Міністерство освіти і науки України

Національний технічний університет України

"Київський політехнічний інститут імені Ігоря Сікорського"

Фізико-технічний інститут

АНАЛIЗ БIНАРНИХ ВРАЗЛИВОСТЕЙ

Лабораторна робота

Робота №6

Виконав студент гр. ФБ-31мп:

Шевченко С. Ш.

Київ – 2024

Тема: Отримати навички динамiчного аналiзу застосункiв методом фаззингу.

Мета роботи: Методи автоматизацiї пошуку вразливостей.

Варіант: 6 (18 у списку групи).

# Виконання завдань:

## 1) Використовуючи AFL проаналiзуйте комадну оболонку за варiантом, остання стабiльна версiя на момент виконання ЛР:

## 6. ksh

Робота виконувалася в наступному середовищі:

Хост ОС: Windows 10 x64

CPU хоста: 6 фізичних / 12 логічних ядер

RAM хоста: 16 Гб

Гіпервізор: VirtualBox 6.1

Гостьова ОС: Ubuntu 20.04 x64

CPU ВМ: 4 ядра

RAM ВМ: 4 Гб

1. Встановимо необхідні пакети:

**$ sudo apt-get install build-essential gcc clang llvm git meson ninja-build -y**

2. Встановимо [AFL](https://github.com/google/AFL):

**$ cd ~/lab6**

**$ git clone https://github.com/google/AFL.git && mv AFL afl**

**$ cd afl && make**

**$ cd llvm\_mode && make**

**$ cd .. && sudo make install**

**$ cd ~/lab6 && which afl-clang-fast**



3. Скомпілюємо [ksh](https://github.com/ksh2020/ksh):

**$ git clone https://github.com/ksh2020/ksh.git**

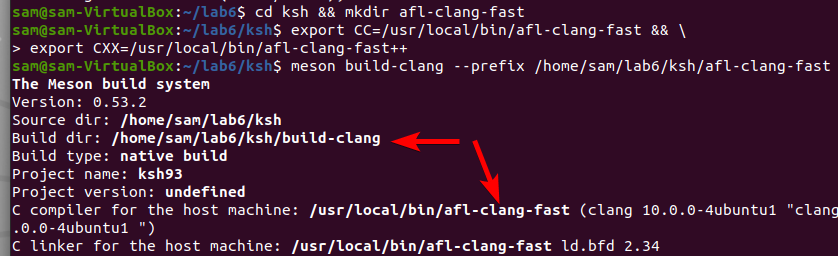
**$ cd ksh && mkdir afl-clang-fast && mkdir gcc**

3.1. Використаємо AFL компілятор afl-clang-fast:

**$ export CC=/usr/local/bin/afl-clang-fast && \**

**export CXX=/usr/local/bin/afl-clang-fast++**

**$ meson build-clang --prefix /home/sam/lab6/ksh/afl-clang-fast**



**$ ninja -C build-clang**

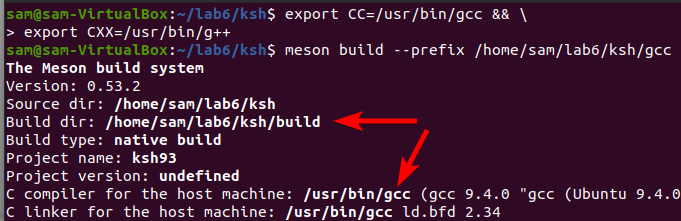
**$ ninja -C build-clang install**

3.2. Використаємо звичайний компілятор gcc:

**$ export CC=/usr/bin/gcc && \**

**export CXX=/usr/bin/g++**

**$ meson build --prefix /home/sam/lab6/ksh/gcc**



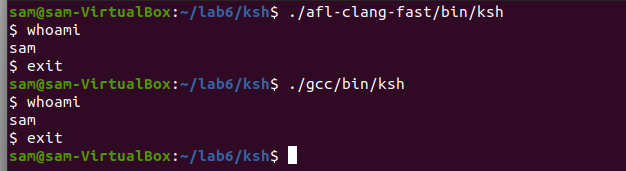
**$ ninja -C build**

**$ ninja -C build install**

3.3. Перевіримо роботу скомпільованих оболонок:

**$ ./afl-clang-fast/bin/ksh**

**$ ./gcc/bin/ksh**



4. Застосуємо деякі [оптимізації](https://github.com/google/AFL/blob/master/docs/perf_tips.txt) на рівні ОС:

**$ sudo su**

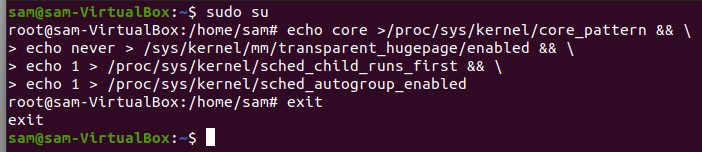
**$ echo core >/proc/sys/kernel/core\_pattern && \**

