Code Obfuscation

Adrian Veto, Junling Zhang

Overview

Code obfuscation is a system in which normal, human-readable code is made more difficult to understand. The code should still function in exactly the same way pre- and post-obfuscation, but obfuscation should make code significantly harder to read.

There are multiple reasons to obfuscate code:

- Protect intellectual property
- Disguise malicious code
- For fun!

Types of Obfuscation

There are a lot of different types of code obfuscation, all of which have different effects on code that make it hard to read and understand. When all the types of obfuscation are put together, they can create a nearly impossible, uncrackable program. For ease of understanding, let's look at each one individually before putting them together.

Rename Obfuscation

Rename obfuscation is probably the simplest and most understandable of any obfuscation method. The idea is simple: you rename all of the variables and functions that you create in a code method to be meaningless strings or even letters.

Before Rename Obfuscation:

```
1  private void
2  CalcPayroll (SpecialList employee-
3  Group) {
4     while(employeeGroup.HasMore()) {
5        employee =
6  employeeGroup.GetNext(true);
7        employee.UpdateSalary();
8        DistributeCheck(employee);
9     }
10 }
```

After Rename Obfuscation:

```
1  private void a(a b) {
2     while (b.a()) {
3        a = b.a(true);
4        a.a();
5        a(a);
6     }
7  }
8
9
10
```

Source: https://www.preemptive.com/what-is-obfuscation/

Debug Obfuscation

Debug obfuscation is another pretty simple obfuscation method. It just removes all comments and debugging functions from the code. This is pretty effective (imagine reading over your code without any comments), but it's almost necessary when coupled with any other obfuscation method. No matter how obscure your code is, if it's written over with comments and debugging information you can figure out what it means.

Control Flow Obfuscation

Control flow obfuscation is essentially messing up the logic of your code so that it's harder to read.

Here are some of the more common ways to accomplish that:

- Dead ends: creating bunches of code that do nothing and just fill up space in a program.
- Splitting up code: Taking functions and processes in code and splitting them up among other, smaller functions.
- Adding conditions: Adds more parts to conditional statements to make them harder to understand.
- Complex looping structures: Instead of simple `for` or `while` loops, control flow obfuscation usually makes much more complex looping structures.

Aggregation Obfuscation

Aggregation obfuscation is an obfuscation method that relies on breaking up arrays and data structures to store their data in separate places. This makes code more confusing and much harder to read.

String Obfuscation

String obfuscation is pretty simple: all strings in the code are encrypted before being decrypted at runtime. This is almost necessary for any code that involves strings in any form.

There are a couple different encryption methods:

- The obfuscator has its own algorithm
- The obfuscator has its own algorithm but it involves a user key
- The user inputs their own encryption algorithm

Data Obfuscation

Data obfuscation is set of ways of disguising personal information so that it can't be read even if it gets stolen.

- **Tokenization:** important information is replaced with sets of characters (tokens)
- **Encryption:** uses some sort of cipher to encode information
- **Generalization:** personal information is separated from its owner
- **Randomization:** more information is added to a dataset to make it confusing

Flaws of Obfuscation

The main downside of code obfuscation is that it limits code performance. Because of this, it's sometimes difficult to choose the right level of code obfuscation. More obfuscation is more secure, but it also comes with worse performance.

The only real solution to this is choosing types of obfuscation that are suitable for the task. It's probably overkill to run string obfuscation when the code only has a couple strings, so that would needlessly slow down code. When a developer obfuscates their code, they have to make decisions about how intensively they want to obfuscate.

Obfuscation Programs

There are a number of obfuscation programs on the internet that can be used for different languages. Obfuscation software can be found for basically anything on the internet, though most of it is paid software. There are also some open-source programs and command line tools for obfuscation. We also made a couple of programs for obfuscating code.

For fun!

Finally, there are code obfuscation contests, where people come up with programs that are graded on what they do and how successful their obfuscation is. Some of these contests even have limits on the number of characters that can be used, which means that people have to write small *and* confusing programs. Some of this stuff gets really complicated, but it's also quite cool.

