April the 26th - Fuzzing project report

Goal 🏆

- **Read docs** about **Intel Pin** and **test** some ManualsExamples
- **Create a script** with **PinTool** to **count branches** of a binary
- **Test** what I have done with a little binary “Hello world !!” with some conditional loop

What I have the last week ? 👨‍💻

Resources

Pin: User Guide : User Guide for PinTool
Pin: User Guide - Examples : Examples of script with PinTool
Intel Pin Tool: Setting up in 5 easy steps : Guide to setup PinTool
Branch Count using Intel Pin Tool : Guide to develop own script which counts branches using Intel Pin Tool

Intel PinTool

I read a lot the **Pin: User Guide** to learn how to use and create script using **Intel PinTool**.
The example **script** `inscount.cpp` helps me to understand the **basic of Intel PinTool scripting**.
The other example **script** `proccount.cpp` helps me to create a **count of branches specific to the binary** (not libraries or such like that).

So, finally, we have here the `procbranchcount.cpp` **script** that I created :

procbranchcount.cpp :

```
#include <fstream>
#include <iomanip>
#include <iostream>
#include <string.h>
#include "pin.H"
using std::ofstream;
using std::string;
using std::hex;
using std::setw;
using std::cerr;
using std::dec;
using std::endl;

ofstream outFile;

// Holds instruction count for a single procedure
typedef struct RtnMain
{
    string _name;
    string _image;
    ADDRINT _address;
    UINT64 _branchcount;
} RTN_MAIN;

// Linked list of instruction counts for each routine
RTN_MAIN * RtnSingle = 0;

// This function is called before every instruction is executed
VOID dobranchcount(UINT64 * branchcounter)
{
    (*branchcounter)++;
}

const char * StripPath(const char * path)
{
    const char * file = strrchr(path,'/');
    if (file)
        return file+1;
    else
        return path;
}

// Pin calls this function every time a new rtn is executed
VOID Routine(RTN rtn, VOID *v)
{
    // We did only the routine when the procedure is the 'main' function
    if (strcmp(RTN_Name(rtn).c_str(), "main") == 0) {

        // Local variable for the routine
        RTN_MAIN * RtnLocal = new RTN_MAIN;

        // The RTN goes away when the image is unloaded, so save it now
        // because we need it in the fini
        RtnLocal->_name = RTN_Name(rtn);
        RtnLocal->_image = StripPath(IMG_Name(SEC_Img(RTN_Sec(rtn))).c_str());
        RtnLocal->_address = RTN_Address(rtn);
        RtnLocal->_branchcount = 0;

        // Assigned to the global variable
        RtnSingle = RtnLocal;

        // Open the routine
        RTN_Open(rtn);

        // For each instruction of the routine
        for (INS ins = RTN_InsHead(rtn); INS_Valid(ins); ins = INS_Next(ins))
        {
                if (INS_IsBranch(ins)) {

                        // Insert a call to dobranchcount to increment the branch counter for
                        INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)dobranchcount, IARG_PTR,
                }
        }

        // Close the routine
        RTN_Close(rtn);
   }
}

// This function is called when the application exits
// It prints the name and count for main procedure
VOID Fini(INT32 code, VOID *v)
{
    outFile << setw(23) << "Procedure" << " "
          << setw(15) << "Image" << " "
          << setw(18) << "Address" << " "
          << setw(12) << "Count of branches" << endl;

    outFile << setw(23) << RtnSingle->_name << " "
          << setw(15) << RtnSingle->_image << " "
          << setw(18) << hex << RtnSingle->_address << dec << " "
          << setw(12) << RtnSingle->_branchcount << endl;
}

int main(int argc, char * argv[])
{
    // Initialize symbol table code, needed for rtn instrumentation
    PIN_InitSymbols();

    // Output of result file
    outFile.open("procbranchcount.out");

    // Initialize pin
    if (PIN_Init(argc, argv)) return Usage();

    // Register Routine to be called to instrument rtn
    RTN_AddInstrumentFunction(Routine, 0);

    // Register Fini to be called when the application exits
    PIN_AddFiniFunction(Fini, 0);

    // Start the program, never returns
    PIN_StartProgram();

    return 0;
}
```

And the little example to **test** (quickly) our script :

prog.c :

```
#include <unistd.h>
#include <string.h>

int main(int argc, char *argv[]) {
    if (argc == 2) {
        if (*(argv[1]) == '1') {
                write(1, "Hello world !!\n", 15);
        }
    }
}
```

Why this example ??

I wanted a little script to test if the script gives me **1** or **2** counted branches according to what I give to the binary.

Results :

When I send **0 argument** to the binary :

```
sh-5.0# ../../../../pin -t /opt/pin-3.18-98332-gaebd7b1e6-gcc-linux/source/tools/ManualExamples/obj-intel64/procbranchcount.so -- ~/prog
sh-5.0# cat procbranchcount.out
              Procedure           Image            Address        Count of branches
                   main            prog       560987cae149                        1
```

When I send **1 argument : ‘1’** to the binary :

```
sh-5.0# ../../../../pin -t /opt/pin-3.18-98332-gaebd7b1e6-gcc-linux/source/tools/ManualExamples/obj-intel64/procbranchcount.so -- ~/prog 1
Hello world !!
sh-5.0# cat procbranchcount.out
              Procedure           Image            Address        Count of branches
                   main            prog       555bf196a149                        2
```

The dynamic analysis focus well ont the `main` function. It send us good result according to the **number of arguments** : **0** or **1 with ‘1’**.

What about the future with Intel PinTool ?

I have to test more with complicated examples but, it’s a second part of my internship.