Kódolás és IT biztonság
(VIHIBB01)
Laboratóriumi gyakorlat

Szoftverek biztonsági tesztelése

Szerző:
Futóné Papp Dorottya

2020. szeptember 27.

Tartalomjegyzék

1. Oktatási célok

Ezen a laboratóriumi gyakorlaton egy széles körben elterjedt biztonsági tesztelési technikával, a fuzzinggal fog megismerkedni. A technika segítségével azáltal találhatunk bugokat, hogy az elemzett szoftver számára véletlenszerű bemeneteket adunk és megfigyeljük a végrehajtását összeomlások és lefagyások után kutatva. A feladatok során össze kell állítania a demonstrált eszköz, a `python-afl`, számára az elemző környezetet és az eszközt felhasználva flageket kell megszereznie. A laboratóriumi gyakorlat elvégzésével képessé válik ezen tesztelési technikai használatára szoftverfejlesztés során.

2. Hattáranyag

A biztonsági tesztelési technikákat statikus vagy dinamikus elemzésekként csoportosíthatjuk. A *statikus* technikák nem hajtják végre a programokat, csak azok forráskód szintű utasításait vagy gépi kódját olvassák végig és értelmezik. Az értelmezés és elemzés az utasítások egy memóriabeli reprezentációján történik. Ezek a technikák jól skálázódnak és nagy kódbázist is képesek kezelni. Azonban nem férnek hozzá futási idejű információkhoz, ezért gyakran adnak hamis pozitív válaszokat: olyan kódrészleteket is sérülékenynek jelölhetnek, amik a valós életben sosem futnának le vagy sosem lehetne kihasználni őket.

A *dinamikus* elemzési technikák végrehajtás közben elemzik a szoftvert. Így ezek a technikák hozzáférnek futási idejű információkhoz, így jóval pontosabb eredményt adnak. A legnagyobb hátrányuk azonban az, hogy nem tudnak olyan viselkedést elemezni, amit nem figyelhetnek meg végrehajtás közben. Ezért ezek a technikák hamis negatív eredményt adhatnak, vagyis gyakran nem képesek minden sérülékenységet megtalálni. Ezen laboratóriumi gyakorlat során egy dinamikus elemzési technikával, a fuzzinggal fogunk megismerkedni.

2.1. Fuzzing

A fuzzoló eszközök általában három fő komponensből állnak. A *generátor* feladat a véletlenszerű bemenetek előállítása elemzés során. A generáláshoz felhasználhat mutáció alapú stratégiát (érvényes bemenetek mutálása), generálás alapú stratégiát (a bemenetek egy konfigurációs fájl alapján kerülnek

előállításra, pl. meghatározott formátummal) vagy evolóciós stratégiát (megfigyelt viselkedés alapján generált bemenetek).

A teszteset végrehajtó feladat a generált adatokat bemenetként adni az elemzett szoftvernek. Az eszközök sokszor csak bemenetek egy korlátozott halmazát képesek bemenetként adni, pl. a sztenderd inputon[1] vagy fájlokon keresztül. Amennyiben az eszköz nem képes kezelni minden olyan bemeneti forrást, amit az elemzett szoftver használ, akkor wrapper szkriptet kell készítenünk, ami átadja a generált inputot az elemzett szoftvernek.

A megfigyelő-rendszer feladat figyelni az elemzett szoftvert végrehajtás közben és detektálni az anomáliákat, összeomlások és időtúllépéseket. Feladata elvégzéséhez a megfigyelő-rendszer aktív vagy passzív szondázást valósíthat meg. Akítv szondázás esetén speciális bemeneteket adunk az elemzett szoftvernek, hogy ellenőrizhessük, válaszol-e. Passzív szondázás esetén anélkül szerzünk információt a végrehajtás állapotáról, hogy befolyásolnánk az elemzett szoftver viselkedését.

Az american fuzzy lop (`afl`[2]) egy biztonsági körökben gyakran használt fuzzoló eszköz. A dokumentációját[3] a laboratóriumi gyakorlat kötelező elolvasni! Az eszközt eredetileg natív végrehajtható fájlok elemzésére fejlesztették ki, ezért magasabb szintű nyelven készült alkalmazások elemzésére csak korlátozottan alkalmas. Egész pontosan a használatához újra kellene fordítani a magasabb szintű nyelvek futtatókörnyezetét (pl. a Python interpretert Python szriptekhez). Bár lehetséges, ez a megoldás ideje nagy részében a futtatókörnyezetben keresni bugokat és nem az elemzett szoftverben. Ezt orvosolandó, több interfész és wrapper is készült az `afl`-hez, hogy nem-natív alkalmazásokhoz is lehessen használni. Ilyen például a `kelinci`[4] Java alkalmazásokhoz és a `python-afl`[5] Pyhton szkriptekhez. A laboratóriumi gyakorlat során ezutóbbit fogjuk használni, ezért a dokumentációját a gyakorlat előtt el kell olvasni!

