УДК 004.056 **DOI** 10.15622/ia.23.3.3

П.Д. Борисов, Ю.В. Косолапов

СПОСОБ КОЛИЧЕСТВЕННОГО СРАВНЕНИЯ ОБФУСЦИРУЮЩИХ ПРЕОБРАЗОВАНИЙ

Борисов П.Д., Косолапов Ю.В. **Способ количественного сравнения обфусцирующих преобразований.**

Аннотация. В работе рассматривается задача количественного сравнения эффективности и стойкости практически применяемых обфусцирующих преобразований программного кода. Предлагается способ нахождения эффективности и стойкости преобразований путем вычисления «понятности» соответственно обфусцированной и деобфусцированной версий программы. В качестве меры понятности программы предлагается использовать похожесть этой программы на аппроксимацию ее «самой понятной» версии. На основе предложенного способа построена модель оценки эффективности и стойкости, основными элементами которой являются: набор исследуемых обфусцирующих преобразований, функция похожести, способ аппроксимации самой понятной версии программы и деобфускатор. Для реализации этой модели 1) выбраны обфусцирующие преобразования, предоставляемые обфускатором Hikari; 2) методами машинного обучения по статическим характеристикам программ из наборов CoreUtils, PolyBench и HashCat построено 8 функций похожести; 3) в качестве аппроксимации самой понятной версии программы выбрана наименьшая по размеру версия программы, найденная среди версий, полученных с помощью опций оптимизации компиляторов GCC, Clang и AOCC; 4) построена и реализована схема деобфускации программ на основе оптимизирующего компилятора из состава LLVM. В работе экспериментально получены результаты оценки эффективности и стойкости для последовательностей преобразований длины один, два и три. Эти результаты показали согласованность с результатами независимых оценок эффективности и стойкости, полученных другими способами. В частности, получено, что наибольшую эффективность и стойкость демонстрируют последовательности преобразований, начинающиеся с преобразований графа потока управления, а наименьшей стойкостью и эффективностью – как правило, последовательности, не содержащие таких преобразований.

Ключевые слова: обфускация, исполняемый код, эффективность, стойкость, похожесть.

1. Введение. Обфусцирующие преобразования обычно применяются для затруднения исследования программ с целью сокрытия алгоритмов и/или данных от аналитика, где под аналитиком может подразумеваться как человек, так и программное средство

автоматического анализа. Такие преобразования применяются как к программам, компилируемым в инструкции целевого процессора (машинный код), так и к интерпретируемым программам (байт-код или промежуточное представление). Способ защиты программ на основе обфускации считается одним из основных в модели MATE (Man At The End), когда нет возможности ограничить аналитика в средствах и методах анализа программного кода [1, 2]. Также обфусцирующие преобразования используются как диверсифицирующие – для защиты от эксплуатации возможных уязвимостей [3].

Исследования в области обфускации ведутся с 70-х годов 20-го века и к настоящему времени выделяются два направления исследований: криптографическое направление (обфускация с доказуемой стойкостью) и направление системного программирования (обфускация с эвристическим обоснованием стойкости) [4]. Криптографами доказано, что не существует запутывающего преобразования, которое бы было стойким в рамках модели «виртуального черного ящика» для всех программ [5]. Не строго говоря, это означает, что для любого запутывающего преобразования найдется программа, по обфусцированному коду которой аналитик может получить больше информации, чем по набору входных и соответствующих выходных данных этой программы. При этом для отдельных классов программ доказуемо стойкая обфускация в модели «виртуального черного ящика» существует [6], а также существуют стойкие запутывающие преобразования для отдельных классов программ в некоторых вариациях этой модели (обзор [4]). Особое место в криптографии занимает неразличимая обфускация, которая позволяет строить стойкие функциональные схемы шифрования [7]. Однако применение неразличимой обфускации на практике затруднено из-за роста размера программы, и пока неясно, насколько такая обфускация затрудняет анализ программ. В целом, имеющиеся результаты в области теоретической (криптографической) обфускации пока далеки от реального применения при защите программ [8]. Напротив, существующие свободно распространяемые и коммерческие средства защиты программ предлагают набор методов обфускации кода, которые получены исследователями, как правило, в области системного программирования. Однако, как замечено в [4], исследования в этой области сравнимы с «состязанием щита и меча»: для используемых на практике способов запутывания кода рано или поздно находятся способы деобфускации (распутывания). Например, обфускация с помощью смешанной булевой и целочисленной арифметики, предложенная в [9], долгое время считалась надежным способом затруднения процесса анализа программ.

Результаты работ [10 – 13] в области упрощения линейных выражений со смешанной булевой и целочисленной арифметикой значительно ослабили позиции этого вида обфускации. В частности, в [13] было показано, что для упрощения могут эффективно применяться методы, используемые при декодировании линейных кодов. Тем не менее, несмотря на отсутствие гарантий стойкости [8], эвристические способы запутывания кода в совокупности с методами противодействия отладке и дизассемблирования могут обеспечить уровень защиты от анализа, достаточный для достижения целей пользователями этих методов [14]: например, анализ программного кода компьютерной игры может потребовать много времени, в течение которого разработчик игры успеет реализовать достаточное для получения прибыли количество лицензий.

В настоящей работе объектом исследования является множество программ $\mathcal{P}$, компилируемых в инструкции целевого процессора. Таким образом, в рамках модели MATE, аналитику для исследования доступен бинарный исполняемый файл программы. Для программ из $\mathcal{P}$ запутывающие преобразования могут применяться на разных уровнях представления: уровне исходного кода, уровне промежуточного представления, уровне машинного кода. Обозначим через

$$\mathcal{O} = \{o_1, ..., o_m\}$$

набор, далее называемый *базовым*, который может состоять из обфусцирующих преобразований, применяемых на разных уровнях представления. В $\mathcal{O}$, например, могут входить такие преобразования, как добавление недостижимого кода, кодирование констант, замена инструкций условного перехода косвенными переходами и т.п. Преобразования, применяемые на одном уровне представления, обычно неприменимы на другом уровне представления, поэтому среди множества всех возможных последовательностей $\mathcal{O}^*$ выделим множество $\mathcal{O}_0^*$ всех *допустимых* последовательностей обфусцирующих преобразований. Аналогично множество всех допустимых последовательностей длины n обозначим $\mathcal{O}_0^n$. Очевидно, что $\mathcal{O}_0^n \subset \mathcal{O}^n$ и $\mathcal{O}_0^* \subset \mathcal{O}^*$. Для $P \in \mathcal{P}$ и $o \in \mathcal{O}$ через $o(P)$ обозначим программу, которая получена по P применением o.

Для пользователя средств обфускации одной из важных задач является выбор для программы $P \in \mathcal{P}$ такой последовательности

$$\mathsf{Obf} = (o_{i_1}, o_{i_2}, ..., o_{i_t}) \in \mathcal{O}_0^*, \tag{1}$$

которая бы при небольшом t «на приемлемом уровне» затрудняла исследование программы

$$o_{i_t}(...(o_{i_1}(P))...)$$

аналитиком. Так как для эвристически стойких обфускаций, как отмечалось выше, отсутствует гарантия стойкости, то, с учетом $|\mathcal{O}_0^*| = \infty$, под обфусцирующей последовательностью «приемлемого уровня» представляется допустимым понимать ту, которая обладает «наилучшими» характеристиками среди последовательностей заданного набора $\mathcal{Q} \subset \mathcal{O}_0^*$ конечной мощности. Поэтому для поиска такой последовательности Obf необходим способ *количественного сравнения* результатов применения обфускаций из $\mathcal{Q}$.

2. Обзор известных результатов. Исследования в области защиты программ представлены достаточно большим число работ, при этом большая часть работ посвящена вопросам защиты на основе обфускации кода [8]. Считается, что первой работой, систематизировавшей подходы к обфускации исходного кода и оценке качества обфускации, является работа [15]. В [15] *качество* обфусцирующего преобразования определяется как интегральный показатель, учитывающий *эффективность* (potensy в [15]) и *стойкость* (resilience в [15]), где под эффективностью преобразования предложено понимать степень запутанности кода, а под стойкостью – степень запутанности кода после применения средства распутывания (деобфускатора).

В [15] в качестве метрик *эффективности* предлагаются, в частности, метрики из области программной инженерии, такие, как метрики Холстеда. Отметим, что такие метрики, как правило, строятся для уровня исходного кода, и их применение затруднительно для других уровней представления кода. Другой подход рассматривается в [16], где для количественной оценки эффективности предложено использовать степень сжимаемости обфусцированного исходного кода, которая в [16] выбрана в качестве аппроксимации сложности программы по Колмогорову. Результаты экспериментов [16], представленные для преобразований исходного кода программ на языке Java, показали, в частности, что обфусцирующие преобразования над данными оказываются более эффективными, чем преобразования графа потока управления.

При оценке *стойкости* преобразований, если не учитывать оценку, полученную путем опроса пула аналитиков-программистов, сложность заключается в выборе модели деобфускатора. В [17] в качестве такой

модели предложено использовать символьный интерпретатор. Этот подход получил дальнейшее развитие в ряде работ. В частности, в [18] набор $\mathcal{O}$, состоящий из девяти преобразований уровня исходного кода, предоставляемых обфускатором Tigress [19] для программ на языке C, поделен на четыре категории, с целью исследования того, как влияет *порядок* применения обфускаций разных категорий (рассматривались все возможные варианты для трех и четырех категорий) на *время* символьного исполнения запутанного тестового кода (в тестовой программе оценивалось время нахождения пароля заданной длины). Таким образом, сравнение стойкости обфусцирующих преобразований в [18] выполняется на основе времени нахождения пароля методами символьной интерпретации. При этом преобразования внутри каждой категории применялись все сразу и в фиксированном порядке, то есть влияние перестановок и комбинаций преобразований внутри каждой категории не исследовалось. Результаты [18] позволяют пользователям Tigress априори выбрать для программы P подходящую последовательность обфускаций. Однако, проверка того, насколько выбранная последовательность лучше другой для конкретной программы P затруднена тем, что в [18] для сравнения результатов используется символьная интерпретация, трудно применимая к большим программам [8]. По этой же причине затруднено применение и подходов, предложенных в [20 – 22], где характеристики символьной интерпретации (время интерпретации, объем затраченной оперативной памяти, степень покрытия кода и т.п.) используются для оценки как эффективности, так и стойкости преобразований.

