Haladó Adatbiztonság

Oláh Norbert

2022

Összefoglaló

1. OWASP Top Ten 2021
   - Sebezhető és elavult összetevők

Sebezhető és elavult összetevők

6 - Sebezhető és elavult összetevők

Sebezhető és elavult összetevők I.

Valószínűleg sebezhető:

- Ha nem ismered az összes felhasznált komponens verzióját (mind a kliens-, mind a szerveroldali). Ez magában foglalja a közvetlenül használt összetevőket, valamint a beágyazott függőségeket.
- Ha a szoftver sebezhető, nem támogatott vagy elavult. Ez magában foglalja az operációs rendszert, a web-/alkalmazáskiszolgálót, az adatbázis-kezelő rendszert (DBMS), az alkalmazásokat, az API-kat és az összes összetevőjét, a futási környezeteket és a könyvtárakat.
- Ha nem ellenőrzi rendszeresen a sebezhetőségeket, és nem iratkozik fel az Ön által használt komponensekkel kapcsolatos biztonsági tájékoztatókra.

Sebezhető és elavult összetevők II.

Valószínűleg sebezhető:

- Ha nem javítja vagy frissíti a mögöttes platformot, keretrendszereket és függőségeket kockázatalapú, időben történő frissítéssel. Ez általában olyan környezetekben fordul elő, ahol a javítás havi vagy negyedéves feladat a változásvezérlés keretében, így a szervezetek napokig vagy hónapokig szükségtelenül ki vannak téve a javított sebezhetőségeknek.
- Ha a szoftverfejlesztők nem tesztelik az updated, upgraded és patchelt könyvtárak kompatibilitását.
- Ha nem biztosítja az összetevők konfigurációit (lásd: A05:2021 – Nem biztonságos konfiguráció).

Forgatókönyv

A komponensek általában ugyanazokkal a jogosultságokkal futnak, mint maga az alkalmazás, így bármelyik komponens hibája komoly következményekkel járhat. Az ilyen hibák lehetnek véletlenek (pl. kódolási hiba) vagy szándékosak (pl. hátsó ajtó egy komponensben). Néhány példa a felfedezett, kihasználható komponensek sebezhetőségére:

A CVE-2017-5638, a Struts 2 távoli kódfuttatási sebezhetőség, amely tetszőleges kód futtatását teszi lehetővé a kiszolgálón, jelentős betörésekért felelős.

Bár a dolgok internetét (IoT) gyakran nehéz vagy lehetetlen javítani, a fontossága jelentős lehet (pl. biomedicinális eszközök).

Sérülékenység menedzsment

OWASP Top Ten 6: Az ismert sebezhetőséggel rendelkező összetevők használata

- A környezet komponensei (network, OS, ...)
- Built-in komponensei (libraries, packages,...)

- **Asset menedzsment** - készlet komponensei
  - Egyértelmű ID-k használata, mint CPE vagy SWID
    - Annak érdekében, hogy tud, ha egy feltárt probléma érint téged
- **Monitoring** - a sérülékenységi repository-k lekérdezése
  - A függőségek ismerete szintén nélkülözhetetlen
- **Értékelni** a fenyegetéseket - kockázatelemzés
  - Prioritások meghatározása pontszám alapján (e.g. CVSS)
- **Kármentesítési** folyamat a kockázat kezelésére
  - Igazítva a patch menedzsment eljárásaihoz

Komment

A sebezhetőség kezelésének magas szintű megbeszélése felhő kontextusban:
http://www.csoonline.com/article/2125921/cloud-security/vulnerabilitymanagement–the-basics.html
Egy sérülékenység menedzsment folyamatának implementálása:
https://www.sans.org/reading-room/whitepapers/threats/implementingvulnerability-management-process-34180
Útmutató az vállalati Patch Management technológiákhoz (NIST):
https://csrc.nist.gov/publications/detail/sp/800-40/rev-3/final
Patch és biztonsági rés menedzsment program (NIST) létrehozása - vegye figyelembe, hogy a dokumentum helyettesítve van, de még mindig hivatkozik és érvényes a régi rendszerekre:
https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-40ver2.pdf

