OBFUSKÁCIA S VYUŽITÍM LLVM

Roman Oravec
Europen 2022

O čom to bude?

1. Obfuskačné transformácie
2. LLVM
3. Implementácie

OBFUSKÁCIA

Transformuje program

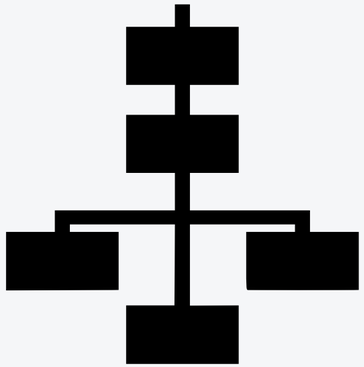
Cieľom je sťažiť analýzu

Funkcionalita musí byť zachovaná

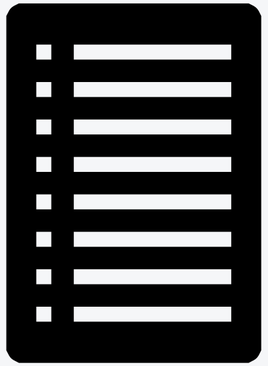
https://www.ioccc.org/2020/yang/prog.c

Čo obfuskovať?

Riadiace štruktúry



Dáta



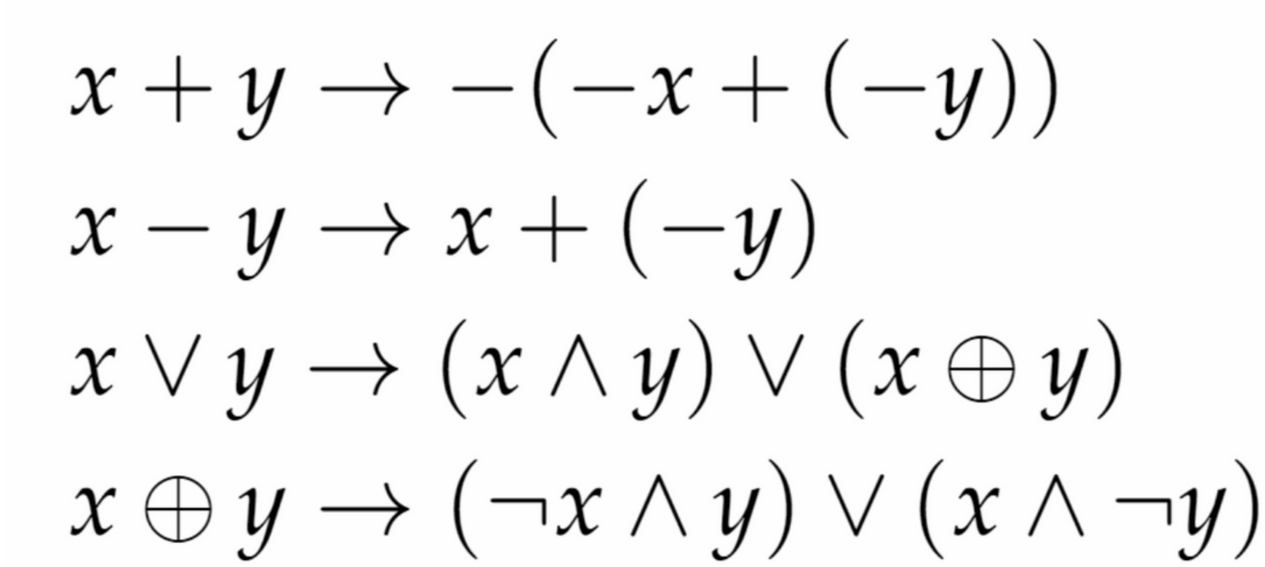
Kto je "utočník"?

Reverzní inžinieri

Nástroje

- Triton
- Angr
- Z3
- ...

Zámena inštrukcií



$$x + y \to -(-x + (-y))$$
$$x - y \to x + (-y)$$
$$x \vee y \to (x \wedge y) \vee (x \oplus y)$$
$$x \oplus y \to (\neg x \wedge y) \vee (x \wedge \neg y)$$

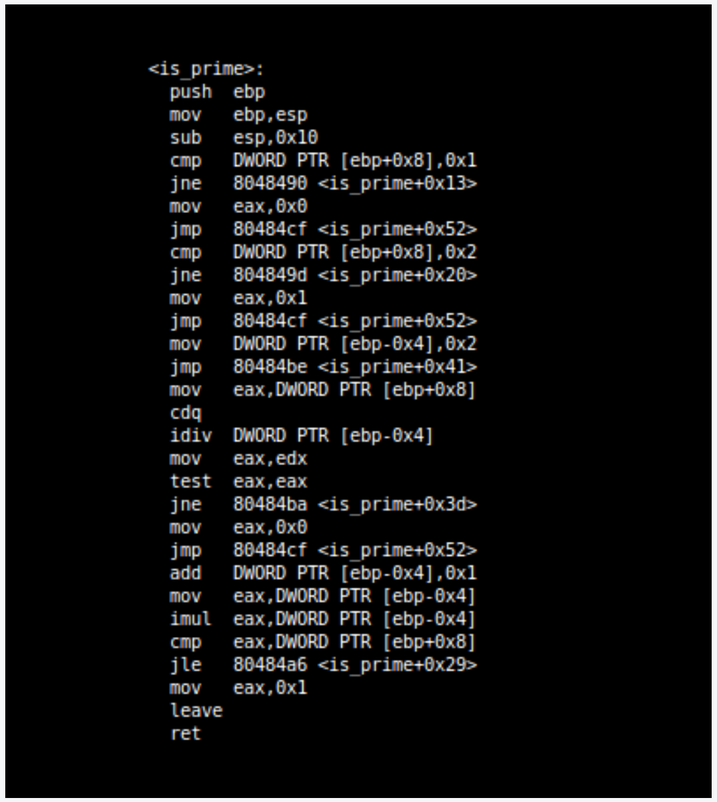
```c
int do_stuff(int r){
  int a = 4;
  int b = 2;
  r /= a ^ b;
  return r;
}
```

```
mov     DWORD PTR [rbp-0x8],0x4
mov     DWORD PTR [rbp-0xc],0x2
mov     eax,DWORD PTR [rbp-0x8]
xor     eax,DWORD PTR [rbp-0xc]
mov     ecx,DWORD PTR [rbp-0x4]
mov     DWORD PTR [rbp-0x10],eax
mov     eax,ecx
cdq
mov     ecx,DWORD PTR [rbp-0x10]
idiv    ecx
```

```
mov     DWORD PTR [rbp-0x8],0x4
mov     DWORD PTR [rbp-0xc],0x2
mov     eax,DWORD PTR [rbp-0x8]
mov     ecx,DWORD PTR [rbp-0xc]
mov     edx,eax
xor     edx,0xffffffff
mov     esi,eax
xor     esi,0xffffffff
or      esi,ecx
or      eax,ecx
sub     eax,esi
add     eax,edx
mov     ecx,DWORD PTR [rbp-0x4]
mov     DWORD PTR [rbp-0x10],eax
mov     eax,ecx
cdq
mov     ecx,DWORD PTR [rbp-0x10]
idiv    ecx
```

```
x ^ y --> (x | y) - (~x | y) + (~x)
```