[1] https://en.wikipedia.org/wiki/Standard_streams
[2] http://lcamtuf.coredump.cx/afl/
[3] http://lcamtuf.coredump.cx/afl/README.txt
[4] https://github.com/isstac/kelinci
[5] https://github.com/jwilk/python-afl

3. Feladatok

A laborgyakorlat egy vezetett és két önálló feladatból áll. A vezetett rész célja az `afl` használatának bemutatása egy példa binárison keresztül. Az önálló részben az `afl` köré készített Python wrappert `python-afl` kell felhasználni két példakód elemzéséhez. A példakódok úgy lettek elkészítve, hogy csak egyetlen bement hatására omlik össze a végrehajtásuk. A feladatok megoldásához ezeket a bemeneteket kell megtalálni a `python-afl` felhasználásával. A bemenetek visszajátszásával a példakódok `RuntimeError` hibával omlanak össze és a kimenetre kiírnak egy *flaget*. A flag egy karaktersorozat, amelyet nehéz a feladat megoldása nélkül kitalálni, megtalálása így jelzi a feladat helyes megoldását.

3.1. Vezetett rész

A labor vezetett részében bemutatjuk az `afl` használatát egy példa binárison keresztül. A bemutató része a tool használatához szükséges környezet kialakítása, a fuzzolás közben megjelenített adatok értelmezése, valamint a tool kimenetének megtekintése és visszajátszása.

3.2. Feladat

Ebben a feladatban a `challenge1/challenge1.py` szkriptbn elrejtett flaget kell megtalálni. A kód a bemenetét a parancssoron megadott fájlból olvassa. A szkript elemzéséhez szükség van egy Pythonban készült wrapper szkriptre, ami jelzi a `python-afl` számára a fuzzolni kívánt kód kezdetét. Egy félig kész wrapper szkript (`challenge1/fuzzer-wrapper.py`) elérhető a feladathoz. A megoldáshoz az alábbi lépéseket kell végrehajtani:

1. Töltse ki a wrapper szkript hiányzó részeit!

2. Készítsen két mappát a `challenge1` mappában `inputs/` és `outputs/` néven! Ezekben a mappákvan fog a `python-afl` dolgozni.

3. Készítsen egy példa bemenetet tartalmazó fájlt a `python-afl` számára (`inputs/1`)! A fájlban egy darab tetszőleges karakter legyen!

4. A `challenge1` mappában állva fuzzolja a wrapper szkriptet a `py-afl-fuzz` parancs kiadásával! A parancsnak az alábbi paramétereket adja meg:

- A kezdeti bemenetek az `inputs` mappában találhatóak
- A kimeneteket az `outputs` mappába kell tenni
- A fuzzolni kívánt parancs:

  ```
  python fuzz-wrapper.py @@
  ```

A `py-afl-fuzz` kapcsolóit a `-h` kapcsolóval listáztathatja.

5. Figyelje a parancs végrehajtását és futtassa, amíg a tool meg nem találja az összeomlást eredményező bemenetet (ez akár 3-5 percig is eltarthat)!

6. Keresse meg az összeomlást eredményező bemenetet az `outputs/crashes` mappában!

7. Futtassa a `challenge1/challenge1.py` szkriptet a megtalált bemenettel!

Sikeres megoldás esetén a szkript egy `RuntimeError` hibával összeomlik és kiírja a Moodle-ben beadandó flaget.

3.3. Feladat

Ebben a feladatban a `challenge2/challenge2.py` szkriptbn elrejtett flaget kell megtalálni. A kód a bemenetét a sztenderd inputról olvassa és külön függvényben dolgozza fel.

Az előző feladatban használt `py-afl-fuzz` képes a sztenderd inputon keresztül bemeneteket adni az elemzett szoftvernek, azonban egyes operációs rendszereken stabilitási problémákba ütközik. A Python3 folyamatok a rendszer alapértelmezett karakterkódolását használják, ami általában UTF-8 a UNIX-alapú rendszerekben. Azonban a fuzzer sokszor csak egy-egy bitet változtat meg a bemenet bináris reprezentációjában, ami hibás UTF-8 kódoláshoz vezet. A probléma megoldásához a fuzzolás során használt wrapper szkriptet úgy kell módosítanunk, hogy ne a sztenderd inputra írja a generált bemenetet, hanem a sztenderd inputot képző pufferbe.

A feladat ennek a wrapper szkriptnek az elkészítése, melyhez elérhető egy skeleton (`challenge2/fuzz-wrapper.py`). A feladat megoldásához az alábbi lépéseket kell végrehajtania:

1. Töltse ki a wrapper szkript hiányzó részeit!

2. Készítsen két mappát a `challenge2` mappában `inputs/` és `outputs/` néven! Ezekben a mappákvan fog a `python-afl` dolgozni.

3. Készítsen egy példa bemenetet tartalmazó fájlt a `python-afl` számára (`inputs/1`)! A fájlban egy darab tetszőleges karakter legyen!

4. A `challenge2` mappában állva fuzzolja a wrapper szkriptet a `py-afl-fuzz` parancs kiadásával! A parancsnak az alábbi paramétereket adja meg:

   - A kezdeti bemenetek az `inputs` mappában találhatóak
   - A kimeneteket az `outputs` mappába kell tenni
   - A fuzzolni kívánt parancs:

     ```
     python fuzz-wrapper.py
     ```

   A `py-afl-fuzz` kapcsolóit a `-h` kapcsolóval listáztathatja.

5. Figyelje a parancs végrehajtását és futtassa, amíg a tool meg nem találja az összeomlást eredményező bemenetet (ez akár 3-5 percig is eltarthat)!

6. Keresse meg az összeomlást eredményező bemenetet az `outputs/crashes` mappában!

7. Futtassa a `challenge2/challenge2.py` szkriptet a megtalált bemenettel!

Sikeres megoldás esetén a szkript egy `RuntimeError` hibával összeomlik és kiírja a Moodle-ben beadandó flaget.