

Vulnb0x à la ALBERT Ferencz

Egy olyan alkalmazás, mely megengedi, hogy különböző github repository-k időnkénti felépítését és tesztelését konfiguráljuk, docker **Dockerfile**-ok segítségével.

Welcome to vulnb0x, mock@aferencz.xyz!




Log out!

Current repository configurations:

git@github.com:AlphaZoneR/attack-2.git

Created: 2022-01-16 22:17:47
Last build: 2022-01-16 22:41:56
No volume mappings
Private key configured


Builds


  

Add a new repository configuration:

Repository URL:
git@example.com/username/repo.git

Private key:
-----BEGIN OPENSSH PRIVATE KEY-----b3BlbnNzaC1rZXktbjEAAAAAAAAABG5vbmUAAAAA...

Volume mappings
Destination: /data/example Source: /data-dir 



Add new configuration

Vulnb0x elindítása

Vulnb0x dependenciák

- [Vagrant by HashiCorp](#)
- Egy olyan virtualizációs rendszer, melyet a vagrant fel tud használni, és a box is elérhető ezen a virtualizációs rendszeren:
- A **Vagrantfile**-ban felhasznált box: [generic/debian10](#). Ugyan ezen a linken tekinthető meg, hogy milyen virtualizációs rendszereket támogat az adott box. A readme létrehozásakor a következők voltak támogatottak:
 - hyperv
 - vmware_desktop
 - virtualbox
 - parallels
 - libvirt

Rendszer elindítása:

```
shell> cd <directory with Vagrantfile>
shell> vagrant up
```

A `vagrant up` parancs végrehajtását követően, a képernyőre kiírodik a virtuális gépen végrehajtott `ip addr` parancs kimenete. Általában az `eth0` interfész IP címén keresztül érhető el a box.

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:c0:45:24 brd ff:ff:ff:ff:ff:ff
    inet 10.90.104.69/16 brd 10.90.255.255 scope global dynamic eth0
        valid_lft 4888sec preferred_lft 4888sec
```

Az alkalmazás a fentebb kapott IP címen és a 8090-es porton érhető el. <http://<ip-of-eth0>:8090/>

Biztonsági rések

Napjainkban az alkalmazások egyre komplexebbé és komplexebbé válnak. Annak érdekében, hogy egy jól levédett, "hibamentes" alkalmazást írjunk, számos biztonsági partikularitásra kell odafigyelnünk. A vuln0x projekt egy olyan alkalmazást próbál használatra bocsátani, mely kívülről megfelelőnek tűnhet, viszont némely biztonsági résekkel rendelkezik, melyek alpból nem feltétlenül feltűnőek. A jelenben az alkalmazások nagy része a web-et célozza. Ennek érdekében a vuln0x projekt 4 releváns biztonsági rést tárgyal.

LFI (Local File Inclusion)

A Local File Inclusion (helyi fájlbefoglalás) egy olyan támadási technika, amelyben a támadók rávesznek egy adott webalkalmazást, hogy fájlokat futtasson vagy tegyen közzé. Az LFI-támadások érzékeny információkat fedhetnek fel, súlyos esetekben pedig több helyek közötti parancsfájl-kezeléshez (XSS) és távoli kód futtatáshoz vezethetnek. Az LFI az OWASP 10 legveszélyesebb webalkalmazás-sebezhetőségei között szerepel.

További információk

Létezik egy [HackTheBox](#) box, mely kihasználja a [CVE-2020-10977](#), ami egy path traversal exploit, mely megenged LFI-t.

JWT cookie manipuláció

Napjainkban szükséges egy olyan funkcionalitás, mely segítségével egy alkalmazásban eltárolhatunk egy felhasználó szesszióhoz tartozó adatokat. Erre több megoldás, több implementáció létezik, viszont amennyiben be szeretnénk tartani a REST konvenciókat, a JWT token az egyik legjobb megoldás. Egy JWT token igazából egy entitás, mely 3 részből áll:

- Header
- Body/Payload
- Signature

Példa JWT token:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Felfigyelhetünk arra, hogy a 3 különböző rész pontok által kerül elválasztásra. Az első két entitás base64 enkódolt szöveg