MOVfuscator



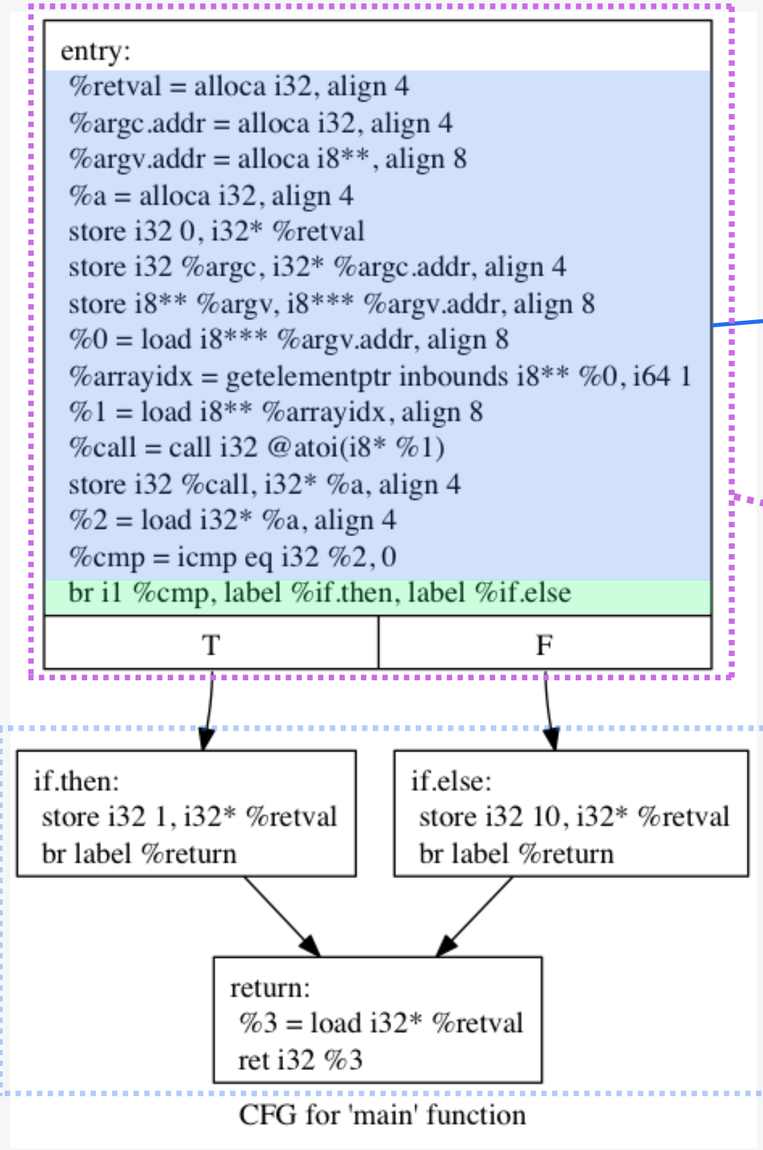
```
<is_prime>:
  push   ebp
  mov    ebp,esp
  sub    esp,0x10
  cmp    DWORD PTR [ebp+0x8],0x1
  jne    8048490 <is_prime+0x13>
  mov    eax,0x0
  jmp    80484cf <is_prime+0x52>
  cmp    DWORD PTR [ebp+0x8],0x2
  jne    804849d <is_prime+0x20>
  mov    eax,0x1
  jmp    80484cf <is_prime+0x52>
  mov    DWORD PTR [ebp-0x4],0x2
  jmp    80484be <is_prime+0x41>
  mov    eax,DWORD PTR [ebp+0x8]
  cdq
  idiv   DWORD PTR [ebp-0x4]
  mov    eax,edx
  test   eax,eax
  jne    80484ba <is_prime+0x3d>
  mov    eax,0x0
  jmp    80484cf <is_prime+0x52>
  add    DWORD PTR [ebp-0x4],0x1
  mov    eax,DWORD PTR [ebp-0x4]
  imul   eax,DWORD PTR [ebp-0x4]
  cmp    eax,DWORD PTR [ebp+0x8]
  jle    80484a6 <is_prime+0x29>
  mov    eax,0x1
  leave
  ret
```

