



**Ofuscadonator 3000**

# DESENVOLVEDORES

**Bruno Rodrigues**

「brunorodrigues.ufs@gmail.com」

Git: BrunoRodriguesDev

**Tulio Caldas**

「actulio@gmail.com」

Git: actulio

**ENGENHARIA  
REVERSA !**

# Engenharia Reversa

Técnica que tem como objetivo entender o funcionamento de um produto.



# Ofuscação de código

Técnica para dificultar o processo de engenharia reversa.



# **ALGORITMO GENÉTICO !**

## Gene

Cada gene representa uma instrução do código.

## Cromossomo

Um conjunto de genes.

Cada cromossomo representa um código.



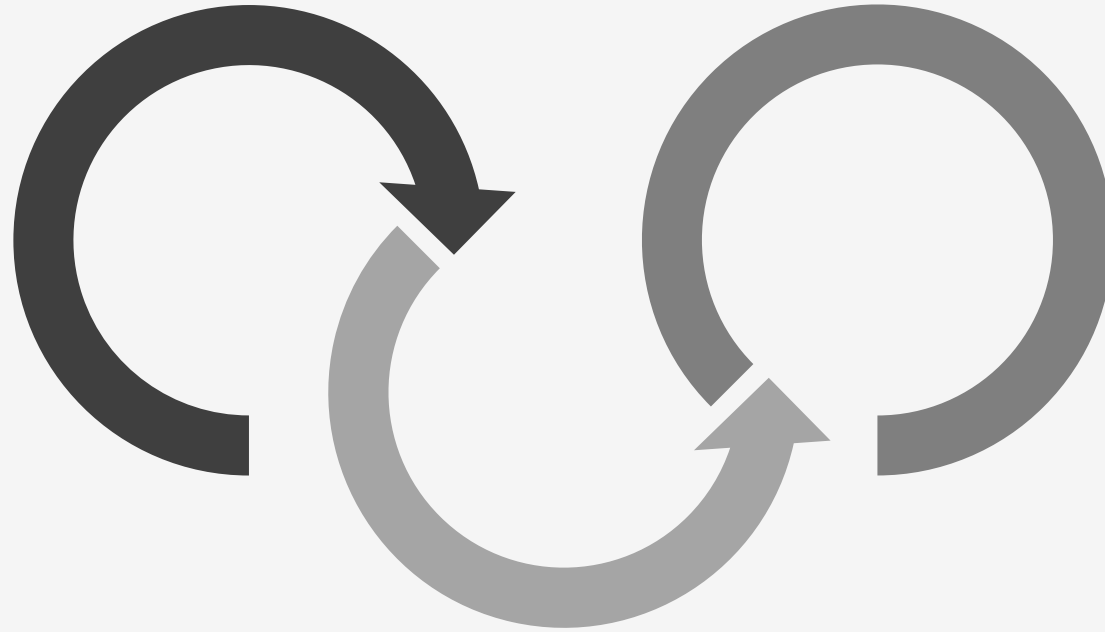
## Função de aptidão

Avalia se o cromossomo é apto ou não apto. A aptidão ocorre quando o cromossomo possui uma instrução a mais que a geração anterior e o comportamento em relação ao Código original não é alterado.

## Mutação

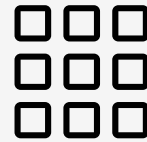
O processo de mutação consiste em adicionar um gene aleatório em um cromossomo dado.

# Processo evolutivo



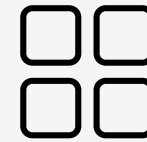
Criação de uma população

População de um indivíduo  
(Código original)



Evolução

Processo de variação do  
cromossomo através de mutação,



Seleção

Seleção de cromossomos aptos  
através de uma função de aptidão.



**ABI**  
Não é Application Programming Interface (API)  
– Bruno Prado



# ABI



## Representação dos dados

Tipo, tamanho, alinhamento.



## Registradores

De pilha, de argumento, de retorno, de propósito geral.



## Padrões de instruções

Intel386 de 32 bits, AMD64 de 64 bits.

**FERRAMENTAS  
UTILIZADAS !**

# x86 and amd64 instruction reference

Derived from the May 2019 version of the [Intel® 64 and IA-32 Architectures Software Developer's Manual](#). Last updated 2019-05-30.

**THIS REFERENCE IS NOT PERFECT.** It's been mechanically separated into distinct files by a dumb script. It may be enough to replace the official documentation on your weekend reverse engineering project where money is at stake, go get the official and freely available documentation.

