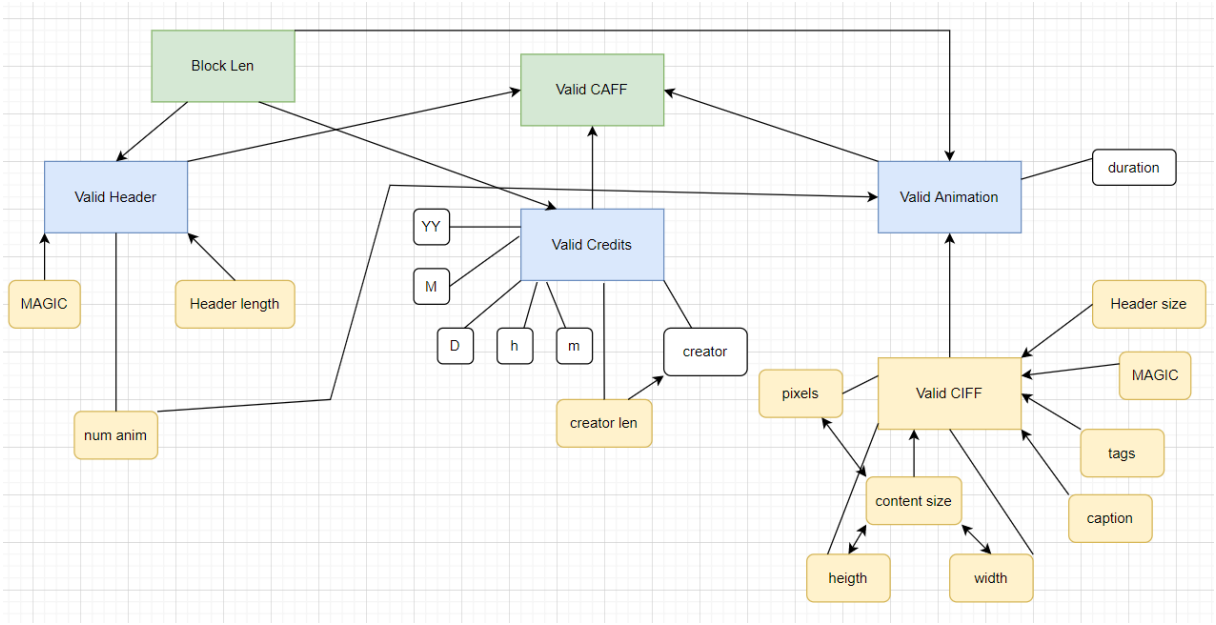
CAFF PARSER

Caff formátum bemutatása nagyvonalakban

A .CAFF (CrySyS Animation File Format) kiterjesztésű fájlok a .gif-ekhez hasonló animációs fájlok. Felépítésük szigorú szabályokat követ. Három blokk fajta fordulhat benne elő. Egy blokk három részből épül fel, első bájt azt az információt tartalmazza, milyen típusú blokkal is van dolgunk. A következő 8 bájton van reprezentálva az adott blokk hossza (length). A harmadik része a blokknak pedig pontosan length hosszú kell legyen. Ezen utolsó blokk belső felépítése változó, annak fényében, milyen típusú blokkról is van szó. Egyik a header, ez olyan alapvető információkat hordoz, mint a tény, hogy CAFF fájllal van dolgunk, a header saját mérete és a fájlban található animációk száma. Következő típusunk a credits ez metaadatokat tárol a fájlról, úgy, mint készítés ideje, készítő a hosszával együtt. Végül pedig magához az animációhoz használt blokk. Az animáció állóképek egymásutánjából áll össze, így az animáció időtartamokat tárol és a képfájlt, amelyet az adott ideig meg kell jeleníteni. Ezen fájlok, formátuma CIFF (CrySyS Image File Format). Egy CIFF fájl két részből áll, a headerből és a tartalomból. Előbbi a szükséges metaadatokat hordozza, pl. kép szélessége, magassága, míg utóbbi a kirajzolandó kép pixeleinek RGB értékét.

