Zama - Obfuscator

Table of contents

A. Approach

Based on:

- my current C/C++/Rust level and general knowledge of the corresponding compiling ecosystem
- my current knowledge in cryptography

I focused on good development techniques rather than advanced obfuscation. I aimed for a full Python approach to end up with a nicely packaged solution, that could act as good base to then build upon.

Hence the concrete obfuscator remains simple, but I wanted to be able to check:

- whether bytecode was changed or not (only through standard compiler, not through optimizer)
- that the function results remained unchanged once obfuscated

This was achieved through the use of CCFI, and using `gcc` and `strip` through `subprocess` call from Python.

To find some obfuscating substitutions I looked in the obfuscator-llvm repository.

B. Fails and Next steps

Some fails:

- Parser: I began on trying out to use an actual parser on the code to get an AST of the code in order to be able to identify function, variables, types, etc. more easily, but I was affraid I wouldn't end up with something usable by the ned of this week.
- Manipulating bytecode: It is how I stumbled upon `obfuscator-llvm` and later the article obfuscating-java-bytecode-with-llvm-and-epona. Looking at the source code of `obfuscator-llvm`, I changed my approach to the one I ending up following.

Next steps:

- Multiple pass: This would need to be tested, but it seems simple enough
- Extend to other operations: implementing the substitutions from obfuscator-llvm would be easily feasible

- Handling other language (only C) : I focused on C, and based on how the tool is architectured and organized there would be quite some work to make support other language the same way we support C.

C. Basic usage

Documentation for installation can be found in the `README.md` at the root of the directory once unzipped.

After installing requirements and if running from source :

```
$ python -m obfuscator --help
$ python -m obfuscator demo --help
```

Let's go through some examples:

Passthrough obfuscator with an example file

```
$ python -m obfuscator obfuscate obfuscator/data/c_function_examples/pi.c
>> Level (0) uses (PassthroughObfuscator)

# include <stdlib.h>
# include <math.h>

/* Returns a very crude approximation of Pi
   given a int: a number of iteration */
float pi_approx(int n){

  double i,x,y,sum=0;

  for(i=0;i<n;i++){

    x=rand();
    y=rand();

    if (sqrt(x*x+y*y) < sqrt((double)RAND_MAX*RAND_MAX))
      sum++; }

  return 4*(float)sum/(float)n; }
```

Specifying a level of obfuscation

```
$ python -m obfuscator obfuscate obfuscator/data/c_function_examples/pi.c --level 5
>> Level (5) uses (HarderToRead)

# include <stdlib.h>
# include <math.h>
/*Returns a very crude approximation of Pi
given a int: a number of iteration*/
```

```
float pi_approx(int n){double i,x,y,sum=0;for(i=0;i<n;i++){x=rand();y=rand();if (sqrt(x*x+y*
```

Specifying a level of obfuscation, running the original and obfuscated version with passed arguments

```
$ python -m obfuscator obfuscate --level 10 obfuscator/data/c_function_examples/sum42.c 1 2
>> Level (10) uses (ReplacementObfuscator)

#include <stdlib.h>
#include <stdint.h>

uint8_t f(uint32_t a, uint32_t b, uint32_t c)
{
    uint8_t res;
    c = (-(-a + (-b)));
    res = (-(-c + (-42)));
    res = (~res & 420) | (res & ~420);
    return res;
}



>> Run original with args ((1, 2, 3))
>> Results: 137

>> Run obfuscated with args ((1, 2, 3))
>> Results: 137
```

**Demoing: pass an example through the 3 different obfuscators, run with passed args and compare results.

```
$ python -m obfuscator demo sum42.c 1 2 3
>> Level (0) uses (PassthroughObfuscator)

#include <stdint.h>

uint8_t f(uint32_t a, uint32_t b, uint32_t c)
{
    uint8_t res;
    c = a + b;
    res = c + 42;
    res = res ^ 420;
    return res;
}



>> Run obfuscated level Level (0) uses (PassthroughObfuscator) with args ((1, 2, 3))
```

```
>> Results: 137

>> Level (5) uses (HarderToRead)

#include <stdint.h>
uint8_t f(uint32_t a,uint32_t b,uint32_t c){uint8_t res;c=a+b;res=c+42;res=res^420;return re


>> Run obfuscated level Level (5) uses (HarderToRead) with args ((1, 2, 3))
>> Results: 137

>> Level (10) uses (ReplacementObfuscator)

#include <stdlib.h>
#include <stdint.h>

uint8_t f(uint32_t a, uint32_t b, uint32_t c)
{
    uint8_t res;
    c = (-(-a + (-b)));
    res = (-(-c + (-42)));
    res = (~res & 420) | (res & ~420);
    return res;
}



>> Run obfuscated level Level (10) uses (ReplacementObfuscator) with args ((1, 2, 3))
>> Results: 137
```