## Core Instructions

Mnemonic	Summary
<a href="#">AAA</a>	ASCII Adjust After Addition
<a href="#">AAD</a>	ASCII Adjust AX Before Division
<a href="#">AAM</a>	ASCII Adjust AX After Multiply
<a href="#">AAS</a>	ASCII Adjust AL After Subtraction
<a href="#">ADC</a>	Add with Carry
<a href="#">ADCX</a>	Unsigned Integer Addition of Two Operands with Carry Flag
<a href="#">ADD</a>	Add
<a href="#">ADDPD</a>	Add Packed Double-Precision Floating-Point Values
<a href="#">ADDPS</a>	Add Packed Single-Precision Floating-Point Values
<a href="#">ADDSD</a>	Add Scalar Double-Precision Floating-Point Values
<a href="#">ADDSS</a>	Add Scalar Single-Precision Floating-Point Values
<a href="#">ADDSUBPD</a>	Packed Double-FP Add/Subtract
<a href="#">ADDSUBPS</a>	Packed Single-FP Add/Subtract
<a href="#">ADOX</a>	Unsigned Integer Addition of Two Operands with Overflow Flag
<a href="#">AESDEC</a>	Perform One Round of an AES Decryption Flow
<a href="#">AESDECLAST</a>	Perform Last Round of an AES Decryption Flow
<a href="#">AESENC</a>	Perform One Round of an AES Encryption Flow
<a href="#">AESENC LAST</a>	Perform Last Round of an AES Encryption Flow
<a href="#">AESIMC</a>	Perform the AES InvMixColumn Transformation
<a href="#">AESKEYGENASSIST</a>	AES Round Key Generation Assist
<a href="#">AND</a>	Logical AND
<a href="#">ANDN</a>	Logical AND NOT
<a href="#">ANDNPD</a>	Bitwise Logical AND NOT of Packed Double Precision Floating-Point Values

## INC — Increment by 1

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
FE /0	INC <i>r/m8</i>	M	Valid	Valid	Increment <i>r/m</i> byte by 1.
REX + FE /0	INC <i>r/m8</i> *	M	Valid	N.E.	Increment <i>r/m</i> byte by 1.
FF /0	INC <i>r/m16</i>	M	Valid	Valid	Increment <i>r/m</i> word by 1.
FF /0	INC <i>r/m32</i>	M	Valid	Valid	Increment <i>r/m</i> doubleword by 1.
REX.W + FF /0	INC <i>r/m64</i>	M	Valid	N.E.	Increment <i>r/m</i> quadword by 1.
40+ <i>rw</i> **	INC <i>r16</i>	O	N.E.	Valid	Increment word register by 1.
40+ <i>rd</i>	INC <i>r32</i>	O	N.E.	Valid	Increment doubleword register by 1.

\* In 64-bit mode, *r/m8* cannot be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

\*\* 40H through 47H are REX prefixes in 64-bit mode.

### Instruction Operand Encoding ¶

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM: <i>r/m</i> ( <i>r</i> , <i>w</i> )	NA	NA	NA
O	opcode + <i>rd</i> ( <i>r</i> , <i>w</i> )	NA	NA	NA

### Description ¶

Adds 1 to the destination operand, while preserving the state of the CF flag. The destination operand can be a register or a memory location. This instruction allows a loop counter to be updated. (This instruction is identical to the `ADD` instruction with an immediate operand of 1 to perform an increment operation that does update the CF flag.)

This instruction can be used with a `LOCK` prefix to allow the instruction to be executed atomically.

In 64-bit mode, `INC r16` and `INC r32` are not encodable (because opcodes 40H through 47H are REX prefixes). Otherwise, the instruction's 64-bit mode default operation size is 32 bits. Use additional registers (R8-R15). Use of the `REX.W` prefix promotes operation to 64 bits.

## Assembly

Raw Hex (zero bytes in bold):

4805785634124881C6785634124801C04801F94901C04901EC

String Literal:

"\x48\x05\x78\x56\x34\x12\x48\x81\xC6\x78\x56\x34\x12\x48\x01\xC0\x48\x01\xF9\x49\x01\xC0\x49\x01\xEC"

Array Literal:


```
{ 0x48, 0x05, 0x78, 0x56, 0x34, 0x12, 0x48, 0x81, 0xC6, 0x78, 0x56, 0x34, 0x12, 0x48,
0x01, 0xC0, 0x48, 0x01, 0xF9, 0x49, 0x01, 0xC0, 0x49, 0x01, 0xEC }
```

Disassembly:

```
0: 48 05 78 56 34 12    add    rax,0x12345678
6: 48 81 c6 78 56 34 12    add    rsi,0x12345678
d: 48 01 c0             add    rax,rax
10: 48 01 f9             add    rcx,rdi
13: 49 01 c0             add    r8,rax
16: 49 01 ec             add    r12,rbp
```

# Instruções implementadas

.....

