

June the 2nd - Fuzzing project report

Goal (26th May - 2nd June)🏆

- **Improve** and **think** about **loop detection false positive** (function executed many times VS loop). We want **metrics** about:
 - **number of call instructions** (all of call instruction)
 - **number of unique functions** (function executed one time)
 - **number of call instructions without ret** (function executed but not returned)
- **manual review**: test with Eclipse **Ghidra** with the binary

What I have done ? 🏠

Improvements/patches

One of the improve is to make the **difference between a function which execute many times in a loop or in different places**.

To do this, we can take the **target address of the function**.

If we have the **same address instruction** and the **same target address** for 10 call, it's a **loop** which execute 10 times the same function.

Otherwise, if we have **different address instruction** but **same target address**, it's the **same function** which execute at different places.

Here is an explanation of a Pin script trace:

```
[*] Main Binary Image: /root/.fuzzing-bug-Depth-Evaluation-Internship/scripts/review_binaries/test-no-exit
[*] Image limits: 0x04134043717632 - 0x04134043726247

ins address: 0x04134043721746 => branch count: 1 => taken count: 1
ins address: 0x04134043721809 => branch count: 1 => taken count: 1
ins address: 0x04134043722054 => branch count: 1 => taken count: 1
ins address: 0x04134043722097 => branch count: 110 => taken count: 110
ins address: 0x04134043722108 => branch count: 11 => taken count: 10

call/ret ins address: 0x04134043721754 - target: 0x0 -> call count: 0 => ret count: 1
call/ret ins address: 0x04134043721864 - target: 0x0 -> call count: 1 => ret count: 0
call/ret ins address: 0x04134043721912 - target: 0x0 -> call count: 0 => ret count: 1
call/ret ins address: 0x04134043721976 - target: 0x0 -> call count: 0 => ret count: 1
call/ret ins address: 0x04134043722018 - target: 0x04134043721792 => call count: 1 => ret count: 0
call/ret ins address: 0x04134043722063 - target: 0x04134043721872 => call count: 1 => ret count: 0
call/ret ins address: 0x04134043722086 - target: 0x0 -> call count: 0 => ret count: 1
call/ret ins address: 0x04134043722103 - target: 0x0 -> call count: 0 => ret count: 10
call/ret ins address: 0x04134043722118 - target: 0x0 -> call count: 0 => ret count: 1
call/ret ins address: 0x04134043722152 - target: 0x04134043722097 => call count: 10 => ret count: 0
call/ret ins address: 0x04134043722175 - target: 0x04134043722104 => call count: 1 => ret count: 0
call/ret ins address: 0x04134043722200 - target: 0x04134043722108 => call count: 1 => ret count: 0
call/ret ins address: 0x04134043722222 - target: 0x04134043722108 => call count: 1 => ret count: 0
call/ret ins address: 0x04134043722233 - target: 0x0 -> call count: 0 => ret count: 1
call/ret ins address: 0x04134043722264 - target: 0x04134043721728 => call count: 1 => ret count: 0
call/ret ins address: 0x04134043722313 - target: 0x0 -> call count: 1 => ret count: 0
call/ret ins address: 0x04134043722340 - target: 0x0 -> call count: 0 => ret count: 1
call/ret ins address: 0x04134043722372 - target: 0x0 -> call count: 0 => ret count: 1
```

MODEL LEVEL_CORE = **PIN**, **AutoDetectControlFlow** (INS, ins)

Returns: TRUE if ins is a **Control flow instruction** and its target address is an offset from the instruction pointer or is an immediate.

When this return FALSE

Here, we have a loop which call 10 times the same function because we have the same instruction address and the same target address of the function.

Here, we have a same target address but not the same instruction address. I can conclude that we call a function in 2 different places.

Result of the Pin script:

```
The bug depth with conditional branches = 3
Number of all conditional branches = 124
The bug depth with call instructions = 1
Number of unique function = 6
Number of all call instructions = 16
```

Another improvement is to have the **number of unique functions** (unique call instruction).

I based on the **number of call** (only 1) and the **target address function** (unique and not 0x0 address).