В [8] отмечено, что несмотря на большое количество работ в области обфускации и деобфускации программ, важной и актуальной является задача развития и совершенствования подходов к *количественной* оценке эффективности и стойкости методов защиты. В частности, эти подходы должны учитывать широкое разнообразие техник, применяемых аналитиками при исследовании программ (такие техники включают отладку, дизассемблирование, символьное исполнение, запуск в «песочнице», применение средств тестирования кода и т.п.). Количественная оценка эффективности и стойкости отдельных преобразований и/или их комбинаций позволит подстроить систему защиты под модель аналитика (выбор системы защиты, адекватной возможностям аналитика, является одним из перспективных направлений исследований, по мнению авторов работы [14]).

3. Постановка задачи. В настоящей работе ставится задача разработки способа сравнения эффективности и стойкости обфусцирущих преобразований программ на основе *статических* характеристик

бинарного исполняемого файла, то есть таких характеристик, которые могли бы быть получены без запуска или символьной интерпретации программы. Использование характеристик именно исполняемого файла, с одной стороны, соответствует модели MATE, а с другой стороны, исключает зависимость предлагаемого метода от применяемых обфусцирующих преобразований и уровней их применения. Актуальность рассмотрения именно исполняемых файлов также подтверждается наибольшим интересом исследователей к обфускации программ, компилируемым в машинный код, что обосновано большей свободой при построении защиты, чем в случае с интерпретируемыми программами [8]. Отметим, что оценка влияния обфускации на понимание программы *в целом*, а не на понимание отдельной локальной функции, представляется наиболее естественной, так как стойкость к пониманию на локальном уровне не влечет стойкости к пониманию всей программы [8].

4. «Понятность» программы. Обфусцирующие преобразования относятся к преобразованиям, сохраняющим семантику программ, к которым принадлежат и оптимизирующие преобразования. Для оптимизирующих преобразований выбор одного из двух Opt_1 и Opt_2 для программы P может осуществляться, например, так: если размер программы $\mathsf{Opt}_1(P)$ меньше минимума размеров P и $\mathsf{Opt}_2(P)$, то наилучшим считается Opt_1; аналогично выбор может быть сделан в пользу Opt_2 (в остальных случаях преобразования Opt_1 и Opt_2 не могут рассматриваться как оптимизирующие для P). В случае аналогичного выбора одного из двух обфусцирующих преобразований (Obf_1 или Obf_2) следует определить *количественную характеристику* программ (бинарных исполняемых файлов), описывающую «понятность» этих программ для аналитика, так как обфусцирующие преобразования направлены на затруднение именно понимания программы (кода и/или данных) [23].

Для введения такой характеристики рассмотрим множество $\mathcal{P}_P$ программ, полученных из программы P преобразованиями, сохраняющими семантику. Пусть $P_0 \in \mathcal{P}$ – самая «понимаемая» версия программы P. В качестве P_0, например, можно рассматривать программу наименьшего размера, так как считается, что чем меньше размер программы, тем она «читабельнее» [24]. Другой подход к выбору P_0 может быть основан на сложности символьной интерпретации машинного кода [18, 25]. В таком случае в качестве P_0 может рассматриваться версия с наименьшим временем символьного исполнения или с наименьшим объемом памяти, затраченной при символьном исполнении. В [21] такой подход обоснован тем, что символьное исполнение можно

рассматривать как модель динамического исследования программы аналитиком-человеком. Однако этот подход имеет ограничения: символьное исполнение осуществимо за обозримое время, как правило, только для небольших программ. Понятность программы P будем рассматривать, как величину похожести программы P на P_0. Для этого рассмотрим функцию $s : \mathcal{P} \times \mathcal{P} \to [0, 1]$ – функцию похожести программ, где 0 соответствует наименьшей степени похожести, а 1 – наибольшей. В этом случае для фиксированных $P_0 \in \mathcal{P}_P$ и s под понятностью программы $P \in \mathcal{P}_P$ можно подразумевать величину $s(P, P_0)$, как степень похожести на самую «понимаемую» версию программы. Однако, как нахождение самой короткой программы, так и нахождение программы с наименьшим временем символьного исполнения, являются вычислительно сложными задачами. Поэтому здесь вместо P_0 предлагается использовать ее *аппроксимацию* $\mathsf{A}(P_0)$, найденную по P. (Аналогичный подход применяется в [16], где при количественной оценке эффективности обфусцирующих преобразований вместо невычислимой колмогоровской сложности программы применяется ее аппроксимация результатом сжатия программы.) Заметим, что аппроксимация $\mathsf{A}(P_0)$ наиболее понятной версии P_0 программы P – это то, что недоступно аналитику, но если бы было доступно, он смог бы это эффективно понять. В качестве аппроксимации $\mathsf{A}(P_0)$ может быть выбрана, например, наименьшая по размеру версия программы P, полученная с помощью доступного набора оптимизирующих преобразований. Представляется, что аппроксимацию можно строить также на основе характеристик символьной интерпретации: например, в качестве $\mathsf{A}(P_0)$ может быть выбрана версия программы (например, среди версий, полученных с помощью разных компиляторов и разных опций компиляции) с наименьшим временем символьной интерпретации или наименьшим использованием памяти при символьной интерпретации.

Понятностью $P \in \mathcal{P}_P$ при фиксированных s и $\mathsf{A}(P_0) \in \mathcal{P}_P$ назовем величину

$$\mathsf{comp}(P) = s(P, \mathsf{A}(P_0)) \in [0, 1]. \tag{2}$$

Таким образом, для зафиксированного алгоритма A понятность программы P зависит от выбора функции похожести s, определенной на парах программ. Известно, что построение функции похожести для программ является нетривиальной задачей и, как правило, зависит от решаемой задачи [26]. Например, если $s = s^1$, где $s^1(P, Q) = 1$ только при совпадении файлов P и Q, и равно нулю в остальных случаях, то

понятность большинства программ будет равна нулю. Другим крайним случаем функции похожести является функция $s = s^2$, такая, что $s^2(P, Q) = 1$, если P и Q на одинаковых входных данных возвращают одинаковые результаты, и равно нулю в остальных случаях. В этом случае понятность большинства программ (всех программ, если A сохраняет функциональность) будет максимальна. Однако функции s^1 и s^2, как будет следовать из определений эффективности и стойкости обфусцирующих преобразований (определения (3)) и (4) в разделе 5), малопригодны для решения поставленной в разделе 3 задачи: при использовании s^1 все обфусцирующие преобразования в равной степени будут эффективными и стойкими, а при использовании s^2 ни одно из обфусцирующих преобразований не будет ни эффективным, ни стойким. Однако практические исследования показывают (например, [27, 28]), что разные запутывающие преобразования могут по-разному влиять на время, затрачиваемое аналитиком для понимания обфусцированной программы. Поэтому для решаемой задачи актуально построение таких функций похожести, значения которых бы коррелировали с результатами практических исследований эффективности и стойкости обфусцирующих преобразований. При этом для рассматриваемой задачи важную роль играет то, как такие функции могут оценить степень похожести программ, реализующих один и тот же алгоритм, и в меньшей степени важно то, как эти функции оценивают степень похожести разных по функционалу программ. В [29] авторами предпринята попытка построения такой функции на основе методов машинного обучения с использованием множества известных функций похожести, определенных на парах бинарных файлов. Эта функция используется в настоящей работе как конкретная реализация абстрактной функции похожести, используемой в определении понятности программы $\mathsf{comp}(P)$ (более подробное описание функции приведено в разделе 5.2). Заметим, что значение $\mathsf{comp}(P)$, когда к P не применяются преобразования, как представляется, может использоваться, например, в программной инженерии для оценки качества программного кода: чем больше $\mathsf{comp}(P)$, тем качество выше, и наоборот.

5. Модель оценки эффективности и стойкости обфускации. С помощью характеристики (2) определим эффективность e обфусцирующего преобразования $\mathsf{Obf} \in \mathcal{O}_0^*$, примененного к программе P, а также стойкость r этого преобразования по отношению к деобфускатору D следующим образом:

$$e(\mathsf{Obf}, P) = 1 - \mathsf{comp}(\mathsf{Obf}(P)) = 1 - s(\mathsf{Obf}(P), \mathsf{A}(P_0)), \quad (3)$$

$$r(\mathsf{D}, \mathsf{Obf}, P) = 1 - \mathsf{comp}(\mathsf{D}(\mathsf{Obf}(P))) = 1 - s(\mathsf{D}(\mathsf{Obf}(P)), \mathsf{A}(P_0)). \quad (4)$$

В рамках определений (3) и (4), задача выбора наиболее эффективного обфусцирующего преобразования для P решается так: Obf_1 эффективнее Obf_2, если

$$e(\mathsf{Obf}_1, P) > \max\{1 - \mathsf{comp}(P), e(\mathsf{Obf}_2, P)\}.$$

Задача выбора среди этих же преобразований наиболее стойкого по отношению к деобфускатору D решается так: Obf_1 является D-устойчивее Obf_2, если

$$r(\mathsf{D}, \mathsf{Obf}_1, P) > r(\mathsf{D}, \mathsf{Obf}_2, P).$$

Набор

$$\mathcal{M} = (\mathcal{O}, \mathcal{Q} \subset \mathcal{O}_0^*, s, \mathsf{D}, \mathsf{A}) \quad (5)$$

с определенными в соответствии с (3) и (4) функциями e и r назовем *моделью оценки эффективности и стойкости обфусцирующих преобразований*. Для реализации этой модели необходимо зафиксировать множество $\mathcal{O}$, выбрать функцию похожести s исполняемых файлов программ, деобфускатор D и способ аппроксимирования A самой понятной программы $P_0 \in \mathcal{P}_P$ для каждого P.

5.1. Выбор множеств $\mathcal{O}$ и $\mathcal{Q}$. В качестве множества $\mathcal{O}$ в настоящей работе выбрано множество преобразований, предоставляемых свободно распространяемым обфускатором Hikari [30], основанным на обфускаторе Obfuscator-LLVM [31]. Обозначения и соответствующие описания выбранных преобразований представлены в таблице 1. В последнем столбце указан тип преобразования, который выбран в соответствии с определениями типов преобразований, введенных в [18] для обфускатора Tigress: A – абстрактные преобразования, C – преобразования графа потока управления, D – кодирование данных и арифметических операций. Для $o \in \mathcal{O}$ тип этого преобразования будем обозначать $\theta(o)$, а тип последовательности Obf вида (1) естественно определить так: $\theta(\mathsf{Obf}) = (\theta(o_{i_1}), ..., \theta(o_{i_t}))$. Например, тип последовательности $\mathsf{Obf} = (\mathsf{ind}, \mathsf{fcw}, \mathsf{sbb})$ будет (C,C,A). Для простоты записи тип последовательности будем указывать без скобок и запятых.