Parser működésének felépítése



A CAFF fájl különböző blokk típusait és azok különböző mezőit egy gráfként ábrázolva, könnyen kivehetőek a függőségek. A parseolás folyamatának elsődleges feladata, az hogy eldöntse azt a kérdést, hogy a fájl formátuma szabályos .caff formátum-e. A gráfban az irányított élek mutatják azt, hogy a nyíl forrása valamilyen hitelesítéssel tartozik afelé a tulajdonság felé, amire mutat. Az irányítatlan élek, egyszerű tartalmazást jeleznek, pl. a Valid Header (maga a header rész) tartalmaz num anim fieldet. A CAFF akkor valid (zöld) ha minden belé mutató nyíl valid, azaz a Valid Header, Valid Credits, Valid Animation csúcsok teljesülnek.

Nézzünk meg még egy példát. A Valid Header csúcs értelmezése a következő módon történik. Van a headerben Magic field, num anim field és header length field. Akkor valid a headerünk, ha a magic stimmel és a header length is rendben van, valamint, a hozzátartozó block_len tényleg validálja a header méretét. Azok a csúcsok, amelyek oda-vissza nyíllal vannak összekötve, azt jelentik, hogy az értékek kölcsönösen validálják egymást. Height esetében, a content_size validálásához jó kell legyen a height de ez vica versa is igaz, eltérés esetén ugyanis nem tudjuk, melyik érték kellene hiteles legyen, ez egy inkonzisztens állapot. Akármelyik validációs folyamat nem megfelelő, a parseolás hibás. Említésre méltó, hogy az egyes értékek bájtonként vannak reprezentálva, tehát a block len egy 8 bájton ábrázolt érték, mondjuk 44 0 0 0 0 0 0 0 az 16os számrendszerből való átkonvertálás után 68-nak felel meg, tehát a block amit definiál 68 bájt hosszú kell legyen.

- Block Len: az adott block hossza, validál minden típust
- Valid CAFF: a CAFF fájl validsága, akkor tekintendő igaznak, ha belé futó élek mindegyikének forrása igaz.
- Valid Header: header helyessége, akkor tekintendő igaznak, ha a belé futó irányított élek mindegyik validációja megfelelő
- Magic: CIFF vagy CAFF szöveg, a CIFF rész, vagy a Header validáláshoz
- num_anim: animációk száma, validálja később azt, hogy hány animáció blokk van
- Header length: a CAFF header részének hosszát validálja
- Valid Credits: a CAFF Credit részének helyessége akkor tekintendő igaznak, ha a belé futó irányított élek mindegyik validációja megfelelő
- YY: készítés éve
- M: készítés hónapja
- D: készítés napja
- h: készítés órája
- m: készítés perce
- creator_len: a creator mezőt validálja, egy dinamikus érték, hogy a creator hány bájton van tárolva
- creator: a készítő, a creator_len mező validálja
- Valid Animation: a CAFF Animation akkor tekintendő igaznak, ha a belé futó irányított élek mindegyik validációja megfelelő
- duration: az animationben található ilyen field, hány miliszekundumig kell megjeleníteni a képet
- Valid CIFF: a CIFF rész, akkor tekintendő igaznak, ha a belé futó irányított élek mindegyik validációja megfelelő
- header size: a CIFF header részének mérete, ha valóban ennyi, akkor lehet csak valid a CIFF
- tags: tagek ami opcionális a értékek a fájl esetében.\0 karakter jelzi egy tag végét. ha nincs tag, csak egy \0 karakter található itt
- caption: a képaláírás, nem lehet többsoros, végén a \n karakter jelzi
- content_size: a CIFF pixeleinek számát tartalmazza, meg kell egyezzen a width*height*3 értékkel
- height: a kép magassága

- width: a kép szélessége
- pixels: a képet alkotó pixelek, számuk meg kell egyezzen a content size-val

A parser másik fő feladata, a validáláson kívül, fájl megfelelő módon történő olvasása. Ismernie kell a fájl felépítésének struktúráját. Mezők fix sorrendjét és fix/dinamikus értékek szerinti léptetéssel kell tudnia megtalálni a kívánt adatokat.

Abban az esetben, ha valamely validáció nem történik meg (nincs Magic) nem egyezik meg 2 érték, ami meg kéne egyezen (pl. a pixelek száma és a content size) a parser nem tekinti érvényesnek a formátumot. Ilyen előállhat abban az esetben is, ha nem megfelelő a fájl összeállítása és pl a num_anim 7 bájton van ábrázolva a megadott 8 helyett. Ekkor mind az érték validáció elbukik, de később a léptetés is hibát fog eredményezni.

