АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

ALGORITHMIC AND PROGRAMMING

УДК 004.051

В. С. Кантарович[1], **П. П. Урбанович**[1,2]
[1]Белорусский государственный технологический университет
[2]Люблинский Католический университет Яна Павла II (Польша)

ЭФФЕКТИВНОСТЬ МЕТОДОВ ОБФУСКАЦИИ ПРОГРАММНОГО КОДА ДЛЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ *PHP* И *JAVASCRIPT*

В статье рассматривается актуальная проблема защиты интеллектуальной собственности на коды компьютерных программ с помощью методов обфускации. Приведены результаты исследования и сравнительного анализа эффективности методов обфускации программного кода, которые применяются в онлайн-обфускаторах для языков программирования *PHP* и *JavaScript.* Рассмотрены известные методы обфускации: на основе упаковки кода, шифрования данных, вставки «мертвого» кода, а также лексических, структурных и семантических изменений кода. Проанализированы влияние методов обфускации на производительность выполнения программного кода, сохранение его функциональности и устойчивость к реверсивной инженерии. Предлагается новый классификатор методов обфускации, который систематизирует их по области применения, динамике выполнения, характеру модификаций и уровню представления кода. Выделяются два основных класса методов: общие, применимые к разным языкам и программам, и специализированные, учитывающие особенности конкретных языков. Предложены критерии и метрики оценки эффективности обфускации, которые включают устойчивость к реверсивной инженерии, уровень защиты, сохранение функциональности, сложность декомпиляции и производительность. На основе метрик выполнена сравнительная оценка обфускаторов. Приведены результаты испытаний онлайн-обфускаторов для двух тестовых файлов с разным уровнем сложности исходного кода для языков программирования *PHP* и *JavaScript*.

Ключевые слова: программный код, интеллектуальная собственность, обфускация, деобфускация, онлайн-обфускаторы, реверсивная инженерия.

Для цитирования: Кантарович В. С., Урбанович П. П. Эффективнось методов обфускации программного кода для языков программирования *PHP* и *JavaScript* // Труды БГТУ. Сер. 3, Физико-математические науки и информатика. 2026. № 1 (302). С. 100–117.

DOI: 10.52065/2520-6141-2026-302-9.

V. S. Kantarovich[1], **P. P. Urbanovich**[1,2]
[1]Belarusian State Technological University
[2] The John Paul II Catholic University of Lublin (Poland)

EFFICIENCY OF PROGRAM CODE OBFUSCATION METHODS FOR PHP AND JAVASCRIPT PROGRAMMING LANGUAGES

This article examines the pressing issue of protecting intellectual property of computer program code using obfuscation methods. It presents the results of a study and comparative analysis of the effectiveness of code obfuscation methods used in online obfuscators for the *PHP* and *JavaScript* programming languages. Common obfuscation methods are considered, including those based on code packing, data encryption, «dead» code insertion, as well as lexical, structural, and semantic code modifications. The impact of obfuscation methods on the performance of program code execution, the preservation of its functionality, and its resistance to reverse engineering are analyzed. A new obfuscation method classifier

is proposed, which systematizes them by area of application, execution dynamics, nature of modifications, and code representation level. Two main classes of methods are distinguished: general ones, applicable to various languages and programs, and specialized ones, taking into account the features of specific languages. Criteria and metrics for evaluating obfuscation effectiveness are proposed, including resistance to reverse engineering, protection level, functionality preservation, decompilation complexity, and performance. Based on these metrics, a comparative evaluation of obfuscators is performed. The results of online obfuscator tests for two files with different levels of source code complexity for the *PHP* and *JavaScript* programming languages are presented.

Keywords: program code, intellectual property, obfuscation, deobfuscation, online obfuscators, reverse engineering.

For citation: Kantarovich V. S., Urbanovich P. P. Efficiency of program code obfuscation methods for *PHP* and *JavaScript* programming languages. *Proceedings of BSTU, issue 3, Physics and mathematics. Informatics*, 2026, no. 1 (302), pp. 100–117 (In Russian).

DOI: 10.52065/2520-6141-2026-302-9.

Введение. С каждым годом растет количество случаев несанкционированного копирования и использования интеллектуальной собственности, что создает необходимость в эффективных механизмах защиты уникальных алгоритмов и логики работы программ. Одним из таких важных инструментов в сфере информационной безопасности является обфускация, которая направлена на защиту интеллектуальной собственности и предотвращение несанкционированного доступа к исходному коду [1–3]. Обфускация помогает затруднить анализ кода и предотвратить его несанкционированное использование, что становится особенно актуальным в условиях современного программирования.

Обфускация – это процесс преднамеренного усложнения исходного кода программы и внесение в него модификаций с целью затруднения его анализа и понимания. Это достигается путем применения различных методов, которые делают код менее читаемым и понятным, но не влияют на его функциональность, что усложняет задачу для злоумышленников, стремящихся восстановить оригинальный код. Таким образом, обфускация представляет собой необходимый компонент для защиты программного кода, обеспечивающий баланс между его доступностью и безопасностью.

Использование оригинальных алгоритмов обфускации способно повысить сложность анализа и декомпиляции, так как современные деобфускаторы программного кода обладают высокой эффективностью в обработке хорошо изученных методов. Эти инструменты, основанные на известных подходах и техниках [3–6], могут быстро идентифицировать и нейтрализовать традиционные механизмы защиты. В связи с этим интеграция инновационных подходов в процесс обфускации может не только затруднить анализ, но и создаст дополнительный уровень защиты, который усложнит задачу реверсивной инженерии и снизит вероятность успешной декомпиляции кода злоумышленниками.

Основным предметом нашего исследования являются онлайн-обфускаторы, которые поддерживают такие интерпретируемые языки программирования, как *JavaScript* и *PHP*. Это объясняется тем, что *JavaScript*-код выполняется на стороне клиента, что делает его уязвимым для несанкционированного доступа и анализа. Учитывая, что большинство веб-приложений зависит от *JavaScript* для взаимодействия с пользователем, защита этого кода становится критически важной. *PHP* также является одним из наиболее распространенных серверных языков программирования, которые используются для создания веб-приложений. С учетом широкого распространения *JavaScript* и *PHP* в веб-разработке эффективная обфускация кода для этих языков программирования становится необходимой для защиты интеллектуальной собственности и снижения рисков несанкционированного копирования.

Эффективность использования алгоритмов обфускации программного кода можно оценивать по нескольким критериям [6]. Важным аспектом является сложность анализа, т. е. оценка того, насколько трудным становится процесс реверсивной инженерии кода, что можно измерить временем, необходимым для декомпиляции, и степенью различий между полученным обфусцированным кодом и оригинальным, что помогает выявить изменения

в структуре и логике, в именах функций и переменных. Еще одним не менее значимым аспектом является оценка того, как обфускация влияет на производительность программы, так как эффективные алгоритмы должны минимально сказываться на скорости выполнения. Эти критерии позволяют оценить эффективность обфускации.

Основная часть. Целью исследования является анализ и сравнение существующих методов обфускации программного кода, которые применяются в известных и доступных онлайн-обфускаторах для языков программирования *PHP* и *JavaScript* и являются основой для реализации инструментальных средств на основе нейросетевых технологий. Более конкретно указанная цель сводилась к определению степени влияния методов обфускации на производительность программного кода, его устойчивость к реверсивной инженерии.

Эффективность методов обфускации по критерию повышения защищенности программного кода от несанкционированного доступа и анализа зависит от нескольких основных факторов [7]. Кратко проанализируем их.

1. *Сложность исходного кода*. Более сложные и структурированные элементы труднее обфусцировать. На это могут влиять количество строк и их структура: например, число условий, циклов, вложенных структур. Кроме того, сложные конструкции могут требовать больших ресурсов для обфускации, что может негативно сказываться на производительности работы программы.