То есть в рассмотренном примере тип будет CCA. Множество {C,A,D} обозначим $\theta(\mathcal{O})$, как множество типов базовых преобразований из $\mathcal{O}$.

Таблица 1. Набор $\mathcal{O}$ преобразований, предоставляемых компилятором Hikari для программ на языке C

	o	**Описание преобразования**	**Тип**
$\mathcal{O}$	bcf	встраивание непрозрачных предикатов	C
	cff	сглаживания графа потока управления	C
	enc	кодирование статических строк	D
	fcw	создание фиктивных функций-прокси	A
	ind	замена инструкций ветвления косвенными переходами	C
	sbb	разбиение базовых блоков	A
	sub	замена инструкций эквивалентными	D
	all	применение всех обфусцирущих преобразований	–

Отметим, что все выбранные преобразования из $\mathcal{O}$ применяются на уровне промежуточного представления (на уровне IR-кода), однако порядок применения преобразований не является произвольным, поэтому $\mathcal{O}_0^* \neq \mathcal{O}^*$. При нахождении наиболее эффективного и/или стойкого обфусцирующего преобразования $\mathsf{Obf} \in \mathcal{Q}$ естественно предполагать, что выбор осуществляется среди последовательностей вида (1) одной и той же длины t. Однако перебор всех $|\mathcal{O}|^t$ последовательностей длины t уже при небольших значениях t может потребовать значительных вычислительных ресурсов. В настоящей работе полный перебор возможных последовательностей осуществляется для $t = 1, 2, 3$. Дополнительно рассматривается последовательность all длины 7, в которой применяются все преобразования из набора $\mathcal{O}$. Преобразование all добавлено в $\mathcal{Q}$ как референсное, так как априорно ожидается, что это преобразование является наиболее эффективным и стойким, поскольку задействованы все преобразования из $\mathcal{O}$ (хотя и в фиксированном порядке).

5.2. Выбор функции похожести. Разнообразие подходов к сравнению бинарных данных велико (обзор [32]). Для реализации модели $\mathcal{M}$ может подойти практически любая функция похожести, определенная на парах бинарных файлов. Эти функции могут как учитывать то, что входными данными являются исполняемые файлы (учитывается структура и характер данных в файле – *специализированные* характеристики), так и не учитывать (учитываются *универсальные* для любых данных характеристики). В первом случае примерами могут быть функции сравнения графов потока управления, полученных с помощью дизассемблера [34], или функции на основе характеристик символьного исполнения кода [18, 21, 22]. Во втором случае примерами функций

похожести являются функции на основе нечетких хеш-функций [33], а также функции на основе алгоритмов сжатия [16]. В настоящей работе используется разработанный авторами в [29] подход *статического* вычисления похожести файлов с использованием методов машинного обучения на основе функций похожести, вычисляемых от статических универсальных и специализированных характеристик бинарных исполняемых файлов. Выбор в пользу статических характеристик сделан по причине применимости этого подхода к программам любого размера и сложности. Обозначение выбранных функций и их краткое описание дано в таблице 2.

Таблица 2. Статические характеристики похожести программ P и Q

f	**Значение характеристики**	$Accuracy(s_f)$	$Impact(f)$
sdhash	результат сравнения хеш-значений функции sdhash для P и Q	0.50	0.01
tlsh	результат сравнения хеш-значений функции tlsh с учетом длины P и Q	0.68	0.06
tlshxlen	результат сравнения хеш-значений функции tlsh без учета длины P и Q	0.53	0.02
lzma	коэффициент относительного сжатия алгоритмом lzma программ P и Q	0.66	0.03
bz2	коэффициент относительного сжатия алгоритмом bz2 программ P и Q	0.65	0.02
deflate	коэффициент относительного сжатия алгоритмом deflate программ P и Q	0.68	0.03
myersdiff	расстояние Майерса для P и Q	0.65	0.05
levenshteindiff	расстояние Левенштейна для P и Q	0.67	0.06
idadiff	результат сравнения утилитой BinDiff графов потока управления P и Q, полученных дизассемблером IDA Pro	0.66	0.12
idadiffc	произведение idadiff на степень уверенности, возвращаемую BinDiff	0.69	0.14
ghydradiff	результат сравнения утилитой BinDiff графов потока управления P и Q, полученных дизассемблером Ghydra	0.64	0.11
ghydradiffc	произведение ghydradiff на степень уверенности, возвращаемую BinDiff	0.68	0.14
ninjadiff	результат сравнения утилитой BinDiff графов потока управления P и Q, полученных дизассемблером Ninja	0.56	0.09
ninjadiffc	произведение ninjadiff на степень уверенности, возвращаемую BinDiff	0.66	0.12

Более подробное описание способов вычисления значений этих функций можно найти в [29]. Отметим, что здесь, в отличие [29], используются четырнадцать функций, а не пятнадцать, так как

результаты [29] показывают, что вклад значений хеш-функции ssdeep практически не влияет на точность классификатора. Также отметим, что в [29] для функции похожести, как некоторого аналога «расстояния» между программами, предельным значениям 0 и 1 поставлено соответствие, обратное принятому в настоящей работе: нулю соответствует совпадение программ, а единице – различие. Здесь же для удобства определения понятности программы применена инверсия предельных значений: чем более программа P похожа на $\mathsf{A}(P_0)$, тем более она понятна. В таблице 2 универсальные характеристики (первые восемь) отделены от специализированных двойной линией. В качестве методов машинного обучения использованы классификаторы, описанные в таблице 3.

Таблица 3. Классификаторы, лежащие в основе соответствующих функций похожести

c	**Описание классификатора**	$Accuracy(s_c)$
RFC	метод «случайный лес»	$0,999$
KNC	метод K ближайших соседей	$0,996$
SVC	метод опорных векторов	$0,985$
DTC	деревья решений	$0,974$
LGC	линейная регрессия	$0,966$
LDAC	линейный дискриминантный анализ	$0,962$
GNBC	наивная байесовская классификация	$0,716$
NNC	нейронная сеть с двумя скрытыми слоями по 10 и 100 нейронов соответственно	$0,962$

Далее функцию похожести, построенную на основе классификатора c из таблицы 3 будем обозначать s_c. В случае, когда классификатор строится с помощью только одной характеристики f из таблицы 2, соответствующую функцию похожести будем обозначать s_f.

5.3. Выбор деобфускатора. В рамках модели MATE, когда аналитику доступен бинарный исполняемый файл, деобфускатор D, по отношению к которому оценивается стойкость обфусцирующих преобразований, должен принимать на вход машинный код. Так как оценка стойкости в соответствии с (4) предполагает сравнение деобфусцированной версии $\mathsf{D}(\mathsf{Obf}(P))$ с аппроксимацией $\mathsf{A}(P_0)$, то выходом деобфускатора D должен быть также бинарный исполняемый файл. Такую возможность предоставляет средство SATURN [35], авторы которого позиционируют его как «state-of-the-art» деобфускатор бинарных исполняемых файлов. Идея, лежащая в основе этого деобфускатора заключается в получении с помощью дизассемблера IDA Pro представления программы в виде графа (включает функции, их

программный код, глобальные переменные/данные и т.д.), трансляции полученного представления в промежуточное представление LLVM с помощью библиотеки Remill из состава McSema [36], применении деобфусцирующих преобразований и компиляции полученного кода в бинарный исполняемый файл. При этом в основе операций деобфускации лежат *оптимизирующие* преобразования. Как отмечено в [25], «трансляция машинного кода» до IR-кода с помощью Remill, его оптимизация с последующей трансляцией обратно в машинный код приводит к тому, что полученная программа представляет собой *интерпретацию* машинных команд исходной программы с помощью последовательностей также машинных команд. Такая трансформация приводит к увеличению программного кода и, как следствие, замедлению работы программы. В [35] применяются оптимизирующие техники для снижения эффекта «интерпретации», однако трудность независимой оценки работы деобфускатора SATURN заключается в отсутствии опубликованного кода этого средства.

Отметим, что идея использования оптимизирующих преобразований, как упрощающих или деобфусцирующих, выдвигалась, помимо работы [35], также в других работах, в частности, в [37, 38, 25]. В настоящее время перечень средств, оптимизирующих непосредственно машинный код, нет так много. Одним из немногих таких оптимизаторов является средство BOLT [39] и его усовершенствованная версия VESPA [40]. Оптимизирующие (упрощающие) преобразования этих средств заключаются в основном в упорядочивании базовых блоков и функций программы таким образом, чтобы вызываемые «по соседству» функции находились в одной странице кода. Это снижает время доступа к инструкциям, так как в кэш процессора код загружается страницами. Однако других упрощений кода (например, удаление недостижимого и мусорного кода, упрощение арифметических выражений и т.п.) не производится, поэтому оптимизаторы BOLT и VESPA вряд ли могут претендовать на роль деобфускатора – средства, упрощающего понимание программы.

В настоящей работе на основе LLVM построена схема получения деобфусцированного кода, показанная на рисунке 1. Идея, лежащая в основе этой схемы состоит в применении оптимизирующих преобразований к обфусцированному IR-коду. Такой подход, с одной стороны, исключает интерпретацию машинных инструкций машинными инструкциями, как это происходит при «трансляции машинного кода» в IR-код, с другой – требует только указания дополнительных опций компилятору LLVM.

Рис. 1. Модель генерации программ P, $\mathsf{Obf}(P)$, и $\mathsf{D}(\mathsf{Obf}(P))$ с помощью LLVM

5.4. Выбор подхода аппроксимирования. В разделе 4 отмечено, что аппроксимацией $\mathsf{A}(P_0)$ самой понятной версии P_0 для программы P можно считать, например, 1) аппроксимацию наименьшей по размеру версии, или 2) аппроксимацию версии с наименьшим временем символьного исполнения. Повторим, что эвристически первый подход может быть обоснован тем, что короткий код считается наиболее читабельным [24]; второй же подход может быть обоснован наличием опубликованных работ, где обфусцирующие преобразования оцениваются при помощи характеристик символьного исполнения [18, 25]. Однако недостатком второго подхода является то, что при символьном исполнении даже небольших программ с малым числом входных параметров, как замечено в [21], затрачиваются значительные вычислительные ресурсы (процессорное время, объем оперативной памяти). Это может препятствовать нахождению $\mathsf{A}(P_0)$ для больших программ, и программ, имеющих большое число входных параметров. Нахождение же $\mathsf{A}(P_0)$, как аппроксимации самой короткой версии программы P, можно эффективно реализовать, например, следующим образом: 1) скомпилировать P с помощью *разных компиляторов*, для каждого из которых использовать *доступное множество опций оптимизации*; 2) среди полученных версий программы P выбрать программу с наименьшим размером; это и будет $\mathsf{A}(P_0)$. В настоящей работе использованы компиляторы и опции компиляции, представленные

в таблице (4). Таким образом, для программы P аппроксимация $\mathsf{A}(P_0)$ находится среди 45 версии этой программы.

