Kódolás és IT biztonság (VIHIBB01)
Laboratóriumi gyakorlat

Szerver oldali biztonság

Szerző:
Gazdag András

2023. október 4.

Tartalomjegyzék

1. Labor célja

A labor gyakorlat során a kód beszúrásos támadások veszélyeit fogja megismerni, különös tekintettel a web alkalmazásoknál felmerülő problémákra. A cél, hogy közvetlen tapasztalatot szerezzen egy való élethez közeli alkalmazáson keresztül egy támadás lehetséges veszélyéről és hatásáról. Az ilyen típusú támadások leggyakrabban az adatbázis kommunikáció során merülnek fel, így a labor alatt is ennek a funkcionalitásnak a megtámadása a cél.

2. Háttéranyag

Egy kód beszúrásos támadás során a támadó kártékony kódot tud bejuttatni egy sérülékenységen keresztül, amit a rendszer az eredeti alkalmazás kódjával együtt végrehajt. Ilyen támadások közé tartozik, amikor a támadó egy sérülékenységen keresztül megvalósít egy operációs rendszer hívást, egy külső alkalmazás elindítását shell parancs segítségével, vagy egy kommunikációt az adatbázis szerverrel (SQL utasítások felhasználásával). Teljes szkripteket (pl: Python, Perl, stb.) lehet bejuttatni, és végrehajtani egy rosszul megírt alkalmazáson keresztül. Minden helyzetben, ahol interpretált nyelv segítségével készül egy alkalmazás, így a forráskód valós időben kerül értelmezésre és végrehajtásra, egy kód beszúrásos támadás veszélye fennáll.

2.1. Adatbázis motorok

Adatbázis motornak nevezzük azt a szoftver komponenst, amelyet egy adatbázis kezelő rendszer használ az adatok beszúrásához, kiolvasásához, módosításához, vagy törléséhez. A legtöbb adatbázis kezelő rendszer definiál valamilyen saját APIt, aminek a segítségével a felhasználó hozzáférhet az adatbázis motor által kezelt adathoz a felhasználói felület manuális használata nélkül.

2.1.1. Relációs adatbázisok

A relációs adatbázis egy olyan adatbázis típus, amely valamilyen kapcsolatban álló adatok tárolására lett kifejlesztve. A felépítés a relációs adatmodellre épít, amely egy intuitív és átlátható formája az adatok táblában tárolásának. Egy táblában minden sor egy rekordnak felel meg, amelyet egy egyedi érték

(kulcs) azonosít. Az oszlopok jelentik az adatok attribútumait, amelyek általában a legtöbb rekord esetén mind ki is vannak töltve.

A relációs modell egy standardizált módja az adatok tárolásának és manipulálásának, melynek nagy erőssége, hogy a táblákban tárolt adatleírást könnyű átlátni, és használni. Idővel egy másik erőssége is megjelent a technológiának, ez pedig a strukturális lekérdező nyelv (SQL - Structured Query Language). Hosszú időn át, ez a nyelv volt az egyeduralkodó az adathozzáférés terén. Relációs algebra alapon megvalósítva, az SQL nyelv egy konzisztens matematikai leíró nyelv, aminek a segítségével hatékony lekérdezések küldhetőek az adatbázisnak.

2.1.2. NoSQL adatbázisok

A NoSQL adatbázisok lazább strukturális felépítést engednek meg. Mivel kevesebb a megkötés az adatok felé relációs szempontból, így a NoSQL adatbázisok nagyobb teljesítményt és skálázhatóságot tudnak elérni bizonyos esetekben. Biztonság szempontjából azonban, a relációs adatbázisokhoz hasonlóan, ezek is lehetnek kód beszúrásos támadás célpontjai, annak ellenére, hogy nem SQL nyelvet használnak az adathozzáféréshez. Sőt, a kommunikáció tipikusan procedurális nyelven történik (pl: JavaScript), így egy támadás hatása még súlyosabb is lehet a deklaratív SQL nyelv elleni támadásokhoz képest.

