Kódolás és IT biztonság (VIHIBB01)
Laboratóriumi gyakorlat

Kliens oldali biztonság

Szerző:
Gazdag András

2020. október 13.

Tartalomjegyzék

1. Labor célja

A labor gyakorlat során a web fejlesztésben a kliens oldalon felmerülő problémákat fogja megismerni. A cél, hogy tapasztalatot szerezzen a JavaScript nyelvvel, és megértse a veszélyét a JavaScript injection (beszúrás) támadásnak, vagyis a Cross-Site Scripting (XSS) támadásnak. A munka során a már korábban megismert Juice-Shop alkalmazással szemben kell támadásokat végrehajtani, amely során demonstrálni kell egy XSS hatását.

2. Háttéranyag

2.1. JavaScript[1]

A JavaScript nyelv (JS), egy interpretált, valós időben forduló programozási nyelv. Kliens oldali alkalmazások fejlesztésére használják első sorban, azonban számos más (nem klasszikusan böngészőben futó) területen is előfordul, úgy mint Node.js alkalmazás fejlesztésnél, az Apache CouchDB használatakor, vagy az Adobe Acrobat környezetben. JavaScript egy prototípus alapú dinamikus programozási nyelv, amely támogatja az objektum orientált fejlesztést, az imperatív és a deklaratív programozást is.

A nyelv alap képességeinek segítségével végre lehet vele hajtani a tipikus programozási feladatokat, úgy mint:

- értékek tárolása változókban.
- szövegen végzett műveletek.
- események hatására történő kódfuttatás.

A nyelv által biztosított lehetőségeken túl azonban a technológia igazi erejét az adja, amikor kliens oldali nyelvként, a böngészők által biztosított API-val (Application Programming Interfaces) együtt használjuk. Ezek a programozási interfészek adnak hozzáférést a böngésző belső képességeihez és adataihoz, aminek a segítségével teljes értékű alkalmazássá lehet változtatni egy statikus weboldalt.

[1] https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

Böngésző API: a böngészők által nyújtott kapcsolódási pont, amelyen keresztül lehetségessé válik az adathozzáférés és kommunikáció a számítógép lokális erőforrásaival. Néhány tipikus használati eset:

- A DOM (Document Object Model)[2] API lehetővé teszi, hogy hozzáférjen a HTML és CSS elemekhez. Ezen keresztül lehet új elemeket létrehozni, meglévőket módosítani vagy törölni.
- A Geolocation API a pozíció információkhoz ad hozzáférési lehetőséget. Ez alapján képes például a google maps is megjeleníteni a felhasználó helyzetét.
- A Canvas és WebGL API segítségével lehet 2D és 3D grafikákat megjeleníteni.
- Audio és Video APIs úgy mint a HTMLMediaElement vagy WebRTC teszi lehetővé a multimédia tartalmak kezelését és megjelenítését.

Third party API (külső API-k): ezek a programozási felületek nem képezik részét a böngésző által nyújtott szolgáltatásoknak. Tipikusan külső szolgáltatásokhoz adnak hozzáférést, amelyek segítségével új funkciókat tudunk összekapcsolni az alkalmazásunkkal. Néhány példa erre:

- A Twitter API, amely lehetővé teszi például egy twitter fiók tweetjeinek a megjelenítését egy weboldalon.
- A Google Maps API vagy az OpenStreetMap API segítségével egy külső térkép funkciót ágyazhatunk be egy weboldalba.

A böngészőbe épített biztonsági szabályok garantálják, hogy minden weboldal (böngésző tab) egy önálló külön környezetben működjön. Ezek a környezetek (execution environments) különítik el egymástól az egyes weboldalak végrehajtandó kódját, ami egy kritikus feladat biztonság szempontjából.

A labor foglalkozásra felkészülés képen, olvassa el a JavaScript nyelv alapjait tartalmazó leírást a Mozilla oldalán: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/A_first_splash.

[2] https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

2.2. Cross-Site Scripting (XSS)

A Cross-Site Scripting (XSS) egy kód beszúrásos támadás. Egy ilyen támadás során a támadó kártékony JavaScript kódot juttat egy legitim, azonban sérülékeny alkalmazásba. A sérülékenységek, amelyek kihasználásával ez lehetségessé válik nagyon széleskörűek, a legtöbb változó, amely tartalmát képes a támadó kontrollálni és szerepet játszik a weboldal megjelenésében potenciálisan veszélyes lehet.