Таблица 4. Используемые компиляторы и опции компиляции: 9 компиляторов, для каждого из которых применяются 5 опций компиляции (опций оптимизации)

Компилятор	**Версии**	**Опции**
GCC [42]	7.5.0, 8.4.0, 9.4.0, 10.3.0	O0, O1
Clang [43]	7.0.1, 8.0.1, 9.0.1, 10.0.0	O2, O3,
AOCC [44]	3.0.0	Os

6. Экспериментальные исследования. Целью экспериментов является исследование применимости на практике предлагаемого метода оценки эффективности и стойкости обфусцирующих преобразований. Другими словами, исследуется корреляция между значением $s(\mathsf{Obf}(P), \mathsf{A}(P_0))$ и «реальной» похожестью $\mathsf{Obf}(P)$ на $\mathsf{A}(P_0)$ (в случае оценки эффективности), а также корреляция между $s(\mathsf{D}(\mathsf{Obf}(P)), \mathsf{A}(P_0))$ и «реальной» сложностью деобфускации $\mathsf{Obf}(P)$, выражаемой как похожесть $\mathsf{D}(\mathsf{Obf}(P))$ на $\mathsf{A}(P_0)$ (в случае оценки стойкости к деобфускатору). Трудность такого исследования заключается в нехватке независимых результатов, которые могли бы быть референсными («реальными») при оценке эффективности и стойкости. Причина этой нехватки, вероятно, заключается в том, что реальная эффективность и стойкость может быть оценена только человеком, а такие исследования требуют больших ресурсов, в том числе, временных. В этом направлении стоит отметить работы [27, 28, 41], где оценка эффективности и стойкости обфусцирующих преобразований выполнена с привлечением аналитиков. Отметим, во всех трех работах исследовалась обфускация исходного кода. Целью работ [27, 41] было исследование влияния обфускации в целом на сложность понимания и изменения исходного кода нескольких программ (на языках Java и C/C++), а не влияние отдельных преобразований. Учитывая то, что в настоящей работе аналитику доступен машинный код, а не исходный, а так же то, что целью работы является механизм сравнения отдельных преобразований, результаты работ [27, 41] не могут использоваться ка референсные. Результаты работы [28] с одной стороны могут быть использованы, так как исследуется влияние двух обфусцирующих преобразований: переименовывание идентификаторов и вставка непрозрачных предикатов, с другой – использование может быть частичным, так как обфусцирующие преобразования исходного кода могут быть нивелированы на уровне машинного кода (например, переименовывание идентификаторов в

исходном коде может быть частично нивелировано опциями компиляции в машинном коде).

Учитывая растущее разнообразие обфусцирующих преобразований и подходов к деобфускации, вряд ли можно ожидать появление масштабных исследований эффективности и стойкости с помощью человека. В то же время вполне ожидаемыми являются исследования, в которых действия человека моделируются автоматическими средствами. Одной из таких моделей является символьное исполнение [17, 25]. Поэтому в настоящей работе при оценке эффективности и стойкости обфусцирующих преобразований в качестве референсных в дополнение к [28] используются результаты из работ [18, 20, 21].

6.1. Параметры модели. В работе исследуется модель (5) при $\mathcal{Q} = \mathcal{O}_0^1 \cup \mathcal{O}_0^2 \cup \mathcal{O}_0^3 \cup \{\mathsf{all}\}$, где $\mathcal{O}$ – набор преобразований из таблицы 1. Отметим, что $|\mathcal{Q}| = 64$. Данные (программы), используемые в экспериментальном исследовании модели (5), условно можно разделить на две части: 1) данные, необходимые для обучения бинарных классификаторов, на основе которых строятся функции похожести $s = s_c$ и $s = s_f$; 2) данные, необходимые для оценки стойкости и эффективности обфусцирующих преобразований из множества $\mathcal{Q}$. Стоит отметить, что построение набора программ, который можно было бы использовать для независимой оценки средств защиты, является отдельной задачей [8]. В настоящей работе делается шаг в направлении построения набора программ для оценки обфусцирующих преобразований.

Функции похожести s_c и s_f, где c – классификатор из таблицы 3, f – признак из таблицы 2, построены (обучены) на основе 164 программ, указанных в таблице 5.

Таблица 5. Набор программ для построения функций похожести

Набор	**Количество**
CoreUtils [45]	106
PolyBench [46]	30
HashCat [47]	28

Отметим, что набор CoreUtils часто применяется для оценки обфусцирующих преобразований, однако в [8] отмечено, что только утилиты операционной системы вряд ли могут являться репрезентативным набором программ, результаты для которого могут быть перенесена на другие типы программы. Поэтому в настоящей работе кроме CoreUtils используются два других набора: набор программ PolyBench, реализующий алгоритмы компьютерной алгебры, и набор HashCat, реализующий алгоритмы криптографических хеш-функций.

С помощью 9-ти компиляторов и 5-ти опций компиляции, указанных в таблице 4, построен набор из $9 \cdot 5 \cdot 164 = 2880$ программ. По этому набору построено множество пар похожих программ $\mathcal{S}$, состоящее из $\binom{9 \cdot 5}{2} \cdot 164 = 162360$ пар, а также множество пар разных программ $\mathcal{D}$, состоящее из $\binom{164}{2} \cdot (9 \cdot 5) = 601470$ пар (рассматривались только сравнения разных программ для одинаковых компиляторов и опций оптимизации, поэтому множитель $9 \cdot 5$, а не $(9 \cdot 5)^2$).

На основе $\mathcal{S}$ и $\mathcal{D}$ построены набор $\mathcal{F}$ векторов длины 15, в каждый из которых входят 14 значений признаков из таблицы 2 и метка похожести M (для пар из $\mathcal{S}$ метка M имеет значение 1, а для пар из $\mathcal{D}$ – значение 0). В набор $\mathcal{F}$ входят векторы, соответствующие всем парам из $\mathcal{S}$, а также набор из случайно выбранных $|\mathcal{S}|$ пар из $\mathcal{D}$. Кроме этого в набор $\mathcal{F}$ включены два вектора: вектор с меткой 0, состоящий из нулей, а также вектор с меткой 1, состоящий из единиц. Эти векторы включены как краевые случаи: вектор из нулей соответствует разным программам, а вектор из единиц похожим программам. Таким образом, набор $\mathcal{F}$ состоит из $324722 = 2|\mathcal{S}| + 2$ векторов длины 15. Напомним, что для функций s_c векторы признаков имеют размерность 15 (14 признаков из таблицы 2 и бинарная метка похожести M), в то время как вектор признаков для функций s_f имеет размерность два (один признак f из таблицы 2 и метка M). Как показали результаты обучения на наборе $\mathcal{F}$, функции s_c обладают большей точностью, чем любая из функций s_f (столбцы $Accuracy(s_c)$ и $Accuracy(s_f)$ в таблицах 3 и 2 соответственно). В таблице 2 в столбце $Impact(f)$ приведена важность характеристики f, рассчитанная с помощью оценки взаимной информации между характеристикой f и меткой похожести M. Видно, что наибольшей важностью обладают характеристики, полученные с помощью дизассемблирования программ (специализированные характеристики).

Для исследования эффективности и стойкости обфусцирующих преобразований используются два множества: множество $\mathcal{P}_1^{test}$, составленное из программ, входящих в наборы из таблицы 5, а также множество $\mathcal{P}_2^{test}$, составленное из программ, не входящих в указанные наборы. В настоящей работе множество $\mathcal{P}_2^{test}$ состоит из 20 программ, которые являются программами с открытым исходным кодом из набора [48]. Множество $\mathcal{P}_1^{test}$ строится путем случайного выбора также 20-ти программ из 58-ми программ, входящих в наборы PolyBench и HashCat из таблицы 5. По набору $\mathcal{P}_i^{test}$, $i = 1, 2$, строится набор $\mathcal{F}_i$ векторов признаков, соответствующих парам $(\mathsf{Obf}(P), \mathsf{A}(P_0))$, где $P \in \mathcal{P}_i^{test}$, $\mathsf{Obf} \in \mathcal{Q}$. Таким образом, $|\mathcal{F}_i| = 1280$, $i = 1, 2$. Аппроксимация $\mathsf{A}(P_0)$ находится в соответствии с описанным в разделе

5.4 подходом: как наименьшая по размеру версия программы P среди 45 версий. Отметим, что, как показали эксперименты, чаще $\mathsf{A}(P_0)$ соответствует версии P, полученной с помощью компиляторов AOCC и GCC с опцией Os: 98% аппроксимаций получены с помощью компилятора AOCC, а остальные – с помощью различных версий компилятора GCC.

6.2. Исследование эффективности обфусцирующих преобразований. В таблице 6 для каждого из $\mathsf{Obf} \in \mathcal{Q}$ (столбцы с именем Obf) в столбцах с именами e_1 и e_2 приведено среднее значение эффективности, вычисленное по формуле (3) на основе функций похожести s_c для всех классификаторов c из таблицы 3. Первый и третий столбец таблицы содержит преобразования, упорядоченные по убыванию значения эффективности e_1 (от более эффективных к менее эффективным, или, что то же самое, от менее похожих на аппроксимацию к более похожим), полученного на множестве $\mathcal{F}_1$, а второй и четвертый – преобразования, упорядоченные по убыванию значения e_2, полученного на множестве $\mathcal{F}_2$. Далее, под обозначением $e_i(\mathsf{Obf})$ будем понимать значение из столбца с именем e_i таблицы 6, соответствующее преобразованию $\mathsf{Obf} \in \mathcal{Q}$ и полученное на множестве $\mathcal{F}_i$, $i = 1, 2$.