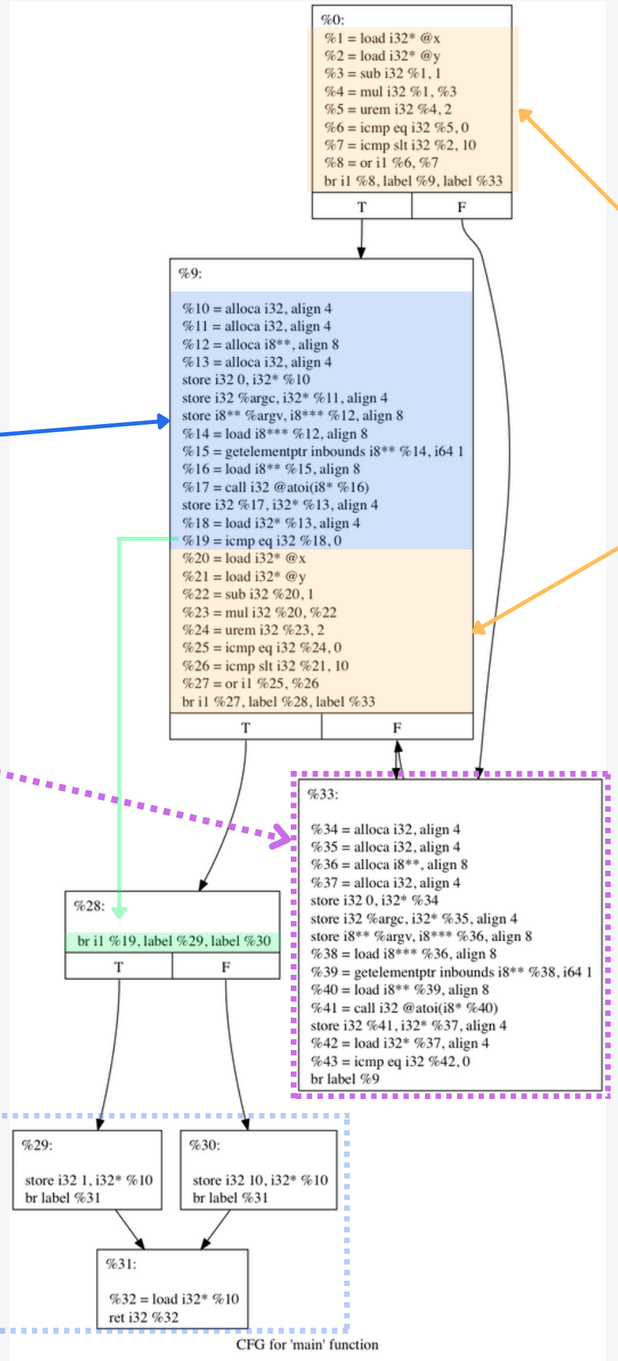
https://github.com/xoreaxeaxeax/movfuscator

Vkladanie kódu

- Nadbytočný kód (garbage code)
- Nedosiahnuteľný kód (dead code)
- "Jednoduchý" spôsob ako oklamať AV
- Optimalizácia?

