Selected techniques for source code obfuscation in scripting languages

M. MOHR

mateusz.mohr@student.wat.edu.pl

Military University of Technology, Faculty of Cybernetics
Kaliskiego Str. 2, 00-908 Warsaw, Poland

The article presents the most common techniques for obfuscating the source code of computer programs. Obfuscation is defined and demonstrated through simple and easy-to-understand examples of code written in scripting languages such as Python and JavaScript. Its applications are discussed, and it is shown how to easily make the analysis of one's programs more difficult.

Keywords: code obfuscation, scripting languages.

DOI: 10.5604/01.3001.0054.6335

1. Introduction

In the modern world, where an innumerable amount of information is processed, it is essential to pay attention to its security. In the early stages of computer science development, programmers could primarily focus on the correct functioning of software. However, with the advent of the Internet, many new threats and challenges have emerged. One of these is the protection of intellectual property, which is particularly challenging in a world where news about cyberattacks is no longer extraordinary. One area that is gaining importance in the context of software security is source code obfuscation [1]. It is important to note, however, that obfuscation can also be used unethically, by hiding malicious code and making it difficult for antivirus programs to detect it [2]. Therefore, it is worth exploring this topic not only to secure one's own code but also to be aware of how these techniques can be used against ordinary computer users with Internet access.

2. Obfucsation

Obfuscation was first defined in 1997 by Christian Collberg in "A Taxonomy of Obfuscating Transformations." The author's definition is as follows [3]:
"Let P → P' be a transformation of a source program P. P → P is an obfuscating transformation, if P and P' have the same observable behavior. More precisely, in order for P → P' to be a legal obfuscating transformation the following conditions must hold:

- If P fails to terminate or terminates with an error conditio, then P' may or may not terminate.
- Otherwisem P' must terminate and produce the same output as P."

Obfuscation, in simple terms, can be described as the process of deliberately making the source code harder to understand without affecting the outcome of its execution. This is achieved by updating the working code using various obfuscation techniques. However, the code remains fully functional [4]. Obfuscation techniques can modify individual method instructions, but this does not affect the program's output. Obfuscation consists of many entirely different techniques that complement each other. In this paper, the classification from the publication "Techniques of Program Code Obfuscation for Secure Software" by Marius Popa was used, which has been extended by one additional technique. In the mentioned article, obfuscation is divided into 8 techniques [5]:

1. Name obfuscation;
2. Data obfuscation;
3. Code flow obfuscation;
4. Incremental obfuscation;
5. Intermediate code optimization;
6. Debug information obfuscation;
7. Watermarking;
8. Source code obfuscation.

The additional 9 technique is – String encryption.

Four of the above techniques, which, due to the publication's focus on simple programs written in scripting languages, will not be presented in detail later in the publication, are briefly defined below:

Incremental obfuscation [6] – this technique involves ensuring the consistency of the code. It guarantees that previously obfuscated names (classes, methods, etc.) will remain consistent with earlier obfuscated versions of the code. In other words, this means that the code that was previously obfuscated does not change between versions, while new elements are subjected to the obfuscation process.

Intermediate code optimization [5] – this technique focuses on improving the efficiency and size of the intermediate code. The technique includes: removal of unused components (such as methods, fields, and strings), constant expression evaluation, assignment of static and final attributes, and inlining of simple methods.

Debug information obfuscation [7] – debug information is valuable because it allows you to understand key aspects of a program's operation and identify errors by decompiling and recompiling the source code. The debug information obfuscation technique involves masking identifiable information by blocking access to debug information altogether or by changing its identifiers and line numbers.

Watermarking [8] – this technique involves adding special sets of data to the code using steganography techniques. This data is used for identification purposes and can include information about both the author (owner) of the application and the client, in order to identify whose copy of the software was made public.

3. Selected obfuscation techniques

The remaining techniques selected and organized for a more detailed analysis are:

- Name obfuscation,
- Data obfuscation,
- Code flow Obfuscations,
- String encryption,
- Source code obfuscation.

