Haladó adatbiztonság

Oláh Norbert

2022

Összefoglaló

1. OWASP Top Tíz 2021
   - Szoftver- és adatintegritási hibák
   - Nem megfelelő logolás és ellenőrzés

Szoftver- és adatintegritási hibák

8 - Szoftver- és adatintegritási hibák

Szoftver- és adatintegritási hibák leírás

A szoftver- és adatintegritási hibák olyan kódokhoz és infrastruktúrához kapcsolódnak, amelyek nem védenek az integritás megsértése ellen.

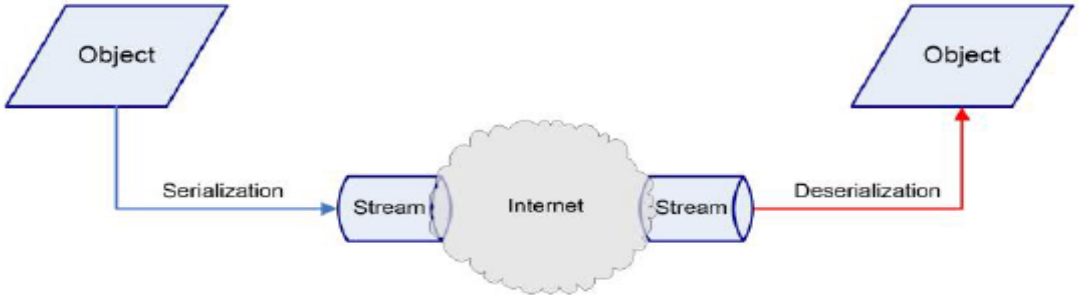
- Példa erre amikor egy alkalmazás nem megbízható forrásokból, adattárakból stb. támaszkodik. A nem biztonságos CI/CD-folyamat jogosulatlan hozzáférést, rosszindulatú kódot vagy rendszerkompromittálódást okozhat.
- Sok alkalmazás már tartalmaz automatikus frissítési funkciót, ahol a frissítések letöltése megfelelő integritásellenőrzés nélkül történik, és a korábban megbízható alkalmazásra alkalmazzák. A támadók potenciálisan feltölthetik saját frissítéseiket, hogy szétküldjék és futtassák az összes telepítést.

Szoftver- és adatintegritási hibák leírás

- Egy másik példa, amikor az objektumokat vagy adatokat olyan struktúrába kódolják vagy szerializálják, amelyet a támadó láthat és módosíthat, és ki van téve a nem biztonságos deszerializálásnak.

Szerializáció és deszerializáció alapjai

- Fő cél: adatátvitel elosztott alkalmazásokban (RPC-k, hitelesítő tokenek stb.)
- Streambe írt objektumok vagy adattípusok
  - Java Serializable, Ruby Marshal, .NET Serializable, Python Pickle, PHP serialize(), Google Protobuf, Apache Avro and Thrift, CBOR/BSON,...
  - Emberek által is olvasható formátumok: tipikusan XML és JSON



Formális definíció:
Link

Deszerializáció biztonsági kihívásai

- Az érzékeny adatok szerializációja (bizalmasság)
- Szerializáció → Stream → deszerializáció lépéseket zárt láncként feltételezünk, bár nyilvánvalóan nem ad integritás védelmet
  - Deszerializáció stream egy lehetséges támadási felület!
  - Probléma: a legtöbb API-ban nem kifejezetten kér egy bizonyos osztálytípus deszerializációját, hanem egy objektum / struktúra típust hoz létre az adatfolyam alapján.
    - Ez lehetővé teszi tetszőleges osztálypéldányok létrehozását
    - A casting megállíthatja a show-t - de már késő...
- A trükkös adatfolyamok felfüggeszthetik a deszerializáció/ parsing-ot (elérhetőség - DoS támadás)
  - XML bomb, ”Billion Laughs”, hatalmas mennyiségű null vagy üres objektumok egy JSON tömbben, ...

Alapvetõen a deszerializáció a rendszer biztonságának minden aspektusát károsíthatja, de a legnagyobb veszélyt a **kódfuttatás** lehetõsége okozza, ha visszaélnek az osztályok deszerializációjával (ami gyakran magában foglalja a technikákat, például a reflexiót).

Egy ’Billion Laughs’ DoS stílus deszerializáció osztályon keresztül: Link

Deszerializáció kérdései - JSON

- Az egyszerű JSON-típusok deszerializálása nem vezethet RCE-hez
  - A JSON szerializációs könyvtárakat azonban általában az osztályok szerializációjára használják
  - Az objektum rekonstrukciója során a mező értékeket reflection-ön keresztül, beállító módszerek vagy speciális konstruktorok segítségével kell beállítani...
    - amely ugyanazt a (POP) támadási lehetőséget okozhatja
- Számos JSON parser sebezhető, így vagy úgy
  - Ne használj: FastJSON, Sweet.Jayson, JSON-IO, FlexSON
  - Veszélyes: FSPickler
  - Óvatosan használj: Json.Net, JavascriptSerializer, DataContractJsonSerializer, Jackson, Genson
  - Biztonságos: Gson
  - Forrás: Link

Parser kategóriák

- Ne használj: parserek, amelyeket ki lehet használni az RCE elvégzéséhez, függetlenül attól, hogy miként használja őket - soha nem szabad használni
- Veszélyes: parserek, amelyek sebezhetőek csökkentett hatással (mint az XXE)
- Óvatosan használd: parserek, amelyek sebezhetők, ha helytelenül vannak konfigurálva (például bekapcsoláskor TypeNameHandling a Json.Net-ben)
- Biztonságos: parserek amelyek nem sebezhetők, azaz eddig nem találtak ismert gyengeségeket és sebezhetőségeket.