2. *Язык программирования*. Разные языки имеют свои особенности и средства для реализации методов обфускации. Для компилируемых языков, таких как *Java*, *C#*, могут использоваться сложные алгоритмы, например, динамическая обфускация, упаковка исполняемых файлов и удаление информации об отладке. Для интерпретируемых языков, например *JavaScript*, могут применяться различные упаковщики кода, такие как *Function Packing*, *String Packing*, при которых весь исходный код упаковывается соответственно в функцию и строку [8, 9].

3. *Методы обфускации*. Они разнообразны, и их реализация зависит от контекста применения, сложности и языка исходного кода. Эффективность отдельных методов может изменяться. Например, для обфускации сложного кода простые методы могут не обеспечить нужную степень защиты. Рекомендуется комбинировать разные виды и методы обфускации, чтобы усложнить анализ кода [10].

4. *Среда выполнения*. Операционная система, аппаратное обеспечение и особенности реализуемого кодом алгоритма также влияют на эффективность обфускации и производительность обфусцированного исходного кода.

Классификация методов обфускации. В основу предлагаемой версии классификатора положены следующие основные критерии: область применения, динамика выполнения, характер модификации кода и уровень его представления. С нашей точки зрения такой подход характеризуется большей общностью и предоставляет возможности для анализа эффективности различных методов обфускации.

Методы обфускации можно классифицировать по следующим признакам.

1. *Область применения*. По этому признаку методы обфускации можно поделить на следующие:

– общие;
– специализированные.

Общие методы ориентированы на широкий круг программных продуктов и на разные языки программирования. Они используют более общие методы и подходы к реализации «запутывания», например лексическую обфускацию, которая может включать в себя переименование переменных, классов, удаление комментариев, изменение структуры кода.

Для исходного кода, представленного в листинге на рис. 1, продемонстрировано использование общих методов на примере применения лексической обфускации, что показано в листинге на рис. 2. Здесь имена переменных `param_1`, `param_2` и функции `output()` заменены на трудно воспринимаемые случайные сгенерированные последовательности символов, и в *JavaScript*-код внедряется функция с именем `bCdEfGh()`, которая никогда не вызывается и манипулирует неиспользуемыми переменными, не влияющими на выполнение основной

функции, что усложняет анализ кода. В *PHP*-код добавлены новые переменные `unusedVar1`, `unusedVar2`, массив `unusedArray`, которые не используются в основной логике, и цикл `for`, содержащий условие и не оказывающий влияния на выполнение функции, но создающий дополнительное запутывание. Также строка «`World`» дополнительно закодирована в `base64`.

```
function output(param_2)
{
  let param_1 = "Hello, " + param_2 + "!";
  console.log(param_1);
}
output("World");
```

а

```
<?php
function output($param_2)
{
    $param_1 = "Hello, " . $param_2 . "!";
    echo $param_1;
}
output("World");
?>
```

б

Рис. 1. Пример исходного *JavaScript*-кода (*а*),
PHP-кода (*б*) до применения общих методов обфускации

```
function aBcDeFg(hIjKlMn) {
  let oPqRsTu = "Hello, " + hIjKlMn + "!";
  console.log(oPqRsTu);

// Вставка неиспользуемого кода, который не влияет на структуру
  if (false) {
    let uVwXyZ = "This variable is never used.";
    console.log("This will never be printed.");
  }
  function bCdEfGh()
{
    let iJkLmNo = 5 + 5;
    return iJkLmNo;
  }}
aBcDeFg("World");
```

а

```
<?php
function a($b)
{
    $c =
base64_decode("SGVsbG8sICR7JHJvfSE=");
    $d = str_replace("{$b}", "{$b}", $c);
    echo $d;

    // Вставка неиспользуемого кода, который
не влияет на структуру
    $unusedVar1 = "This is unused.";
    $unusedVar2 = 12345;
    $unusedArray = array(1, 2, 3, 4, 5);
    for ($i = 0; $i < 100; $i++)
{
        if ($i === 50)
{
            continue;
}
}}
a("V29ybGQ=");
?>
```

б

Рис. 2. Пример использования общих методов обфускации
для исходного *JavaScript*-кода (*а*), *PHP*-кода (*б*)

К онлайн-обфускаторам, которые используют общие методы обфускации, относятся JavaScript *Obfuscator Tool* [11], *EvalPacker Obfuscator* [12], *JavaScript Obfuscator* [13], *PHP obfuscator* [14], *WB PHP Obfuscator* [15].

Специализированные методы обфускации основываются на специфических особенностях языков программирования и требуют более глубокого понимания принципов их функционирования. Эффективность этих методов значительно зависит от особенностей конкретного программного кода, а также от используемых средств для деобфускации.

К специализированным методам относятся:

– полиморфная обфускация (изменяет структуру кода таким образом, чтобы он выполнял одну и ту же функцию, но при каждом запуске обфускатора структура кода будет изменяться);

– обфускация потока управления (распараллеливает код, изменяет порядок операторов выполнения программы, реструктуризирует циклы);

– обфускация структур данных (упаковка кода, шифрование строк кода);

– динамическая обфускация;

– удаление отладочной информации;

– скрытие или удаление сигнатур методов и интерфейсов.

Для исходного кода на языках *JavaScript* и *PHP*, представленного в листинге на рис. 1, продемонстрировано использование обфускации потока управления и упаковки кода в константу в листинге на рис. 3.

а

```
const _0x51e847 = _0x1c51;
(function(_0x2812f8, _0x17f43d) {const _0x73f48 = _0x1c51,
        _0x12a5c4 = _0x2812f8();
    while (!![]) {  try {
            const _0x243243 = -parseInt(_0x73f48(0x181)) / (-
0x2ad * -0x9 + -0x1cc * 0x8 + -0x9b4) + par-
seInt(_0x73f48(0x182)) / (-0x1 * -0x1606 + 0x1c11 + -0x3215)
* (parseInt(_0x73f48(0x180)) / (0x1 * -0x1f5d + 0x19 * -0xd8
+ -0x248 * -0x17)) + -parseInt(_0x73f48(0x184)) / (-0x2 *
0xa7 + -0x859 * -0x1 + -0x707 * 0x1) * (-par-
seInt(_0x73f48(0x17e)) / (0x1739 + -0x1 * 0x20d2 + 0x99e)) +
parseInt(_0x73f48(0x17b)) / (0x22d3 + -0x26f9 + 0x42c) + par-
seInt(_0x73f48(0x178)) / (0x1065 + -0x780 + -0x8de) * (par-
seInt(_0x73f48(0x185)) / (-0x56 * 0x56 + -0x1ca3 + 0x398f)) +
-parseInt(_0x73f48(0x175)) / (0x1 * -0x3a + -0x55b + 0x59e) +
-parseInt(_0x73f48(0x17c)) / (-0x1 * 0xdd5 + 0x5 * -0x62 +
0x3 * 0x543) * (-parseInt(_0x73f48(0x183)) / (-0x3f * 0x51 +
-0x19c1 * -0x1 + -0x1ed * 0x3));
            if (_0x243243 === _0x17f43d) break;
            else _0x12a5c4['push'](_0x12a5c4['shift']());
        } catch (_0x20bdab) {
            _0x12a5c4['push'](_0x12a5c4['shift']()); }}
}(_0x3446, 0x1bf7 + -0x5bdb8 + 0xea3d4));
function output(_0x193e35) {
    const _0x4b8e90 = _0x1c51,
        _0x47ae3a = {
            'vnmYo': function(_0x4f0013, _0x2f4f93) {
                return _0x4f0013 + _0x2f4f93;},
            'sTsIO': function(_0x36a291, _0x3e6da0) {
                return _0x36a291 + _0x3e6da0;},
            'ucCeZ': _0x4b8e90(0x177)};
    let _0x1fc114 =
_0x47ae3a[_0x4b8e90(0x17a)](_0x47ae3a[_0x4b8e90(0x176)](_0x47
ae3a[_0x4b8e90(0x17d)], _0x193e35), '!');
    console[_0x4b8e90(0x17f)](_0x1fc114);}
function _0x1c51(_0x4ad23c, _0x4581d1) {
    _0x4ad23c = _0x4ad23c - (-0x3d * -0x4f + -0x1b * -0x10a +
-0x2d6c);
    const _0x3853c9 = _0x3446();
    let _0x1594fb = _0x3853c9[_0x4ad23c];
    return _0x1594fb;}output(_0x51e847(0x179));
function _0x3446() {
    const _0x22f36f = ['224DdzRhd', 'World', 'vnmYo',
'6725718edYHmY', '1650AkOVKQ', 'ucCeZ', '313665PCNaDR',
'log', '32727GglWDn', '522279YLiOPV', '2xxwPKe', '297PYFZvh',
'4veGhVe', '64544qEIqsO', '3101328opxLiW', 'sTsIO',
'Hello,\x20'];
    _0x3446 = function() {
        return _0x22f36f;};  return _0x3446();}
```