To illustrate the obfuscation process using the aforementioned techniques more accurately, two programs written in scripting languages were developed. Each of these techniques should be examined individually; however, it is important to remember that obfuscation typically involves using many of them simultaneously. The first program, prepared to demonstrate the operation of these techniques, is a simple calculator written in the Python programming language:

Code 1 – Calculator

```
def add(x, y):
  return x + y

def subtract(x, y):
  return x - y

def multiply(x, y):
  return x * y

def divide(x, y):
  if y == 0:
    return "You can't divide by 0!"
  return x / y

def calculator():
  while True:
    print("Operations:\n1.         Addition\n2.
Subtraction\n3. Multiplication\n4. Division\n5.
Exit ")
    choice     =     input("Select     operation
(1/2/3/4/5): ")

    if choice == '5':
      break

    if choice in ('1', '2', '3', '4'):
      num1   =   float(input("Enter   the   first
number: "))
      num2   =   float(input("Enter   the   second
number: "))

      if choice == '1':
        print("Result: ", add(num1, num2))

      elif choice == '2':
        print("Result: ", subtract(num1, num2))

      elif choice == '3':
        print("Result: ", multiply(num1, num2))

      elif choice == '4':
        result = divide(num1, num2)
        print("Result: " if isinstance(result,
float) else "", result)
    else:
      print("Incorrect choice")

calculator()
```

The second example presented is code responsible for changing an image on a website, written in JavaScript. This program works with a webpage created using HTML technology.

Code 2 – Website

```
<!DOCTYPE html>
<html>
<head>
    <title>Changing the picture</title>
</head>
<body>
    <img src="image1.png" class="changeable-
image" alt="Image">
    <script src="script.js"></script>
</body>
</html>
```

Code 3 – Script for changing an image on a website

```
const images =
document.querySelectorAll('.changeable-image');
const imageSources = ['image1.png',
'image2.png', 'image3.png'];

images.forEach(image => {
    image.addEventListener('click', function() {
        let currentImageIndex =
imageSources.indexOf(this.src.split('/').pop());
        let nextImageIndex = (currentImageIndex
+ 1) % imageSources.length;
        this.src = imageSources[nextImageIndex];
    });
});
```

The source code of both programs will change with the application of each selected obfuscation technique, and their operation should become increasingly difficult to analyze. Additionally, it is assumed that in the example of changing an image on a website, the code written in HTML remains unchanged.

Name obfuscation – is the simplest code obfuscation technique to understand. It involves changing the names of variables, functions, classes, and other identifiers, which are usually meaningful for better understanding the source code, to names that are meaningless (convey no information) [9]. After changing the identifiers' names, it is mandatory to ensure the consistency of the entire application by replacing the old names with the new ones. It is worth mentioning that this obfuscation technique also includes method overloading, which allows a single method name to refer to different functions depending on the context of its invocation. Method name obfuscation in this context involves using the same obfuscated name (identifier) for all overloaded method versions, even though they have different signatures [5]. This means that instead of assigning a unique name to each version of a method, all of them have the same name. However, the programming language can distinguish which version of the method to call based on the passed arguments. Unfortunately, in scripting languages, which are statically typed (such as Python, JavaScript, or Ruby), method overloading does not work well. Below are examples of name obfuscation in Python.

Code 4 – Calculating Sum

```
def calculateTheSum(a, b):
  result = a + b
  return result

sum = calculateTheSum(2, 3)
print(sum)
```

In the above program code, the *calculateTheSum* function takes two arguments and returns their sum. This code is clear and understandable even to people who do not know how to program.

Code 5 – Calculating Sum After Applying Name Obfuscation Technique

```
def s3wzY(tYx, xCb):
  P3wn = tYx + xCb
  return P3wn

rTb = s3wzY(2, 3)
print(rTb)
```

In the presented example, the variable names have been changed to meaningless strings. Although the program's functionality remains unchanged, its comprehension has become more difficult.

