基于控制流代码混淆方法的混淆点选择策略研究

张海宁

2015.11.15

一、问题
二、策略
三、当前进展
四、下一步工作

一.问题

基于控制流混淆的混淆方法

1. 混淆数量过少：难以达到代码混淆的目的；
2. 全部混淆：时间开销、空间开销难以接受；
3. 混淆一部分：选择哪些可混淆点？

最好的方法：

代码作者指定混淆哪部分，因为代码作者最了解程序中哪一部分代码需要额外的保护。

一.问题

如果不需要代码作者手动制定，可不可以用一些自动化的手段对代码混淆点的选择给出建议？

二 . 策略

想法：从攻击者角度看待这个问题，混淆逆向攻击者感兴趣的可混淆点。

逆向攻击者：

1. 自动分析：污点标记、符号执行是逆向攻击者常用的方法。
2. 手动软件破解，关注与用户输入数据相关的代码。（比如与注册码相关的代码）

二 . 策略

混淆与输入数据相关的可混淆点

逆向攻击者：程序输入与程序行为之间的对应关系，

二、策略

选择 C 代码理由：

1. 足够复杂，应用广泛，有代表性。
2. 仅仅关注与程序功能密切相关的部分。

中间语言缺点：

1. 没有标准的中间语言，且经常变化。

汇编语言缺点：

1. 许多逻辑结构被隐藏；引入许多无关代码。
2. 即使找到合适的可混淆点，也需要在与之对应的中间语言或高级语言上做混淆，混淆汇编代码，基本不可能。

验证本策略的正确性：

只混淆与输入数据相关的可混淆点，对逆向攻击者造成的困难，与全部混淆可混淆点所带来的困难，在一个数量级上。

如何衡量逆向攻击者逆向分析难度？

利用 bitblaze 等常用二进制分析平台，比较符号执行、约束求解的难度。

三 . 当前进展

输入 C 代码

词法分析

语法分析

污点标记

记录污点传播过程

找出与输入数据相关的可混淆点

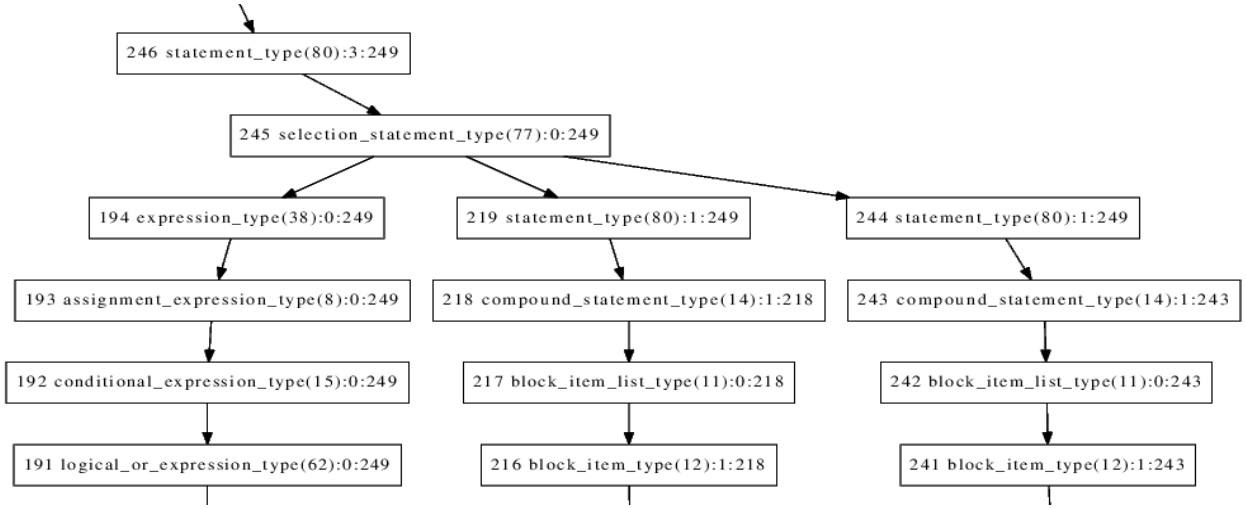
三 . 当前进展

词法分析，语法分析部分：利用 lex 、 yacc 等工具简化代码编写难度。所需 lex 、 yacc 文件来自

http://quut.com/c/ANSI-C-grammar-y.html

http://quut.com/c/ANSI-C-grammar-l-2011.html

```
1
2  void main()
3  {
4    int a=12;
5    int b=13;
6    int c;
7
8    c+=a-b+c;
9    if(c>15)
10   {
11     ++a;
12   }
13   else
14   {
15     if(b<12)
16     {
17       b++;
18     }
19     else
20     {
21       b--;
22     }
23   }
24 }
25
```



三 . 当前进展

污点标记：用户输入制定的数据进行污点标记。

标记第四行变量 a

./myFunc -t 2.taint -g 2.dot 2.c

2.taint:

4 a 12　// 标记第四行的 a ，污点级别为 12

三、当前进展

记录污点传播过程：

```
2794 symbol_a_table_num:7
2795 ----------------------------------------
2796 index:0 declaration_specifiers_index:1 IDENTIFIER_index:1 symbol_name:***a(line_num:4)*** action_scope:254
2797 pointer_index:-1 array_dimension:0
2798 taint_m:12 taint_src:0
2799 ----------------------------------------
2800 index:1 declaration_specifiers_index:2 IDENTIFIER_index:2 symbol_name:***b(line_num:5)*** action_scope:254
2801 pointer_index:-1 array_dimension:0
2802 taint_m:0 taint_src:1
2803 ----------------------------------------