б

```
<?php
$GLOBALS['_1843820376_']=Ar-
ray(); ?><? function
_1011077025($i){$a=Ar-
ray('SGVsbG8sIA==','IQ==','V29
ybGQ=');return base64_de-
code($a[$i]);} ?><?php func-
tion
l__0($_0){$_1=_1011077025(0)
.$_0 ._1011077025(1);echo
$_1;}l__0(_1011077025(2)); ?>
```

Рис. 3. Пример использования специализированных методов обфускации для исходного *JavaScript*-кода (*а*), *PHP*-кода (*б*)

В онлайн-обфускаторах специализированные методы обфускации используются редко и большинство из них предлагают базовые варианты методов: внедрение искусственных циклов, ветвлений и кодирование в формат *Base64*. Это связано с ресурсоемкостью такой обфускации, и часто онлайн-сервисы имеют ограничения на вычислительные ресурсы. К онлайн-обфускаторам, использующим специализированные методы обфускации, относятся *JavaScript Obfuscator Tool*, *EvalPacker Obfuscator*, *JavaScript Obfuscator*.

2. *Динамика выполнения*. Данный признак позволяет разделить методы обфускации на основе их воздействия на поведение программного продукта в процессе выполнения программы. К данной категории относятся:

– статическая обфускация;

– динамическая обфускация;

– обфускация с динамической проверкой подлинности.

Статическая обфускация включает методы, применяемые к исходному коду единожды, до его выполнения, т. е. на этапе компиляции. Эти методы модифицируют структуру кода без изменения его поведения в процессе выполнения программы. К данной категории обфускации относятся методы:

– замена имен переменных, функций и классов;

– удаление пробелов и комментариев;

– добавление неиспользуемого кода;

– изменение структуры кода;

– шифрование и упаковка.

Можно утверждать, что к этому типу принадлежат методы лексической обфускации, обфускации структур данных и обфускации потока управления. К обфускаторам, использующим статическую обфускацию, относятся все онлайн-обфускаторы, которые вносят изменения в исходный код или скомпилированный байт-код.

Динамическая обфускация включает методы, которые изменяют код в процессе выполнения программы. Это означает, что код будет трансформироваться каждый раз при запуске программы. К данной категории относятся:

– полиморфная обфускация (генерация различных вариантов кода, выполняющего одну и ту же функцию);

– полное изменение кода во время выполнения;

– шифрование и расшифрование кода в процессе исполнения.

Обфускаторы этого типа предназначены для языков программирования с компиляцией в машинный и байт-код (*C*, *C++*, *Java*, *Kotlin*, *C#*), например, *VMProtect*, *Code Virtualizer*, *DexGuard*, *Obfuscator-LLVM*. Для интерпретируемых языков программирования (*Python*, *JavaScript*, *PHP*) применяются такие методы динамической обфускации, как динамическое шифрование, динамическая упаковка кода.

Для исходного кода на языках *JavaScript* и *PHP*, представленного в листинге на рис. 1, приведен пример (рис. 4) применения динамического шифрования строк и упаковщика *Function Packing*. Упаковщик *Function Packing* примечателен тем, что он позволяет уменьшить размер кода и скрыть его логику, что затрудняет реверсивную инженерию. Этот метод объединяет несколько функций в одну и может применять динамическое шифрование строк, что делает анализ кода значительно сложнее.

```
eval(function(p, a, c, k, e, d) {
    e = function(c) {
        return c };
    if (!''.replace(/^/, String)) {
        while (c--) {
            d[c] = k[c] || c }
        k = [function(e) {
            return d[e]  }];
        e = function() {
            return '\\w+'    };
        c = 1  };
    while (c--) {
        if (k[c]) {
            p = p.replace(new RegExp('\\b' + e(c) +
'\\b', 'g'), k[c])} }
    return p
}('8  0(2){7  1="6,  "+2+"!";5.4(1)}0("3");',  9,  9,
'output|param_1|param_2|World|log|con-
sole|Hello|let|function'.split('|'), 0, {}))
```

а

```
< ? php
eval(base64_decode(
'CiBnb3RvIERRal9lOyBaRXpNMzogb-
3V0cHV0KCJcMTI3XHg2Zlx4NzJcMTU0-
XHg2NCIpOyBnb3RvIGxKUDdiOyBEWpf-
ZTogZVuY3Rpb24gb3V0cHV0KCRwYX-
JhbV8yKSB7ICRwYXJhbV8xID0gIlx4-
NDhceDY1XHg2Y1wxNTRceDZmXHgyY1
w0MCIgLiAkcGFyYW1fMiAuICJceDIxIjsg-
ZWNobyAkcGFyYW1fMTsgfSBnb3RvIFpFek0zOy-
BsSlA3Yjog'));
? >
```

б

Рис. 4. Пример использования динамической обфускации для исходных *JavaScript*-кода (*а*), *PHP*-кода (*б*)

Обфускация с динамической проверкой подлинности предполагает взаимодействие между обфусцированным кодом и внешней средой. К методам данной обфускации относится:

– привязка к определенному устройству или платформе, при которой код проверяет идентификационные данные устройства и прекращает свою работу, если он запущен на другом устройстве;

– лицензирование и активация кода через сервер с периодической проверкой срока действия лицензии;

– водяные знаки, которые внедряются в код, чтобы отслеживать его распространение;

– проверка целостности файлов;

– динамическое изменение обфусцированного кода, которое может включать в себя постоянное обращение к серверу для получения новых алгоритмов, модулей, библиотек.

3. *Характер модификации кода*. Здесь методы обфускации могут отличаться типом изменений, вносимых в исходный программный код. Можно выделить методы, добавляющие следующие типы изменений в код:

– синтаксические;

– семантические;

– структурные.

Методы обфускации, которые добавляют синтаксические изменения, затрагивают структуру кода, но никак не влияют на его выполнение. К таким методам относятся:

– переименование переменных, классов, методов;

– форматирование кода;

– изменение порядка операторов.

Пример обфускации кода с применением синтаксических изменений исходного кода приведен в листинге на рис. 5.

```
class X {
    outputMessage(y) {
        let z = (y ? "Hello, " + y + "!" :
"Hello, Guest!");
        console.log(z);
    }}
const a = new X();
const b = "World";
a.outputMessage(b);
```

а

```
<?php
class A {
    public function showMessage($b) {
        $c = "Hello, " . $b . "!";
        echo $c; }
}
$instance = new A();
$message = "World";
$instance->showMessage($message); ?>
```

б

Рис. 5. Пример использования методов обфускации, которые добавляют синтаксические изменения для исходных *JavaScript*-кода (*а*), *PHP*-кода (*б*)

В данные примеры были внесены такие синтаксические изменения, как создание объекта класса для изменения порядка выполнения операторов, переименование переменных и функций, удаление пробелов и переносов строк, использование тернарного оператора.