Bogus Control Flow





predikáty

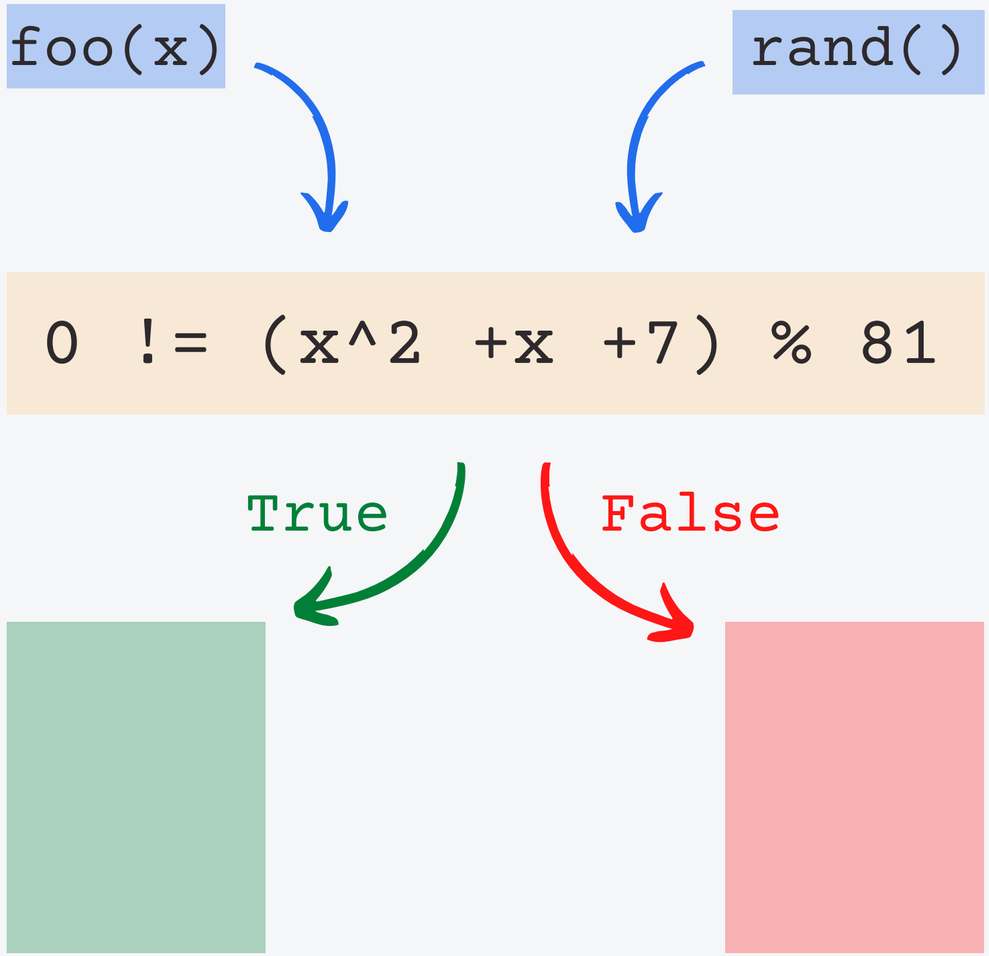
klon

https://github.com/obfuscator-llvm/obfuscator

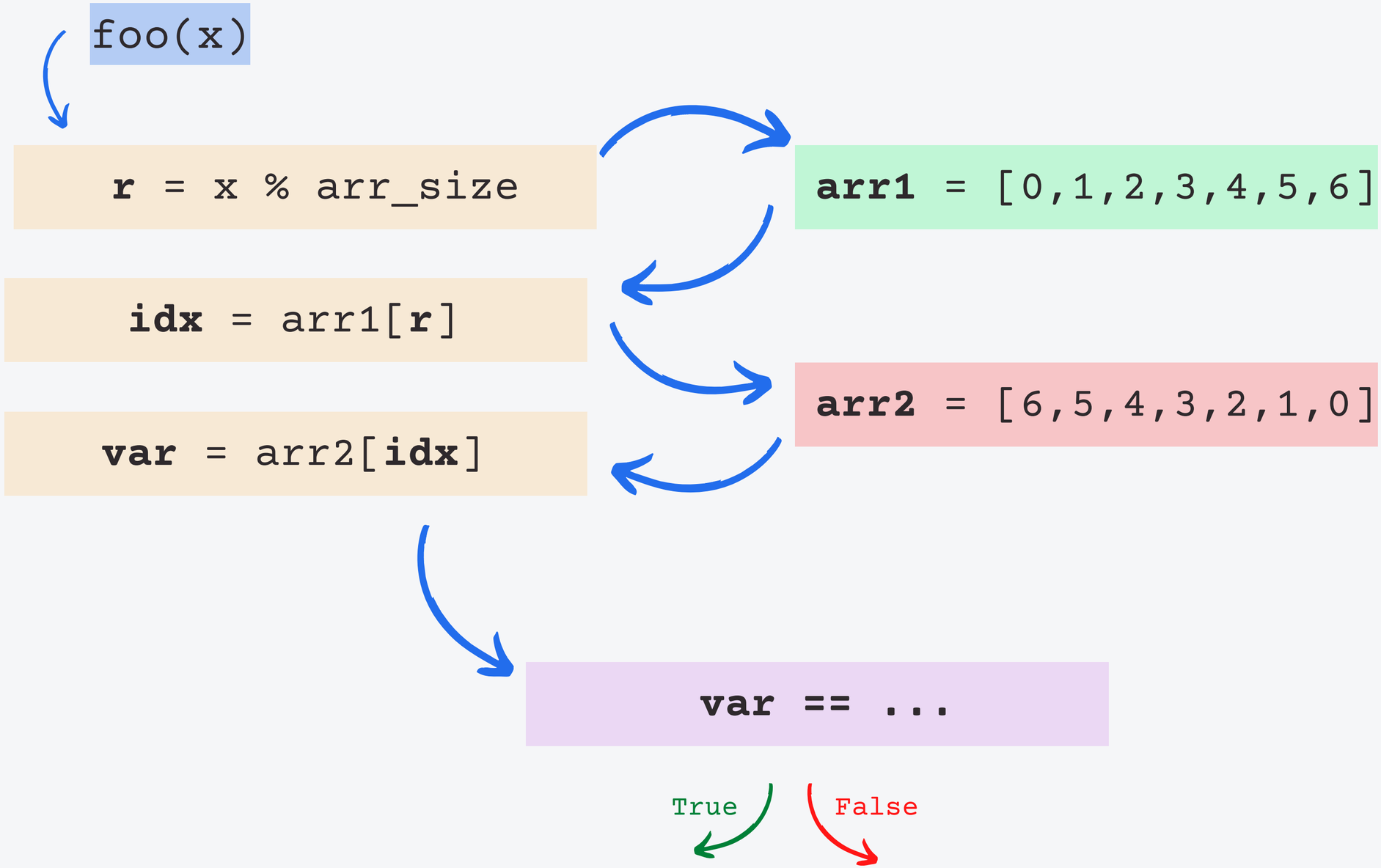
Obfuskačné predikáty
(Opaque predicates)

- Základ mnohých transformácií
  - BCF, CFG Flatenning
- Mnoho spôsobov konštrukcie
- Zvyčajne invariantné