Sérülékenységi attribútumok

- Az ismert sebezhetőség azonosítása érdekében azt általános és egyértelmű módon kell leírni
  - Security Content Automation Protocol (SCAP)
- És használjon általános attribútumokat
  - Platform: Common Platform Enumeration (CPE)
  - Type: Common Weakness Enumeration (CWE)
  - Severity: Common Vulnerability Scoring System (CVSS)
  - Security description: Open Vulnerability and Assessment Language (OVAL)
  - ID: Common Vulnerability Enumeration (CVE) or NVD ID

A szoftver azonosító tag szabvány (SWID) nem része a SCAP-nak, de erőfeszítések történnek a SWID és a CPE / SCAP közötti átjárhatóság érdekében. Ne feledje, hogy a gyakorlatban az SCAP-t az operációs rendszer keményítésére (hardening) használják, nem pedig az általános alkotóelemek / környezet keményítésére.

SWID integration project from NIST:
https://csrc.nist.gov/projects/Software-Identification-SWID

Common Vulnerability Scoring System - CVSS

- Ipari szabvány a sebezhetőség súlyosságának értékelésére (CVSSv2 since 2007, CVSSv3 since 2015)
  - Sérülékenységek pontszáma 0-tól-10ig
  - Helps to compare vulnerabilites
  - Megkönnyíti a rangsorolást és a sérülékenységi menedzsment politikát
  - v3 pontszámok általában magasabbak, mint a v2 ugyanazon sérülékenységeknél
- Az alap mérők (metrikák) a következő jellemzőkből állnak
  - **Kihasználhatóság**
    - Hozzáférési vektor, hozzáférési bonyolultság, hitelesítés
  - **Impact**
    - Confidentiality, integrity, availability

Általános érvelés az, hogy a 9-10 bázisponttal rendelkező sebezhetőség kritikus, 7-8 magas, 4-6 közepes és 0-3 alacsony. Általában véve a v3 sokkal leíróbb, mint a v2 - például a v3 figyelembe veszi a kihasználás következményeit, amelyek más biztonsági követelményeket is károsíthatnak. Tehát az információszivárgás veszélyeztetése csak a v2 Confidentiality veszélyeztetheti, de a v3-ban veszélyeztetheti Confidentiality-t és Integrity-t.

CVSS:
https://en.wikipedia.org/wiki/CVSS CVSSv3 útmutató:
https://www.first.org/cvss/user-guide
CVSSv2 útmutató:
https://www.first.org/cvss/v2/guide
CVSSv2 pontszámok megoszlása a CVE összes sebezhetősége között:
https://www.cvedetails.com/cvss-score-distribution.php
Kommentár a pontszám inflációval a CVSS v3 vs v2-nél:
https://www.riskbasedsecurity.com/2017/05/cvssv3-when-every-vulnerabilityappears-to-be-high-priority/

Azonosítási és hitelesítési hibák

7 - Azonosítási és hitelesítési hibák

Azonosítási és hitelesítési hibák

A felhasználó személyazonosságának megerősítése, a hitelesítés és a munkamenet-kezelés kritikus fontosságú a hitelesítéssel kapcsolatos támadások elleni védelem szempontjából. Hitelesítési gyengeségek lehetnek, ha az alkalmazás:

- Lehetővé teszi az automatikus támadásokat, például a hitelesítő adatok kitöltését, amikor a támadó rendelkezik az érvényes felhasználónevek és jelszavak listájával.
- Engedélyezi a nyers erővel vagy más automatizált támadásokat.
- Alapértelmezett, gyenge vagy jól ismert jelszavakat engedélyez, mint például ”Password1” vagy ”admin/admin”.
- Gyenge vagy nem hatékony hitelesítő adatok helyreállítását és elfelejtett jelszavakat használ, például ”tudásalapú válaszokat”, amelyeket nem lehet biztonságossá tenni.