Методы обфускации, вносящие семантические изменения в код, затрагивают логику его выполнения. К таким изменениям относятся:

– шифрование строк;

– добавление «мертвого» кода;

– модификация обработки ошибок;

– замена значений переменных;

– изменение порядка операций и использование функций высшего порядка для выполнения простых операций.

Методы обфускации, вносящие структурные изменения в код, воздействуют на его организацию, не изменяя при этом логику или поведение программы. К структурным изменениям можно отнести добавление избыточных условных операторов или циклов, замену функций их содержимым, изменение типов данных, манипуляции с комментариями, а также добавление новых функций и классов.

Современные обфускаторы используют методы, реализующие совместно несколько типов изменений программного кода для усложнения его анализа [16].

4. *Уровни представления кода*. Процесс обфускации может быть осуществлен над любым из видов представления кода. Следовательно, можно выделить 3 уровня процесса обфускации:

– уровень машинного кода;

– уровень промежуточного кода (байт-кода);

– уровень исходного кода.

Обфускация на уровне машинного кода происходит, когда преобразование осуществляется над ассемблерным кодом программы или над двоичным файлом, хранящим машинный код. К таким обфускаторам относятся популярные коммерческие продукты *Code Virtualizer*, *VMProtect*, *Themida*. Данный тип обфускаторов использует виртуализацию, преобразуя части оригинального кода в байт-код для виртуальной машины следующим образом:

– заменяют простые машинные инструкции на более сложные, одинаковые по функциональности;

– используют полиморфизм;

– применяют обфускацию потока управления;

– используют различные регистры для хранения одних и тех же данных в разных частях кода;

– добавляют код, который не влияет на функциональность программы.

Обфускация на уровне промежуточного кода применяется для языков *.Net*, *Java*, которые компилируют свой исходный код в байт-код. При таком преобразовании применяются методы:

– лексической обфускации;

– обфускации структур данных;

– обфускации потока управления;

– виртуализации [17].

Примерами обфускаторов промежуточного кода являются *VMProtect*, *Code Virtualizer*, *Obfuscator-LLVM*.

Обфускация на уровне исходного кода осуществляется непосредственно над исходным кодом программы, написанным на языке высокого уровня. К данному типу относятся методы:

– лексической обфускации;

– обфускации структур данных;

– обфускации потока управления.

Обфускацию исходного кода следует использовать в сочетании с другими методами для обеспечения более надежной защиты программного обеспечения (*PHP Obfuscator*, *WB PHP Obfuscator*, *JavaScript Obfuscator Tool*, *EvalPacker Obfuscator*, *JavaScript Obfuscator*).

Иерархическое представление рассмотренной классификации методов обфускации приведено на рис. 6.

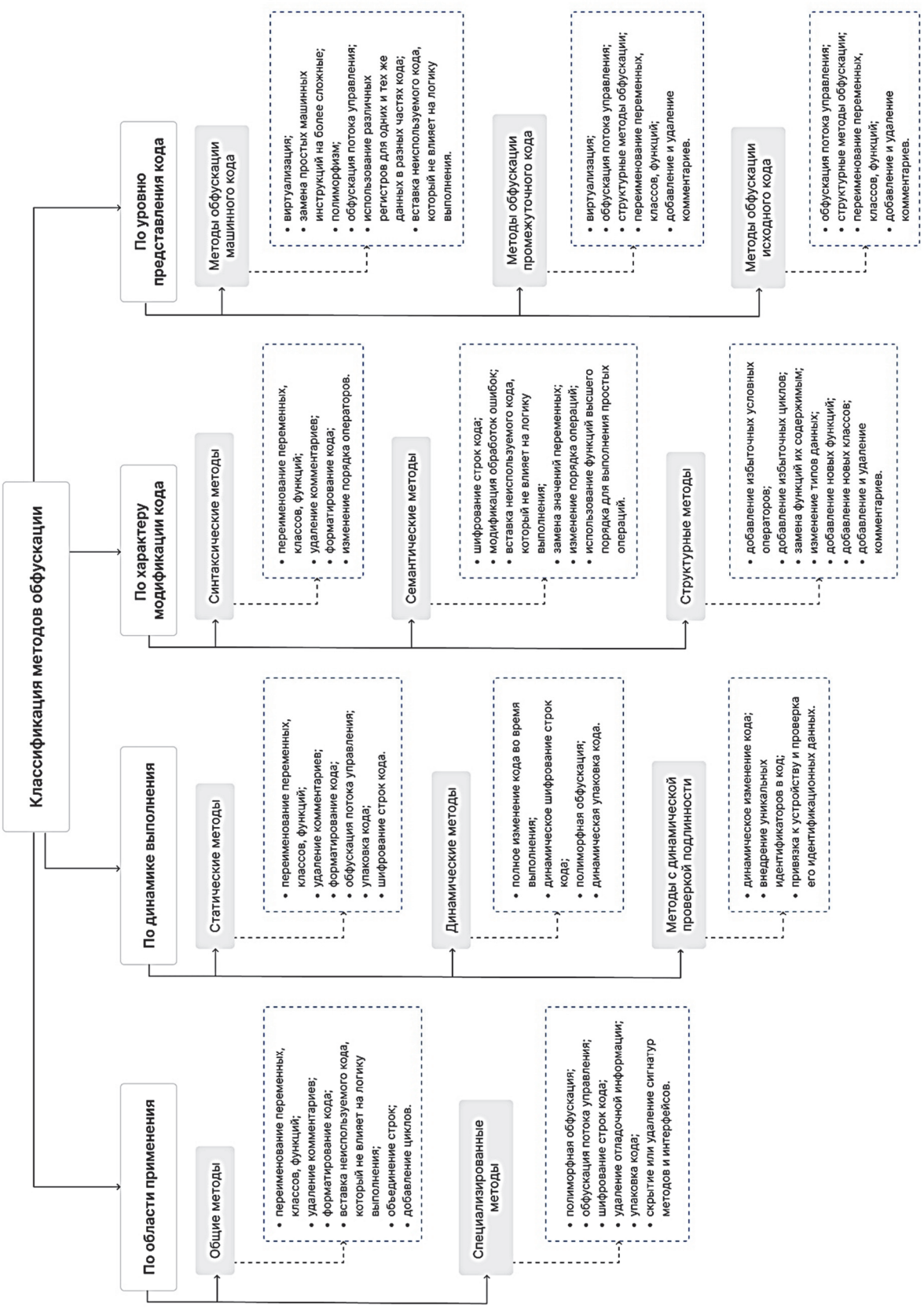


Рис. 6. Иерархическое представление классификации методов обфускации

Оценка эффективности обфускаторов программного кода. Оценка эффективности обфускаторов кода может включать несколько критериев или метрик [18–22]. В наших исследованиях мы использовали следующие:

– *устойчивость к реверсивной инженерии R* (*Resistance to Reverse Engineering*) – используется для описания уровня сложности реализации реверсивной инженерии над программой после обфускации кода;

– *уровень защиты P* (*Protection Level*) – характеризует степень сложности анализа и понимания кода злоумышленниками (на основе сравнения читабельности кода до и после обфускации);

– *сохранение функциональности F* (*Functionality Preservation*) – характеризует степень влияния обфускации на функциональность программы;

– *сложность декомпиляции D* (*Decompilation Difficulty*) – позволяет оценить сложность восстановления исходного кода после его обфускации;

– *производительность* O_p (*Perfomance*) – означает влияние обфускации на скорость выполнения программы;

– *цена «запутывания» кода* O_r (*Obfuscation Overhead*) – отражает объем ресурсов устройства, необходимого для запуска обфусцированного и необфусцированного кодов;

– *размер кода S* (*Code Size*) – характеризует влияние обфускации на общий размер программного продукта;

– технические возможности *U* (*Technical Capabilities*) – параметр характеризует поддержку различных языков программирования, а также наличие инструментов для анализа и отладки защищенного кода.