Code 6 – Finding Maximum Value

```
def findMax(numbers):
  maxNumber = numbers[0]
  for num in numbers:
    if num > maxNumber:
      maxNumber = num
  return maxNumber

print(findMax([1, 3, 2, 8, 5]))
```

In the presented example, the program that searches for the largest number in an array is well and clearly described using variable and function names. This makes it easy to understand without requiring code analysis. This changes after the obfuscation shown below.

Code 7 – Finding Maximum Value After Applying Name Obfuscation Technique

```
def vZ16p(b3P2):
  lNxX = b3P2[0]
  for uIQf in b3P2:
    if uIQf > lNxX:
      lNxX = uIQf
  return lNxX

print(vZ16p([1, 3, 2, 8, 5]))
```

Returning to the programs from the beginning of the chapter, after applying the discussed obfuscation technique, they might look as follows:

Code 8 – Calculator After Applying Name Obfuscation Technique

```
def xZ15(rTv, T47):
    return rTv + T47

def Ts3D(eiK, vZ4):
    return eiK - vZ4

def P1W9(P0d, vVZ):
    return P0d * vVZ

def Il1y(bJS, AI1):
    if AI1 == 0:
        return "You can't divide by 0!!"
```

```
    return bJS / AI1

def NwwC():
    while True:
        print("Operations:\n1. Addition\n2.
Subtraction\n3. Multiplication\n4. Division\n5.
Exit ")
        uM = input("Select operation
(1/2/3/4/5): ")

        if uM == '5':
            break

        if uM in ('1', '2', '3', '4'):
            Xx4 = float(input("Enter the first
number: "))
            Go2 = float(input("Enter the second
number: "))

            if uM == '1':
                print("Result: ", xZ15(Xx4,
Go2))

            elif uM == '2':
                print("Result: ", Ts3D(Xx4,
Go2))

            elif uM == '3':
                print("Result: ", P1W9(Xx4,
Go2))

            elif uM == '4':
                Er = Il1y(Xx4, Go2)
                print("Result: " if
isinstance(Er, float) else "", Er)
        else:
            print("Incorrect choice")

NwwC()
```

The above program has been changed by renaming the *calculator*, *add*, *subtract*, *multiply* and *divide* functions to the meaningless strings. In a similar way, the names of the variables *num1*, *num2*, *choice* and *result* were changed to *Xx4*, *Go2* and *uM* and *Er*, respectively. In each of the functions performing arithmetic calculations, the names of the *x* and *y* variables have been changed to random strings of characters.

The script for changing photos on the website was changed using the same technique as follows:

Code 9 – Script for Changing an Image on a Website After Applying Name Obfuscation Technique

```
const myE4 =
document.querySelectorAll('.changeable-image');
const Af13 = ['image1.png', 'image2.png',
'image3.png'];

myE4.forEach(e3r => {
    e3r.addEventListener('click', function() {
        let hji =
Af13.indexOf(this.src.split('/').pop());
        let W6r = (hji + 1) % Af13.length;
        this.src = Af13[W6r];
    });
});
```

Once again, this program becomes more difficult to understand. However, for the sake of your own intellectual rights, it is also worth turning to other obfuscation techniques.

Data obfuscation [10] – obfuscation technique that involves modifying the data structure in software. Data obfuscation can be divided into three subcategories:

Data aggregation [10] involves changes in data grouping through: merging two or more scalar variables into a single composite variable (e.g., in a structure or class), inheriting interfaces to create complex class hierarchies (adding additional, unnecessary classes or methods, increasing code complexity, but also creating dummy classes that are not used at all in the program), and reorganizing data in arrays (e.g., by splitting one array into several smaller ones or vice versa, or by converting one-dimensional arrays into multidimensional ones and vice versa).