The best example of a contest like this is the *International Obfuscated C Code Contest*, a biannual contest where programmers from all over the world submit obfuscated C programs. The judges of the contests look at both the finished product and the structure of the code to determine winners and placements. All code is then posted online, where people try to figure out what the program does solely by looking at the source code.

```c
#include <stdio.h>

#define N(a)        "%"#a"$hhn"
#define O(a,b)      "%10$"#a"d"N(b)
#define U           "%10$.*37$d"
#define G(a)        "%"#a"$s"
#define H(a,b)      G(a)G(b)
#define T(a)        a a
#define s(a)        T(a)T(a)
#define A(a)        s(a)T(a)a
#define n(a)        A(a)a
#define D(a)        n(a)A(a)
#define C(a)        D(a)a
#define R           C(C(N(12)G(12)))
#define o(a,b,c)    C(H(a,a))D(G(a))C(H(b,b)G(b))n(G(b))O(32,c)R
#define SS          O(78,55)R "\n\033[2J\n%26$s";
#define E(a,b,c,d) H(a,b)G(c)O(253,11)R G(11)O(255,11)R H(11,d)N(d)O(253,35)R
#define S(a,b)      O(254,11)H(a,b)N(68)R G(68)O(255,68)N(12)H(12,68)G(67)N(67)

char* fmt = O(10,39)N(40)N(41)N(42)N(43)N(66)N(69)N(24)O(22,65)O(5,70)O(8,44)N(
            45)N(46)N    (47)N(48)N(    49)N( 50)N(     51)N(52)N(53    )O( 28,
            54)O(5,        55) O(2,    56)O(3,57)O(      4,58 )O(13,    73)O(4,
            71 )N(   72)O   (20,59    )N(60)N(61)N(       62)N (63)N    (64)R R
            E(1,2,   3,13   )E(4,    5,6,13)E(7,8,9        ,13)E(1,4    ,7,13)E
            (2,5,8,        13)E(    3,6,9,13)E(1,5,         9,13)E(3    ,5,7,13
            )E(14,15,    16,23)    E(17,18,19,23)E(          20, 21,    22,23)E
            (14,17,20,23)E(15,    18,21,23)E(16,19,    22     ,23)E(    14, 18,
            22,23)E(16,18,20,    23)R U O(255 ,38)R    G (     38)O(    255,36)
            R H(13,23)O(255,    11)R H(11,36) O(254    ,36)     R G(    36 ) O(
            255,36)R S(1,14    )S(2,15)S(3, 16)S(4,    17 )S     (5,    18)S(6,
            19)S(7,20)S(8,    21)S(9    ,22)H(13,23    )H(36,     67    )N(11)R
            G(11)""O(255,    25 )R        s(C(G(11)    ))n (G(          11) )G(
            11)N(54)R C(    "aa")   s(A(   G(25)))T    (G(25))N         (69)R o
            (14,1,26)o(    15, 2,   27)o   (16,3,28    )o( 17,4,        29)o(18
            ,5,30)o(19    ,6,31)o(        20,7,32)o    (21,8,33)o       (22 ,9,
            34)n(C(U)    )N( 68)R H(    36,13)G(23)    N(11)R C(D(      G(11)))
            D(G(11))G(68)N(68)R G(68)O(49,35)R H(13,23)G(67)N(11)R C(H(11,11)G(
            11))A(G(11))C(H(36,36)G(36))s(G(36))O(32,58)R C(D(G(36)))A(G(36))SS

#define arg d+6,d+8,d+10,d+12,d+14,d+16,d+18,d+20,d+22,0,d+46,d+52,d+48,d+24,d\
            +26,d+28,d+30,d+32,d+34,d+36,d+38,d+40,d+50,(scanf(d+126,d+4),d+(6\
            -2)+18*(1-d[2]%2)+d[4]*2),d,d+66,d+68,d+70, d+78,d+80,d+82,d+90,d+\
            92,d+94,d+97,d+54,d[2],d+2,d+71,d+77,d+83,d+89,d+95,d+72,d+73,d+74\
            ,d+75,d+76,d+84,d+85,d+86,d+87,d+88,d+100,d+101,d+96,d+102,d+99,d+\
            67,d+69,d+79,d+81,d+91,d+93,d+98,d+103,d+58,d+60,d+98,d+126,d+127,\
            d+128,d+129

char d[538] = {1,0,10,0,10};

int main() {
    while(*d) printf(fmt, arg);
}
```