В рамках исследования был проведен анализ эффективности онлайн-обфускации для языков программирования *PHP* и *JavaScript*. Для этого использован подход, основанный на анализе работы обфускаторов с двумя различными тестовыми файлами (*test_1* и *test_2*). Время выполнения и размер выходных данных оценивались для нескольких режимов. Для более компактного представления результатов эксперимента производилось их усреднение. Время выполнения программного кода тестировалось на устройстве со следующими характеристиками: MacBook Pro, 2019 год выпуска, 2,4 GHz 4-ядерный процессор Intel Core i5, 8 ГБ оперативной памяти.

Количественные оценки рассмотренных выше характеристик вычислялись с помощью простых математических выражений, которые проанализируем ниже.

Устойчивость к реверсивной инженерии *R* представляется в виде функции роста энтропии кода и времени деобфускации:

$$R = (H_{obf} - H_{orig}) / T_{deobf}, \quad (1)$$

где H_{obf} – энтропия обфусцированного кода; H_{orig} – энтропия исходного кода; T_{deob} – время деобфускации в миллисекундах. Степень устойчивости повышается с возрастанием *R*.

Характеристика *P* формально измеряется степенью снижения читабельности кода:

$$P = 1 - C_{obf} / C_{orig}, \quad (2)$$

где C_{obf} – индекс для оценки читабельности кода после обфускации и C_{orig} – до обфускации.

Предлагается использовать индекс читабельности кода (*C*), который изменяется в диапазоне от 0 до 1; при $C = 0$ – код полностью нечитабелен и при $C = 1$ – код полностью читабелен и легко понимается. На основе экспертных оценок мы вводим следующую шкалу для *C*:

0,0–0,1 – крайне низкая читабельность; код полностью нечитабелен;

0,1–0,3 – низкая читабельность; код практически невозможно понять, структура и логика полностью скрыты;

0,3–0,5 – средняя читабельность; код частично читабелен, некоторые части могут быть понятны, но общая логика запутана; применяется обфускация с сохранением структуры кода;

0,5–0,7 – высокая читабельность; код в значительной степени читабелен, но содержит некоторые элементы, затрудняющие понимание; применяется обфускация только некоторых частей кода с сохранением значений и структуры переменных;

0,7–0,9 – очень высокая читабельность; код почти полностью читабелен, небольшие изменения могут быть трудными для обнаружения; основные структурные компоненты и логика понятны; применяется минимальная обфускация, например, с заменой имен функций;

0,9–1,0 – полная читабельность; код полностью читабелен.

Параметр F отождествляется с относительным числом (процентом) успешных выполнений обфусцированной программы и рассчитывается по формуле

$$F = N_{success} / N_{tot}, \tag{3}$$

здесь $N_{success}$ – число успешных тестов; N_{tot} – общее количество тестирований файла.

Численно характеристика D имеет бинарную форму: 0 – код после деобфускации восстановлен полностью или частично; 1 – код не восстановлен. Вычисляется по формуле, представленной ниже:

$$D = B(1 - N_{restored} / N_{tot}), \tag{4}$$

где B – бинарный флаг для процесса деобфускации, $B \in (0, 1)$; $N_{restored}$ – количество успешно восстановленных тестов; N_{tot} – общее число тестов.

Производительность O_p обфускатора (оценка замедления выполнения программного кода) рассчитывается по формуле

$$O_P = (T_{obf} - T_{orig}) / T_{orig}, \tag{5}$$

где T_{obf} – время выполнения обфусцированного программного кода (усредненное для нескольких итераций); T_{orig} – время выполнения исходного варианта программы (усредненное для нескольких итераций).

Цена «запутывания» O_r зависит от затрачиваемых ресурсов: время выполнения (как основной критерий производительности) и потребления оперативной памяти, необходимой для выполнения кода. Объем оперативной памяти влияет на цену «запутывания» потому, что в процессе обфускации часто добавляется «мертвый» код, дополнительные переменные, циклы, функции, а это увеличивает размер исполняемого кода, объем потребляемой памяти и негативно сказывается на времени выполнения программы, замедляя ее.

Значение O_r вычисляем по формуле

$$O_r = \alpha O_P + (1 - \alpha)(M_{obf} - M_{orig}) / M_{orig}, \tag{6}$$

здесь M_{obf} – необходимый объем оперативной памяти для размещения обфусцированного кода (усредненное для нескольких итераций); M_{orig} – объем потребляемой памяти исходного варианта программы (усредненное для нескольких итераций); α – условный коэффициент, отражающий важность ресурсов.

Время выполнения является основным критерием производительности, и замедление скорости выполнения программы на 60–70% уже является очень критичным. Таким образом, значение коэффициента α мы установили на уровне 0,7, имея в виду значение времени для работы с веб-приложениями [23].

Значение S вычисляется по формуле

$$S = S_{obf} / S_{orig}, \quad (7)$$

где S_{obf} – размер файла после обфускации (усредненно для разных режимов работы онлайн-обфускаторов); S_{orig} – размер файла до обфускации (усредненно для разных режимов работы онлайн-обфускаторов).

Технические возможности могут быть оценены при помощи индекса универсальности U, как взвешенная сумма учитываемых возможностей, которая рассчитывается по формуле

$$U = w_L L + w_N N + w_D D + w_A A, \quad (8)$$

где L (Language Support) – количество поддерживаемых языков; N (Number of modes) – количество режимов, используемых в обфускаторах; D (Dynamic Obfuscation Support) – коэффициент, отражающий применение дополнительных специализированных методов обфускации в значениях от 0 до 1: 0 – отсутствуют, 0,5 – базовые настройки и методы; 1 – используются специализированные методы/динамическая обфускация; A (Analysis Tools) – отражает наличие инструментов для отладки кода: $A = 0$ – не поддерживает, $A = 1$ – поддержка инструментов отладки обфусцированного кода); w_L, w_N, w_D, w_A – взвешенные коэффициенты, которые отражают веса отдельных характеристик при оценке общей универсальности обфускатора. Весовые коэффициенты выбраны на основе возможностей рассмотренных онлайн-обфускаторов.

Числовые значения взвешенных коэффициентов основаны на иерархии приоритетов возможностей обфускаторов. Коэффициент $w_L = 0{,}3$, так как поддержка нескольких языков программирования повышает универсальность и влияет на эффективность используемых методов обфускации; $w_N = 0{,}4$. Режимы у обфускаторов являются основным фактором технических возможностей, поскольку они определяют разнообразие применяемых методов для сокрытия исходного кода и напрямую влияют на читабельность и производительность. Таким образом, данный критерий является наиболее важным. Для критерия D весовой коэффициент $w_D = 0{,}2$, он имеет среднее значение, так как не все онлайн-обфускаторы поддерживают динамическую обфускацию, которая усложняет анализ кода. Для критерия A, обозначающего наличие инструментов для отладки программного кода, мы приняли $w_A = 0{,}1$, поскольку приоритетными в обфускаторах являются инструменты для запутывания кода, а не для отладки программы.