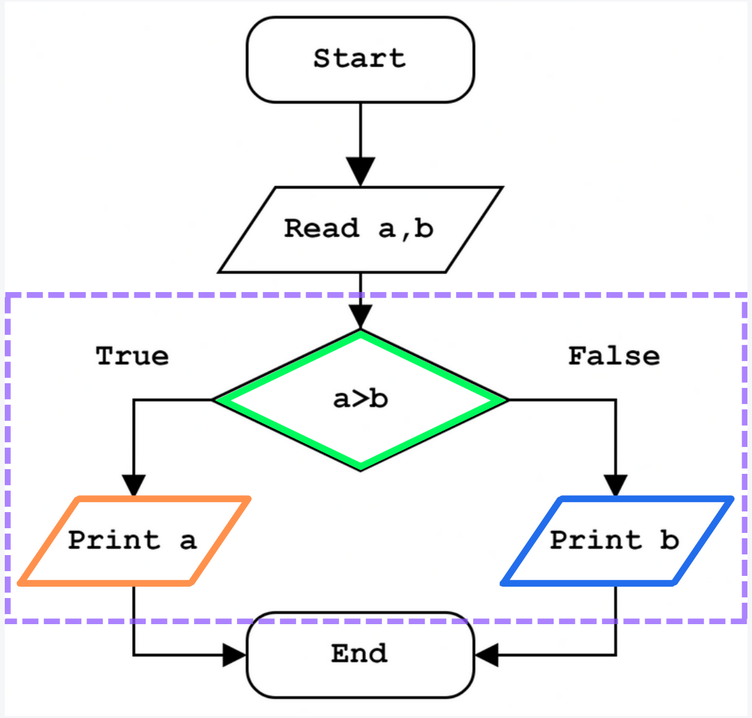
Obfuskačné predikáty #1

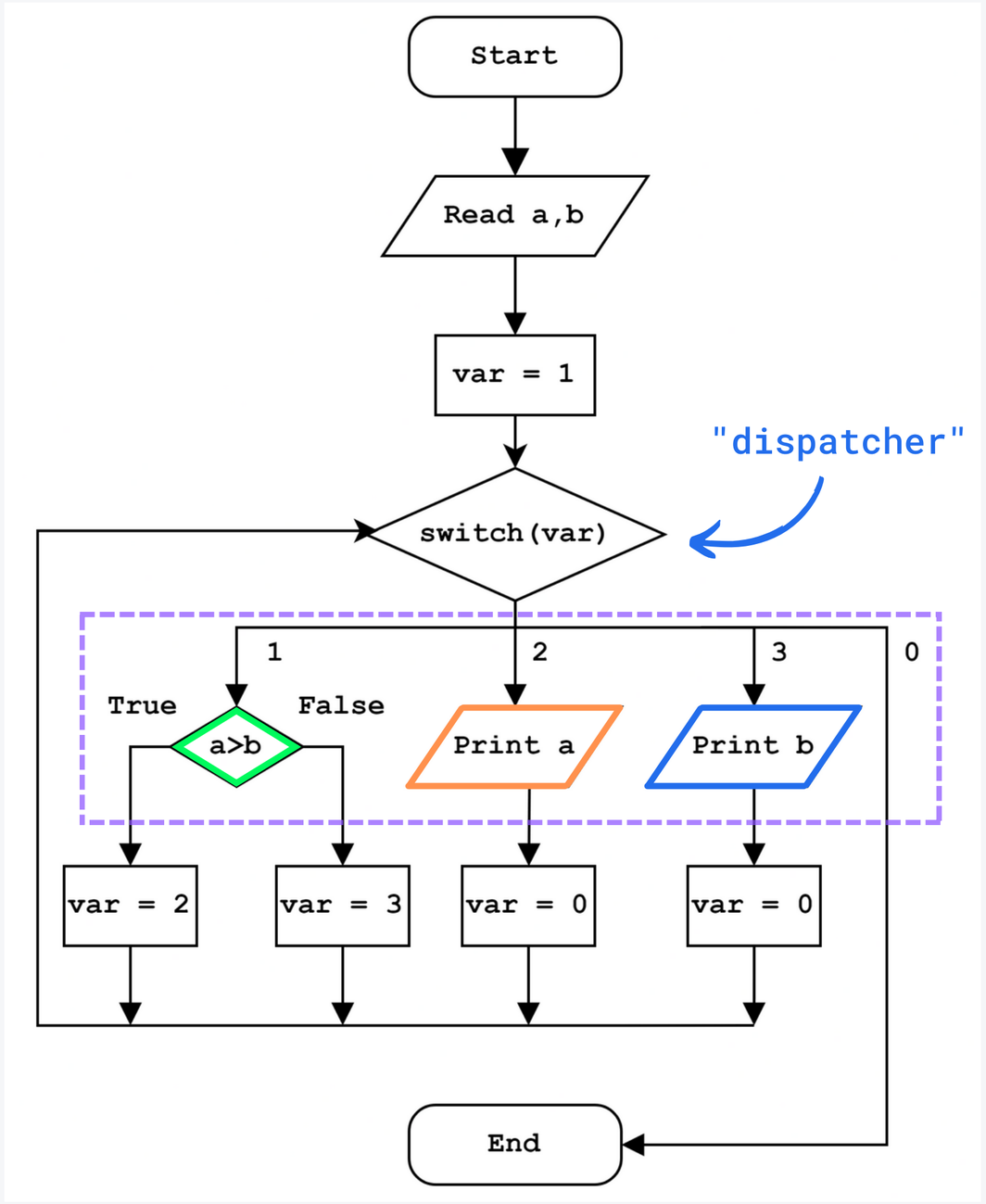


Obfuskačné predikáty #2

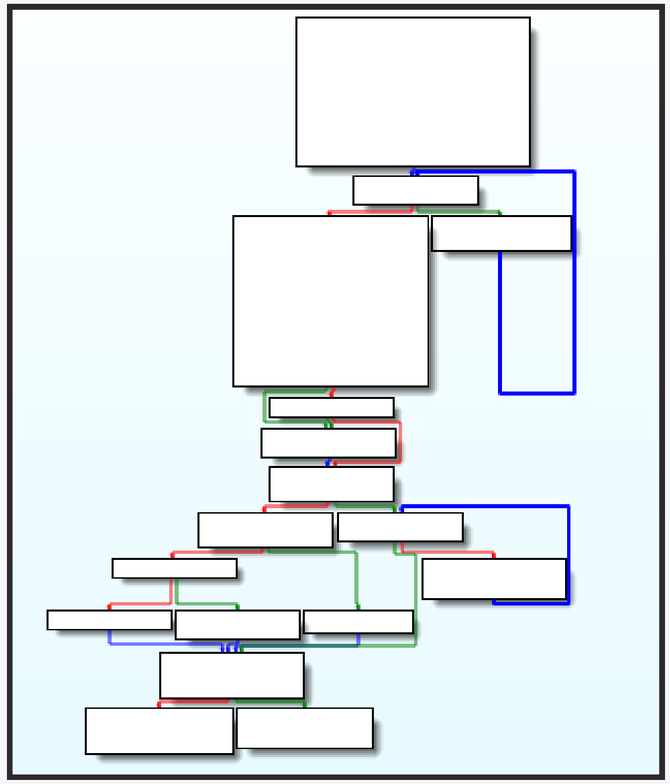


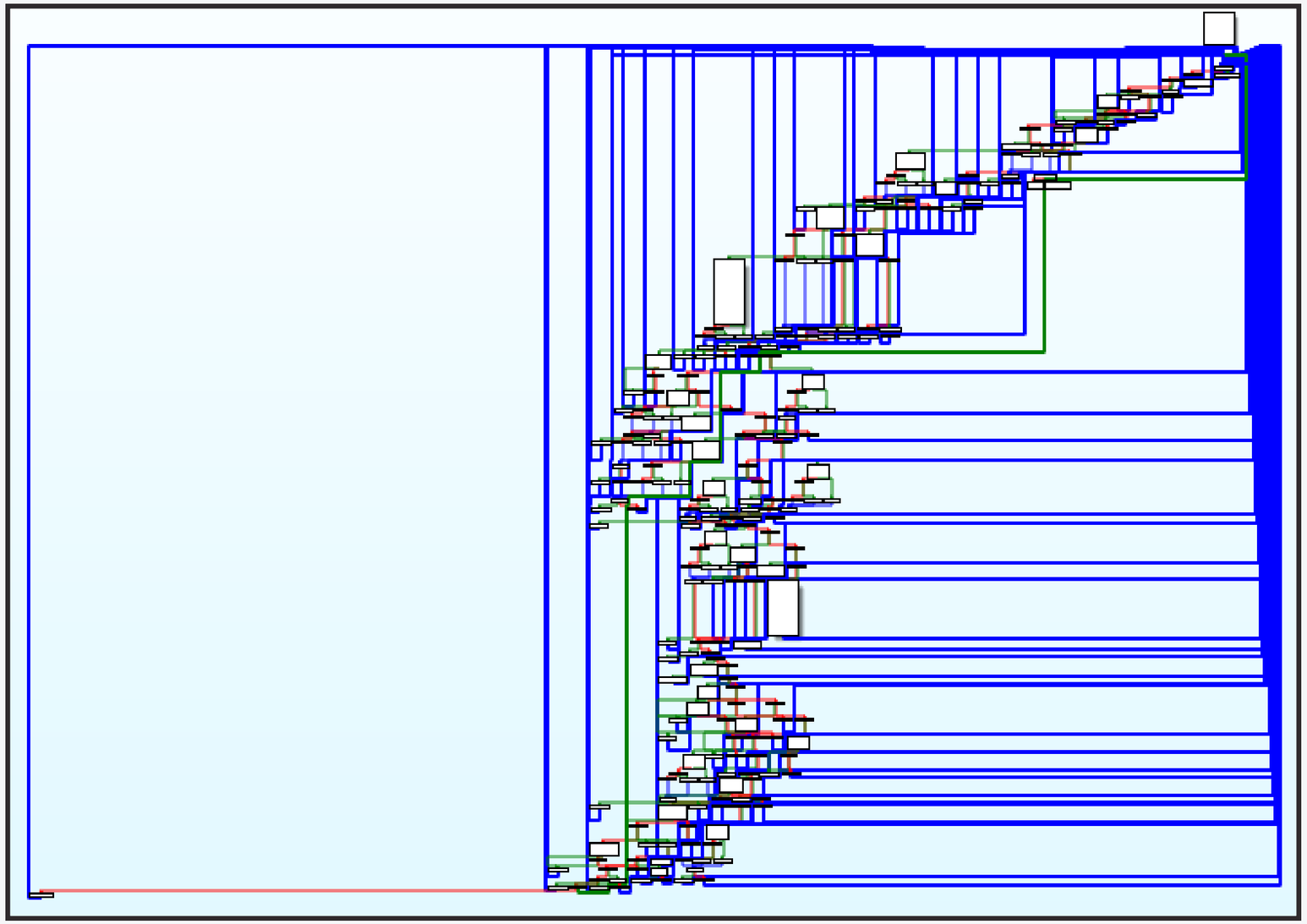
Control Flow Flattening





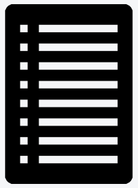
BCF + Flattening





https://eshard.com/posts/D810-a-journey-into-control-flow-unflattening

Obfuskácia konštánt



$$f(x) = ax + b$$
$$g = f^{(-1)}$$
$$\Big\} \; g(f(x)) = \mathbf{x}$$