**echo never > /sys/kernel/mm/transparent\_hugepage/enabled && \**

**echo 1 > /proc/sys/kernel/sched\_child\_runs\_first && \**

**echo 1 > /proc/sys/kernel/sched\_autogroup\_enabled**

**$ exit**



5. Підготуємося перед фаззингом. Вкажемо вхідні дані "id", "whoami", "echo", "pwd":

**$ cd ~/lab6 && mkdir fuzz && cd fuzz**

**$ cp /home/sam/lab6/ksh/afl-clang-fast/bin/ksh .**

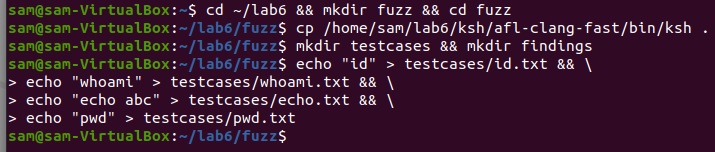
**$ mkdir testcases && mkdir findings**

**$ echo "id" > testcases/id.txt && \**

**echo "whoami" > testcases/whoami.txt && \**

**echo "echo abc" > testcases/echo.txt && \**

**echo "pwd" > testcases/pwd.txt**



6. Запустимо фаззер. [Розпаралелимо](https://github.com/google/AFL/blob/master/docs/parallel_fuzzing.txt) задачу – створимо 4 процеси: 1 головний (M=master) і 3 дитячих (S=slave). Кожний процес створюємо в своєму терміналі (TTY):

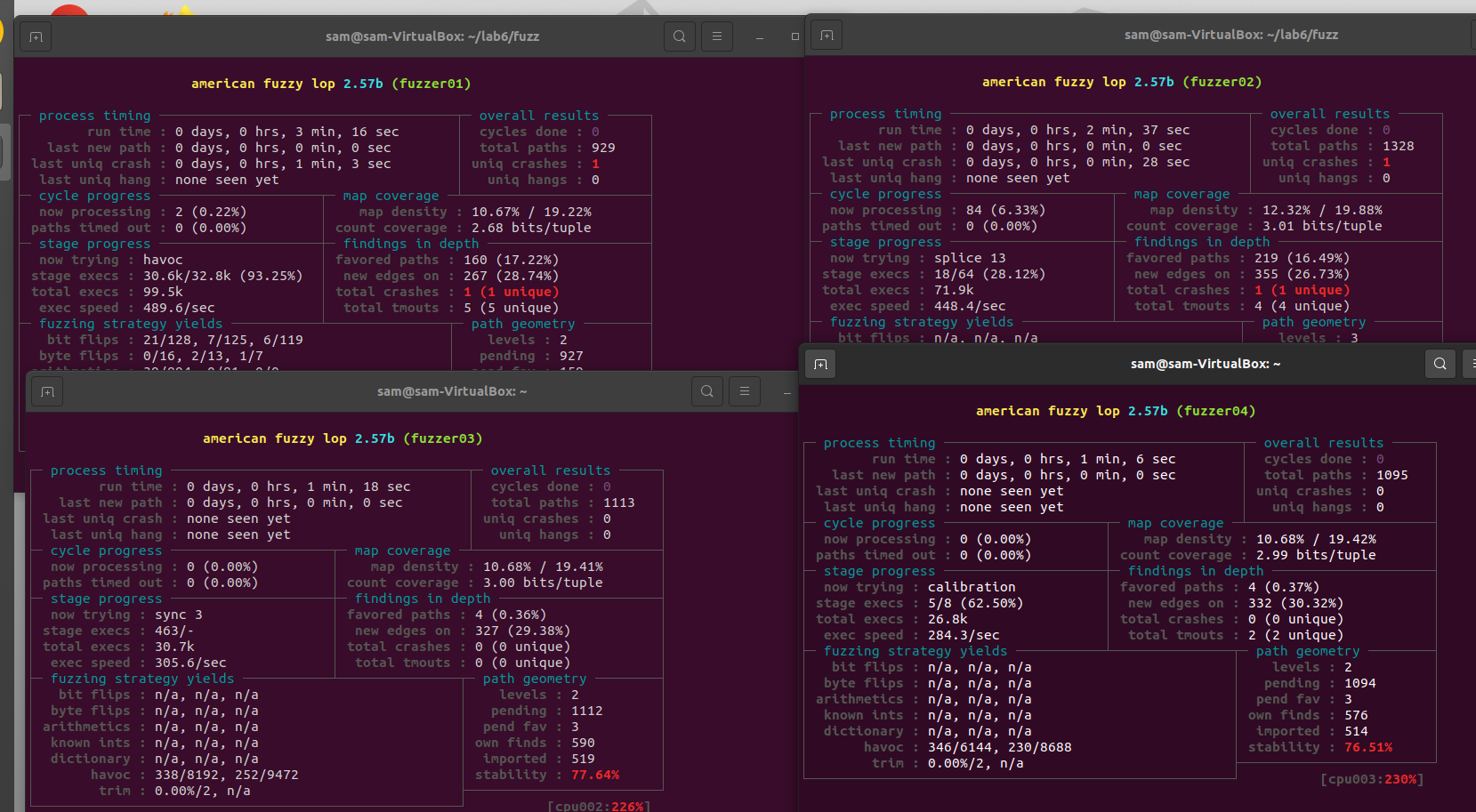
**$ cd ~/lab6/fuzz && afl-fuzz -i testcases -o findings -M fuzzer01 ./ksh**