Code 10 – Adding Scalar Variables

```
num1 = 10
num2 = 20
sum = num1 + num2
print(sum)
```

Code 11 – Adding Variables in an Array

```
data = [10, 20]
sum = data[0] + data[1]
print(sum)
```

By merging two scalar variables into a single composite variable (a tuple), the data structure of the program was changed.

Code 12 – Class Performing Simple Arithmetic Operations

```
class Calculator:
    def add(self, a, b):
        return a + b

    def subtract(self, a, b):
        return a - b
```

Code 13 – Several Classes Performing Arithmetic Operations

```
class BasicMath:
    def add(self, a, b):
        return a + b

class AdvancedMath(BasicMath):
    def subtract(self, a, b):
        return a - b

class Unused:
    def do_nothing(self):
        pass

calc = AdvancedMath()
```

After adding the class hierarchy, the code became more complex. Additionally, the "Unused" class was introduced as an extra element, not contributing to the actual functionality of the application.

Code 14 – One-Dimensional Array Storing Values

```
array = [1, 2, 3, 4, 5]
```

Code 15 – Two-Dimensional Array Storing Values

```
array = [[1, 2], [3, 4], [5]]
```

Changing the array structure from one-dimensional to multidimensional results in the need for multi-level indexing, which can make data understanding and manipulation more difficult.

Data Storage and Encoding [10] affects the way data is stored and interpreted through: converting local variables into global ones, changing simple variable value declarations into mathematical or logical expressions, and, as with data aggregation, inheriting interfaces to create complex class hierarchies and reorganizing data in arrays.

Code 16 – Using a Local Variable as a Counter Inside a For Loop

```
def increment():
    counter = 0
    for i in range(10):
        counter += 1
    return counter

print(increment())
```

Code 17 – Using a Global Variable as a Counter Inside a For Loop

```
counter = 0

def increment():
    global counter
    for i in range(10):
        counter += 1

increment()
print(counter)
```

Converting the local variable *counter* into a global one can introduce confusion about the sources of changes to its value, especially in larger programs where many functions may modify global variables.

Code 18 – Simple Variable Declaration

```
number = 10
```

Code 19 – Variable Declaration Using a Mathematical Expression

```
number = (100 - 90) * (2 ** 1) + (6 - 1)
```

The use of a mathematical expression makes it difficult to quickly understand the code during analysis without using additional tools.

Data Ordering [10] involves modifying the order of data declarations in a program by: changing the order in which methods are declared, randomizing the order of method parameters, using a mapping function, and randomizing the order of variable declarations in classes.

Code 20 – Calculator with Logical Order of Methods

```
class Calculator:
    def __init__(self, value=0):
        self.value = value

    def add(self, number):
        self.value += number

    def subtract(self, number):
        self.value -= number

    def get_value(self):
        return self.value

calc = Calculator()
calc.add(10)
calc.subtract(5)
print(calc.get_value())
```

Code 21 – Calculator with Changed Order of Methods

```
class Calculator:
    def subtract(self, number):
        self.value -= number

    def get_value(self):
        return self.value

    def __init__(self, value=0):
        self.value = value

    def add(self, number):
        self.value += number

calc = Calculator()
calc.add(10)
calc.subtract(5)
print(calc.get_value())
```

The methods were rearranged in a non-intuitive order, which can cause confusion during code analysis.

Code 22 – Method with Logical Order of Parameters

```
def process_data(data, flag):
    if flag == 'special':
        return data * 2
    return data + 10

print(process_data(5, 'special'))
```

Code 23 – Method with Changed Order of Parameters

```
def process_data(flag, data):
    if flag == 'special':
        return data * 2
    return data + 10

print(process_data('special', 5))
```

Changing the order of parameters using randomization can make functions difficult to understand, especially when a large project has many functions with a lot of parameters.