Azonosítási és hitelesítési hibák

- Sima szöveges, titkosított vagy gyengén hashelt jelszavak adattárolóit használja (lásd A02:2021-Cryptographic Failures).
- Hiányzó vagy nem hatékony többfaktoros hitelesítéssel rendelkezik.
- Munkamenet-azonosítót tesz közzé az URL-ben.
- Sikeres bejelentkezés után újrafelhasználja a munkamenet-azonosítót.
- Nem érvényteleníti helyesen a munkamenet-azonosítókat. A felhasználói munkamenetek vagy hitelesítési tokenek (főként az egyszeri bejelentkezési (SSO) tokenek) nem érvénytelenülnek megfelelően a kijelentkezés vagy inaktivitási időszak alatt.

Session fenyegetések kezelése

- A támadó megtanulhatja az áldozat session ID-ját, és megszemélyesítheti azt egy **session hijacking** támadással,
  - A nyílt (nem titkosított) hálózati csomagok elkapása,
  - Szkript beszúrása az oldalra (XSS) és a sütik (cookies) elolvasása (ha lehetséges)
  - A session ID kitalálása vagy brute force, ha gyenge a véletlen része
- Ha beállítható a session ID-ja (például ha az URL továbbításra kerül), a támadó végrehajthat egy **session fixation** támadást
  - A támadó belép egy webhelyre és megkapja a session ID-ját
  - A támadó bejelentkezik
  - Az URL-t a(z) (potenciális) áldozatoknak elküldi ($\approx$ adathalász támadás)
  - Az áldozatot ráveszik, hogy nyisson meg egy adott linket, amellyel végrehajt valamilyen műveletet

Sok esetben a támadó végső célja a felhasználói fiókhoz való hozzáférés, amelyet a felhasználói session megszerzése segíthet. Számos sebezhetőség és támadás szól arról, hogy a támadó valamilyen módon ellopja a felhasználó session ID-ját.
Formális definíció:
Link Session hijacking:
Link Session fixation:
Link

Session kezelésének legjobb módszerei

- A session ID-val kapcsolatos követelmények
  - Legalább 128 bit hosszúságú legyen
  - Muszáj, hogy **véletlen** és **megjósolhatatlan** legyen
    - Legalább 64 bit entrópia (véletlenszerűség) az ID-ban
  - A fennmaradó nem véletlenszerű rész nem szabad, hogy feltárjon bármilyen érzékeny információt
    - A kriptográfiai hash-t kell alkalmazni a payload-on
  - Egy sütiben kell tárolni, és megfelelően védeni kell
- Sikeres hitelesítés esetén
  - Mindig egy új session (ID) létrehozása
  - Másolja az összes adatot a régiból az új sessionbe
  - Ezután érvényteleníteni kell a régi session-t

Ne implementálj egyéni session menedzsmentet és session ID generálást

Ökölszabályként kerülje el a saját session menedzsment implementálást, mivel rengeteg szempontnak kell megfelelnie.
OWASP összefoglaló a biztonságos session kezelésről:
Link
Egy cikk a session kezelésről és biztonsági rétegről (Hogyan kezeljük ezt a node.js-ben):
Link
Vita a cikkről:
Link

Session kezelése - legjobb módszerek

- A biztonság (secure) megakadályozza, hogy a süti titkosítatlanul eljusson (HTTP)
  - Ha nem állítja be, akkor kiszivároghat a session ID, ha a HTTP használata is lehetséges
- HttpOnly megakadályozza a szkriptek hozzáférését a sütihez
  - A mai böngészőkben ez egyébként nem lehetséges
- A hitelesített session-nek nem kell tartósnak lennie (memória tárolás)
  - Ne használj Max-Age vagy Expires -t, ezek fenntartják a sessiont (fájl tárolás)
- Fontos, hogy a domain-t megfelelő módon kell beállítani
  - Meg lehet határozni azt az útvonalat is, amely tovább korlátozza a süti használatát
- Be állíthatod a web.xml -n (alapértelmezett értékként)
  - Ezek az attribútumok beállítják a megfelelő HTTP header értékeket