Для комплексной оценки эффективности обфускаторов на основе всех критериев, описанных выше, предлагается добавить интегральный показатель E – эффективность обфускатора, который учитывает все метрики (в соответствии с выражениями (1)–(8)) и помогает в поиске баланса между уровнем защиты исходного кода, производительностью, нагрузкой и ресурсами устройства:

$$E = \alpha_R R + \alpha_P P + \alpha_F F + \alpha_D D + \alpha_{O_P} O_P + \alpha_{O_r} O_r + \alpha_S S + \alpha_U U, \quad (9)$$

здесь α_i – вес критерия i.

Веса критериев α_i, общая сумма которых равна 1, выбраны на основе приоритетов их влияния на качество процесса обфускации: $\alpha_R = 0{,}20$; $\alpha_P = 0{,}15$; $\alpha_F = 0{,}15$; $\alpha_D = 0{,}15$; $\alpha_{OP} = 0{,}15$; $\alpha_{Or} = 0{,}10$; $\alpha_S = 0{,}05$; $\alpha_U = 0{,}05$.

Результаты исследований. Файл *test_1* – простой код с небольшим количеством строк, позволяющий оценить базовую функциональность обфускатора. Его размер составлял 98 байт,

а среднее время выполнения необфусцированного кода – 2,28 мс. Файл *test_2* содержал более сложный код, намеренно усложненный за счет дополнительных операций и увеличения количества итераций, что позволяло оценить влияние методов обфускации на производительность. Размер этого файла составлял 6833 байт, а среднее время выполнения необфусцированного кода для языка *JavaScript* – 38,26 мс., а на языке *PHP* – 112,38 мс. Использование файлов с разными характеристиками позволило оценить устойчивость обфускатора к различным типам кода и воздействие применяемых методов на производительность программы.

Для анализа выбраны следующие онлайн-обфускаторы для *JavaScript*-кода: *JavaScript Obfuscator Tool* (рассмотрено 2 режима), *EvalPacker Obfuscator* (рассмотрено 4 режима), *JavaScript Obfuscator* (рассмотрено 4 режима). Для *PHP*-кода для анализа выбраны следующие онлайн-обфускаторы: *PHP obfuscator* (рассмотрено 2 режима), *WB PHP Obfuscator* (рассмотрено 2 режима). Для последующего сравнения онлайн-обфускаторам присвоены свои уникальные имена, заданные в соответствии с порядком упоминания в тексте: соответственно *obfJS_1*, *obfJS_2*, *obfJS_3*, *obfPHP_1*, *obfPHP_2*.

В ходе исследования было установлено, что все рассматриваемые обфускаторы позволяют реализовать несколько режимов обфускации. Однако в обфускаторах *EvalPacker Obfuscator* и *JavaScript Obfuscator* некоторые из этих режимов не обеспечили сохранения функциональности программного кода после обфускации, что было выявлено при тестировании на обоих тестовых файлах. Более того, режимы обфускации в *EvalPacker Obfuscator*, *JavaScript Obfuscator* и *PHP Obfuscator* не выявили устойчивости тестовых файлов к декомпиляции. Обфусцированный код, полученный при использовании этих средств, был успешно восстановлен с помощью онлайн-деобфускаторов, что также было подтверждено тестированием на двух тестовых файлах.

Сравнения обфускаторов по таким критериям, как читабельность кода, среднее время выполнения программы, средний размер файла, для разных режимов рассматриваемых онлайн-обфускаторов приведены в табл. 1. Полученные оценки имеют особое значение для программного кода, написанного на *JavaScript* и *PHP*, поскольку они непосредственно влияют на эффективность обработки кода на сервере и, соответственно, на производительность веб-приложений.

Таблица 1. **Результаты сравнения онлайн-обфускаторов по определенным критериям**

Язык программирования	Обфускатор	Читабельность кода C	Среднее время выполнения T_{obf}, мс		Средний размер файлов S_{obf}, байт	
			Файл *test_1*	Файл *test_2*	Файл *test_1*	Файл *test_2*
JavaScript	*obfJS_1*	0,3	2,9	62	1473	6243
JavaScript	*obfJS_2*	0,2	1,45	22	4015	72048
JavaScript	*obfJS_3*	0,4	2,53	54	650	4779
PHP	*obfPHP_1*	0,5	0,00011	91,19	740	4466
PHP	*obfPHP_2*	0,4	0,0015	399,78	610	6043

В табл. 2 представлены данные, полученные после вычислений всех критериев эффективности, описанных выше, для каждого онлайн-обфускатора для языка программирования *JavaScript*, а в табл. 3 – для *PHP*. Каждое значение в табл. 2 и табл. 3 отражает результаты, полученные экспериментальным путем, и помогает определить наиболее эффективный обфускатор. Значение E_{avg} получено как среднее значение интегрального показателя E для двух тестируемых файлов, что позволяет более точно оценить эффективность обфускации с учетом различных характеристик кода.

Таблица 2. **Результаты количественного сравнения онлайн-обфускаторов для языка программирования *JavaScript***

Критерий	Обфускатор					
	obfJS_1		*obfJS_2*		*obfJS_3*	
	Файл *test_1*	Файл *test_2*	Файл *test_1*	Файл *test_1*	Файл *test_2*	Файл *test_1*
R, %	8,62	0,05	– 83	– 6,77	11	0,017
P, %	60	70	80	80	60	60
F, %	50	50	75	50	66	66
D	0	0,5	0,18	0,5	0,2	0,2
O_p, %	27	62	– 36	– 42	11	41
O_r, %	18,9	44,0	– 26,0	– 29,5	77,0	29,0
S	1,84	1,23	11,79	11,13	2,53	0,66
U	2,2		1,9		1,6	
E	0,53	0,17	– 0,32	– 0,11	0,1	0,18
E_{avg}	0,23		– 0,22		0,14	

Таким образом, наиболее эффективным из рассмотренных онлайн-обфускаторов для языка программирования *JavaScript* оказался *JavaScript Obfuscator Tool* (*obfJS_1*), обладающий множеством дополнительных технических возможностей и настроек, что позволяет существенно повышать уровень защиты кода.

Таблица 3. **Результаты количественного сравнения онлайн-обфускаторов для языка программирования *PHP***

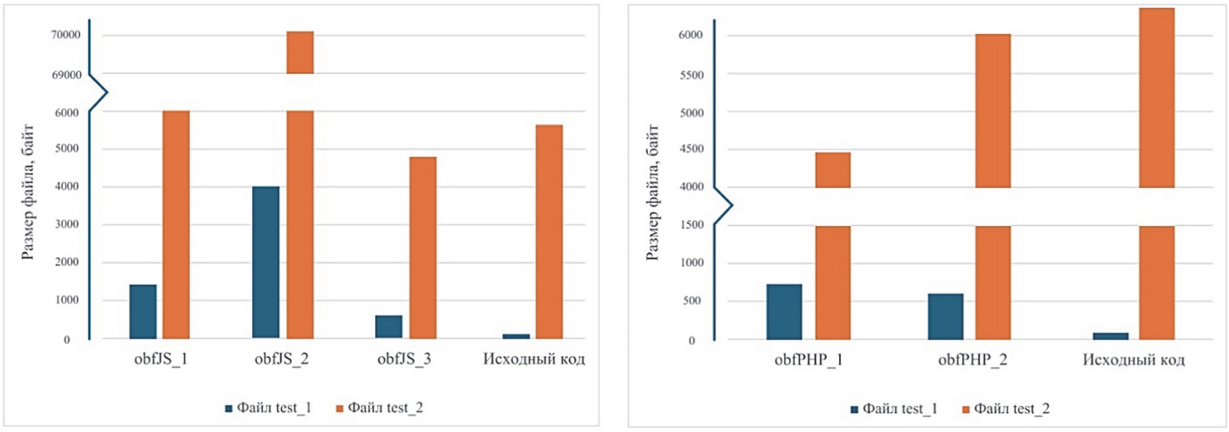
Критерий	Обфускатор			
	obfPHP_1		*obfPHP_2*	
	Файл *test_1*	Файл *test_2*	Файл *test_1*	Файл *test_2*
R, %	–4,98	–15,00	6,57	–2,28
P, %	50	50	60	70
F, %	100	100	50	50
D	1	1	0	0
O_p, %	10,0	3,6	52,5	355,0
O_r, %	–349	–1050	460	910
S	1,98	1,57	1,67	0,797
U	1,2		2,1	
E	1,00	1,72	– 0,34	–1,20
E_{avg}	1,36		– 0,77	

Следовательно, наиболее эффективным из рассмотренных онлайн-обфускаторов для языка программирования *PHP* был выявлен *PHP Obfuscator* (*obfPHP_1*).