На основе данных из таблицы 6 вычислен *рейтинг эффективности* отдельных преобразований, отдельных пар преобразований и отдельных троек преобразований, входящих в обфусцирующие последовательности. Рейтинг эффективности преобразований рассчитан двумя способами: с учетом порядка следования базовых преобразований и без учета этого порядка. В первом случае набор преобразований $n = (X_1, ..., X_v) \in \mathcal{O}^v$ считается входящим в последовательность $\mathsf{Obf} = (o_{i_1}, ..., o_{i_t})$, если существует j, $1 \leqslant j \leqslant t - v$, что $o_{i_j} = X_1$, $o_{i_{j+1}} = X_2$, ..., $o_{i_{j+v}} = X_v$. Во втором случае этот набор считается входящим в последовательность Obf, если существуют такие $1 \leqslant j_1 < ... < j_v \leqslant t$, что $o_{i_{j_1}} = X_1$, $o_{i_{j_2}} = X_2$, ..., $o_{i_{j_v}} = X_v$. Вхождение набора n в Obf, учитывающее порядок, будем обозначать $n \in_+ \mathsf{Obf}$, а вхождение без учета порядка – сооветственно $n \in_- \mathsf{Obf}$. Для набора n его рейтинг эффективности для заданных $\Delta \in \{+, -\}$ и $i = 1, 2$ определим формулой:

$$\rho_{\Delta,i}(n) = \frac{\sum_{\mathsf{Obf} \in Q_\Delta(n)} e_i(\mathsf{Obf})}{|Q_\Delta(n)|},$$

где $Q_\Delta(n)$ – множество обфусцирующих последовательностей из $\mathcal{Q} \setminus \{\mathsf{all}\}$, содержащих с учетом порядка или без учета (в зависимости от Δ) преобразование n:

$$Q_{\Delta}(n) = \{\mathrm{Obf} \in \mathcal{Q} \setminus \{\mathsf{all}\} : n \in_{\Delta} \mathrm{Obf}\}. \quad (6)$$

В таблице 7 приведен вычисленный рейтинг эффективности одиночных преобразований; для сравнения в таблицу добавлено соответствующее значение для преобразования all из таблицы 6.

Таблица 6. Упорядоченные по убыванию значений эффективности e_1 и e_2 обфусцирующие преобразования длины один, два и три

$\mathcal{F}_1$		$\mathcal{F}_2$		$\mathcal{F}_1$		$\mathcal{F}_2$	
Obf	e_1	Obf	e_2	Obf	e_1	Obf	e_2
ind,fcw,sbb	0,85	all	0,68	bcf,sbb	0,335	ind,cff	0,364
bcf,ind,fcw	0,802	bcf,ind,fcw	0,601	bcf,cff,sub	0,324	ind,cff,sub	0,364
all	0,797	bcf,fcw,sbb	0,583	enc,bcf,sub	0,313	bcf,fcw,sub	0,358
enc,ind,fcw	0,735	bcf,ind,sbb	0,574	enc,bcf,cff	0,308	bcf	0,355
bcf,ind,sub	0,732	enc,bcf,ind	0,544	enc,bcf	0,307	enc,fcw,sbb	0,335
ind,fcw	0,65	ind,fcw,sbb	0,531	enc,fcw	0,304	bcf,sub	0,317
enc,ind,sbb	0,626	bcf,cff,sbb	0,528	enc,fcw,sub	0,291	enc,sbb	0,31
bcf,ind	0,623	enc,bcf,sbb	0,508	bcf,sub	0,29	enc,fcw,sub	0,309
ind,fcw,cff	0,617	bcf,sbb	0,501	fcw	0,288	enc,cff,sbb	0,306
bcf,ind,cff	0,611	bcf,ind,cff	0,486	fcw,cff,sub	0,277	enc,sub,sbb	0,304
ind,sub,sbb	0,611	bcf,ind	0,477	enc,fcw,cff	0,272	enc,fcw,cff	0,303
ind,sbb	0,607	ind,sub,sbb	0,476	bcf	0,271	enc,sub	0,301
ind,fcw,sub	0,6	ind,cff,sbb	0,473	fcw,sub	0,269	enc	0,298
ind,cff,sbb	0,59	ind,sbb	0,473	fcw,cff,sbb	0,268	enc,fcw	0,295
bcf,fcw	0,576	bcf,sub,sbb	0,472	fcw,cff	0,267	fcw,sbb	0,291
enc,bcf,ind	0,564	bcf,fcw	0,472	fcw,sub,sbb	0,259	enc,cff	0,287
enc,bcf,fcw	0,563	enc,ind,sbb	0,471	fcw,sbb	0,25	fcw,sub,sbb	0,285
bcf,ind,sbb	0,559	bcf,fcw,cff	0,465	bcf,cff	0,24	fcw,cff,sbb	0,281
ind,sub	0,557	enc,ind,fcw	0,462	enc,sbb	0,178	fcw,cff,sub	0,279
ind	0,549	bcf,ind,sub	0,453	enc,cff	0,176	fcw	0,278
ind,cff,sub	0,541	enc,bcf,fcw	0,445	enc,cff,sbb	0,175	enc,cff,sub	0,277
enc,ind	0,538	ind,fcw,sub	0,434	enc	0,175	fcw,cff	0,275
enc,ind,cff	0,538	ind,fcw,cff	0,43	enc,cff,sub	0,174	bcf,cff,sub	0,273
bcf,fcw,cff	0,534	enc,ind	0,422	enc,sub,sbb	0,169	fcw,sub	0,272
ind,cff	0,526	ind,fcw	0,414	enc,sub	0,165	enc,bcf,sub	0,267
bcf,fcw,sub	0,522	enc,ind,sub	0,406	cff	0,155	sub,sbb	0,264
bcf,fcw,sbb	0,513	enc,ind,cff	0,387	sub	0,151	sbb	0,262
enc,ind,sub	0,491	ind	0,372	cff,sub	0,144	cff,sub,sbb	0,261
bcf,sub,sbb	0,365	enc,bcf,cff	0,37	cff,sub,sbb	0,14	cff,sub	0,259
enc,bcf,sbb	0,354	bcf,cff	0,367	cff,sbb	0,135	sub	0,257
enc,fcw,sbb	0,348	enc,bcf	0,365	sbb	0,13	cff	0,255
bcf,cff,sbb	0,341	ind,sub	0,365	sub,sbb	0,13	cff,sbb	0,255

Также в таблице 7 представлены результаты работ [20, 21], где оценка эффективности одиночных обфусцирующих преобразований и преобразования all выполнена с привлечением символьного

исполнения (столбец e_{SE}). Отметим, что для одиночных преобразований рейтинг с учетом порядка совпадает с рейтингом без учета порядка: $\rho_{+,i}(n) = \rho_{-,i}(n)$ для всех $n \in \mathcal{O}$. Для пар и троек преобразований результаты вычисления рейтинга показаны в таблицах 8 и 9 соответственно. Преобразования в таблицах 7–9 выстроены в порядке убывания значений $\rho_{+,i}$, $i = 1, 2$. Заметим, что в работе рассматриваются последовательности длины не более три, и для троек преобразований также выполняется равенство $\rho_{+,i}(n) = \rho_{-,i}(n)$.

Таблица 7. Рейтинг эффективности *одиночных* обфусцирующих преобразований

$\mathcal{F}_1$		$\mathcal{F}_2$		[20, 21]	
n	$\rho_{\pm,1}$	n	$\rho_{\pm,2}$	n	e_{SE}
all	0,797	all	0,680	all	0,72
ind	0,614	ind	0,454	ind	0,30
bcf	0,457	bcf	0,445	bcf	0,11
fcw	0,457	sbb	0,397	sbb	0,08
sbb	0,361	fcw	0,382	enc	0,08
enc	0,353	enc	0,362	fcw	0,03
sub	0,342	cff	0,343	cff	0,02
cff	0,334	sub	0,33	sub	0,01

Таблица 8. Рейтинг эффективности *пар* обфусцирующих преобразований

С учетом порядка				Без учета порядка			
$\mathcal{F}_1$		$\mathcal{F}_2$		$\mathcal{F}_1$		$\mathcal{F}_2$	
n	$\rho_{+,1}$	n	$\rho_{+,2}$	n	$\rho_{-,1}$	n	$\rho_{-,2}$
ind,fcw	0,709	bcf,ind	0,522	ind,fcw	0,709	bcf,sbb	0,528
bcf,ind	0,649	ind,sbb	0,506	ind,bcf	0,649	ind,bcf	0,522
ind,sub	0,598	bcf,sbb	0,505	ind,sbb	0,641	ind,sbb	0,5
ind,sbb	0,597	ind,fcw	0,479	ind,sub	0,589	bcf,fcw	0,487
enc,ind	0,586	bcf,fcw	0,465	bcf,fcw	0,585	ind,fcw	0,479
ind,cff	0,561	fcw,sbb	0,435	ind,enc	0,582	ind,enc	0,449
bcf,fcw	0,542	enc,ind	0,43	ind,cff	0,571	ind,cff	0,417
fcw,sbb	0,49	ind,sub	0,425	bcf,sub	0,424	bcf,enc	0,416
enc,bcf	0,401	enc,bcf	0,416	enc,fcw	0,419	ind,sub	0,416
fcw,sub	0,388	ind,cff	0,415	fcw,sbb	0,415	bcf,cff	0,415
fcw,cff	0,372	bcf,cff	0,385	bcf,sbb	0,411	fcw,sbb	0,384
bcf,sbb	0,345	cff,sbb	0,369	bcf,enc	0,401	enc,sbb	0,372
bcf,sub	0,323	bcf,sub	0,352	bcf,cff	0,393	enc,fcw	0,358
enc,fcw	0,304	sub,sbb	0,344	cff,fcw	0,372	bcf,sub	0,357
bcf,cff	0,303	fcw,cff	0,339	fcw,sub	0,37	cff,sbb	0,351
cff,sbb	0,302	fcw,sub	0,332	enc,sbb	0,308	sbb,sub	0,344
sub,sbb	0,279	enc,fcw	0,31	sbb,sub	0,279	cff,fcw	0,339
cff,sub	0,267	enc,sbb	0,31	cff,sbb	0,275	fcw,sub	0,323
enc,sbb	0,178	enc,sub	0,302	cff,enc	0,274	cff,enc	0,322
enc,cff	0,175	enc,cff	0,29	enc,sub	0,267	enc,sub	0,311
enc,sub	0,167	cff,sub	0,285	cff,sub	0,267	cff,sub	0,285