**$ cd ~/lab6/fuzz && afl-fuzz -i testcases -o findings -S fuzzer02 ./ksh**

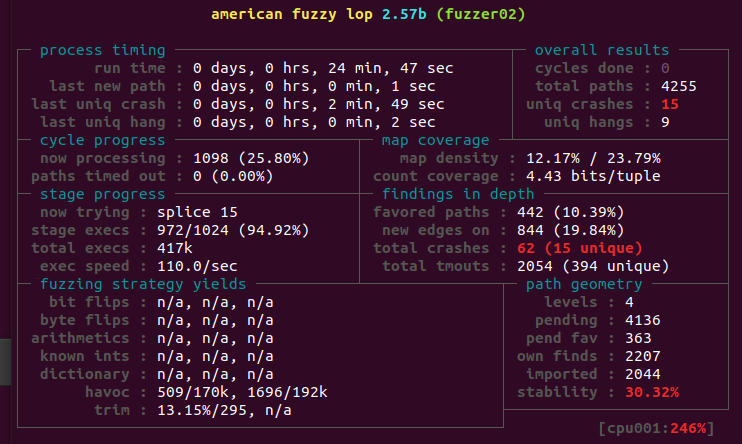
**$ cd ~/lab6/fuzz && afl-fuzz -i testcases -o findings -S fuzzer03 ./ksh**

**$ cd ~/lab6/fuzz && afl-fuzz -i testcases -o findings -S fuzzer04 ./ksh**

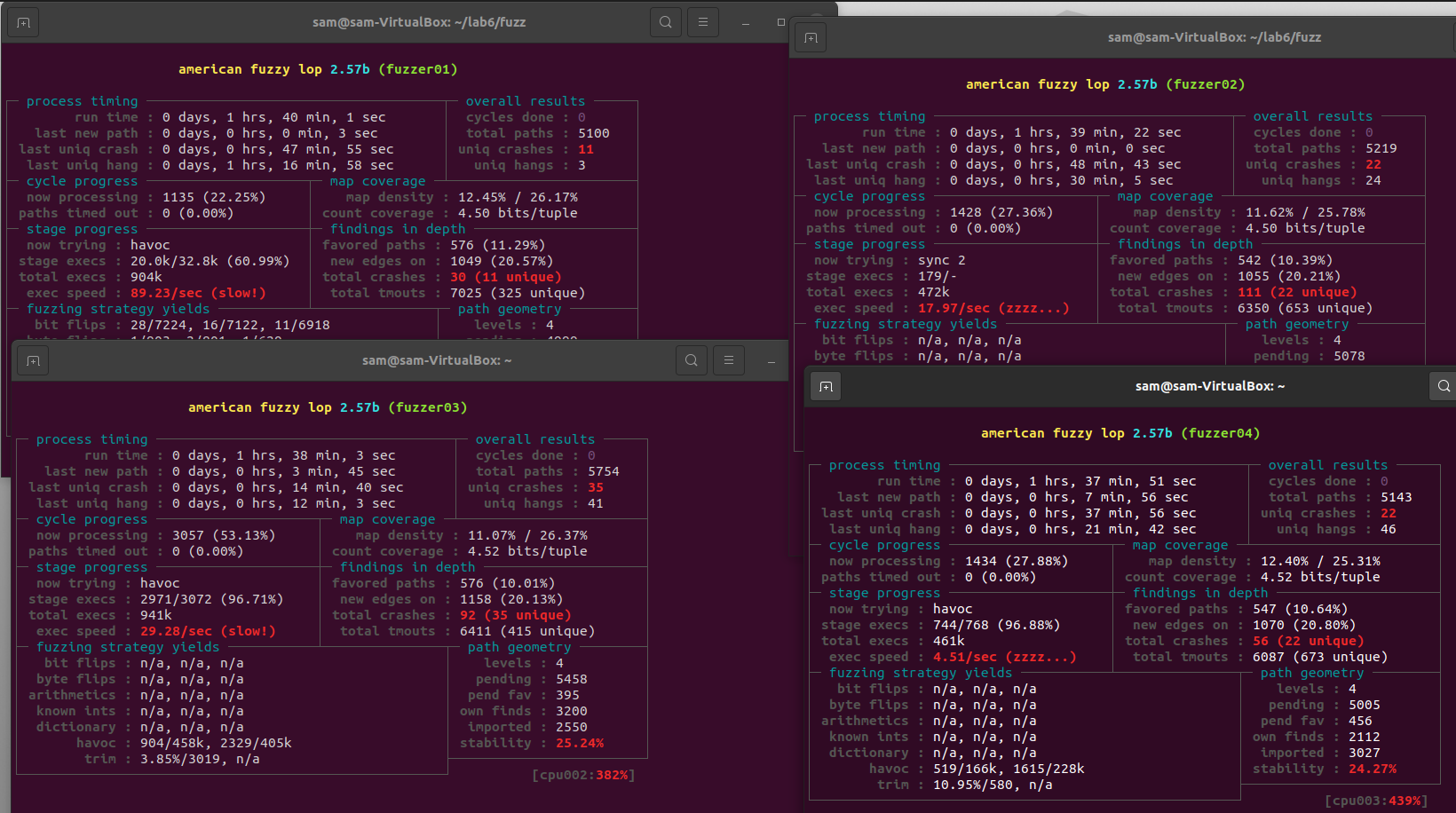
Перші ж унікальні збої знаходимо вже через 3 хв фаззінгу:



Вже через 24 хв роботи fuzzer02 знайшов 15 збоїв:

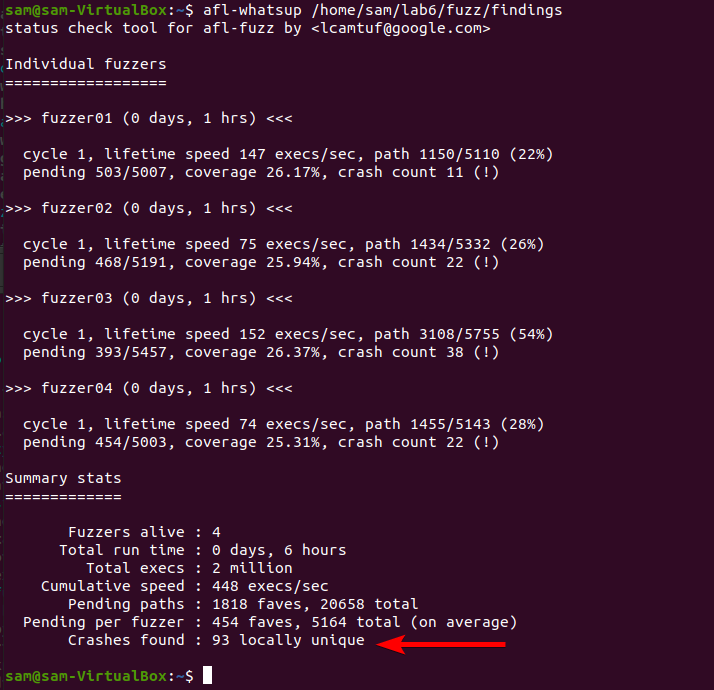


Через 1 год 40 хв роботи маємо купу збоїв:



Виведемо статус всіх процесів:

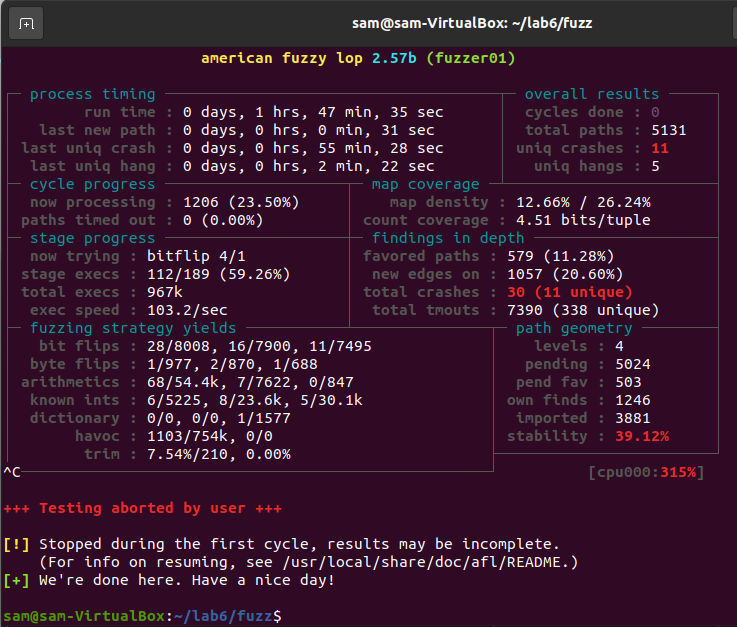
**$ afl-whatsup /home/sam/lab6/fuzz/findings**



