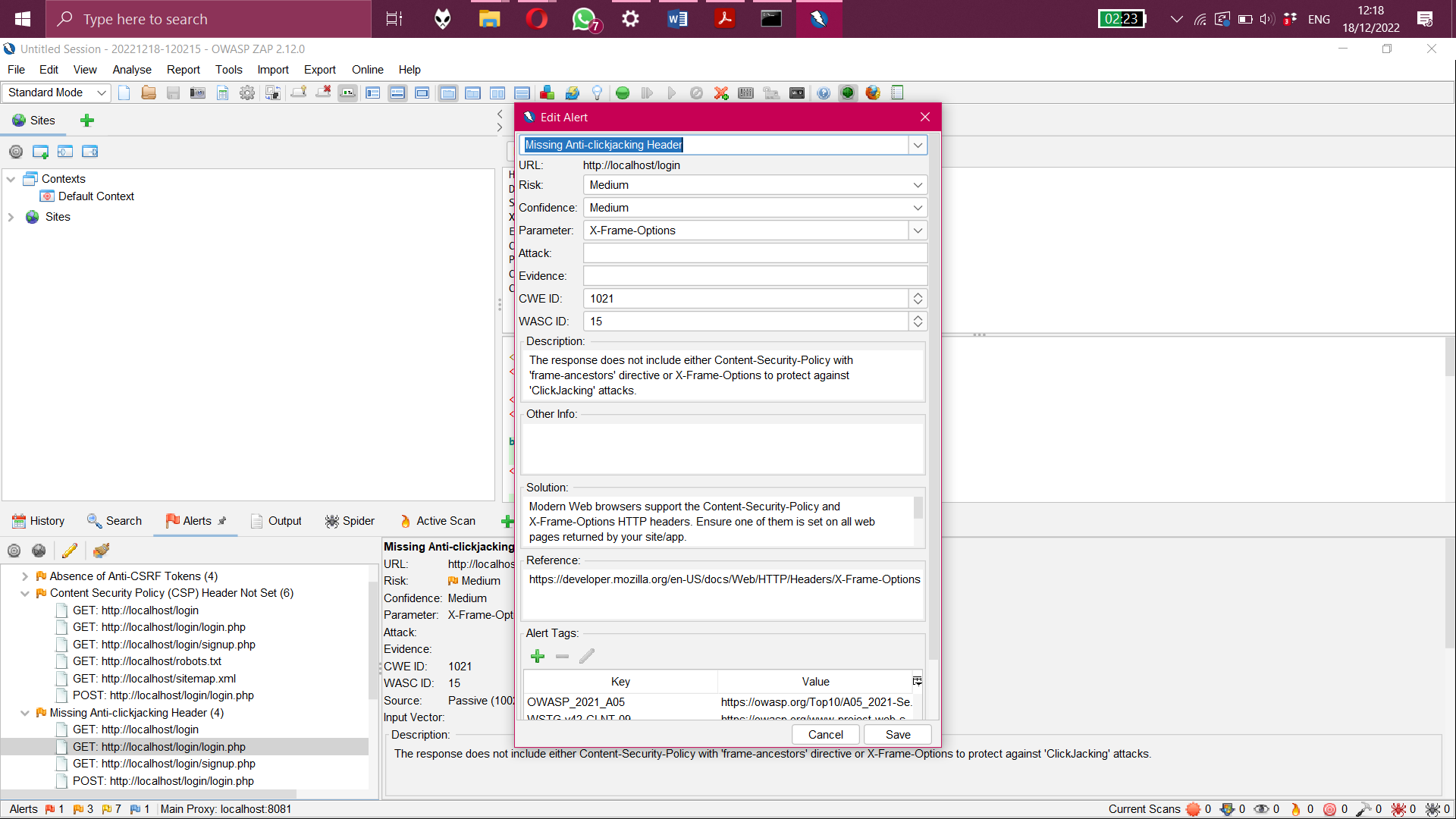
-In baza de date, in useri a aparut o intrare pe nume “thisshouldnotexistandhopefullyitwillnot” cu parola ZAP, probabil semnificand o injectie sql;



