# CPPCHECK

Cppcheck je nástroj na statickú analýzu kódu napísaného v jazykoch C a C++. Umožňuje odhaliť chyby, ktoré sú bežné pri písaní kódu (napr. dereferenciu NULL ukazateľov, neinicializované premenné alebo chybné prístupy do pamäte). Zameriava sa na nájdenie skutočných chýb, a nie len na bežné štylistické problémy. Knižnica sme inštalovali pomocou správcu balíkov na našom OS, čo v tomto prípade bol apt-get pre Ubuntu, a to príkazom

```
sudo apt-get install cppcheck
```

Po inštalácii jednoducho spustíme tento nástroj príkazom

```
cppcheck --enable=all main.c 2> cppcheck_output.txt
```

```
main.c:81:29: error: Uninitialized variable: buff3 [uninitvar]
                char OOBR = buff3[size3];
                            ^
main.c:15:10: style: Variable 'buff' is not assigned a value.
[unassignedVariable]
    char buff[0x1000];
         ^
main.c:81:27: style: Variable 'OOBR' is assigned a value that is never
used. [unreadVariable]
                char OOBR = buff3[size3];
                          ^
main.c:82:32: style: Variable 'OOBR_heap' is assigned a value that is
never used. [unreadVariable]
                char OOBR_heap = buff4[size3];
                                 ^
main.c:83:30: style: Variable 'buff3[size3]' is assigned a value that is
never used. [unreadVariable]
                buff3[size3] = 'c';
                             ^
nofile:0:0: information: Cppcheck cannot find all the include files (use
--check-config for details) [missingIncludeSystem]
```

Na základe výstupu z **Cppcheck** identifikujeme a popíšeme zraniteľnosti, vrátane ich potenciálnych následkov a možných opráv.

## 1. Nenačítaná premenná `buff3` (Uninitialized variable)

**Popis zraniteľnosti**: Premenná `buff3` je deklarovaná, ale nie je inicializovaná. To znamená, že premenná `buff3[size3]` môže obsahovať náhodné hodnoty z pamäte, čo spôsobuje nepredvídateľné správanie programu. Môže to viesť k chybe segmentácie alebo k použitiu neočakávaných hodnôt, čo umožňuje útočníkovi manipulovať s programom pomocou techník ako je buffer overflow.

**Oprava**: Pred použitím inicializujte `buff3`. Napríklad:

```
char buff3[10] = {0}; // inicializácia na nulové hodnoty
```

## 2. Nepriradená hodnota premennej `buff` (Unused variable)

**Popis zraniteľnosti**: `cppcheck` upozorňuje, že premenná `OOBR` dostane priradenú hodnotu, ktorá sa však nikdy nepoužije. Ide teda o zbytočné priradenie a deklaráciu premennej, ktorá nemá v kóde žiaden praktický účel

Premenná `OOBR` v riadku `char OOBR = buff3[size3];` je inicializovaná hodnotou z `buff3[size3]`, ale nikde v kóde sa nepoužíva. Takéto priradenia sú považované za neoptimálny kód

**Oprava**: Premennú `OOBR` a súvisiaci riadok môžeme jednoducho vymazať, pretože to nemá žiadny vplyv na logiku ani funkčnosť programu.

## 3. Nevyužitá hodnota `OOBR` (Unused assigned value)

**Popis zraniteľnosti**: Premenná `OOBR` je inicializovaná hodnotou `buff3[size3]`, ale táto hodnota sa nikdy nepoužíva. Navyše `buff3` môže byť neinicializovaná, takže pokus o prístup k `buff3[size3]` môže spôsobiť out-of-bounds read.

Prístup k neinicializovaným alebo mimo rozsah hodnotám môže spôsobiť, že program spadne alebo umožní útočníkovi prezerať citlivé údaje v pamäti.

**Oprava**: Odstránenie premennej `OOBR`

## 4. Nevyužitá hodnota `OOBR_heap` (Unused assigned value)

**Popis zraniteľnosti**: Rovnako ako v prípade `OOBR`, premenná `OOBR_heap` je inicializovaná hodnotou `buff4[size3]`, ktorá sa nikdy nepoužíva. Navyše `size3` môže byť mimo rozsahu `buff4`, čo spôsobí out-of-bounds read.

Rovnako ako pri `OOBR`, môže to viesť k pádu programu alebo k vyzradeniu pamäťových údajov.

**Oprava**: Ak sa nebude využívať táto premenná, tak ju jednoducho odstránime

## 5. Nevyužitá hodnota `buff3[size3]`(Unused assigned value)

**Popis zraniteľnosti**: `cppcheck` signalizuje, že hodnota priradená do `buff3[size3]` (`'c'`) nie je v programe nikdy použitá, čo znamená, že sa v programe na nič nepoužíva a pravdepodobne tam nie je potrebná

Hodnota priradená do `buff3[size3]` je zbytočná, pretože sa nikde v kóde nevyužíva. `cppcheck` to rozpoznal ako neoptimálny kód, ktorý by sa dal odstrániť

**Oprava**: Aby sme odstránili túto zbytočnú operáciu, môžeme jednoducho vymazať riadok `buff3[size3] = 'c';`. Ak nemá iný účel, odstránenie nemá žiadny dopad na logiku programu.