Code 24 – Variable Declarations in a Class in Logical Order

```
class DataHolder:
    def __init__(self):
        self.data = []
        self.size = 0

    def add_data(self, value):
        self.data.append(value)
        self.size += 1

holder = DataHolder()
holder.add_data(10)
print(holder.size)
```

Code 25 – Variable Declarations in a Class in Changed Order

```
class DataHolder:
    def __init__(self):
        self.size = 0
        self.data = []

    def add_data(self, value):
        self.data.append(value)
        self.size += 1

holder = DataHolder()
holder.add_data(10)
print(holder.size)
```

Furthermore, changing the order of variable declarations in a class can impact the difficulty of understanding the code.

Code 26 – Calculator After Applying Data Obfuscation Technique

```
gS = 2

def xZ15(rTv):
    return rTv[0] + rTv[1]

def Ts3D(eiK, vZ4):
    return eiK - vZ4 * (gS - 1)

def NwwC():
    while True:
        print("Operations:\n1.     Addition\n2.
Subtraction\n3. Multiplication\n4. Division\n5.
Exit ")
        uM     =     input("Select     operation
(1/2/3/4/5): ")

        if uM == '5':
            break

        if uM in ('1', '2', '3', '4'):
            Xx4 = float(input("Enter the first
number: "))
            Go2 = float(input("Enter the second
number: "))

            if uM == '1':
                mM3 = (Xx4, Go2)
                print("Result: ", xZ15(mM3))

            elif uM == '2':
                print("Result:   ",   Ts3D(Xx4,
Go2))

            elif uM == '3':
                mp9 = (Go2, Xx4)
                print("Result: ", P1W9(mp9))

            elif uM == '4':
                Ep2 = 1
                TiX = Go2 * (Xx4 * gS - 2 *
Xx4) * 14 + Ep2 * Go2
                Er = Il1y(Xx4, TiX)
                print("Result:       "       if
isinstance(Er, float) else "", Er)
        else:
            print("Incorrect choice")

def P1W9(P0d):
    return P0d[0] * P0d[1]

def Il1y(bJS, AI1):
    if AI1 == 0:
        return "You can't divide by 0!!"
    return bJS / AI1

NwwC()
```

In the calculator from the beginning of the chapter, besides using the name obfuscation technique, the data obfuscation technique was also applied. Thus, in the functions xZ15 and P1W9, data structures containing both values were used instead of two separate parameters. Moreover, the program includes mathematical expressions involving an additional global variable, gS. The final changes introduced as part of the technique were altering the order of method and parameter declarations.

Code 27 – Script for Changing an Image on a Website After Applying Data Obfuscation Technique

```
const myE4 =
document.querySelectorAll('.changeable-image');
const Af13 = ['image3.png', 'image2.png',
'image1.png'];

myE4.forEach(image => {
    image.addEventListener('click', function()
{
        let hji =
Af13.indexOf(this.src.split('/').pop());
        let W6r = (hji - 1 + Af13.length) %
Af13.length;
```

```
        this.src = Af13[W6r];
    });
});
```

In the script for changing images, using the data obfuscation technique, the order of images in the array Af13 was changed and the expression W6r was complicated in such a way that the images are displayed on the page in their original order.

Code flow obfuscation (also known as Control flow obfuscation) is a technique of intentionally compiling the structure of a program to make it more difficult to analyze. It uses methods such as changing the order of operations, introducing intermediate jumps, using branching functions and changing the order of code blocks without affecting functionality. For example, the if-else condition can be replaced with a switch statement [11]. This technique also uses dead code, which, despite appearing in the program code, is never executed [7].