Statikus toolok alkalmazása

Szoftverek fejlesztésekor, fontos, hogy a megírt kód minősége egységes legyen, függetlenül attól, hogy ki írja ezt a kódot. Ez azért nagyon fontos, mert a rosszabb minőségű kód több hibát tartalmaz. Ezek nem feltétlenül működésbeli, vagy szintaktikai hibák. Nagyon sok esetben olyan csendes hibák, amelyeket emberi szemmel, intelligenciával nem, vagy csak nehezen veszünk észre, ha még nem találkoztunk megfelelő mennyiségű ilyen hibával. A kód minőségének javítását ilyen téren, pedig különböző statikus kód elemzőkkel lehet elérni, melyek meghatározott sémákat ellenőriznek a kódban.

Cppcheck:

A parser kódján futtatot statikus kód elemző szoftverek egyike a cppcheck, amelynek kimenete a képen megfigyelhető.

```
[modko42@Benis-MBP CaffParser % cppcheck cd ..
Checking ../CaffParser/CAFFParser.cpp ...
1/4 files checked 43% done
Checking ../CaffParser/CIFFParser.cpp ...
2/4 files checked 81% done
Checking ../CaffParser/CustomByteArray.cpp ...
3/4 files checked 88% done
Checking ../CaffParser/main.cpp ...
4/4 files checked 100% done
```

Cert c++:

A másik elemző, amit használtunk az pedig a SEI CERT C++ szabványt hitelesítette, ennek eredménye pedig a következő lett.

```
modko42@Benis-MBP CaffParser % python cert.py main.cpp.dump
Checking main.cpp.dump...
Checking main.cpp.dump, config ...
modko42@Benis-MBP CaffParser % python cert.py CustomByteArray.cpp.dump
Checking CustomByteArray.cpp.dump...
Checking CustomByteArray.cpp.dump, config ...
modko42@Benis-MBP CaffParser % python cert.py CAFFParser.cpp.dump
Checking CAFFParser.cpp.dump...
Checking CAFFParser.cpp.dump, config ...
modko42@Benis-MBP Downloads % python cert.py CIFFParser.cpp.dump
Checking CIFFParser.cpp.dump...
Checking CIFFParser.cpp.dump, config ...
modko42@Benis-MBP Downloads %
```