A NoSQL adatbázis hívások az alkalmazás fejlesztéséhez használt programozási nyelven készülnek. A kommunikáció egyedi API hívásokkal történik, amelyek néha megkövetelnek megfelelően megformázott adatot (pl: XML, JSON, LINQ, stb.). Egy ilyen formátumban elrejtett támadás nem feltétlenül fog fennakadni az elsődleges nyelv által használt biztonsági szabályokon. Például, a HTML vezérlőkarakterek kiszűrése, úgy mint a `<>& ;` karakterek, nem fognak megállítani egy JSON API elleni támadást, ahol a speciális karakterek közé tartoznak még a `\{}:` karakterek is.

Manapság már több mint 150 különböző NoSQL adatbázis közül választhatnak a fejlesztők, amelyek a programozási nyelvek széles skáláján nyújtanak API támogatást. Ezek általában mind valamennyire eltérő funkcionalitásokat valósítanak meg, különböző megszorítások mellett. Mivel nincs egy egységes nyelv a NoSQL adatbázisok kezelésére, így az egyes támadások sem lesznek egyformán hatásosak mindegyik ellen. Emiatt, aki NoSQL elleni támadást szeretne megérteni, első lépésként, a háttérben használt konkrét technológia sajátosságait (pl: szintaxis) kell megismernie. Van egy jelen-

tős eltérés a kétféle adatbázis típus elleni támadás között: NoSQL esetben nem magától értetődő, hogy a támadás hol fog végrehajtódni. Egy SQL beszúrásos támadást mindig az adatbázis motor értelmez. Egy NoSQL elleni támadás, a használt API és adatmodell függvényében, vagy az adatbázis szintéjén, vagy akár még az alkalmazás szintjén is végrehajtódhat.

Tipikus NoSQL elleni támadás akkor szokott előfordulni, ha a támadó által módosított sztring egy NoSQL API hívásba közvetlenül felhasználásra kerül.

2.2. Az SQL nyelv

Az SQL, ami a Structured Query Language rövidítése, egy strukturális lekérdező nyelv. Ezt a nyelvet használja a legtöbb relációs adatbázis a kommunikációra. SQL utasítások segítségével lehet adatot beszúrni, lekérdezni, módosítani, vagy törölni. Az SQL nyelvet használja, többek között, a népszerű adatbázis motorok közül az Oracle, a Microsoft SQL Server, MySQL, és az SQLite is. Habár mindegyik motor hivatalosan az SQL nyelvet használja, az egyes megvalósítások kis mértékben eltérnek a motor specifikus kiegészítések miatt. Így például egyes esetekben a komment kezdetét jelző karakter eltér. Az alap utasítások, úgy mint `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CREATE` és `DROP` megegyezik mindegyik implementációban, így a legtöbb gyakorlati probléma megoldása során nem érezhető a különbség. A labor teljesítéséhez az SQL nyelv alapvető ismerete szükséges, amely elsajátítható például erről az oldalról: https://www.w3schools.com/sql/default.asp.

2.3. Támadások

A beszúrásos támadások közül az SQL, NoSQL és OS command injection, valamint az ORM vagy LDAP injection szokott a leggyakrabban előfordulni. Ezeknek a támadásoknak az alap ötlete elég hasonló. A sérülékenység alapja mindig az, hogy az legitim kód és a külső forrásból származó adat összekeveredik a feldolgozás során.

2.3.1. SQL Injection

Az SQL injection vagyis az SQL beszúrásos támadás a kategóriának a legismertebb és legjobban elterjedt példája. Egy ilyen sérülékenység kihasználásához a támadónak először találnia kell egy paramétert, amit a web alkalmazás

továbbít az adatbázis felé valamelyik kérés során. Ezután, ha a támadó képes egy speciálisan megválasztott értéket megadni a paraméter értékéül, akkor ráveheti az alkalmazást, hogy egy módosított kérést küldjön el az adatbázis felé. Egy ilyen támadás végrehajtása nem kíván meg speciális szakértelmet, és a felmerülő nehézségek megoldására egyre több automatizált eszköz áll rendelkezésre.