Session kezelése -legjobb módszerek

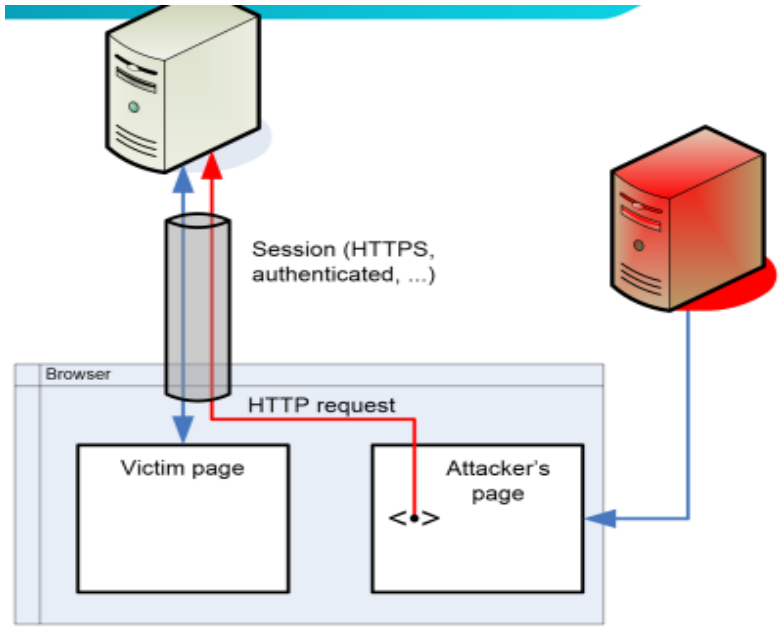
```
<session-config>
  <cookie-config>
    <secure>true</secure>
    <http-only>true</http-only>
  </cookie-config>
</session-config>
```

```
Cookie cookie = new Cookie("myvar", "xyz");
cookie.setDomain("example.com");
//cookie.setMaxAge(24 * 60 * 60);
cookie.setSecure(true);
cookie.setHttpOnly(true); //since JEE 6
response.addCookie(cookie);
```

```
Set-Cookie: myvar=xyz; Domain=example.com; Secure; HttpOnly;
```

Nagy különbség van a szerver oldali session lejárati idő és a süti életkorának (a session-timeout és a web.xml max-age) között! Az előbbi megmondja a szervernek, hogy mennyi ideig kell maradnia a session-nek a felhasználó utolsó tevékenysége után, míg az utóbbi megmondja a böngészőnek, mennyi ideig kell a sütit tárolni. A szerver oldali munkamenet időtúllépésének beállítása jó ötlet, a Max-Age beállítása nem.
Megjegyzések a süti jellemzőkről és biztonságról:
Link
A süti-domainek korlátozásának okai:
Link
Wikipédia:
https://en.wikipedia.org/wiki/Secure_cookies

Cross Site Request Forgery (CSRF)



- A támadó kéri, hogy a hitelesített felhasználó küldjön egy kérést
  - Social engineering + malicious code
- Lehetséges, mert a böngészők automatikusan küldik a session-sütiket
  - A rosszindulatú kód megörökli a hitelesített felhasználó ID-ját

Cross Site Request Forgery (CSRF)

```
<img src="http://www.mybank.com/passchange?pass=attack">
```

- XSS: a rosszindulatú kód a támadott webhely része
  - Kihasználja a felhasználó bizalmát egy adott webhelyen
- A CSRF rosszindulatú URL-je nem kívánt HTTP-kérést okoz a megtámadott webhelynél
  - Kihasználja a webhely bizalmát a felhasználó böngészőjében

A képen jól védett a csatorna HTTPS, kétfaktoros hitelesítés stb. útján. A szervernek azonban nincs más választása, mint a kérés elfogadása (nem tud megkülönböztetni jó és rosszindulatú kéréseket). Ezt **session riding**-nak is nevezik.