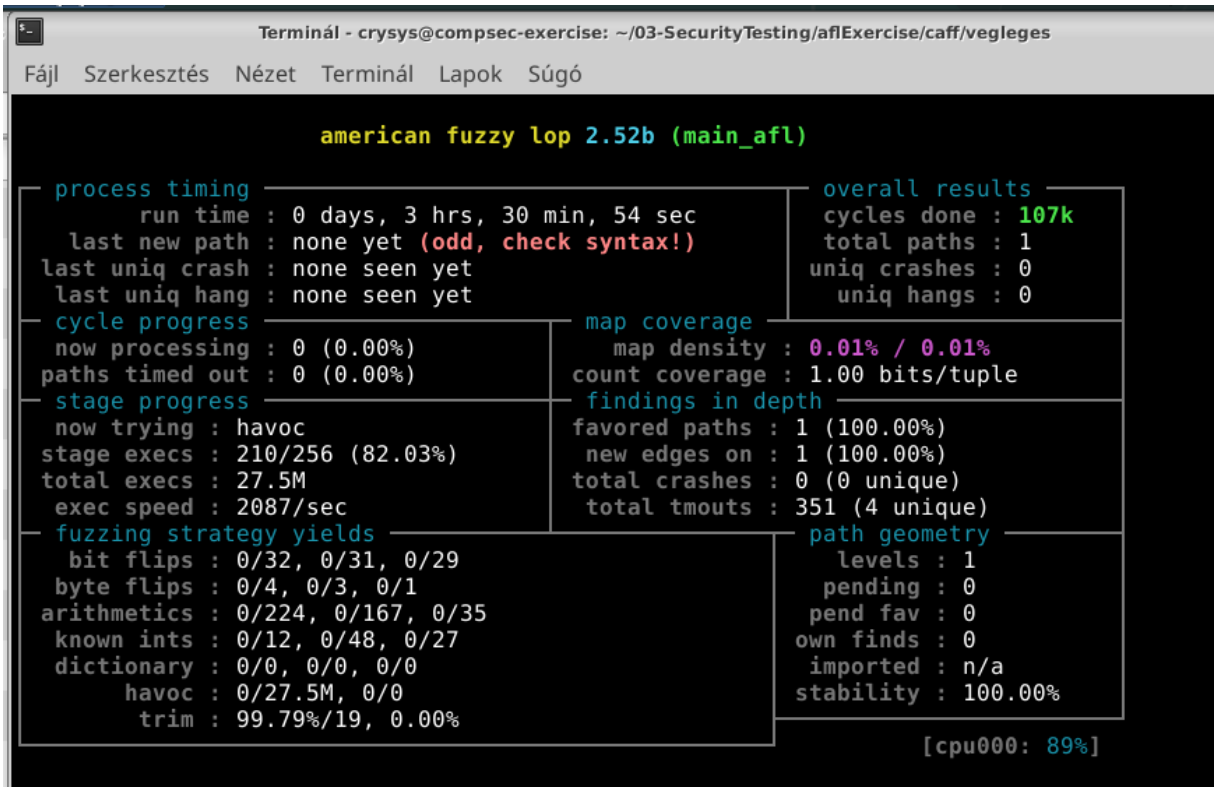
Dinamikus toolok alkalmazása

Unit tesztek:

A parser ugyan a teljes platform egyetlen eleme, viszont ezen a komponensen belül is megfigyelhetők különböző kisebb részegységek. Ezeknek a részegységeknek pedig a megfelelő működését, unit tesztekkel tökéletesen tudjuk validálni. Ezek a unit tesztek a test.cpp fájlban vannak összegyűjtve. Ahogy megfigyelhettük a CAFF Parser 3(header, credits, animation) validációt végez el, (melyek további validációkat tartalmazhatnak) hogy megfelelőnek jelezzen egy fájlt. Ezen 3 részre vannak unit tesztek, megfelelő bemenetekkel. Illetve teszteljük a parser-t egészében egy teljes caff fájllal. Valamint az animation rész hitelesítésénél, bejön a képbe a CIFF rész ellenőrzése is. Ezen részre is készültünk egy unit teszttel, mert a CIFF megfelelősége a CAFF fájl megfelelőségének feltétele. Illetve egy segéd osztályt is valósítottunk meg a CustomByteArray-t ez a különböző blokkokban lévő byte dekódolása 10es számrendszerbeli számokra. Ennek a megfelelő működésén múlik a többi osztály megfelelő működése, így ezt is unit tesztben ellenőrizzük.

Fuzzing:

A fuzzolás az a folyamat, amikor különböző bemenetekkel próbáljuk ki az alkalmazást, ezzel is tesztelve azt, hogy biztonságos és nem lehet semmilyen módon váratlan működésre bírni. A fuzzoláshoz, az AFL-fuzzert használtuk. Egy megfelelő bemenetet adva neki, a fuzzer, ezt a bemenetet módosítja ezzel keres újabb és újabb bemeneteket, ezzel tesztelve az alkalmazást.



A fuzzing nem eredményezett semmi olyan bemenetet, amitől a parser elcrashelne.

Valgrind:

A valgrind többféle beépülő modul haszálatával, ellenőrzi a bináris futtatása során a memóriahasználatot. Ellenőrzi, hogy mennyi memóriát használ az alkalmazás, hol foglal dinamikusan, ezeket hogyan kezeli, mikor ír bele, mikor olvas, összességében mikor és mennyit használja a dinamikusan kért memóriát. A dinamikus memóriakezelésnek pedig kulcsmomentuma a memória felszabadítása, ugyanis ha mi explicit kérünk memóriát az operációs rendszertől, akkor ezt valamikor vissza is kell adjuk, különben memóriaszivárgás lép fel. A fuzzing nem eredményezett olyan bemenetet, amitől az alkalmazás elcrashelne, ezáltal a már korábban használt teszt inputokat használva futtattuk a valgrindot. Ezek mindegyike hasonlóan makulátlan eredményt produkált, mint ami a mellékelt képen is látható.

```
crysys@compsec-exercise:~/03-SecurityTesting/aflExercise/caff/vegleges$ valgrind ./main
==9726== Memcheck, a memory error detector
==9726== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==9726== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==9726== Command: ./main
==9726==
1.caff
Correct format
==9726==
==9726== HEAP SUMMARY:
==9726==     in use at exit: 0 bytes in 0 blocks
==9726==   total heap usage: 60 allocs, 60 frees, 16,484,381 bytes allocated
==9726==
==9726== All heap blocks were freed -- no leaks are possible
==9726==
==9726== For counts of detected and suppressed errors, rerun with: -v
==9726== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
crysys@compsec-exercise:~/03-SecurityTesting/aflExercise/caff/vegleges$
```