Таблица 9. Рейтинг эффективности *троек* обфусцирующих преобразований

$\mathcal{F}_1$		$\mathcal{F}_2$		$\mathcal{F}_1$		$\mathcal{F}_2$	
n	$\rho_{\pm,1}$	n	$\rho_{\pm,2}$	n	$\rho_{\pm,1}$	n	$\rho_{\pm,2}$
ind,fcw,sbb	0.850	bcf,ind,fcw	0.601	enc,ind,sub	0.491	enc,ind,sub	0.406
bcf,ind,fcw	0.802	bcf,fcw,sbb	0.583	bcf,sub,sbb	0.365	enc,ind,cff	0.387
enc,ind,fcw	0.735	bcf,ind,sbb	0.574	enc,bcf,sbb	0.354	enc,bcf,cff	0.370
bcf,ind,sub	0.732	enc,bcf,ind	0.544	enc,fcw,sbb	0.348	ind,cff,sub	0.364
enc,ind,sbb	0.626	ind,fcw,sbb	0.531	bcf,cff,sbb	0.341	bcf,fcw,sub	0.358
ind,fcw,cff	0.617	bcf,cff,sbb	0.528	bcf,cff,sub	0.324	enc,fcw,sbb	0.335
bcf,ind,cff	0.611	enc,bcf,sbb	0.508	enc,bcf,sub	0.313	enc,fcw,sub	0.309
ind,sub,sbb	0.611	bcf,ind,cff	0.486	enc,bcf,cff	0.308	enc,cff,sbb	0.306
ind,fcw,sub	0.600	ind,sub,sbb	0.476	enc,fcw,sub	0.291	enc,sub,sbb	0.304
ind,cff,sbb	0.590	ind,cff,sbb	0.473	fcw,cff,sub	0.277	enc,fcw,cff	0.303
enc,bcf,ind	0.564	bcf,sub,sbb	0.472	enc,fcw,cff	0.272	fcw,sub,sbb	0.285
enc,bcf,fcw	0.563	enc,ind,sbb	0.471	fcw,cff,sbb	0.268	fcw,cff,sbb	0.281
bcf,ind,sbb	0.559	bcf,fcw,cff	0.465	fcw,sub,sbb	0.259	fcw,cff,sub	0.279
ind,cff,sub	0.541	enc,ind,fcw	0.462	enc,cff,sbb	0.175	enc,cff,sub	0.277
enc,ind,cff	0.538	bcf,ind,sub	0.453	enc,cff,sub	0.174	bcf,cff,sub	0.273
bcf,fcw,cff	0.534	enc,bcf,fcw	0.445	enc,sub,sbb	0.169	enc,bcf,sub	0.267
bcf,fcw,sub	0.522	ind,fcw,sub	0.434	cff,sub,sbb	0.140	cff,sub,sbb	0.261
bcf,fcw,sbb	0.513	ind,fcw,cff	0.430				

Для типов последовательностей определим отношения вхождения τ в $\theta(\mathsf{Obf})$ с учетом ($\in_+$) и без учета порядка ($\in_-$) также, как такие отношения определены выше для последовательностей. Тогда для типа τ его рейтинг при заданных $\Delta \in \{+,-\}$ и $i = 1, 2$ определим формулой:

$$\rho^T_{\Delta,i}(\tau) = \frac{\sum_{\mathsf{Obf} \in Q^T_\Delta(\tau)} e_i(\mathsf{Obf})}{|Q^T_\Delta(\tau)|},$$

где $Q^T_\Delta(\tau)$ – множество обфусцирующих последовательностей из $\mathcal{Q} \setminus \{\mathsf{all}\}$, содержащих преобразования типа τ:

$$Q^T_\Delta(\tau) = \{\mathrm{Obf} \in \mathcal{Q} \setminus \{\mathsf{all}\} : \tau \in_\Delta \theta(\mathrm{Obf})\}. \tag{7}$$

Результаты вычисления рейтинга для одиночных типовых, пар типов и троек типов приведены соответственно в таблицах 10, 11 и 12.

Таблица 10. Рейтинг эффективности типов *одиночных* обфусцирующих преобразований

$\mathcal{F}_1$		$\mathcal{F}_2$		[20, 21]	
τ	$\rho^T_{\pm,1}$	τ	$\rho^T_{\pm,2}$	τ	e_{SE}
C	0,446	C	0,401	C	0.143
A	0,408	A	0,390	A	0.050
D	0,360	D	0,352	D	0.045

Таблица 11. Рейтинг эффективности типов *пар* обфусцирующих преобразований

С учетом порядка				Без учета порядка			
$\mathcal{F}_1$		$\mathcal{F}_2$		$\mathcal{F}_1$		$\mathcal{F}_2$	
τ	$\rho^T_{+,1}$	τ	$\rho^T_{+,2}$	τ	$\rho^T_{-,1}$	τ	$\rho^T_{-,2}$
CA	0.522	CA	0.455	CC	0.528	CC	0.447
CC	0.521	CC	0.447	CA	0.477	CA	0.431
AA	0.490	AA	0.435	CD	0.416	AA	0.384
DC	0.419	DC	0.394	AA	0.415	CD	0.375
AD	0.388	CD	0.344	AD	0.359	AD	0.357
CD	0.381	AC	0.339	DD	0.267	DD	0.311
AC	0.373	AD	0.332				
DA	0.279	DA	0.329				
DD	0.167	DD	0.303				

Таблица 12. Рейтинг эффективности типов *троек* обфусцирующих преобразований

С учетом порядка				Без учета порядка			
$\mathcal{F}_1$		$\mathcal{F}_2$		$\mathcal{F}_1$		$\mathcal{F}_2$	
τ	$\rho^T_{+,1}$	τ	$\rho^T_{+,2}$	τ	$\rho^T_{-,1}$	τ	$\rho^T_{-,2}$
CAA	0.681	CAA	0.557	CCC	0.611	CCA	0.512
CCC	0.611	CCA	0.544	CCA	0.574	CCC	0.486
CAC	0.576	CCC	0.486	CAA	0.544	CAA	0.465
CCA	0.573	CAC	0.447	CCD	0.501	CCD	0.398
CAD	0.561	DCA	0.438	CAD	0.437	CAD	0.398
CCD	0.532	DCC	0.434	CDD	0.326	CDD	0.317
DCA	0.491	CDA	0.403	AAD	0.303	AAD	0.310
DCC	0.470	CAD	0.396	ADD	0.230	ADD	0.307
CDA	0.372	CCD	0.363				
DAA	0.348	DAA	0.335				
DCD	0.326	DCD	0.317				
DAD	0.291	DAD	0.309				
ACD	0.277	DDA	0.304				
DAC	0.272	DAC	0.303				
ACA	0.268	ADA	0.285				
ADA	0.259	ACA	0.281				
DDA	0.169	ACD	0.279				

Отметим, что в таблицах 7-12, кроме упорядочения результатов по убыванию рейтинга для наборов $\mathcal{F}_1$ и $\mathcal{F}_2$, строки этих таблиц сгруппированы специальным образом. Группирование выполнено отдельно для случая, когда порядок применения преобразований учитывается, и для случая, когда такой порядок не учитывается. В каждом из этих случаев столбец с названиями наборов преобразований для $\mathcal{F}_2$ может быть получен путем перестановки строк из столбца с названиями наборов преобразований для $\mathcal{F}_1$. Строки таблиц сгруппированы так, что каждая из групп имеет наименьшее число строк и при этом строки для

$\mathcal{F}_2$ могут быть получены из строк для $\mathcal{F}_1$ путем перестановок строк внутри групп. Такое упорядочение будем использовать для оценки зависимости эффективности преобразований от набора программ. Степень зависимости определим как среднее число строк в группе (результат деления общего числа строк на число групп). Результаты вычисления степени зависимости, полученные по таблицам 7-12, приведены в таблице 13, откуда видно, что с ростом числа преобразований эта зависимость проявляется больше, как в случае учета порядка преобразований, так и без учета.

Таблица 13. Оценка степени зависимости эффективности от набора программ: $|n|$ и $|\theta(n)|$ – соответственно число базовых преобразований и число типов в наборе $n \in \mathcal{O}^i$, $|n| = |\theta(n)| = i$

Δ	$\lvert n\rvert$			$\lvert\theta(n)\rvert$		
	1	2	3	1	2	3
$+$	$1,4$	$10,5$	$17,5$	1	$\approx 1,28$	$\approx 2,83$
$-$					$1,2$	$\approx 1,14$

6.3. Оценка эффективности обфусцирующих преобразований. Анализ таблиц 7-13, помимо выводов о соотношении эффективности конкретных преобразований и типов преобразований, позволяет сделать следующие выводы.

Результаты оценки эффективности с помощью предложенных функций похожести (на основе статических признаков) коррелируют с результатами работ [20, 21], где оценка эффективности выполнена с использованием символьного исполнения (на основе динамических признаков). В частности, для различных подходов к оценке эффективности, согласно, таблице 7, преобразования ind и bcf, демонстрирует наибольшую эффективность, а преобразования sub и cff – наименьшую. Отметим, что особое внимание к преобразованию bcf (непрозрачные или неявные предикаты), например, уделяемое в диссертациях [53] и [6], подтверждает результаты настоящей работы. Более того, для множества $\mathcal{F}_2$ упорядоченный набор преобразований практически совпадает с упорядоченным набором преобразований, полученным в [20, 21]; отличие заключается только в порядке следования соседних преобразований enc и fcw. Отметим, что множество $\mathcal{F}_2$ построено на основе набора программ [48], который использовался и в [20, 21]. Поэтому близость результатов оценки эффективности при различных подходах их получения может свидетельствовать в пользу достоверности этих результатов.