Code 28 – Checking for Even Numbers

```
number = int(input("Enter a number: "))
if number % 2 == 0:
    print(f"{number} is even.")
else:
    print(f"{number} is odd.")
```

Code 29 – Checking for Even Numbers After Applying Code Flow Obfuscation Technique

```
def is_even(number):
    try:
        if number % 2:
            raise  ValueError("Number  is
odd")
        else:
            return True
    except ValueError as e:
        print(e)
        return False

number = int(input("Enter a number: "))
result = is_even(number)
if result:
    print(f"{number} is even.")
else:
    _unused_var = None
    print(f"{number} is odd.")
```

In the above program that checks the parity of an entered number, the direct check for evenness was replaced with a function, an unnecessary and unused variable was added, and the if-else statement was unusually replaced with a try-except block, thereby controlling the program flow.

Code 30 – Calculator After Applying Code Flow Obfuscation Technique

```
gS = 2

def xZ15(rTv):
    return rTv[0] + rTv[1]

def Ts3D(eiK, vZ4):
    return eiK - vZ4 * (gS - 1)

def iC5(Ad):
    return Ad[0] * gS - Ad[1]

def NwwC():
    op = {
        '1': lambda x, y: iC5((x, y + gS)),
        '2': Ts3D,
        '3': lambda x, y: iC5((y, x + gS)),
        '4': lambda x, y: Il1y(x, y * (x * gS -
2 * x) * 14 + 1 * y)
    }

    while True:
        print("Operations:\n1.     Addition\n2.
Subtraction\n3. Multiplication\n4. Division\n5.
Exit ")
        uM    =    input("Select    operation
(1/2/3/4/5): ")

        if uM == '5':
            break

        if uM in op:
            Xx4 = float(input("Enter the first
number: "))
            Go2 = float(input("Enter the second
number: "))

            if uM == '1':
                mM3 = (Xx4, Go2)
                print("Result: ", xZ15(mM3))

            elif uM == '2':
                print("Result:  ",  op[uM](Xx4,
Go2))

            elif uM == '3':
                mp9 = (Go2, Xx4)
                print("Result: ", P1W9(mp9))

            elif uM == '4':
                Ep2 = 1
                TiX = Go2 * (Xx4 * gS - 2 *
Xx4) * 14 + Ep2 * Go2
                Er = Il1y(Xx4, TiX)
                print("Result:       "       if
isinstance(Er, float) else "", Er)
        else:
            print("Incorrect choice")

def P1W9(P0d):
    return P0d[0] * P0d[1]

def Il1y(bJS, AI1):
    if AI1 == 0:
        return "You can't divide by 0!!"
    return bJS / AI1

NwwC()
```

In the initially prepared calculator program, additional changes were introduced by adding the function *iC5*, which is an example of dead code. Mapping instructions were also added, which can replace if-else statements. However,

in the proposed example, most of the mapped operations will never be executed, further complicating the program's analysis.

Code 31 – Script for Changing an Image on a Website After Applying Code Flow Obfuscation Technique

```
const myE4 =
document.querySelectorAll('.changeable-image');
const Af13 = ['image3.png', 'image2.png',
'image1.png'];
let b5 = false

myE4.forEach(image => {
    image.addEventListener('click', function()
{
        let hji =
Af13.indexOf(this.src.split('/').pop());
        let W6r = (hji - 1 + Af13.length) %
Af13.length;
        this.src = Af13[W6r];

        if (b5) {
            let uVrs = Math.sqrt(-1);
            let aUd = uVrs + 100;
            aUd = aUd * uVrs;
            b5 = !b5;
        }
    });
});
```

Źródło: badania własne

A variable *b5* and a conditional statement that will never be executed were added to the above code.

String encryption [12] – this technique involves encrypting strings of characters in the program code in order to make it more difficult to analyze and search for specific code fragments. Encrypted strings of characters stored in the program code are decrypted when needed.

Code 32 – Program Printing "Hello World"

```
message = "Hello World"
print(message)
```

Code 33 – Program Printing "Hello World" After Applying String Encryption Technique

```
import base64

def decrypt_string(encoded_text):
    decoded_bytes                          =
base64.b64decode(encoded_text.encode('utf-
8'))
    return str(decoded_bytes, 'utf-8')

message = "SGVsbG8gV29ybGQ="
print(decrypt_string(message))
```

In the above program, Base64 encoding was used to hide the meaning of the printed text. However, when the program is executed, this text is readable and does not differ from the original version of the program.