Всього унікальних збоїв: 93

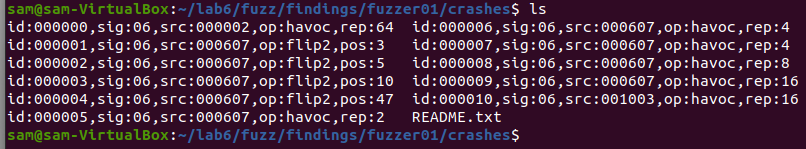
Зупинимо виконання всіх процесів (CTRL+C).

Fuzzer01:

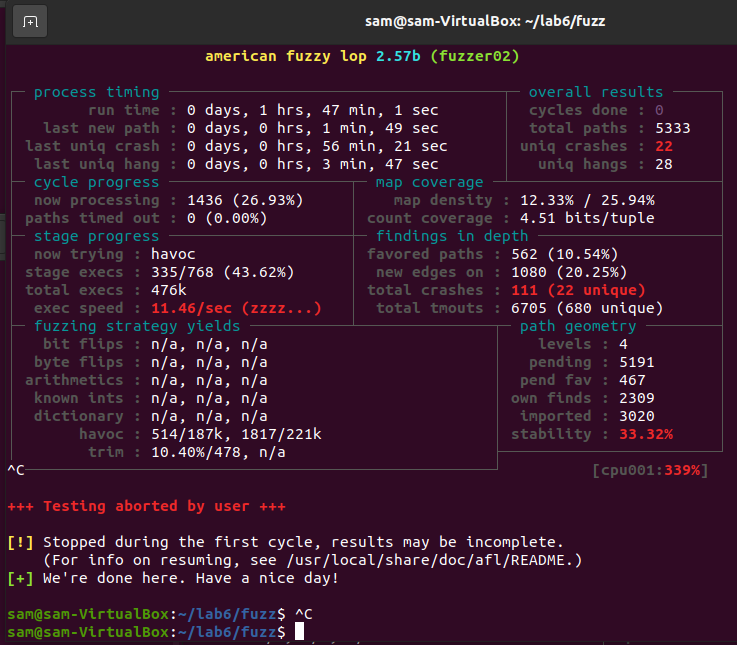


Унікальних збоїв: 11

**$ cd ~/lab6/fuzz/findings/fuzzer01/crashes**



Fuzzer02:

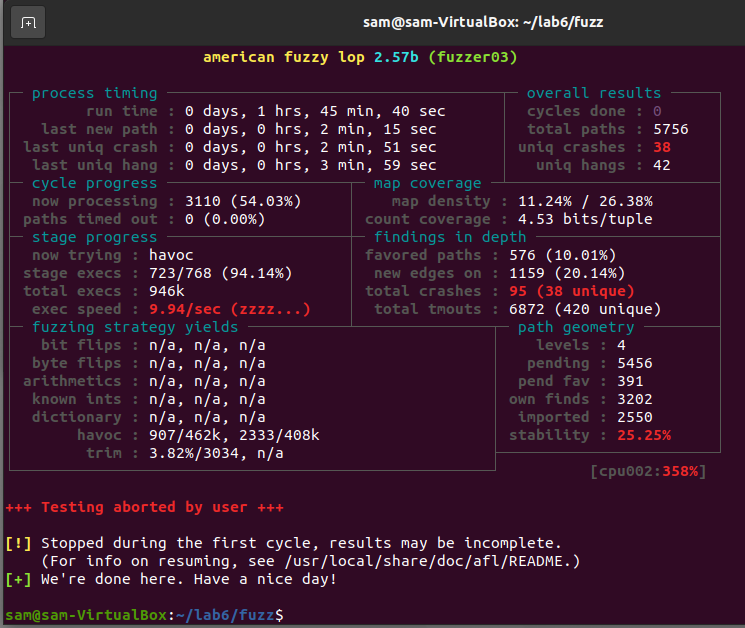


Унікальних збоїв: 22

**$ cd ~/lab6/fuzz/findings/fuzzer02/crashes**

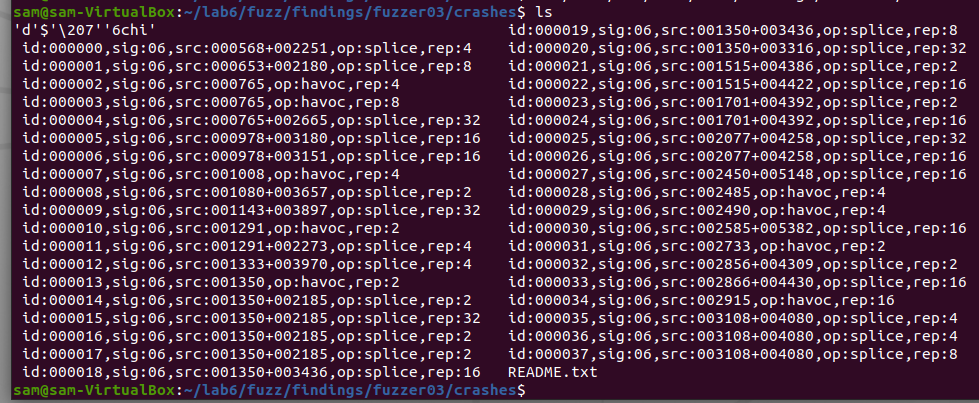


Fuzzer03:

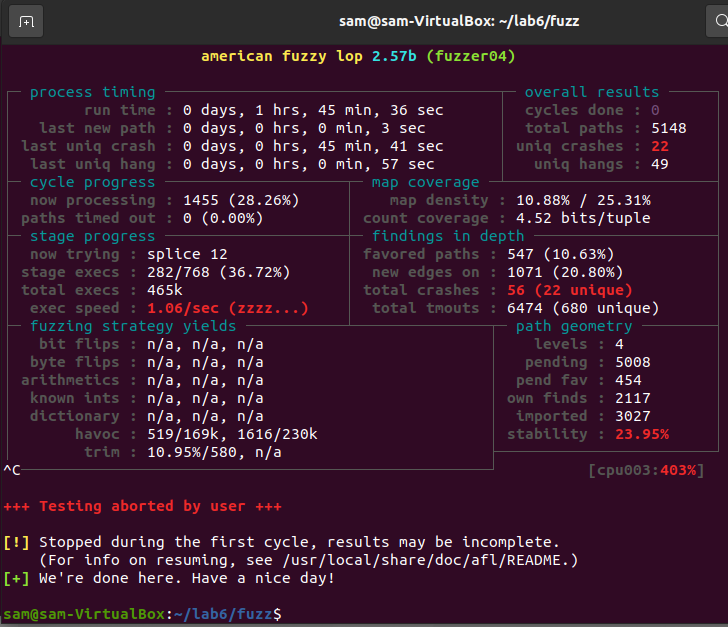


Унікальних збоїв: 38

**$ cd ~/lab6/fuzz/findings/fuzzer03/crashes**



Fuzzer04:



Унікальних збоїв: 22

**$ cd ~/lab6/fuzz/findings/fuzzer04/crashes**



## 2) Проаналiзуйте знайденi збої. Мiнiмiзуйте вхiднi данi, залиште тiльки частину, що веде безпосередньо до виключення виконання.

## 3) Оцiнiть можливостi експлуатацiї.

Вхідні дані можна мінімізувати за допомогою afl-cmin. Проте в моєму випадку мінімізація не потрібна (ввід скорочується на 0 байтів).

Отже, всього було знайдено 93 унікальних збоїв (11 + 22 + 38 + 22 = 93):

Fuzzer01: 11

Fuzzer02: 22

Fuzzer03: 38

Fuzzer04: 22

Всього: 93

1. Заходимо в кожну папку зі збоями і перевіряємо кожний файл в gdb:

**$ cd ~/lab6/fuzz/findings/fuzzer01/crashes**

**$ gdb -q ../../../ksh**

**gef➤ run < id:000000,\***

**gef➤ run < id:000001,\***

…

**gef➤ run < id:000009,\***

**gef➤ run < id:000010,\***

Таким чином було знайдено:

* 14 вхідних файлів, що призводять до SIGSEGV
* 79 вхідних файлів, що призводять до SIGABRT

[!] **Дійсно** унікальних збоїв SIGSEGV всього **4**. При подальшому ручному аналiзi iнших 10 збоїв видно, що вони зводяться до цих чотирьох.

2. Продебажимо ksh, скомпільований звичайним gcc. Використаємо в якості вводу 4 файли, про які сказано вище:

**$ cd ~/lab6 && mkdir debug && cd debug**

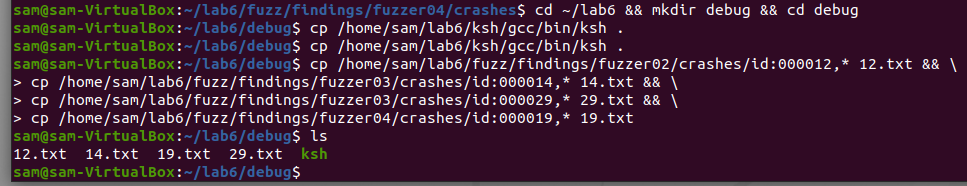
**$ cp /home/sam/lab6/ksh/gcc/bin/ksh .**