Эффективность преобразования зависит от выбранной функции похожести в модели (5). Наибольшей эффективностью для построенных в работе функций похожести обладают преобразования типа C, которые направлены на изменение графа потока управления (таблица 10), а наименьшей эффективностью – преобразования типа D. Высокая эффективность преобразований типа C подтверждается независимыми исследованиями, например, [52]. Заметим, что в [16] получен обратный результат: обфусцирующие преобразования над данными оказываются более эффективными, чем преобразования графа потока управления. Причина такого различия, как представляется, заключается в использовании разных функций похожести. В [16] эффективность оценивается с помощью сложности программы по Колмогорову, которая аппроксимируется сжимаемостью программы алгоритмами сжатия. Преобразования данных, в частности, шифрование, в большей степени, чем другие преобразования, делают программы более похожими на наборы случайных данных, что приводит к меньшей сжимаемости программ, и объясняет их высокую сложность по Колмогорову. В настоящей работе для построения функции похожести помимо степени сжатия используются специализированные характеристики, учитывающие структуру и характер данных исполняемого файла. Представляется, что демонстрируемая в настоящем исследовании высокая эффективность преобразований типа C объясняется тем, что построенные функции похожести используют, в том числе, информацию о графе потока управления. Другими словами, преобразования типа C меняют граф программы P, делая его менее похожим на граф программы $\mathsf{A}(P_0)$. Другими словами, эффективность обфусцирующих преобразований зависит от используемой функции похожести, которую можно рассматривать как модель аналитика, против которого выбирается эффективное защитное обфусцирующее преобразование. Тем не менее, влияние обфусцирующих преобразований типа C на эффективность обфусцирующей последовательности независимо подтверждается особым внимаем исследователей к таким преобразованиям (например, [6, 13, 28, 50, 53 – 56]). Такой интерес может быть обоснован тем, что часто аналитик начинает исследование программы с построения графа потока управления [57], поэтому естественным направлением исследований в области защиты программ является затруднение восстановления такого графа. При этом стоит отметить, что не все преобразования этого типа могут обеспечивать высокую эффективность. В частности, из таблиц 7-9 видно, что группы, содержащие наименее эффективные наборы, содержат наборы, в которые из преобразований типа C входит

только преобразование cff. Слабая эффективность этого преобразования показана с применением техники символьного исполнения в [20, 21] (крайние правые два столбца в таблице 7) и независимо показана в [51] также с помощью символьного исполнения.

Эффективность обфусцирующего преобразования зависит от программы, к которой оно применяется, и порядка применения базовых преобразований. В частности, согласно таблице 7, на наборе $\mathcal{F}_1$ преобразование fcw эффективнее sbb, а sub эффективнее cff. В то же время на наборе $\mathcal{F}_2$ для указанных пар наблюдается обратная ситуация. В таблице 8 эта зависимость проявляется в большей степени для пар базовых преобразований. Например, можно выделить только три пары преобразований, находящихся в первой пятерке наиболее эффективных преобразований и для $\mathcal{F}_1$ и для $\mathcal{F}_2$: (ind,fcw), (bcf,ind) и (ind,sbb). Также можно выделить только два преобразования, которые независимо от набора программ и порядка применения базовых преобразований, находятся в пятерке наименее эффективных:(cff,sub) и (enc,cff). В остальных случаях наблюдается зависимость от порядка применения преобразований и/или набора программ. Для троек преобразований (напомним, что в этом случае $\rho_{+,i} = \rho_{-,i}$), как следует из таблицы 12, в первой пятерке наиболее эффективных троек независимо от рассматриваемого набора находятся преобразования (ind,fcw,sbb) и (bcf,ind,fcw). Независимость от набора программ демонстрирует также наименее эффективные преобразования (cff,sub,sbb) и (enc,cff,sub). В остальных случаях для троек наблюдается зависимость от набора программ. Степень зависимости эффективности от набора программ отражена в таблице 13. Особенно эта зависимость проявляется с ростом числа преобразований в последовательности. Для типов преобразований зависимость от программ проявляется в меньшей степени. В большей степени наблюдается зависимость от порядка и типа применяемых преобразований (следующий вывод).

Эффективность зависит от типов преобразования и порядка следования этих типов. Согласно таблицам 11 и 12, комбинации преобразований типа C и A позволяют строить эффективные обфусцирующие последовательности, причем эффективность зависит от порядка применения типов: последовательности, в которых сначала применяется преобразование типа C, а затем – типа A, существенно эффективнее обратного порядка. Этот эффект может быть объяснен тем, что преобразование типа A может изменить программу так, что применение преобразования C становится затруднительным. Результаты таблицы 12, в которых учитывается порядок применения преобразований,

можно условно разделить на две части: первые 9 строк, последние 8 строк. В первой части, как для $\mathcal{F}_1$, так и для $\mathcal{F}_2$, в каждом из типов на первом или втором месте есть преобразование типа C, причем нет типов, начинающихся типом A, и тип A всегда следует после типа C. Во второй части, как для $\mathcal{F}_1$, так и для $\mathcal{F}_2$, расположены типы, либо начинающиеся типом A, либо начинающиеся D, в которых тип A, при наличии, следует после D, либо начинающиеся D и содержащие два преобразования типа D. Таким образом, множество из первых 9-ти типов преобразований для набора $\mathcal{F}_1$ совпадает с точностью до перестановки со множеством из первых 9-ти типов преобразований для набора $\mathcal{F}_2$. Аналогичный вывод справедлив и для типов из второй части. Последовательности, в которых не используются пребразования типа C, обладают наименьшей эффективностью. Отметим, что в соответствии с результатами работы [8] в среднем преобразования типов CDA и DAC являются *дефективными* (время символьного анализа уменьшилось по сравнению со временем символьного исполнения оригинальной программы), что согласуется с полученными в настоящей работе результатами. Именно, не смотря на то, что CDA относится к первой части, оно занимает последнее 9-е место для набора $\mathcal{F}_1$ и 7-е место для набора $\mathcal{F}_2$. Отметим, что в типах CDA и DAC преобразование типа A идет после преобразования типа D, что выше для результатов, полученных в этой работе, отмечено как признак малоэффективных преобразований. Тип DAC относится ко второй части и занимает в этой части 5-е место среди восьми для обоих наборов программ. Преобразования CAD и DCA содержатся на пятом месте первой части для обоих наборов и в соответствии с [18] также являются эффективными.

В [18] не исследовались пары преобразований, а также тройки преобразований вида (X,Y,Z), в которых преобразования X и Z одного типа разделяются преобразованием Y другого типа. Однако результаты показывают, что такие преобразования могут быть более эффективными. В частности, для набора $\mathcal{F}_1$ наибольшую эффективность показало преобразование (ind,fcw,sbb) (последовательность типа CAA).

Увеличение числа преобразований в обфусцирующих последовательностях по-разному влияет на изменение эффективности (уменьшение похожести). В частности, на наборе $\mathcal{F}_1$ добавление преобразования cff к последовательностям X и (X,Y) приводит к уменьшению эффективности в 8-ми из 10-ти случаев. Добавление преобразования sub к последовательностям X и (X,Y) в 8-ми из 15-ти случаев приводит к уменьшению эффективности. Добавление преобразования sbb к последовательностям X и (X,Y) в 8-ми из 21-ого

случаев приводит к уменьшению эффективности. На множестве $\mathcal{F}_2$ в 5-ти из 10-ти случаев добавление преобразования cff к последовательностям X и (X,Y) приводит к уменьшению эффективности. Добавление преобразования sub к последовательностям X и (X,Y) в 10-ти из 15-ти случаев приводит к уменьшению эффективности. Добавление преобразования sbb к последовательностям X и (X,Y) только в одном случае из 21-го приводит к уменьшению эффективности.

Имеется группа преобразований: {cff, sub, (cff,sub), (cff,sub,sbb), (cff,sbb), sbb, (cff,sbb), sbb, (sub,sbb)}, которые показывают наименьшую эффективность на обоих наборах $\mathcal{F}_1$ и $\mathcal{F}_2$. То есть можно отметить обфусцирующие преобразования, которые, как представляется, обладают малой эффективностью независимо от обусцируемой программы. Аналогичный вывод можно сделать и для наиболее эффективных преобразований {all, (bcf,ind,fcw)}, которые демонстрируют наибольшую эффективность на наборах $\mathcal{F}_1$ и $\mathcal{F}_2$. В остальных случаях наблюдается зависимость эффективности от обфусцируемой программы.

6.4. Исследование стойкости обфусцирующих преобразований. Для оценки стойкости реализована схема, изображенная на рисунке 1. В качестве деобфускатора применен оптимизатор компилятора Clang для IR-кода (параметр оптимизации Os). Отметим, что в работах [20, 21] в качестве деобфускатора при оценке стойкости использовался такой же оптимизатор. В таблице 14 для каждого из $\mathsf{Obf} \in \mathcal{Q}$ в столбцах с именами r_1 и r_2 приведено среднее значение стойкости, вычисленное по формуле (4) на основе функций похожести s_c для всех классификаторов c из таблицы 3. Первый и третий столбец таблицы содержит преобразования, упорядоченные по убыванию значения стойкости r_1 (от более эффективных к менее эффективным, или, что то же самое, от менее похожих на аппроксимацию к более похожим), полученного на множестве $\mathcal{F}_1$, а второй и четвертый – преобразования, упорядоченные по убыванию значения r_2, полученного на множестве $\mathcal{F}_2$.

По аналогии с рейтингом эффективности для заданных $\mathcal{F}_i$, $i = 1, 2$, и $\Delta \in \{+, -\}$ определим рейтинг стойкости одиночных преобразований, пар преобразований, троек преобразований и соответствующих комбинаций типов преобразований:

$$\varrho_{\Delta,i}(n) = \frac{\sum_{\mathsf{Obf} \in Q_\Delta(n)} r_i(\mathsf{Obf}, \mathsf{D})}{|Q_\Delta(n)|},$$

$$\varrho^T_{\Delta,i}(t) = \frac{\sum_{\mathsf{Obf} \in Q^T_\Delta(t)} r_i(\mathsf{Obf}, \mathsf{D})}{|Q^T_\Delta(t)|},$$

где символом $r_i(\mathsf{Obf}, \mathsf{D})$ здесь обозначается стойкость r_i из таблицы 14 преобразования Obf по отношению к деобфускатору D, а множества $Q_\Delta(n)$ и $Q_\Delta^T(n)$ определены соответственно в (6) и (7). Результаты вычисления рейтинга стойкости, упорядоченные по убыванию стойкости, представлены в таблицах 15,16,17,18,19,20. В таблице 21 представлены результаты оценки зависимости значений стойкости для выбранных функций похожести от используемого тестового набора программ. Как и для эффективности такая зависимость определяется как среднее число строк в группе.