Egy támadó felhasználhat egy XSS támadást, hogy kártékony kódot juttasson az áldozat böngészőjébe. A böngészőnek nincs esélye ezt a kódot megkülönböztetni az alkalmazás legitim kódbázisától, így azzal azonos módon a támadó kódot is végrehajtja. A végrehajtás során a legitim kódhoz hasonlóan a támadó is hozzá tud férni a sütikhez, más session információhoz, vagy bármilyen érzékeny adathoz a weboldalon belül. A kód akár a weboldal HTML tartalmát is módosíthatja, ezzel további károkat okozva az áldozatnak. Az XSS támadásoknak 3 fajtáját különböztetjük meg.

2.2.1. Stored XSS (más néven Persistent XSS)

Stored XSS támadásról (eltárolt XSS) akkor beszélünk, ha a kártékony kód a sérülékeny alkalmazásban (tipikusan egy adatbázisban) eltárolásra kerül. Erre tipikus példaként szolgálhat egy fórum alkalmazás, vagy egy kommentelési lehetőség. Fontos azonban szem előtt tartani, hogy maga a támadás ekkor is kliens oldalon történik, azonban a kártékony kód ebben az esetben egy adatbázison keresztül jut el a böngészőbe. A támadást így megelőzi egy másik (időben független) lépés, amely során a támadó a kártékony kódot bejuttatja az adatbázisba. Az előző példákból kiindulva ez úgy valósítható meg, ha közzétesz egy új fórum bejegyzést, vagy ír egy kommentet. A fejlesztő által elkövetett hiba ilyenkor az, hogy egy felhasználótól származó adatot sosem szabad közvetlenül, ellenőrzés nélkül eltárolni (input validáció szükséges), továbbá, adatbázisból kiolvasott adatot, sosem szabad közvetlenül a HTML kódba beleírni (escaping szükséges).

A modern HTML5 alapú technológiák elterjedésével elképzelhető ennek a támadásnak olyan verziója is már, ahol a böngésző nyújtotta adattárolási képességeket használja ki a támadó, így a kártékony kód nem a szerver oldali adatbázisba kerül, hanem a kliens oldalon marad.

2.2.2. Reflected XSS (más néven Non-Persistent XSS)

Reflected XSS esetén a felhasználótól kapott adatot nem tárolja el a szerver oldal, hanem azt közvetlenül felhasználja a válasz elkészítése során. Tipikus példa erre egy keresést megvalósító oldal, ahol az oldal tetején általában megjelenik az a bemenet, amire a felhasználó rákeresett. Egy ilyen funkcionalitás esetén, ha a szerver oldalra érkezett adat ellenőrzés nélkül (input validáció) bekerül a válasz HTML-be, akkor fenn áll az XSS támadás lehetősége.

Ez a támadás is elképzelhető úgy, hogy a kártékony kód nem jut el a szerver oldalra, ezt azonban egy új kategóriának tekintjük a megvalósítás technikai tulajdonságai miatt: ez a DOM alapú XSS.

2.2.3. DOM Based XSS (más néven DOM alapú XSS)

DOM alapú XSS támadásnak azt tekintjük, amikor a támadás teljes folyamata a kliens oldalon valósul meg, és a sérülékenység a DOM kliens oldali manipulálásában található. Modern web alkalmazások (például az SPA-k[3]) a legtöbb esetben a kliens oldalon renderelik ki a végső weboldalt. A legtöbb keretrendszer rendelkezik alapvető védelemmel, azonban a használatuk során lehetséges hibát elkövetni. Azok az alkalmazások, amelyek pedig keretrendszer nélkül valósítják meg ezt a funkcionalitást, tipikus célpontjai lehetnek egy ilyen támadásnak. Egy gyakran elforduló eset, ha a renderelés során az aktuális URL (vagy egy része) felhasználásra kerül. Ha ilyen esetben, ez az adat közvetlenül megjelenik a HTML kódban, akkor ez kihasználható egy DOM alapú XSS támadás részeként.

2.3. XSS támadás megakadályozása[4]

A következő ajánlások célja, hogy az összes korábban említett XSS támadást megállítsák egy alkalmazásban. Ezek alkalmazása esetén nincs lehetőség külső forrásból származó bármilyen tetszőleges HTML tartalom megjelenítésére, azonban ezen korlátozás mellett elérhető biztonság egy általában elfogadható mértékű kompromisszumnak számít. Egy átlagos szoftver fejlesztése során valószínűleg nincs szüksége az összes itt megemlített szabály betartására. A

[3] https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58

[4] https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

legtöbb esetben az első két pont elegendő lehet, azonban érdemes minden projekt során megvizsgálni a felmerülő veszélyeket, és az alapján dönteni.