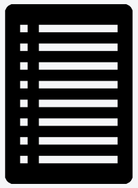
$$E = x + y - (x \;||\; y) - (!x \;||\; y) + (!x) = \mathbf{0}$$

$$C = g(x + y - (x \;||\; y) - (!x \;||\; y) + (!x) + f(C))$$

$$C = g(0 + f(C))$$

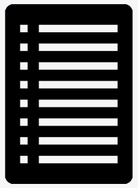
$$C = g(f(C)) = \mathbf{C}$$

Obfuskácia reťazcov



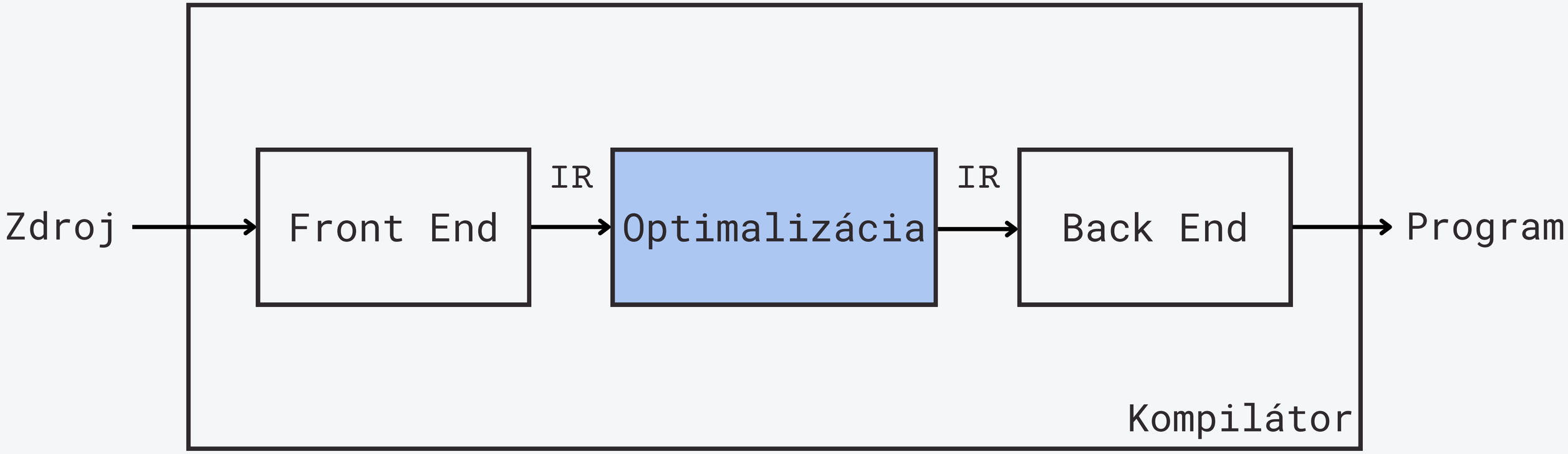
- Textové reťazce sa zakódujú počas kompilácie
- Do programu sa vloží dekódovacia funkcia
- Najčastejšie XOR
- Vhodné kombinovať s ďalšími transformáciami