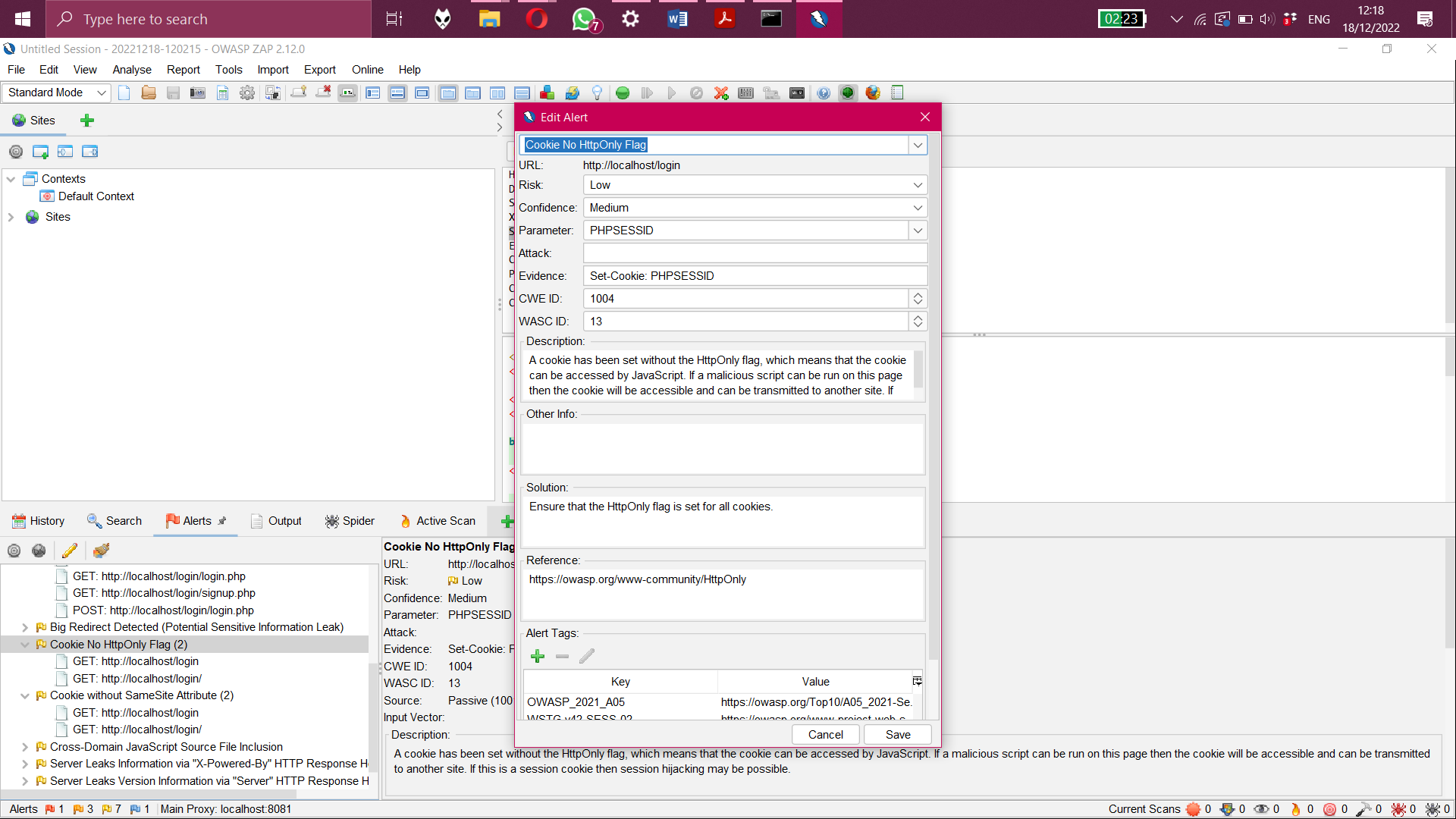
-Putem vedea cum incearca sa acceseze fisiere de sistem;