**$ cp /home/sam/lab6/fuzz/findings/fuzzer02/crashes/id:000012,\* 12.txt && \**

**cp /home/sam/lab6/fuzz/findings/fuzzer03/crashes/id:000014,\* 14.txt && \**

**cp /home/sam/lab6/fuzz/findings/fuzzer03/crashes/id:000029,\* 29.txt && \**

**cp /home/sam/lab6/fuzz/findings/fuzzer04/crashes/id:000019,\* 19.txt**

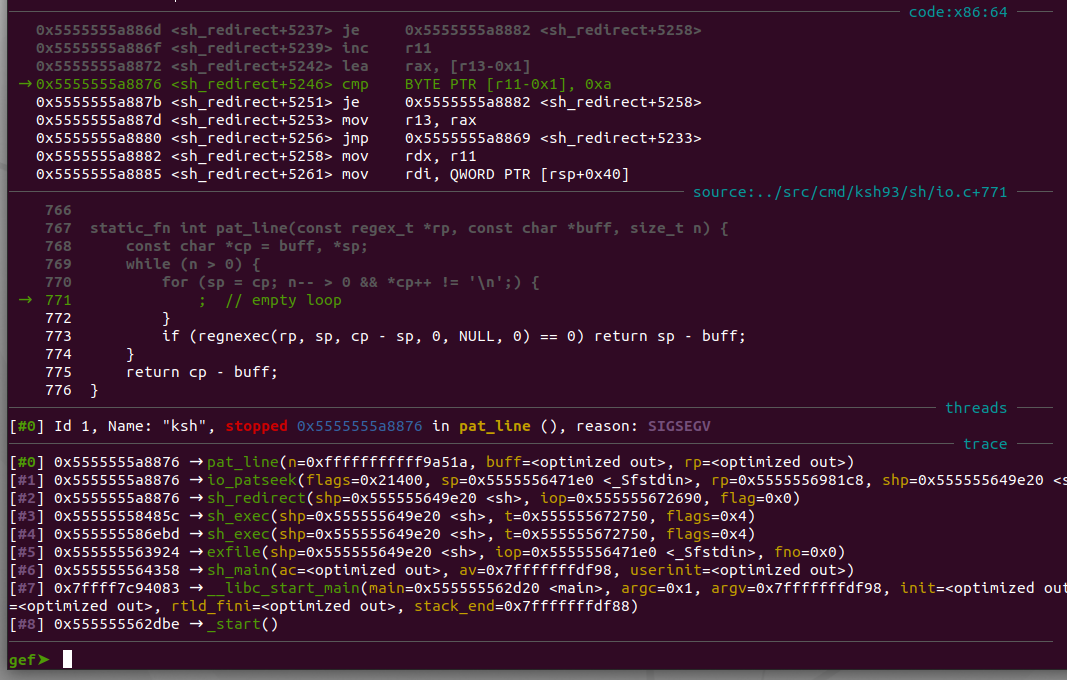


**$ gdb -q ksh**

2.1. Вхідний файл 12.txt:

**gef➤ run < 12.txt**



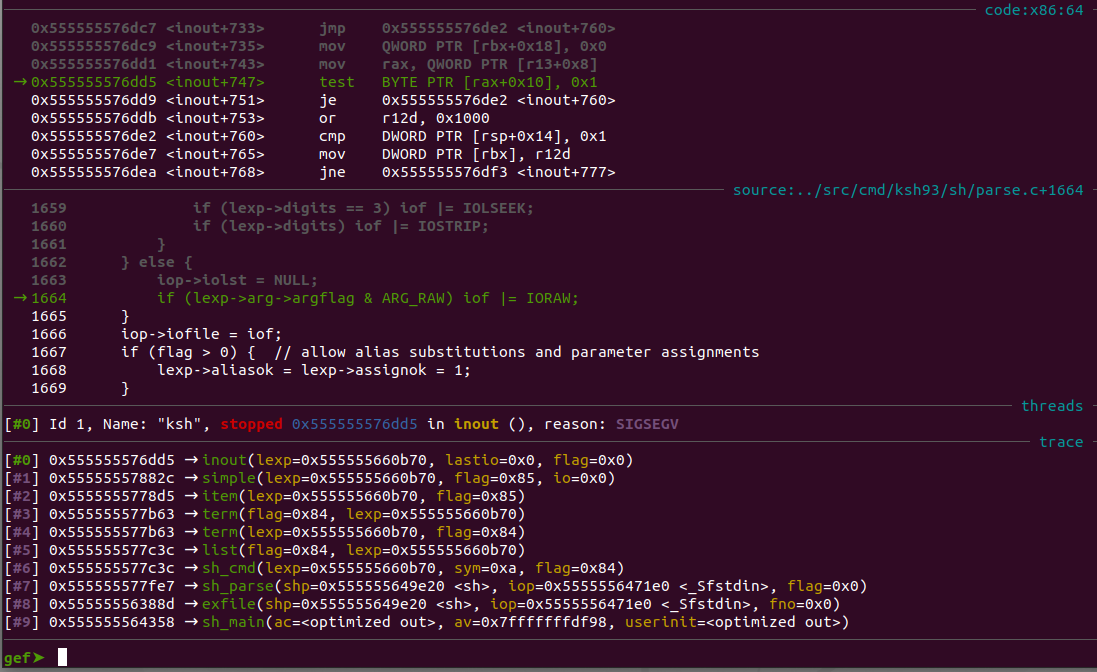


Розiменування некоректного вказiвника [r11-0x1] у рядку 771 /src/cmd/ksh93/sh/io.c. Можливiсть експлуатацiї малоймовiрна.

2.2. Вхідний файл 14.txt:

**gef➤ run < 14.txt**

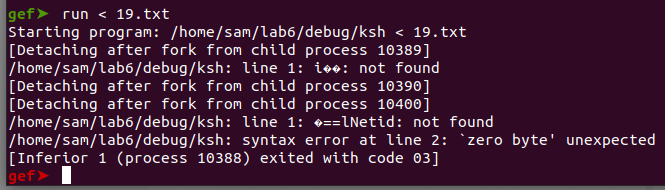




Розiменування некоректного вказiвника [rax+0x10] у рядку 1664 /src/cmd/ksh93/sh/parse.c. Можливiсть експлуатацiї малоймовiрна.

2.3. Вхідний файл 19.txt:

**gef➤ run < 19.txt**

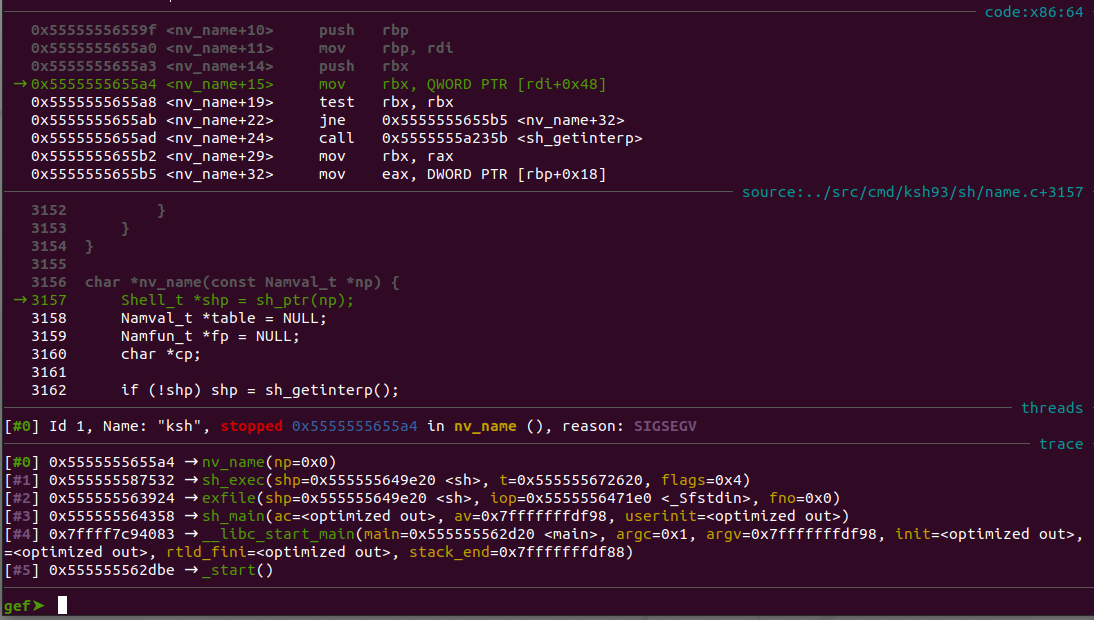


Створення 2 дитячих процесів. Існує серйозна можливість експлуатацiї.

2.4. Вхідний файл 29.txt:

**gef➤ run < 29.txt**





Розiменування некоректного вказiвника [rdi+0x48] у рядку 3157 /src/cmd/ksh93/sh/name.c. Можливiсть експлуатацiї малоймовiрна.

3. Дампи всіх 4 файлів:

**$ for file in \*.txt; do echo "=== $file"; hexdump -C "$file"; echo; done**



Особливо цікавий файл 29.txt – ksh 2020 року можна крашнути, ввівши рядок ".()T":