Сравнение влияния метода обфускации на размер файла для языков *JavaScript* и *PHP* для выбранных обфускаторов представлено на рис. 7.

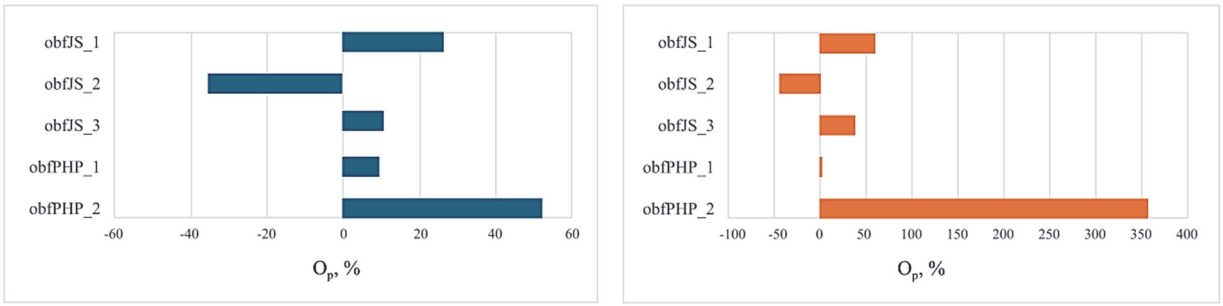
Сравнение влияния метода обфускации на производительность выполнения программного кода для выбранных обфускаторов для файлов *test_1* и *test_2* приведено на рис. 8.

Анализ представленных данных позволяет сделать вывод о том, что некоторые методы обфускации, реализованные различными инструментальными средствами, приводят к значительному увеличению размера исходного файла. Это, в свою очередь, часто негативно сказывается на скорости загрузки и общей производительности приложения. Кроме того, результаты исследования выявили, что при использовании некоторых методов обфускации наблюдается существенное увеличение времени выполнения кода. Данный факт является критическим недостатком, поскольку значительное замедление работы программы после обфускации делает ее практически непригодной для использования.



а *б*

Рис. 7. Сравнение влияния метода обфускации
на размер файла *test_1* (*а*) и *test_2* (*б*) для языков *JavaScript* и *PHP*



а *б*

Рис. 8. Сравнение влияния метода обфускации на производительность выполнения
файла *test_1* (*а*) и *test_2* (*б*) для языков *JavaScript* и *PHP*

Заключение. Результаты исследования подчеркивают острую необходимость в разработке надежных и эффективных методов обфускации и деобфускации программного кода. Анализ существующих онлайн-обфускаторов показал, что многие из них компрометируют либо размер файла, либо скорость выполнения программы, или устойчивость к декомпиляции. Это указывает на необходимость дальнейших исследований данной предметной области, а также на важность создания высокоэффективных решений, способных обеспечивать надежную защиту кода.

В рамках исследований также был разработан классификатор методов обфускации, который интегрирует различные подходы с учетом их эффективности и применимости. Он позволяет не только лучше понять существующие возможности, но и выбрать оптимальные методы для решения конкретных задач.

Список литературы

1. Пласковицкий В. А., Урбанович П. П. Защита программного обеспечения от несанкционированного использования и модификации методами обфускации // Труды БГТУ. 2011. № 6: Физико-математические науки и информатика. С. 173–176.

2. Алгоритмы и методы защиты программного кода на базе обфускации / А. В. Красов [и др.] // Информатика. Вычислительная техника и управление. 2020. Т. 12, № 1. С. 1–12.

3. Урбанович П. П. Защита информации методами криптографии, стеганографии и обфускации: учеб.-метод. пособие. Минск: БГТУ, 2016. 220 с.

4. Никольская К. Ю., Хлестов А. Д. Обфускация и методы защиты программных продуктов // Вестник УрФО. Безопасность в информационной сфере. 2015. Вып. 2 (16). С. 7–10.

5. Code Obfuscation: A Comprehensive Approach to Detection, Classification, and Ethical Challenges / T. Raitsis [et al.] // Algorithms. 2025. Vol. 18, no. 2. P. 54–74. DOI:10.3390/a18020054.

6. Assessment of Source Code Obfuscation Techniques / A. Viticchie [et al.] // 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM), Raleigh, NC, USA, 2016. P. 11–20. DOI: 10.48550/arXiv.1704.02307.

7. De Sutter B. A New Framework of Software Obfuscation Evaluation Criteria // arXiv preprint arXiv:2502.14093. 2025. DOI: 10.48550/arXiv.2502.14093

8. Herrera A. Optimizing away JavaScript obfuscation // 2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM). Adelaide, SA, Australia, 2020, pp. 215–220. DOI: 10.48550/arXiv.2009.09170.

9. From obfuscated to obvious: A comprehensive JavaScript deobfuscation tool for security analysis / D. Zhou [et al.] // arXiv preprint arXiv:2512.14070. 2025. DOI: 10.48550/arXiv.2512.14070.

10. Statically detecting JavaScript obfuscation and minification techniques in the wild / M. Moog [et al.] // Proceedings of the 2021 International Conference on Security and Privacy in Communication Networks (SecureComm). 2021. DOI: 10.1109/DSN48987.2021.00065.

11. Obfuscator.io // JavaScript obfuscator tool – Protect Your JS Code. URL: https:// obfuscator.io/ (дата обращения: 15.12.2025).

12. EvalPacker Obfuscator // Обфускаторы JavaScript кода. URL: https://tools.100zona.com/ evalpacker.html (дата обращения: 17.12.2024).

13. BeautifyTools.com // Online Javascript Obfuscator. URL: https://beautifytools.com/ javascript-obfuscator.php (дата обращения: 17.12.2024).

14. PHP Minify // PHP Obfuscator. URL: https://php-minify.com/php-obfuscator/ (дата обращения: 11.01.2025).

15. WB0.ru // PHP Obfuscator. URL: http://wb0.ru/phpobf.php (дата обращения: 11.01.2025).

16. CASCADE: LLM-powered JavaScript Deobfuscator at Google / S. Jiang [et al.] // arXiv preprint arXiv:2507.17691. 2025. DOI: 10.48550/arXiv.2507.17691.

17. Popa M. Techniques of program code obfuscation for secure software // Journal of Mobile, Embedded and Distributed Systems. 2011. Vol. 3, no. 4. P. 205–218.

18. Кантарович В. С., Урбанович П. П. Критерии эффективности алгоритмов обфускации и деобфускации программного кода и их сравнительный анализ на основе онлайн-обфускаторов // Беспилотные аппараты «БПЛА–2025»: сб. ст. II Междунар. форума по беспилотным аппаратам, Минск, 30 сент. – 2 окт. 2025 г.: в 2 ч. Минск: БГТУ, 2025. Ч. 1. С. 149–153.

19. Chen G., Jin X., Lin Z. JsDeObsBench: Measuring and Benchmarking LLMs for JavaScript Deobfuscation // Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security. 2025. P. 36–50. DOI: 10.48550/arXiv.2506.20170.