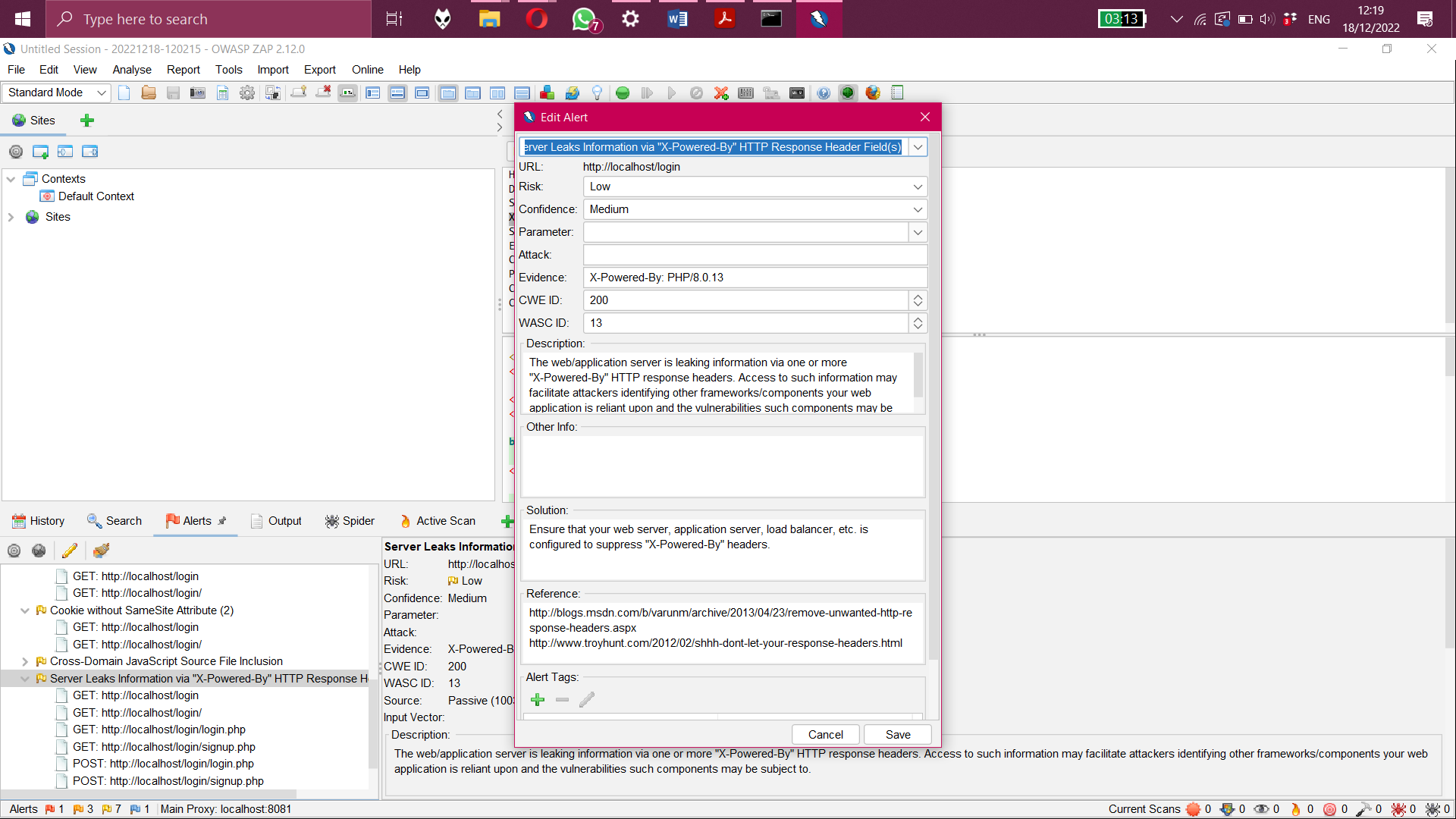
-Posibilitatea de Cross Site Request Forgery din pricina lipsei unor token-uri de securitate dintr-un formular;



-Posibilitatea unor atacuri de tip ClickJacking;



-Cookie-uri ce nu au flag-ul HttpOnly, ceea ce inseamna ca pot fi accesate de cod javascript si manipulate;



-Scurgeri de informatii despre server;

Dar si alte vulnerabilitati. In total a gasit 12 vulnerabilitati, din care posibilitatea unei injectii Sql este cea mai grava.