JSON és .NET deszerializációs issue-k:
Link

Nem megfelelő logolás és ellenőrzés

9 - Nem megfelelő logolás és ellenőrzés

Nem elegendő védelem

- Mivel nincs 100% -os biztonság, fel kell készülnie a sebezhetőséget kihasználó kísérletek észlelésére, naplózására, megválaszolására és blokkolására
  - Még akkor is, ha vannak sebezhető pontok, de nem fedték fel és nem javították ezeket (még!)
    - Potenciálisan védelmet nyújt még a zero days ellen is
  - Mind a **kézi (manuális)**, mind **automatizált** támadások ellen
- A detektálási és reagálási megoldásoknak helyben kell lennie
  - Megfelelő loggolás
  - Behatolás-Érzékelő / Megelőző Rendszerek (IDS / IPS)
  - Web Application Firewall (WAF)
- Ez a fejlesztési folyamatokra is vonatkozik
  - Például. patch menedzsment - milyen gyorsan tudnak válaszolni

A **zero-day** (más néven 0-day) biztonsági rés olyan számítógépes szoftver sebezhetőség, amely ismeretlen azok számára, akik érdekeltek lennének a biztonsági rés megszüntetésében (ideértve a célszoftvert gyártót is). Amíg a biztonsági rést nem szüntetik meg, a hackerek kihasználhatják azt a számítógépes programoknál, az adatoknál, a további számítógépeknél vagy a hálózatoknál. A **zero-day** sebezhetőségre irányuló kihasználást **zero-day exploit** vagy **zero-day** támadásnak nevezik.

Zero-day:
Link

Észlelésekkel való visszaélés

- Észlelésekkel való visszaélés vagy rossz célra használása
  - Olyan használati minták (pattern), amelyeket egy valid felhasználó nem okozhat
    - Például egy érték előfordulása a legördülő menüben, amely nem szerepel a listában
  - Használati minták, amelyek nincsenek használatban
    - A felhasználói interakció páratlan dinamikája (egér, billentyűzet)
    - Túl gyorsan ismételt kérések hasonló adatokkal
- Válasz
  - Logolni és elemezni a rosszul viselkedő kéréseket
    - Blokkolja magát a fiókot, vagy az IP-címeket / tartományokat
  - Patching - a feltárt sebezhetőségekre adott válasz
    - Virtuális patch: megakadályozza a kihasználást a sebezhetőség javítása helyett (forgalom blokkolása vagy kódfuttatás)

A behatolás észlelésének kritikus része a **szokatlan vagy rendellenes tevékenységek észlelése** (akár úgy, hogy megtörténik, akár az esemény utáni bűnügyi erőfeszítések részeként) .

A felhasználó szempontjából ez azt jelenti, hogy megpróbál olyan dolgokat végrehajtani, amelyeket a felhasználói felület nem engedélyez.

Logolás és log elemzés

- Logolás célja
  - Azonosítani a biztonsági incidenseket
  - A szabálysértések nyomon követése
  - Információ nyújtása a problémákról és a szokatlan körülményekről, illetve feltételekről
  - További alkalmazás-specifikus adatok közlése az események kivizsgálásához, amelyek hiányoznak más naplóforrásokból
  - A támadások felismerése segítséget ad a sebezhetőség azonosításához és a sebezhetőségek kihasználása elleni védekezéshez

OWASP Logging Cheat Sheet:
Link

Behatolásjelző rendszerek és webes alkalmazások tűzfalai

- Behatolásjelző rendszerek (Intrusion Detection Systems - IDS)
  - **Aláírás-alapú:** megvizsgálja az egyes csomagokat, és egy mintát illeszt egy ismert aláíráshoz
  - **Anomália-alapú:** a hálózati forgalom összesített folyamának elemzése és minta illesztése
- Web Application Firewall (WAF) - szabályok a HTTP kommunikációhoz a támadási kísérletek észlelésére és leállítására
- Példa nyílt forrású hálózati behatolás-érzékelő rendszerre (network intrusion detection system- NIDS) - SNORT
  - Tartalomkeresés / egyezés a könnyű (lightweight) szabályleírási nyelv alapján
  - Csomagok naplózása és valós idejű forgalom elemzése az IP hálózatokon

Ezeknek a rendszereknek az együttes megnevezésére az IDPS-t használjuk (behatolás-érzékelő / megelőző rendszerek).

Az IDS csak a behatolás bizonyítékait keresi, míg az IPS proaktív is lehet, pl. bizonyos IP-címek feketelistájával vagy bizonyos szolgáltatásokhoz való hozzáférés gátlásával.

A ”Deep Packet Inspection” (DPI) megközelítés a hálózati forgalmat az egyes csomagok tartalma alapján vizsgálja és szűri, amelynek további felhasználási esetei vannak a bűnüldözésben és a cenzúrában.

Ezek közül sok tekinthető hálózat biztonsági aggálynak, szemben az alkalmazások biztonságával.

Bro (egy free/open forrás alternatívája a Snort-nak):
Link

Snort:
Link

Deep packet inspection:
Link

Köszönöm a figyelmet!