Th script: [Pin script - bugdepthevaluation.cpp](#)

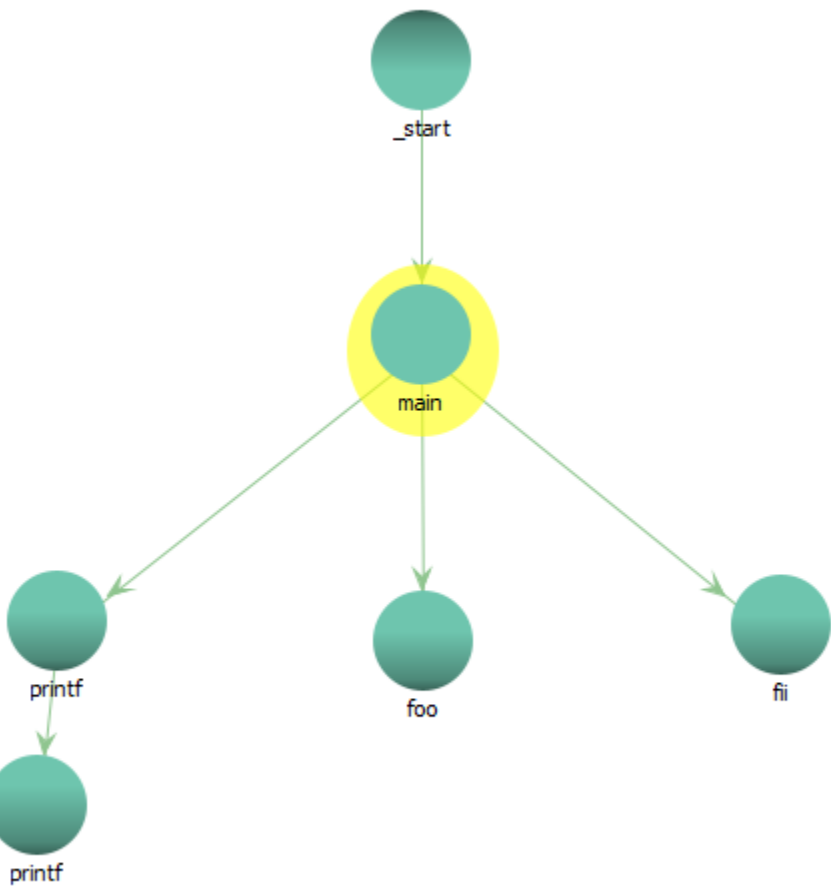
Manual review (Ghidra)

I use Ghidra to do manual review and to have a function call graph. After these improvements/patches, we have to know if they works.

We have this test script:

```
1 #include <stdio.h>
2
3 int foo() {
4     int i, x = 0;
5
6     // exit-2
7     //exit(1);
8
9     for (i = 0; i < 10; i++)
10        x += i * 2;
11
12    // exit-3
13    //exit(1);
14
15    return x;
16
17 }
18
19 int fii() {
20     // exit-1
21     //exit(1);
22
23     return 1;
24
25 }
26
27
28 int main()
29 {
30     int y, z = 0;
31
32     for (int i = 0; i < 10; i++) {
33         y += foo();
34     }
35
36     z = fii();
37
38     printf("%d\n", y);
39     printf("%d\n", z);
40 }
```

On Ghidra, we have this:



And the Pin result:

```
The bug depth with conditional branches = 3
Number of all conditional branches = 124
The bug depth with call instructions = 1
Number of unique function = 6
Number of all call instructions = 16
```

I don't understand why I have 6 for Number of unique functions and on Ghidra, 4 functions (main, printf, foo and fi). The function _start is not in the instrumentation I think because we focused on the main executable.

However, Ghidra can not find precise callgraph for indirect function calls! - Sanjay Rawat. It's maybe cause of that ?? Or it count the _start function and the printf function in double (because of the dynamically linked of the binary).

The results: [Results \(libxml and manual reviews\)](#)

Where I stopped ? 🛑

We are in the end of the first part of the internship. I can improve and fix some problems about the development of the Pin script. The goal of the next week s is to concentrate on the second part: **experiment part** !!