```
shell> echo "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9" | base64 -d
{ "alg": "HS256", "typ": "JWT" }

shell> echo
"eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ" |
base64 -d
{ "sub": "1234567890", "name": "John Doe", "iat": 1516239022}
```

A legfontosabb rész viszont talán a signature. Az aláírás biztosítja azt, hogy a kliens nem császárkált az adatokkal. Az aláírást különböző módokon keresztül generálhatjuk. Használhatunk egy egyirányú sajátos titkot, vagy egy publikus-privát kulcs párost. Az alkalmazások nagyrésze egy titkot használ, és ez segítségével generálja le a 3 komponenst. Innen származhatnak biztonsági rések. Amennyiben ismert a titok, bárki módosíthatja az adott JWT token.

Példa JWT token manipulációs biztonsági rés, amely közel áll a python flask alkalmazásunkhoz: [RCE On a Facebook Server](#)

Docker network isolation és privát adatok tárolása

Az alkalmazás támadásakor kihasználásra kerül a veszélyesen konfigurált hálózat illetve az, hogy a fejlesztők nem fektettek figyelmet a privát adatok enkriptálására. További infó:

- [Docker hálózat izolációjának javítása](#)
- [Privát kulcsok használata](#)

Támadási útvonal

- Felhasználó regisztrál
- Felhasználó belép
- Felhasználó konfigurál egy github repository, amely tartalmazza a támadás első részét [git repo](#):
 - A rendszer megengedi a volume-ok hozzácsatolását a konténerekhez, azzal az ötlettel, hogy úgyis csak lokális foldereket enged meg a docker: amennyiben relatív útvonalat adunk meg, mint például `docker run -v ./thisdirectory/./directory` a következő error-t dobja "docker: Error response from daemon: create ./../.: "/./../" includes invalid characters for a local volume name, only "[a-zA-Z0-9][a-zA-Z0-9_-.]" are allowed. If you intended to pass a host directory, use absolute path."
 - A probléma az, hogy a így nem tudnánk relatív útvonalakat megadni, mégpedig ezt akarjuk, ezért a program mindig hozzáragassztja a felhasználó által megadott volume nevéhez a jelenlegi útvonalat.
 - Mivel intuitívan az első szabály szerint arra várnánk, hogy a docker ne engedje meg a mappában visszafelé történő navigálását, mint felhasználók a fenti folyamatot biztonságosnak tartjuk

- A probléma az, hogy míg `./randomdirector:/<mountpoint>` helytelennek bizonyul, `/home/<username>/../../../../:<mountpoint>` nem az.
- Ez megengedi a LFI(Local File Inclusion)-t, mely segítségével mountolhatjuk a lokális `/` foldert, majd innen `cat`-vel kiírhatjuk bizonyos fájlok tartalmát
- A felhasználó rátalál a `tree` parancs segítségével a jelenleg futó alkalmazás kódjára
- Kiderül, hogy az alkalmazás python-ban íródott, és `flask`-et használ
- Flask-en belül használja a `flask.session` funkcionlilitást
- Flask a `session` információkat cookie-ban tárolja, melyeket elküld a kliensnek
- A cookie-k enkódolásához, egy titkot használ, melyet az alkalmazás `.prod` fájlban tárol
- A felhasználó észreveszi, hogy az alkalmazás az adminisztrátor jogokat egy a `session`-on tárolt flag segítségével biztosítja
- A felhasználó, a megszerzett titok segítségével, módosítja a saját cookie-ját