Egy tipikus támadás így néz ki: <img src="http://www.mybank.com/passchange.jsp?pass=attack" width="0" height="0"> (a kép láthatatlan a felhasználó számára).
Formális definíció:
Link
OWASP:
Link

Feladat - Cross Site Request Forgery (CSRF)

CSRF feladat

- Keresd fel a http://www.insecar.com oldalt
  - Lépj be egy érvényes felhasználóval (használd test/testpass)
- Ezután nyisd meg egy másik ablakban: http://attacker.com/CSRF_clickonthis.html
  - Lépj vissza http://www.insecar.com oldalra
  - Kattints a → logged out-ra!
  - Próbálj újra bejelentkezni
  - Mik az új hitelesítő adatok?
- Ellenőrízd:CSRF_clickonthis.html használd a changepass-trick.html (see in C:/Ex/WebExample_jsp/Attacker_com)

CSRF feladat

- Mi történt, miután végrehajtottad a lépéseket?
- Hogyan változtatta meg a támadó a jelszót?

Bejelentkezés CSRF

- A támadó arra kényszeríti a felhasználót, hogy jelentkezzen be egy másik helyre a támadó hitelesítő adataival - **account fixation**
  - A támadó fiókot hoz létre a célhelyen
  - A támadó a kiaknáz egy CSRF-t egy támadó sztringgel, például úgy mint

```
<img src="http://www.example.com/login?user=attacker&pw=attackerpw">
```

  - A felhasználót láthatatlanul bejelentkezteti a célhelyre
  - Amikor a felhasználó a célhelyet használja, akkor is bejelentkezik a támadó fiókjába
- Lehetséges célok
  - Kereső motorok → a támadó keresési előzményeket megszerezheti
  - Fiókok, amelyek kezelik a hitelkártyát → a támadó becsaphatja a felhasználót, hogy hitelkártyákat adjon a támadó fiókjához

CSRF megelőzés

- **Megjósolhatatlan token** felvétele a bodyba vagy az egyes HTTP-kérelmek URL-jébe - gyakran használt megoldás a visszajátszásos támadásokkal szemben
  - A tokent a böngésző nem küldi el automatikusan
  - A támadó nem ismeri
  - Egyedi token a felhasználói session-höz vagy minden kéréshez
- Példa - rejtett mezőben

```
<form action="passchange" method="post">
<input type="hidden"
    name="680754807546"
    value="54656897546" >
...
</form>
```

A következőképpen tudjuk megakadályozni, hogy a CSRF egy kétrétegű védelembe beépüljön.

- Az első réteg ellenőrzi, hogy a felhasználó kezdeményezi-e a kérést az eredetivel és a hivatkozó header-rel.
- A második réteg véletlenszerű értéket küld a felhasználónak, és megköveteli ennek használatát a következő kérelmekben.
  - Ezt úgy lehet megtenni, hogy ezt a token értékét minden létrehozott oldalra elhelyezi - általában rejtett mezőként vagy csak szerverre vonatkozó titkos kulccsal titkosítva.
  - Ezt követően elküldi azt minden egyes következő kéréssel vagy - ha az előző lehetőség nem kivitelezhető - azzal a megkötéssel, hogy a böngésző küldje el a véletlenszerű értéket URL-paraméterként, úgy mint egy süti-értéket.

A CSRF-támadás végrehajtásakor a támadó nem tudja megszerezni ezeket a véletlenszerű értékeket (kivéve, ha a cross-site scripting támadást is sikerül kiaknázni), megakadályozva ezzel a támadás sikeres végrehajtását.

CSRF Prevention Cheat Sheet:
Link

Thank you for your attention!