Tieto zraniteľnosti sú vo väčšine prípadov spôsobené nekorektnou manipuláciou s pamäťou a neinicializovanými premennými

# AFL (American Fuzzy Lop)

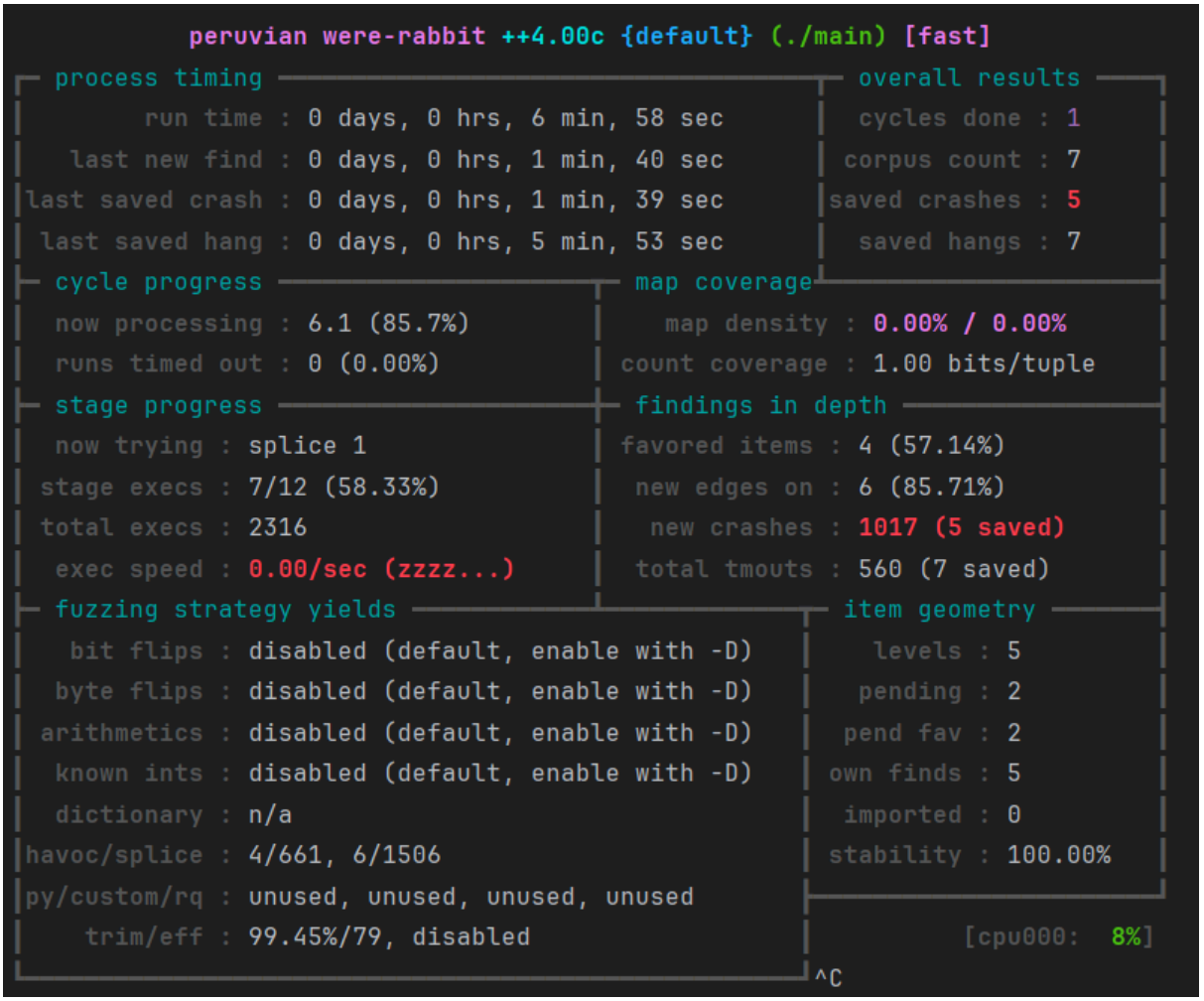
AFL je nástroj na dynamické fuzzovanie, ktorý automaticky generuje testovacie vstupy s cieľom nájsť chyby, ktoré sa prejavia pri spustení programu (ako sú buffer overflow, pamäťové úniky, atď.). Podobne ako pri cppcheck, inštalovali sme ho príkazom

```
sudo apt-get install afl
```

Spustenie nástroja prebehne príkazmi

```
 afl-gcc -o main main.c
```

```
afl-fuzz -i testcases -o findings -C -- ./main @@
```



**`new crashes: 1017 (5 saved)`**: AFL detegovalo 1017 nových pádov, pričom 5 z nich bolo uložených. Tento vysoký počet pádov naznačuje, že program je náchylný na chyby pri spracovaní niektorých vstupov

**`saved hangs: 7`**: Zablokovania programu (hangs) môžu byť spôsobené nekonečnými cyklami alebo nekonečnou rekurziou

**`total timeouts: 560 (7 saved)`**: Časové vypršania sú prípady, keď spracovanie testovacieho prípadu trvá príliš dlho, čo môže byť dôsledkom nekonečných cyklov alebo vyčerpania pamäte.

**`new edges: 6 (85.71%)`**: AFL objavilo nové hrany v ceste spracovania, čo znamená, že určité vstupy viedli program novými cestami. Táto metrika potvrdzuje, že AFL testovalo rôzne časti kódu, pravdepodobne v rôznych podmienkach (napr. keď `img.height` je 0 alebo iné hodnoty, čo môže spôsobovať rôzne chyby ako delenie nulou)

**`map density: 0.00% / 0.00%`** a **`count coverage: 1.00 bits/tuple`**: Nízka hustota pokrytia mapy môže naznačovať, že AFL nemá prístup ku všetkým častiam kódu alebo že fuzzer zatiaľ nedokázal preskúmať hlbšie časti