```
(pyenv) => import flask
(pyenv) => app = flask.Flask(__name__)
(pyenv) => app.secret_key = <acquired-secret-key>
(pyenv) => sessioni = flask.sessions.SecureCookieSessionInterface()
(pyenv) => serializer = sessioni.get_signing_serializer(app)
(pyenv) => serializer.dumps({"user": {"email": "<user_email>", "permissions":
"admin"}})
```

- Firefox-ban Cookie Editor-val módosítja a session nevezetű cookie-t a fenti lépések segítségével kapott értékre
- Megjelenik egy új panel a kilensben, mely listázza az összes létező repository-t
- A felhasználó észreveszi, hogy még van egy repo, amely konfiurálva van, root felhasználó által
- Az alkalmazás megengedi az egyszeri ssh privát kód konfigurálását, a klónozás érdekében
- Ezeket a kódokat a REST-api nem küldi vissza
- További elemzés után a felhasználó rájön, hogy az alkalmazás *docker in docker* módban fut, és létezik egy MongoDB szerver is, melyben eltárolja az adatokat
- Egy újabb `repository` segítségével a felhasználó rácsatlakozik a MongoDB szerverre, és kinyeri a konfigurált repositorykat.
- Az így kinyert privát kulccsal megpróbál rácsatlakozni a szerverre ssh-val, és sikerül. Kiderül, hogy bob root jogosultsággal rendelkezik.

Javítási lehetőségek

LFI kijavítása

Mivel a mountpontok meghatározása egy fontos funkcionlilitása a programnak, fontos kijavítani ezt úgy, hogy felhasználható maradjon továbbá is. A legegyszerűbb megoldás a lokális fájlrendszerhez megadott mountpont validálása. Mivel az egyetlen megengedett útvonal, a repository-n belül kell elhelyezkedjen, végezhetünk egy ellenőrzést, hogy a teljesen kiértékelt útvonal benne legyen a jelenlegi repository mappájában. Ezzel a lépéssel kizárjuk azt, hogy az alkalmazás felhasználója valaha is rájöjjön a flask-ben használt titokra, vagy akármilyen más nem megengedett LFI-t végezzen. Amennyiben biztosítani tudjuk azt, hogy ez a titok soha nem kerül publikussá, a fentebb meghatározott támadási útvonal már nem használható. Ennek ellenére, fontos, hogy több szinten is levédjük az alkalmazásunkat. A következő probléma a JWT token módosítása volt.

JWT cookie manipulation kijavítása

Erre a problémára több megoldás is létezhet. Az első megoldás abban az esetben lehet érvényes, mikor a alkalmazás kódja legacy-nak bizonyul, és nem engedhetjük meg magunknak, hogy nagy módosításokat végezzünk: pl. más session dependencia használata, új szerver hozzáadása. Amennyiben ragaszkodunk a JWT token használatához, fontos, hogy ne tároljunk benne jogosultságokat. Míg a WEB 3.0 standard szerint, a web is P2P és block-chain irányba mozog, a jelenlegi infrastruktúra megengedi azt, hogy a szerveren tároljuk az alkalmazáshoz tartozó kritikus adatokat. Fontos, hogy a JWT tokenben csak olyan információt tároljunk mely esetleges módon elősegíti a felhasználó beazonosítását, de soha nem bízunk meg benne. A példaalkalmazás esetében, a legelső megoldás a `permissions` flag migrálása lenne az adatbázisban. A JWT tokenben csak és kizárólag a felhasználó email címét tároljuk, melyet szükség esetén arra használunk, hogy az adatbázisból betöltsük a felhasználó reprezentációját, és ebből kinyerjük a `permissions` flag-et. Egy mégjobb megoldás talán az lenne, hogy ne a felhasználót partikulárisan beazonosító adatot küldjük el JWT token-ként, hanem egy random generált UUID-t, amelyet hozzákötünk egy felhasználóhoz. Ezen megoldással mitíghatjuk a brute force támadások hatékonyságát, abban az esetben ha valaha véletlenül mégis kikerülne az alkalmazás titkos kulcsa, mivel talán a véletlen szerűen generált azonosítókat nehezebb helyesen eltalálni.