Diverzifikácia



- Použitie PRNG
- Seed --> determinizmus
  - Vhodné pre testovanie
- Substitúcie, náhodne vybrané bloky, predikáty
- Náhodný kľúč pre kódovanie reťazcov

LLVM a obfuskácia



LLVM IR

```
define i32 @foo() {
entry:
  %add = add i32 7, 42
  %cmp = icmp sgt i32 %add, 56
  br i1 %cmp, label %if.then, label %if.end

if.then:
  br label %return

if.end:
  br label %return

return:
  %retval.0 = phi i32 [7, %if.then], [%add, %if.end]
  ret i32 %retval.0
}
```

LLVM a obfuskácia

- In-tree vs. out-of-tree LLVM Pass
- Generovanie IR a aplikácia transformácií:

```
$ clang -S -emit-llvm source.c  --> *.ll
$ opt -S -load libMyPass.so -my-pass-flag *.ll -o out.ll
$ clang out.ll -o my_binary
```

- Generovanie CFG pomocou "**opt -dot-cfg**"
- Ďalšie užitočné prepínače:

  -disable-O0-optnone --> použitie **opt** aj pri **-O0**
  -fno-discard-value-names --> zachová názvy premenných

Open-source LLVM obfuskátory

- **O-LLVM**
  - BCF, Flattening, Substitúcia
- **Armariris**
  - Reťazce
- **Hikari**
  - AntiClassDump, Objective-C, volania funkcií
- **Kryptonite**
  - PoC, anti-debug

O-LLVM - https://github.com/obfuscator-llvm/obfuscator
Hikari - https://github.com/HikariObfuscator/Hikari
Armariris - https://github.com/GoSSIP-SJTU/Armariris
Kryptonite - http://0vercl0k.tuxfamily.org/bl0g/?p=260

Vlastná implementácia

- Odolné predikáty v BCF
- MBA substitúcie
- Kódovanie reťazcov ľubovoľnou funkciou
- Obfuskácia konštánt

https://github.com/roman-oravec/obfuscation-passes

Metriky

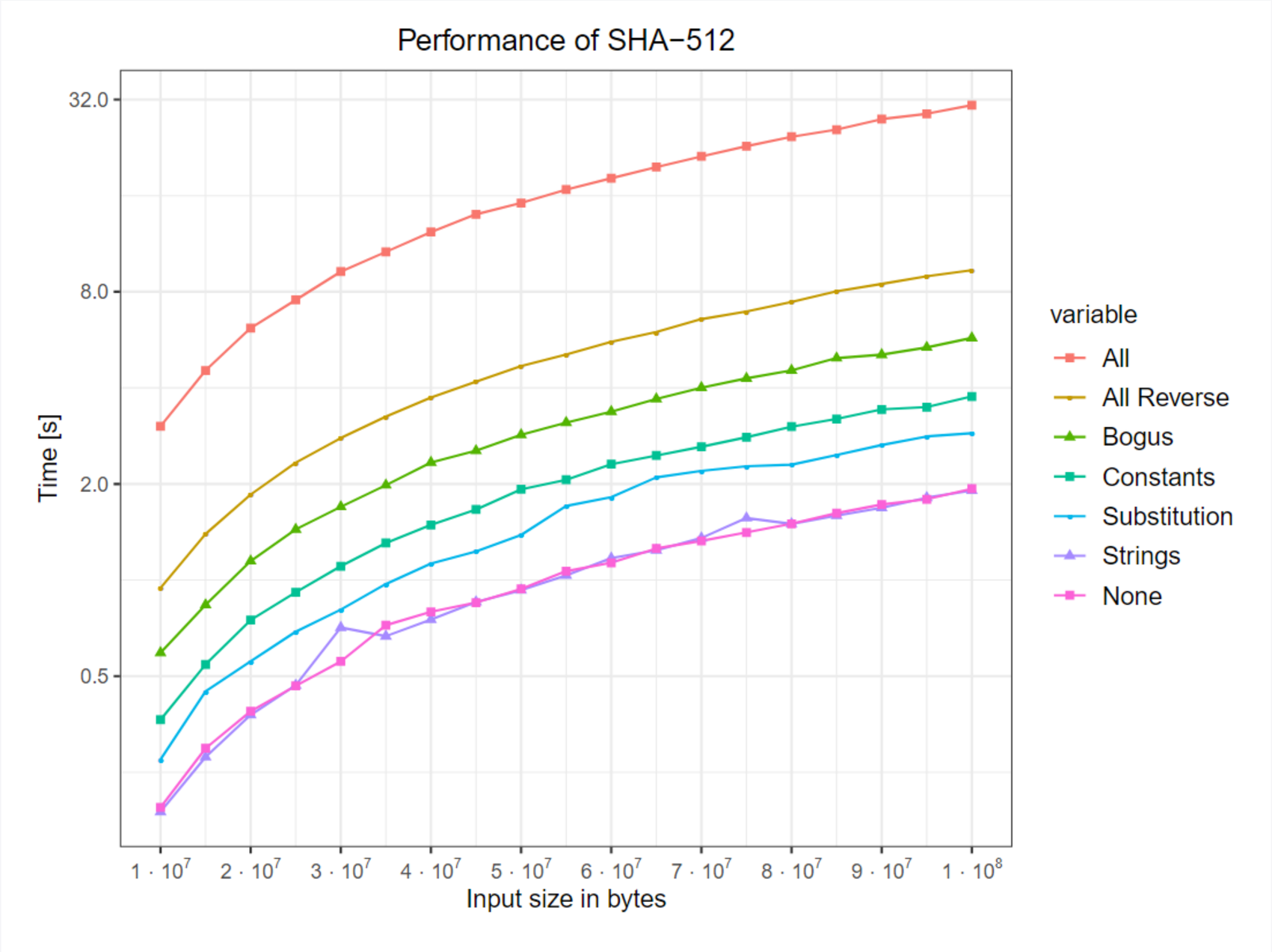
- **Odolnosť** (Resilience)
  - číselné metriky
- **Potencia**
  - zhotovenie automatického deobfuskátora
- **Výkon** (Cost)
  - benchmark