<code>inc reg</code> <code>dec reg</code> <code>cmp reg, reg</code> <code>xor reg, reg</code> <code>xor reg, im32</code> <code>add reg, reg</code> <code>add reg, im32</code>	<code>bswap reg</code> <code>not reg</code> <code>neg reg</code> <code>and reg, reg</code> <code>and reg, im32</code> <code>or reg, reg</code> <code>or reg, im32</code> <code>clc</code> 
---	--

`clc` – Limpa flag do carry (clear carry flag)  
`bswap` – Troca a "endianess" dos bytes (byteswap)  
    `not` - complemento de um  
    `neg` - complemento de dois

```
[New Thread 0x7ffff7a53700 (LWP 8980)]  
[New Thread 0x7ffff7252700 (LWP 8981)]  
[Thread 0x7ffff7a53700 (LWP 8980) exited]  
[New Thread 0x7ffff7252700 (LWP 8982)]  
[Thread 0x7ffff7252700 (LWP 8981) exited]  
[New Thread 0x7ffff7a53700 (LWP 8983)]
```

Thread 88 "main" received signal SIGSEGV, Segmentation fault.

```
[Switching to Thread 0x7ffff7252700 (LWP 8982)]
```

```
0x00007ffff7fc703b in ?? ()
```

```
(gdb) l
```

```
520
```

```
521     uint32_t getChromossomeSize(Chromossome &chromossome){
```

```
522         uint32_t size = 0;
```

```
523         for(uint32_t k = 0; k < chromossome.chromossome.size(); k++){
```

```
524             size += chromossome.chromossome[k].size;
```

```
525         }
```

```
526         return size;
```

```
527     }
```

```
528
```

```
529     int main(){
```

```
(gdb) █
```

「GDB」



**PROBLEMAS  
TÉCNICOS !**

# Tamanho das instruções

.....—.....

Criar uma estrutura do tipo MAP com o tamanho de cada tipo de instrução usada. Porém o jeito certo seria ter uma espécie de assembler, que por si só seria um projeto.



## Possibilidade do programa entrar em loop durante a inserção das instruções

.....—.....

Criação de uma thread que roda o código e uma thread que serve de Watcher verificando se um tempo determinado passou ou não. Caso o tempo tenha passado, a thread é cancelada.



# População muito grande

.....—.....

Estudo de caso: Se temos 5 mutações para cada cromossomo, supondo todos dando certo, em 10 gerações temos aproximadamente 10 milhões de cromossomos.



Solução: Variável limitadora de quantos genes aptos por geração são suficientes.

## Cancelamento de uma thread em loop

.....—.....

Para que uma thread que entrou em loop responda a um sinal ela precisa que seu tipo de cancelamento seja alterado de `PTHREAD_CANCEL_DEFERRED` (espera até que um ponto de cancelamento seja encontrado) para `PTHREAD_CANCEL_ASYNCCHRONOUS` (pode ser cancelada a qualquer momento).

## Cancelamento de thread de forma assíncrona

Explicação: Cancelar a thread de forma assíncrona poderia deixar algumas estruturas como o Vector em um estado inconsistente.

Solução: Mover tudo o que fosse possível para fora da thread e passar apenas as referências.

## Alteração dos desvios relativos a cada mutação

.....—.....

Criada uma estrutura com um mapeamento da localização de todas as instruções de desvio de um determinado cromossomo e seus devidos endereços de destino. A cada mutação, os endereços eram recalculados.

## SIGFPE devido a operação de divisão por zero

.....—.....

Criado um signal handler que utiliza de uma estrutura `ucontext_t` que contém o contexto do caminho de execução do processo e é alterada para pular a instrução que gerou o problema.



## Pthread\_cond\_timedwait demorando muito tempo

.....—.....

Explicação: Uma função que espera um TIMEOUT ou uma Condição.

Problema: A sincronização entre a thread que enviava o sinal para a variável de condição e a thread que verificava o sinal demorava muito mais que a execução na memória do cromossomo.

Solução: usar um loop e uma variável para verificar se o tempo passou.

## SIGSEGV na execução do cromossomo

.....—.....

Explicação: Como os registradores usados nas instruções são escolhidos de forma aleatória, mudar os registradores \$rsp e \$rbp (topo e quadro da pilha, respectivamente) causa problemas no acesso ao segmento de memória pela função subsequente.

Solução: Não permitir que os registradores \$rsp e \$rbp fossem escolhidos.

## SIGSEGV em funções aleatórias

.....—.....

Empilhar e desempilhar manualmente os registradores não preservados (uma vez que pusha e popa não são compatíveis com o modo 64 bits) pareceu resolver o problema.

## \*Executar código na memória

.....—.....

Por algum bug do GCC, só funciona quando compilado com diretivas de otimização, o que infelizmente dificulta na hora de depurar no GDB.

# **CÓDIGO IMPLEMENTADO**

<https://github.com/BrunoRodriguesDev/Ofuscador>