A másik megoldás, ami talán egyszerűbb, a flask-sessions dependencia használata. Ezen modul megengedi a szesszió információk szerveroldali eltárolását egy absztrakt és perzisztens módon. Lehetőséget ad különböző perzisztencia szintek felhasználására, mint például: MongoDB, redis, lokális fájlrendszer, PostgreSQL.... Amennyiben különbséget szeretnénk tenni a felhasználó és a hozzátartozó jelenlegi szesszió között, tegyük fel böngészési adatok esetében, ezt kézzel kellene perzisztáljuk. A flask-sessions modul annyival bővíti a már létező JWT alapú modult, hogy a szesszióhoz tartozó adatokat szerveroldalon tárolja, és csak egy azonosítót küld cookie-ként a kliensnek. Hasonló a fentebb tárgyalt JWT tokenes UUIDs példához, annyi különbséggel, hogy az azonosítót nem mi határozzuk meg, hanem a platform kezeli valamilyen szinten.

A docker hálózat felépítéséből származó biztonsági rés kijavítása

Az alkalmazás ezen biztonsági rése már kicsit kényesebb és nehezebb javítani. Felvetődnek a következő megoldások/lehetőségek:

1. A docker konténereknek kapcsolatának betiltása a külvilághoz

Ebben az esetben rontunk az alkalmazás felhasználhatóságán, mivel lehetnek olyan konténerek melyek működése függ néhány külső API-hívástól, külső dependenciák installálásától.

2. Subnet blokkolása a konténer szintjén

Nehéz konfigurálni, előre meghatározni, hogy milyen subnet-en lesz a mongo szerver, új docker network meghatározása stb: Komplex

3. docker-compose network isolation:

Az alkalmazás architektúráját tanulmányozva arra a következtetésre juthatunk, hogy a `vulnb0x-docker` és `vulnb0x-app` közötti dependencia egyirányú. Nincs arra szükség, hogy a `vulnb0x-docker`-t akármilyen más is elérje, vagy ő is elérjen más konténereket. A legnagyobb problémát az okozza, hogy a `vulnb0x-docker` konténer ugyan abban a hálózatban helyezkedik el, mint a `vulnb0x-mongod`. Mivel docker in docker módban fut, ezért a konténerben indított konténerek is elérhetik az első szinten futó konténereket. Ebből származik a biztonsági rés is.

Javaslat:

Módosítsunk a rendszer hálózati felépítésén. Legyen *net1* és *net2* egymástól független hálózat, melyeken elhelyezzük *vulnb0x-docker*, *vulnb0x-mongodb* és *vulnb0x-app* konténereket

Network	vulnb0x-docker	vulnb0x-mongo	vulnb0x-app
net1	x		x
net2		x	x

Az így kapott felépítésben, bármilyen konténer ami a *vulnb0x-docker*-ben fut, izolált a kritikus infrastruktúrától.

Privát kulcsok használata

Az alkalmazás ezen funkcionalitása már első látásra is szemrebbenéseket okozhat. Egy privát kulcs beillesztése talán mindig is piros zászlót emelne. E funkcionalitás azzal az ötlettel kerül implementálásra, miszerint a felhasználó kimondottan generál egy privát kulcsot mely csakis ebben az alkalmazásban használatos. Mint ahogyan azt a támadási lépésekben is megtekinthettük, a felhasználó nem mindig figyelmes. Mi, mint az alkalmazás író, némi kötelességet kellene érezzünk a felhasználóink adataival kapcsolatosan. A legnagyobb hiba, amit talán sok már létező alkalmazás elkövet az, hogy a privát adatokat semmilyen képpen nem enkriptálja, csak free-text-ben tárolja az adatbázisban. Az alkalmazásunkban ezt a hibát talán a legkönnyebben azzal lehetne kijavítani, hogy az alkalmazás titkával enkódoljuk a *private_key* mezőt az adatbázisban. Így, amennyiben a titok nem publikus, az adatok privátak maradnak. Egy másik megoldás az lenne, hogy az adott privát kulcsot a felhasználó jelszavával enkriptáljuk. Ez maga után vonná természetesen azt is, hogy a privát kulcs bármilyen felhasználásakor a felhasználó meg kell adja a jelszavát. Ameddig a jelszó privát marad, a privát kulcs is privát marad.

Egy harmadik megoldás az API kulcsok felhasználásának implementálása. Ez nem egy triviális javítás, ezért a javított verzió nem ezt fogja implementálni.