1. HTML escape-elést kell alkalmazni minden nem megbízható tartalom HTML elembe történő beágyazása előtt.

2. Attribútum escape-elést kell alkalmazni minden attribútum érték megadása előtt.

3. Tetszőleges helyre sosem szabad nem megbízható tartalmat beágyazni. (Csak az erre szándékosan meghatározott helyre.)

4. JavaScript escape-elést kell alkalmazni minden nem megbízható adat esetén, amit értékül adunk egy JavaScript változónak.

5. HTML escape-elést kell alkalmazni minden nem megbízható JSON értékre, és adatot csak a JSON.parse metódus segítségével érdemes beolvasni.

6. CSS escape-elést kell alkalmazni, és szigorú érték ellenőrzést végrehajtani, mielőtt stílus leírást illesztünk be egy HTML oldalba.

7. URL escape-elést kell alkalmazni mielőtt egy értéket felhasználunk URL paraméterként.

8. A HTML tartalmat mindig kifejezetten erre a célra kifejlesztett könyvtár segítségével ellenőrizzük a felhasználás előtt.

9. Kerüljük a JavaScript URLek használatát.

3. Feladatok

Ez a labor gyakorlat 4 feladatból áll. Az egyes feladatok nehézség szerint vannak növekvő sorrendben, így a megoldásuk a leírt sorrendben javasolt. A labor során a korábbi mérésről megismert Juice-shop alkalmazást kell a továbbiakban is használni.

A gyakran használt XSS támadásokat több helyen is összegyűjtötték már. Egy ilyen lista a feladatok megoldásában is segítség lehet, a próbálkozásokhoz innen javasolt ötleteket gyűjteni: https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet#Tests

Amennyiben valamelyik feladat megoldásához szükséges egy bejelentkezett felhasználó, akkor az adminisztrátor felhasználó használata javasolt. Ehhez a felhasználónév: `admin@juice-sh.op` valamint a jelszó: `admin123`

A feladatok megoldása során a cél mindig a Javascript kód futtatás demonstrálása, amit a legegyszerűbben a JavaScriptben elérhető `alert()` függvény meghívásával lehet megtenni.

A támadásokat gyakran megnehezíti, hogy string beszúrása során figyelni kell, hogy az idézőjelek megfelelően legyen használva. Az eredeti forráskód is rend szerint használja a ' és a " karaktereket. Ilyen esetekben célszerű lehet még a ` (backtick) karaktert is kipróbálni, amely szintén használható string eleje és vége jelként.

3.1. Vezetett Feladat

Cél: Egy persistent XSS támadás végrehajtása A Juice-Shop alkalmazás profil oldala sérülékeny persistent XSS támadásra. Ezt kisználva módosítsa a felhasználó nevét úgy, hogy `<script>alert(`xss`)</script>` kódot lefuttatja, ezzel demonstálva, hogy egy támadás végrehajtása lehetséges!

3.2. Vezetett Feladat

Cél: Egy reflected XSS támadás végrehajtása Keressen az alkalmazásban olyan bemenetet, amely tartalma a szerverrel megvalósított kommunikáció során megjelenik a válaszban is.

Segítség: A sérülékenység a megrendelések nyomonkövetése szolgáltatásban van.

Hajtson végre XSS támadást, amely során egy HTML elemet szúr be a weboldalba, amelyet könnyű felismerni (például: `<iframe>` tag)!

3.3. Önálló feladat

Cél: Hajtson végre DOM alapú XSS támadást! Ez a feladat nagyon hasonló az előző feladatban végrehajtotthoz. Ismételje meg a támadást a web oldal keresés funkciója ellene, majd keresse meg a forrás kódban, hogy hol található a sérülékeny kódrészlet. A keresés feldolgozását megvalósító kódot ezen a linken keresztül éri el: https://github.com/bkimminich/juice-shop/blob/master/frontend/src/app/search-result/search-result.component.html

3.4. Önálló feladat

Cél: Hajtson végre egy persistent XSS támadást a kliens oldali védelem megkerülés után! Minden új felhasználó adata eltárolásra kerül a adatbázisban. A regisztráció folyama során minden adat megfelelő ellenőrzése egy kritikus feladat a biztonság szempontjából.

A Juice-Shop alkalmazásban ezt a védelmet a kliens oldalra implementálta a fejlesztő, ami így a támadó által megkerülhető. A kliens oldalra írt védelmek a weboldal HTML tartalmának a kézzel végrejatott módosításaival könnyen megkerülhető.

Hajtoson végre persistent XSS támadást, hogy demonstrálja ezt a sérülékenységet! A sikeres támadás eredményét a `/#/administration` URL megnyitása után lehet látni!