20. A Framework to Quantify the Quality of Source Code Obfuscation / H. Jin [et al.] // Appl. Sci. 2024. Vol. 14. P. 50–56. DOI:10.3390/app14125056.

21. A family of experiments to assess the effectiveness and efficiency of source code obfuscation techniques / M. Ceccato [et al.] // Empirical Software Engineering. 2014. Vol. 19, no. 3. P. 64–75. DOI: 10.1007/s10664-013-9248-x.

22. Пласковицкий В. А. Применение метрик программного обеспечения для оценки сложности исполняемого кода // Труды БГТУ. 2013. № 6: Физико-математические науки и информатика. С. 145–148.

23. Chen J., Shang W. An exploratory study of performance regression introducing code changes // Proceedings of the International Conference on Software Maintenance and Evolution (ICSME). Melbourne, Australia, 2017. P. 1–12.

References

1. Plaskovitskiy V. A., Urbanovich P. P. Protecting software from unauthorized use and modifications using obfuscation methods. *Trudy BGTU* [Proceedings of BSTU]. 2011, no. 6: Physics and Mathematics, pp. 173–176 (In Russian).

2. Krasov A. V., Radynskaya V. E., Zuyev I. P., Geraskina V. S., Karelsky P. V. Algorithms and method of protecting the software code on the basis of obfuscation. *Informatika. Vychislitel'naya tekhnika i upravleniye* [Computer Science. Computer Engineering and Control]. 2020, vol. 12, no. 1, pp. 1–12 (In Russian).

3. Urbanovich P. P. *Zashchita informatsii metodami kriptografii, steganografii i obfuskatsii* [Information protection by cryptography, steganography and obfuscation methods]. Minsk, BGTU Publ., 2016. 220 p. (In Russian).

4. Nikolskaya K. U., Hlestov A. D. Obfuscation and methods of protection software. *Vestnik UrFO. Bezopasnost' v informatsionnoy sfere*. 2015. Vol. 2, no. 16, pp. 7–10 (In Russian).

5. Raitsis T., Elgazari Y., Toibin G. E, Lurie Y., Mark S., Margalit O. Code Obfuscation: A Comprehensive Approach to Detection, Classification, and Ethical Challenges. *Algorithms*, 2025, vol. 18, no. 2, pp. 54–74. DOI: 10.3390/a18020054.

6. Viticchie A., Regano L., Torchiano M., Basile C., Ceccato M., Tonella P., Tiella R. Assessment of Source Code Obfuscation Techniques. *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, Raleigh, NC, USA, 2016, pp. 11–20. DOI: 10.48550/arXiv.1704.02307.

7. De Sutter B. A New Framework of Software Obfuscation Evaluation Criteria. *arXiv preprint arXiv*:*2502.14093*, 2025. DOI: 10.48550/arXiv.2502.14093.

8. Herrera A. Optimizing away JavaScript obfuscation. *2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. Adelaide, SA, Australia, 2020, pp. 215–220. DOI: 10.48550/arXiv.2009.09170.

9. Zhou D., Ying L., Chai H., Wang D. From obfuscated to obvious: A comprehensive JavaScript deobfuscation tool for security analysis. arXiv preprint arXiv:2512.14070. 2025. DOI: 10.48550/arXiv.2512.14070.

10. Moog M., Demmel M., Backes M., Fass A. Statically detecting JavaScript obfuscation and minification techniques in the wild. *Proceedings of the 2021 International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2021. DOI: 10.1109/ DSN48987.2021.00065.

11. Obfuscator.io. *JavaScript obfuscator tool – Protect Your JS Code*. Available at: https:// obfuscator.io/ (accessed 15.12.2025).

12. EvalPacker Obfuscator. *JavaScript code obfuscators*. Available at: https://tools.100zona.com/ evalpacker.html (accessed 17.12.2024).

13. BeautifyTools.com. *Online Javascript Obfuscator*. Available at: https://beautifytools.com/ javascript-obfuscator.php (accessed 17.12.2024).

14. PHP Minify. *PHP Obfuscator*. Available at: https://php-minify.com/php-obfuscator/ (accessed 11.01.2025).

15. WB0.ru. *PHP Obfuscator*. Available at: http://wb0.ru/phpobf.php (accessed 11.01.2025).

16. Jiang S., Kovuri P., Tao D., Tan Z. CASCADE: LLM-powered JavaScript Deobfuscator at Google. *arXiv preprint arXiv:2507.17691*, 2025. DOI: 10.48550/arXiv.2507.17691.

17. Popa M. Techniques of program code obfuscation for secure software. *Journal of Mobile, Embedded and Distributed Systems*, 2011, vol. 3, no. 4, pp. 205–218.

18. Kantarovich V. S., Urbanovich P. P. Efficiency criteria for software code obfuscation and deobfuscation algorithms and their comparative analysis based on online obfuscation tools. *Bespilotnyye apparaty "BPLA–2025": sbornik statey II Mezhdunarodnogo foruma po bespilotnym apparatam* [Unmanned aerial vehicles "UAV–2025": a collection of articles from the II International Forum on Unmanned Aircraft]. Minsk, September 30 – October 2, 2025: in 2 parts. Minsk, 2025, part 1, pp. 149–153 (In Russian).

19. Chen G., Jin X., Lin Z. JsDeObsBench: Measuring and Benchmarking LLMs for JavaScript Deobfuscation. *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, 2025, pp. 36–50. DOI: 10.48550/arXiv.2506.20170.

20. Jin H., Lee J., Yang S., Kim K., Lee D. H. A Framework to Quantify the Quality of Source Code Obfuscation. *Appl. Sci*., 2024, vol. 14, pp. 50–56. DOI: 10.3390/app14125056.

21. Ceccato M., Di Penta M., Falcarin P., Ricca F., Torchiano M., Tonella P. A family of experiments to assess the effectiveness and efficiency of source code obfuscation techniques. *Empirical Software Engineering*, 2014, vol. 19, no. 3, pp. 64–75. DOI: 10.1007/s10664-013-9248-x.

22. Plaskovitskiy V. A. Using software metrics to assess the complexity of executable code. *Trudy BGTU* [Proceedings of BSTU]. 2013, no. 6: Physics and Mathematics, pp. 145–148 (In Russian).

23. Chen J., Shang W. An exploratory study of performance regression introducing code changes. *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. Melbourne, Australia, 2017, pp. 1–12.

Информация об авторах

Кантарович Виктория Сергеевна – старший преподаватель кафедры информационных систем и технологий. Белорусский государственный технологический университет (ул. Свердлова, 13а, 220006, г. Минск, Республика Беларусь). E-mail: kantarovich@belstu.by. ORCID: 0009-0002-0586-8506.

Урбанович Павел Павлович – доктор технических наук, профессор, профессор кафедры информационных систем и технологий. Белорусский государственный технологический университет (ул. Свердлова, 13а, г. Минск, Республика Беларусь)[1], приглашенный профессор (ал. Рацлавицке, 14, Люблин 20-950, Польша)[2]. E-mail: p.urbanovich@belstu.by. ORCID: 0000-0003-2825-4777.

Information about the authors

Kantarovich Victoria Sergeevna – Senior Lecturer, the Department of Information Systems and Technologies. Belarusian State Technological University (13a Sverdlova str., 220006, Minsk, Republic of Belarus). E-mail: kantarovich.v@gmail.com. ORCID: 0009-0002-0586-8506.

Urbanovich Pavel Pavlovich – DSc (Engineering), Professor, Professor, the Department of Information Systems and Technologies. Belarusian State Technological University (13a Sverdlova str., 220006, Minsk, Republic of Belarus)[1], visiting Professor (14 Racławickie Alley, Lublin 20-950, Poland)[2]. E-mail: p.urbanovich@belstu.by. ORCID: 0000-0003-2825-4777.