Code 34 – Calculator After Applying String Encryption Technique

```
gS = 2

def xZ15(rTv):
    return rTv[0] + rTv[1]

def Ts3D(eiK, vZ4):
    return eiK - vZ4 * (gS - 1)

def iC5(Ad):
    return Ad[0] * gS - Ad[1]

def NwwC():
    op = {
        '1': lambda x, y: iC5((x, y + gS)),
        '2': Ts3D,
        '3': lambda x, y: iC5((y, x + gS)),
        '4': lambda x, y: Il1y(x, y * (x * gS -
2 * x) * 14 + 1 * y)
    }

    while True:
        print(''.join(chr(code)  for  code  in
[79, 112, 101, 114, 97, 116, 105, 111, 110,
115, 58, 10, 49, 46, 32, 65, 100, 100, 105,
116, 105, 111, 110, 10, 50, 46, 32, 83, 117,
98, 116, 114, 97, 99, 116, 105, 111, 110, 10,
51, 46, 32, 77, 117, 108, 116, 105, 112, 108,
105, 99, 97, 116, 105, 111, 110, 10, 52, 46,
32, 68, 105, 118, 105, 115, 105, 111, 110, 10,
53, 46, 32, 69, 120, 105, 116, 32]))
        uM = input(''.join(chr(code) for code
in [83, 101, 108, 101, 99, 116, 32, 111, 112,
101, 114, 97, 116, 105, 111, 110, 32, 40, 49,
47, 50, 47, 51, 47, 52, 47, 53, 41, 58, 32]))

        if uM == '5':
            break

        if uM in op:
            Xx4 = float(input(''.join(chr(code)
for code in [69, 110, 116, 101, 114, 32, 116,
104, 101, 32, 102, 105, 114, 115, 116, 32, 110,
117, 109, 98, 101, 114, 58, 32])))
            Go2 = float(input(''.join(chr(code)
for code in [69, 110, 116, 101, 114, 32, 116,
104, 101, 32, 115, 101, 99, 111, 110, 100, 32,
110, 117, 109, 98, 101, 114, 58, 32])))

            if uM == '1':
                mM3 = (Xx4, Go2)
                print(''.join(chr(code)     for
code in [82, 101, 115, 117, 108, 116, 58, 32]),
xZ15(mM3))

            elif uM == '2':
                print(''.join(chr(code)     for
code in [82, 101, 115, 117, 108, 116, 58, 32]),
op[uM](Xx4, Go2))

            elif uM == '3':
                mp9 = (Go2, Xx4)
                print(''.join(chr(code)     for
code in [82, 101, 115, 117, 108, 116, 58, 32]),
P1W9(mp9))

            elif uM == '4':
                Ep2 = 1
                TiX = Go2 * (Xx4 * gS - 2 *
Xx4) * 14 + Ep2 * Go2
                Er = Il1y(Xx4, TiX)
```

```
                print(''.join(chr(code)     for
code in [82, 101, 115, 117, 108, 116, 58, 32])
if isinstance(Er, float) else "", Er)
        else:
            print(''.join(chr(code) for code in
[73, 110, 99, 111, 114, 114, 101, 99, 116, 32,
99, 104, 111, 105, 99, 101]))

def P1W9(P0d):
    return P0d[0] * P0d[1]

def Il1y(bJS, AI1):
    if AI1 == 0:
        return  ''.join(chr(code)  for  code  in
[89, 111, 117, 32, 99, 97, 110, 39, 116, 32,
100, 105, 118, 105, 100, 101, 32, 98, 121, 32,
48, 33]
)
    return bJS / AI1

NwwC()
```