Az ilyen sérülékenységek az alkalmazások minden részében elfordulhatnak, így megtalálásuk a triviálistól a rendkívül nehézig változhat. Egy sikeres támadás hatása szintén nagyon széles lehet, egyszerűbb adatszivárgástól az adatbázis teljes elvesztéséig minden. Általánosságban, egy bonyolultabb alkalmazás esetén érdemes azt feltételezni, hogy a forráskód tartalmaz ilyen sérülékenységet, mivel erre van nagyobb valószínűség.

Egy alkalmazás sérülékeny, ha a következő feltételek közül valamelyik teljesül:

- A felhasználó által megadott adatot nem ellenőrzi az alkalmazás mielőtt azt használja (input validation és filtering hiánya).
- Dinamikus lekérdezések vagy nem-parametrikus lekérdezések vannak használatban a környezetnek megfelelő escape-elés nélkül.
- Felhasználótól származó adat kerül felhasználásra egy ORM rendszer használata során, további érzékeny adatok lekérdezéséhez.
- Felhasználótól származó adat van ellenőrzés nélkül kézzel hozzáfűzve egy SQL utasításhoz (dinamikus lekérdezés vagy tárolt eljárás is lehet sérülékeny).

Egy tipikus sérülékeny és emiatt hibás megvalósítása egy SQL lekérdezésnek:

```
String query = "SELECT * FROM users WHERE name = '" + userName
               + "' AND password = '" + password + "'";
```

Ez az implementáció funkcionálisan elfogadhatóan működik, amíg a felhasználó ténylegesen csak felhasználónevet és jelszót ad meg bemenetként. Egy támadó azonban felhasználhatja bármelyik mezőt arra, hogy egy SQL injection támadást valósítson meg. Egy ilyen esetben a támadó megadhatja bemenetként például a következő értékeket:

- username: `whoever`
- password: `asd' OR 'a'='a`

Ezzel módosul az alkalmazás viselkedése. Az így létrejött lekérdezés már eltér az eredetitől:

```
SELECT * FROM users WHERE name = 'whoever' AND
                          password = 'asd' OR 'a'='a'
```

A pluszban hozzáfűzött `OR 'a'='a'` rész alapjaiban módosítja az eredeti lekérdezés funkcióját. Ahelyett, hogy csak egy rekordot adna vissza az adatbázis, az összes rekordot visszaadja a táblából. A támadó így meg tudta kerül a jelszó ellenőrzést az alkalmazásban. Amennyiben az alkalmazás kódja a válasz első sorát használja fel automatikusan (mivel arra számít, hogy csak 1 rekordot fog visszakapni), akkor a támadó több célt is elérhet egyből: egy másik felhasználó nevében jelentkezik be. A felhasználók közül az első tipikusan az adminisztrátor szokott lenni, így ez még további különösen nagy problémát is okozhat.

Egy másik tipikus trükk, amit a támadók fel szoktak használni, az az SQL komment karakter használata. Ez a karakter, ahogy korábban is volt már róla szó, eltér az egyes adatbázis motorok között. A komment karakter használata, egy támadás során, képes "megállítani" a lekérdezés végrehajtását, mivel a komment karakter hatására megáll a lekérdezés még hátra lévő részének a feldolgozása. A következő példa szemlélteti ezt az esetet:

- username: `name' OR 'a'='a' #`
- password: `whatever`

Ezen bemenetek hatására az alábbi SQL lekérdezés fog előállni.

```
SELECT * FROM users WHERE name = 'name' OR 'a'='a' #' AND ...
```

A feldolgozás során az adatbázismotor nem fogja már kiértékelni a jelszóra vonatkozó feltételt.

Megjegyzés: A korábban bemutatott példakódok nem teljesen helyesek szintaktikailag. A céljuk a koncepciók bemutatása, nem pedig konkrét támadások terjesztése.

2.3.2. A támadás megakadályozása

Kód beszúrásos támadást a leghatékonyabban forráskód ellenőrzéssel, és alapos teszteléssel lehet elkerülni. A tesztelés során érdemes minden paramétert, fejléc mezőt, URL részt, sütit, JSON dokumentumot, SOAP vagy XML bemenetet tesztelni, mert ezek játszanak tipikusan szerepet egy ilyen támadásban. Ilyen teszteléseket érdemes a CI/CD folyamat részként megvalósítani, hogy minden forráskód módosításkor ezek automatikusan lefussanak.