2804 index:2 declaration_specifiers_index:3 IDENTIFIER_index:3 symbol_name:***c(line_num:6)*** action_scope:254
2805 pointer_index:-1 array_dimension:0
2806 taint_m:12 taint_src:0
2807 ----------------------------------------
2808 index:3 declaration_specifiers_index:-1 IDENTIFIER_index:-1 symbol_name:***(null)(line_num:8)*** action_sco
2809 pointer_index:-1 array_dimension:0
2810 taint_m:12 taint_src:0
2811 ----------------------------------------
2812 index:4 declaration_specifiers_index:-1 IDENTIFIER_index:-1 symbol_name:***(null)(line_num:8)*** action_sco
2813 pointer_index:-1 array_dimension:0
2814 taint_m:12 taint_src:0
2815 ----------------------------------------
2816 index:5 declaration_specifiers_index:-1 IDENTIFIER_index:-1 symbol_name:***(null)(line_num:9)*** action_sco
2817 pointer_index:-1 array_dimension:0
2818 taint_m:12 taint_src:0
2819 ----------------------------------------
2820 index:6 declaration_specifiers_index:-1 IDENTIFIER_index:-1 symbol_name:***(null)(line_num:15)*** action_sc

2821 pointer_index:-1 array_dimension:0
```

三、当前进展

记录污点传播过程：

```
2871 index:9 sequence:9 node_index:251 prev:0 next:0 ins_type:58 ins_set_num:0
2872 line_num:23 inst_type:58 ins_ret:-1 ins_data1:10 ins_data2:13 ins_data3:17 ins_taint_level:12 ins_taint_src:0
2873 instruction description: selection_statement_0_ins(58)
```

```
2911 index:18 sequence:18 node_index:245 prev:0 next:0 ins_type:58 ins_set_num:0
2912 line_num:22 inst_type:58 ins_ret:-1 ins_data1:19 ins_data2:22 ins_data3:24 ins_taint_level:0 ins_taint_src:6
2913 instruction description: selection_statement_0_ins(58)
```

```
 1
 2 void main()
 3 {
 4   int a=12;
 5   int b=13;
 6   int c;
 7
 8   c+=a-b+c;
 9   if(c>15)
10   {
11     ++a;
12   }
13   else
14   {
15     if(b<12)
16     {
17       b++;
18     }
19     else
20     {
21       b--;
22     }
23   }
24 }
25
```

四.下一步工作

完善污点传播过程

支持函数功能

支持指针、数组、结构等复杂语法