Code 35 – Script for Changing an Image on a Website After Applying String Encryption Technique

```
const myE4 =
document.querySelectorAll(String.fromCharCode(..
.[46, 99, 104, 97, 110, 103, 101, 97, 98, 108,
101, 45, 105, 109, 97, 103, 101]));
const Af13 = [String.fromCharCode(...[105, 109,
97, 103, 101, 51, 46, 112, 110, 103]),
String.fromCharCode(...[105, 109, 97, 103, 101,
50, 46, 112, 110, 103]),
String.fromCharCode(...[105, 109, 97, 103, 101,
49, 46, 112, 110, 103])];
let b5 = false

myE4.forEach(image => {

image.addEventListener(String.fromCharCode(...[9
9, 108, 105, 99, 107]), function() {
        let hji =
Af13.indexOf(this.src.split('/').pop());
        let W6r = (hji - 1 + Af13.length) %
Af13.length;
        this.src = Af13[W6r];

        if (b5) {
            let uVrs = Math.sqrt(-1);
            let aUd = uVrs + 100;
            aUd = aUd * uVrs;
            b5 = !b5;
        }
    });
});
```

In both of the above source codes, strings have been replaced with arrays of ASCII characters.

Source code obfuscation – is a technique that involves hiding the meaning of the code by removing comments and changing identifier names before it is handed over for maintenance or testing [5]. The code proposed at the beginning has no comments, and its identifier names have already been changed in the obfuscation process.

4. Conclusion

Code obfuscation is a relatively simple process that you can use to protect your intellectual property. It should be remembered that this process can always be reversed. However, analyzing obfuscated code is more difficult and resource-consuming than analyzing non-obfuscated code. The easiest way to prove this is to look again at the program codes numbered 1 and 3, then 34 and 35.

5. Bibliography

[1] Kovacevic A., “What is Code Obfuscation? How to Disguise Your Code to Make it More Secure”, #CYBERSECURITY, 20.11.2020.

[2] Król K., „Wpływ dekompresji kodu źródłowego (unminify process) na wydajność aplikacji mapowej”, [online, 29.10.2020].

[3] Collberg Ch., Thomborson C., Low D., “A Taxonomy of Obfuscating Transformations”, [online, 01.1997].

[4] Brzozowski M., Yarmolik V.N., „Obfuskacja – narzędzie zabezpieczające prawa autorskie do projektów sprzętowych”, *Pomiary Automatyka Kontrola*, Vol. 54(8), 477–479 (2008).

[5] Popa M., “Techniques of Program Code Obfuscation for Secure Software”, *Journal of Mobile, Embedded and Distributed Systems*, Vol. 3, No. 4 (2011).

[6] allatori.com, “Incremental Obfuscation”, (dostęp: 19.05.2024)

[7] Govindraj B., “Code Obfuscation: A Comprehensive Guide Against Reverse-Engineering Attempts”, [online, 05.06.2023].

[8] allatori.com, “Watermarking”, (dostęp: 19.05.2024).

[9] Kubiak S., „Obfuskacja kodu”, [online, 07.06.2019].

[10] Faruki P., et al., “Android Code Protection via Obfuscation Techniques: Past, Present and Future Directions”, *arXiv1611.10231* (2016).

[11] Brook Ch., “What is Code Obfuscation & How Does It Work?” [online, 02.05.2024].

[12] Brook Ch., “What is Code Encryption and How Does it Work?” [online, 26.06.2023].

Wybrane techniki obfuskacji kodu źródłowego w językach skryptowych

M. MOHR

Artykuł przedstawia najpopularniejsze techniki zaciemniania kodu źródłowego programów komputerowych. Obfuskacja jest zdefiniowana i przedstawiona na prostych i łatwych do zrozumienia przykładach kodu napisanego w językach skryptowych, takich jak Python i JavaScript. Omówiono jej zastosowania i pokazano, jak w prosty sposób utrudnić analizę własnych programów.

Słowa kluczowe: obfuskacja kodu, języki skryptowe.