Egy kód beszúrásos támadás megállításának a kulcs kérdése, hogy sikerüljön az eredeti kódbázist, és a feldolgozandó adatot elkülönítve kezelni.

- Ideális esetben a fejlesztés során csak megbízható APIt használjunk, amely nem interpretálja a bemenetet. Erre a legelterjedtebb opció egy parametrizált interfészt (parameterized interface) használata. SQL injection esetén, ezt a legtöbb keretrendszerben vagy nyelven prepared statementnek nevezik. Emellett, szintén segíthet egy ORM (Object Relational Mapping) keretrendszer használata, azonban fontos figyelni rá, hogy ez nem jelent automatikus védelmet, így érdemes ellenőrizni, hogy a keretrendszer fejlesztői hogyan kezelik a problémát.

- Egy engedélyező lista (allow list) használata is javít a helyzeten, az input validáció részeként. Teljes védelmet ez a megoldás általában nem ad, mivel szükséges lehet néhány speciális karakter engedélyezése az alkalmazás funkcionalitásához.

- Ha az előző két módszer nem alkalmazható valamilyen okból kifolyólag, akkor a dinamikus lekérdezések összeállítása során escape-eljük a bemenetet, az adott interpreter által használt szabályoknak megfelelően. *Megjegyzés:* Az SQL adatstruktúra neveit (pl: tábla-, vagy oszlopneveket, stb.) nem lehet escape-elni, emiatt, ha egy struktúra neve külső adatként érkezik az alkalmazásba, az komoly biztonsági kockázatot jelent. Ez a probléma gyakran szokott előfordulni jelentést vagy kimutatást készítő alkalmazások fejlesztése során.

- A `LIMIT` és ahhoz hasonló kulcsszavak használata szintén javasolt, mivel ezzel korlátozni lehet egy támadás során, az egy lépésben kiszivárgó adatok mennyiségét.

2.3.3. Denial-of-service (más néven szolgáltatás megtagadás)

Egy denial-of-service (DoS) támadás során, a támadó célja, hogy leterhelje az elérhető erőforrásokat annyira, hogy a legitim felhasználók számára már ne maradjon szabad kapacitás. Például elképzelhető, hogy egy támadó nagy terhelést generál egy átlag számítógép processzora és hálózati kártyája ellen, amivel rövidebb-hosszabb kiesést érhet el a gépet használni próbáló felhasználó munkájában: akadozhat az internet elérés, email letöltés, vagy bármelyik közepes, vagy nagyobb erőforrást igénylő feladat végrehajtása.

3. Feladatok

Ez a labor gyakorlat 3 feladatból áll. Az egyes feladatok nehézség szerint vannak növekvő sorrendben, így a megoldásuk a leírt sorrendben javasolt. A labor során a Juice Shop alkalmazást kell használni.

Juice Shop beállítása és elindítása

Néhány feladat alapbeállítások mellett nem elérhető, mivel azok potenciálisan veszélyesek lehetnek. A mérés során használt környezetben engedélyezhetjük ennek a működését, mivel ez kifejezetten oktatási-gyakorlási célra van. Az ehhez szükséges lépések a következőek:

- Nyissa meg a következő konfigurációs fájlt:
  `nano ~/juice-shop/config/default.yml`
- A 81. sorban módosítsa az ott található értéket `true`-ra:
  `safetyOverride: true`
- Mentse el a konfigurációs fájlt, majd zárba be a szerkesztőt.
- Lépjen vissza a Juice Shop alap könyvtárába:
  `cd ~/juice-shop/`
- Indítsa el a Juice-shop alkalmazást:
  `npm start`
- Az indulás a gépek terheltségétől függően pár percig is eltarhat.

A Juice Shop interaktív módban fut, így az eddig használt terminálban nem fog tudni további parancsokat kiadni. Ha bezárja a terminál ablakot, akkor leáll az alkalmazás is. A mérés végén a `CTRL + C` billentyűkombináció segítségével lehet majd leállítani az alkalmazást szabályosan.