Potencia

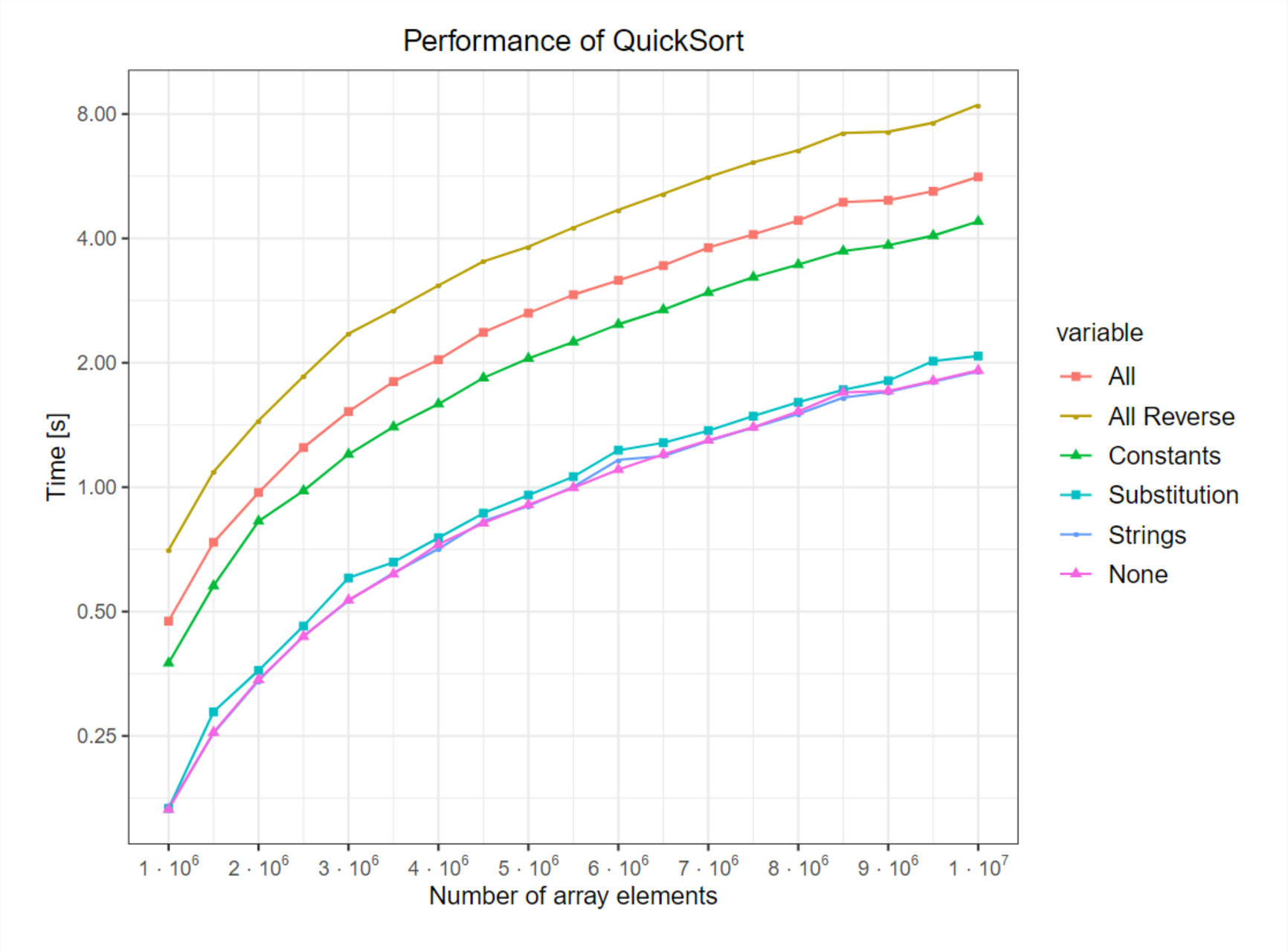
μ_1 - Počet inštrukcií
μ_2 - Kolmogorova zložitosť

Program	SHA-512		AES		QuickSort	
Metric	μ_1	μ_2	μ_1	μ_2	μ_1	μ_2
Substitution	1.37	1.07	1.09	1.04	1.23	1.03
Substitution $\times 2$	2.25	1.26	1.34	1.13	1.82	1.11
Opaque constants	2.66	1.86	3.29	2.40	3.76	1.96
String obfuscation 1	1.05	1.04	1.01	1.02	1.19	1.12
String obfuscation 2	1.09	1.05	1.02	1.03	1.35	1.16
Bogus Control Flow	3.52	2.23	4.28	3.49	7.18	3.40
Str 2 + Bogus	3.72	2.35	4.40	3.64	10.64	4.44
Str 2 + Bogus + Const	14.26	8.08	19.16	16.23	32.43	12.27
All	26.98	11.03	36.05	23.12	57.88	16.23
Reverse order	8.10	4.24	9.50	7.05	21.57	7.84

Vplyv na výkon



Vplyv na výkon



Výhody a nevýhody využitia LLVM

- Podporované jazyky
  - C, C++, Go, Haskell, Rust, Swift...
- Podporované architektúry
  - **x86, ARM**, PowerPC, MIPS...
- Rozsiahle API
- Kompatibilita verzií

Otázky?

romanoravec1@gmail.com