Az indulás után az alkalmazás a `http://localhost:3000`-es címen érhető el. Aki sikeresen teljesítette az első mérést, annak a `http://juice-shop.crysys.hu` címen is elérhető a weboldal.

Támadások kivitelezése

A sérülékenységek elleni támadásokat először érdemes egy egyszerűbb verzióban kipróbálni, hogy kiderüljön, hogy valóban a megfelelő helyen próbálja megoldani a feladatokat. Erre egy jó példa, hogy egy SQL injection támadást először csak egy olyan minimális bemenettel érdemes tesztelni mint például a `'` vagy a `';` karakterek. Az ilyen bemenetekre adott válaszok elemzésével megállapíthatja, hogy a sérülékenység valóban kihasználható, és így ezután érdemes a feladat megoldását célzó összetettebb bemenetet összeállítani.

3.1. Vezetett Feladat

Cél: Adminisztrátori hozzáférést szerezni a rendszerhez! Találjon egy sérülékeny bemenetet, ahol a hibát kihasználva képes az adminisztrátor nevében bejelentkezni. A feladat SQL injection vagy jelszó próbálkozás segítségével is megvalósítható.

Segítség: A bejelentkezést végző szerver oldali kód az alábbi linken elérhető: https://github.com/juice-shop/juice-shop/blob/master/routes/login.ts

3.2. Vezetett Feladat

Cél: Hagyja a szerver végre pihenni egy kicsit! A megoldandó feladat arról szól, hogy a folyamatos terhelés után végre pihenhessen a szerver egy kicsit, ami a felhasználók szempontjából egy leegyszerűsített denial-of-service támadásként érzékelhető.

A Juice Shop web alkalmazás egy MongoDB nevű NoSQL adatbázist használ információk tárolására. Ebben található többek között az egyes termékek értékelése. A feladat során a először az értékelések betöltéséhez használt hálózati kommunikáció megkeresése szükséges, majd a kérésben található sérülékenység megtalálása után, a támadás végrehajtása. A támadáshoz a MongoDB által használt sleep függvény használata javasolt, melynek dokumentációja megtalálható itt: https://docs.mongodb.com/manual/reference/method/sleep/.

Segítség: A szerver oldali kód, amely kiszolgálja a kéréseket megtalálható ezen a linken: https://github.com/juice-shop/juice-shop/blob/

master/routes/showProductReviews.ts

Megjegyzés: a szervert elárasztani nagy számú kérdésekkel nem fogja megoldani ezt a feladatot. Ezzel valószínűleg csak a teljes környezet túlterhelését lehet elérni, ami nem számít helyes megoldásnak.

3.3. Önálló Feladat

Cél: Jim jelszavának a megváltoztatása az 'Elfelejtett jelszó' funkció segítségével. Használja az alkalmazásba épített elfelejtett jelszó funkciót, hogy módosítsa Jim (jim@juice-sh.op) jelszavát. A jelszó helyreállítás rendelkezik egy beépített védelemmel, amely az engedély nélküli módosításokat hívatott megállítani: egy biztonsági kérdésre kell válaszolni. Ezt a védelmet azonban social engineering támadás segítségével ki lehet játszani.

Jim az egyik lehető legrosszabb biztonsági kérdést választotta a regisztrációja során. Ez, azzal együtt, hogy Jim egy híresség semmilyen védelmet nem nyújt a felhasználói fiókja számára, mivel a kérdésre a választ ki lehet találni. A feladat megoldásához a következő lépések javasoltak:

- Keresse meg az 'Elfelejtett jelszó' funkciót a web alkalmazásban!
- Jim e-mail címének a megadása után jegyezze meg Jim biztonsági kérdését!
- Keressen olyan Juice shop termékeket amelyekhez írt Jim értékelést! A második oldalon érdemes keresgélni. Az így talált információ árulkodhat Jim személyéről.
- Az első feladatból ismert SQL injection módszerhez hasonlóval jelentkezzen be Jim nevében, majd keresse meg az általa megadott szállítási címeket!
- Az értékelésekből és a címekből megszerzett információk alapján derítse ki az interneten, hogy ki Jim valójában.
- A Jimről található elérhető információkból válaszolja meg a korábban megtalált biztonsági kérdést!