Таблица 14. Упорядоченные по убыванию значений стойкости r_1 и r_2 обфусцирующие преобразования длины один, два и три

$\mathcal{F}_1$		$\mathcal{F}_2$		$\mathcal{F}_1$		$\mathcal{F}_2$	
Obf	r_1	Obf	r_2	Obf	r_1	Obf	r_2
all	0,905	all	0,585	fcw,sub,sbb	0,276	enc,cff,sub	0,288
ind,fcw,sbb	0,883	bcf,ind,fcw	0,567	fcw,cff,sbb	0,272	enc,cff,sbb	0,288
bcf,ind,fcw	0,869	bcf,ind,sbb	0,545	fcw,cff,sub	0,269	fcw,sub,sbb	0,284
bcf,ind,sbb	0,766	ind,fcw,sbb	0,505	bcf,fcw,sub	0,267	fcw,cff	0,282
bcf,ind	0,703	bcf,ind	0,5	fcw,cff	0,262	fcw,sbb	0,281
bcf,ind,cff	0,692	enc,bcf,ind	0,497	bcf,sub,sbb	0,261	fcw	0,276
ind,sbb	0,689	bcf,ind,sub	0,495	enc,cff,sbb	0,253	fcw,sub	0,276
enc,bcf,ind	0,675	enc,ind,sbb	0,493	enc,sub,sbb	0,252	fcw,cff,sbb	0,276
bcf,ind,sub	0,675	bcf,ind,cff	0,491	bcf,cff,sbb	0,244	sbb	0,274
ind,cff,sbb	0,657	ind,cff,sbb	0,479	enc,bcf,sbb	0,244	cff,sub	0,273
ind,sub,sbb	0,65	ind,sbb	0,478	fcw	0,243	fcw,cff,sub	0,271
enc,ind,fcw	0,65	ind,sub,sbb	0,475	fcw,sub	0,242	cff,sub,sbb	0,271
ind,fcw,sub	0,649	enc,ind,fcw	0,454	enc	0,24	cff	0,27
ind,fcw,cff	0,637	ind,fcw	0,433	enc,sbb	0,238	sub,sbb	0,269
ind,fcw	0,63	ind,fcw,sub	0,432	enc,cff	0,235	cff,sbb	0,266
enc,ind,sbb	0,58	ind,fcw,cff	0,431	bcf,sbb	0,228	enc,bcf,cff	0,265
ind	0,519	enc,ind	0,411	enc,bcf,cff	0,213	enc,bcf,fcw	0,259
ind,sub	0,514	enc,ind,cff	0,409	bcf	0,206	sub	0,259
ind,cff,sub	0,512	ind,cff,sub	0,394	cff,sub	0,205	bcf,fcw	0,257
ind,cff	0,505	enc,ind,sub	0,391	sub	0,203	enc,bcf	0,256
enc,ind,sub	0,49	ind,cff	0,391	cff	0,203	enc,bcf,sub	0,256
enc,ind	0,449	ind,sub	0,385	bcf,cff,sub	0,196	bcf,fcw,sub	0,253
enc,ind,cff	0,438	ind	0,382	enc,bcf	0,194	bcf,sub	0,253
enc,fcw,sbb	0,43	enc,fcw,sbb	0,316	bcf,cff	0,189	bcf	0,253
enc,bcf,fcw	0,375	enc,sbb	0,308	bcf,sub	0,188	bcf,cff	0,251
enc,fcw,cff	0,34	enc,sub,sbb	0,306	cff,sub,sbb	0,186	bcf,fcw,cff	0,244
bcf,fcw,sbb	0,315	enc,fcw,cff	0,297	enc,sub	0,184	bcf,fcw,sbb	0,238
bcf,fcw,cff	0,303	enc,fcw	0,295	enc,cff,sub	0,175	bcf,cff,sub	0,237
enc,fcw	0,29	enc,cff	0,294	sbb	0,174	bcf,cff,sbb	0,235
bcf,fcw	0,287	enc	0,293	cff,sbb	0,17	enc,bcf,sbb	0,229
enc,fcw,sub	0,277	enc,fcw,sub	0,292	sub,sbb	0,17	bcf,sub,sbb	0,219
fcw,sbb	0,277	enc,sub	0,291	enc,bcf,sub	0,168	bcf,sbb	0,21

Таблица 15. Рейтинг стойкости *одиночных* обфусцирующих преобразований

$\mathcal{F}_1$		$\mathcal{F}_2$		[20, 21]	
n	$\varrho_{\pm,1}$	n	$\varrho_{\pm,2}$	n	r_{SE}
all	0,905	all	0,585	all	0.709
ind	0,629	ind	0,456	ind	0.300
fcw	0,411	sbb	0,329	bcf	0.103
bcf	0,376	fcw	0,328	sbb	0.075
sbb	0,373	enc	0,327	enc	0.070
enc	0,336	bcf	0,319	fcw	0.025
cff	0,325	cff	0,314	cff	0.022
sub	0,319	sub	0,312	sub	0.019

Таблица 16. Рейтинг стойкости *пар* обфусцирующих преобразований

С учетом порядка				Без учета порядка			
$\mathcal{F}_1$		$\mathcal{F}_2$		$\mathcal{F}_1$		$\mathcal{F}_2$	
n	$\varrho_{+,1}$	n	$\varrho_{+,2}$	n	$\varrho_{-,1}$	n	$\varrho_{-,2}$
bcf,ind	0,732	bcf,ind	0,516	ind,bcf	0,732	ind,bcf	0,516
ind,fcw	0,720	ind,sbb	0,505	ind,fcw	0,720	ind,sbb	0,496
ind,sbb	0,678	ind,fcw	0,470	ind,sbb	0,702	ind,fcw	0,470
ind,sub	0,582	ind,sub	0,436	ind,sub	0,582	ind,enc	0,443
ind,cff	0,561	ind,cff	0,433	ind,cff	0,574	ind,cff	0,433
enc,ind	0,521	enc,ind	0,432	ind,enc	0,547	ind,sub	0,429
fcw,sbb	0,473	fcw,sbb	0,335	fcw,sbb	0,407	enc,sbb	0,323
fcw,cff	0,347	cff,sbb	0,309	bcf,fcw	0,405	enc,fcw	0,319
fcw,sub	0,342	enc,sbb	0,308	enc,fcw	0,394	fcw,sbb	0,317
enc,fcw	0,334	fcw,sub	0,307	cff,fcw	0,347	cff,enc	0,307
cff,sbb	0,319	sub,sbb	0,304	bcf,sbb	0,343	enc,sub	0,304
enc,bcf	0,312	fcw,cff	0,300	enc,sbb	0,333	sbb,sub	0,304
bcf,fcw	0,309	enc,fcw	0,300	fcw,sub	0,330	bcf,fcw	0,303
sub,sbb	0,299	enc,sub	0,299	bcf,enc	0,312	cff,sbb	0,302
cff,sub	0,257	enc,bcf	0,294	bcf,cff	0,306	fcw,sub	0,301
enc,sbb	0,238	enc,cff	0,290	sbb,sub	0,299	cff,fcw	0,300
bcf,sbb	0,236	cff,sub	0,289	cff,sbb	0,297	bcf,enc	0,294
enc,cff	0,221	bcf,fcw	0,250	bcf,sub	0,292	cff,sub	0,289
enc,sub	0,218	bcf,cff	0,247	cff,enc	0,276	bcf,cff	0,287
bcf,cff	0,210	bcf,sub	0,243	enc,sub	0,258	bcf,sub	0,286
bcf,sub	0,206	bcf,sbb	0,219	cff,sub	0,257	bcf,sbb	0,279

6.5. Оценка стойкости обфусцирующих преобразований. Анализ результатов в таблицах, помимо выводов о соотношении рейтинга стойкости отдельных наборов преобразований и их типов, позволяет сделать следуюие выводы.

Таблица 17. Рейтинг стойкости *троек* обфусцирующих преобразований

$\mathcal{F}_1$		$\mathcal{F}_2$		$\mathcal{F}_1$		$\mathcal{F}_2$	
n	$\varrho_{\pm,1}$	n	$\varrho_{\pm,2}$	n	$\varrho_{\pm,1}$	n	$\varrho_{\pm,2}$
bcf,ind,fcw	0,883	bcf,ind,fcw	0,567	bcf,fcw,sbb	0,315	enc,fcw,sub	0,292
ind,fcw,sbb	0,869	bcf,ind,sbb	0,545	bcf,fcw,cff	0,303	enc,cff,sub	0,288
bcf,ind,sbb	0,766	ind,fcw,sbb	0,505	enc,fcw,sub	0,277	enc,cff,sbb	0,288
bcf,ind,cff	0,692	enc,bcf,ind	0,497	fcw,sub,sbb	0,276	fcw,sub,sbb	0,284
bcf,ind,sub	0,675	bcf,ind,sub	0,495	fcw,cff,sbb	0,272	fcw,cff,sbb	0,276
enc,bcf,ind	0,675	enc,ind,sbb	0,493	fcw,cff,sub	0,269	fcw,cff,sub	0,271
ind,cff,sbb	0,657	bcf,ind,cff	0,491	bcf,fcw,sub	0,267	cff,sub,sbb	0,271
ind,sub,sbb	0,650	ind,cff,sbb	0,479	bcf,sub,sbb	0,261	enc,bcf,cff	0,265
enc,ind,fcw	0,650	ind,sub,sbb	0,475	enc,cff,sbb	0,253	enc,bcf,fcw	0,259
ind,fcw,sub	0,649	enc,ind,fcw	0,454	enc,sub,sbb	0,252	enc,bcf,sub	0,256
ind,fcw,cff	0,637	ind,fcw,sub	0,432	enc,bcf,sbb	0,244	bcf,fcw,sub	0,253
enc,ind,sbb	0,580	ind,fcw,cff	0,431	bcf,cff,sbb	0,244	bcf,fcw,cff	0,244
ind,cff,sub	0,512	enc,ind,cff	0,409	enc,bcf,cff	0,213	bcf,fcw,sbb	0,238
enc,ind,sub	0,490	ind,cff,sub	0,394	bcf,cff,sub	0,196	bcf,cff,sub	0,237
enc,ind,cff	0,438	enc,ind,sub	0,391	cff,sub,sbb	0,186	bcf,cff,sbb	0,235
enc,fcw,sbb	0,430	enc,fcw,sbb	0,316	enc,cff,sub	0,175	enc,bcf,sbb	0,229
enc,bcf,fcw	0,375	enc,sub,sbb	0,306	enc,bcf,sub	0,168	bcf,sub,sbb	0,219
enc,fcw,cff	0,340	enc,fcw,cff	0,297				

Таблица 18. Рейтинг стойкости типов *одиночных* обфусцирующих преобразований

$\mathcal{F}_1$		$\mathcal{F}_2$		[20, 21]	
t	$\varrho^T_{\pm,1}$	t	$\varrho^T_{\pm,2}$	t	r_{SE}
C	0,414	C	0.348	C	0.208
A	0,390	A	0.331	A	0.050
D	0,338	D	0.322	D	0.045

Таблица 19. Рейтинг стойкости типов *пар* обфусцирующих преобразований

С учетом порядка				Без учета порядка			
$\mathcal{F}_1$		$\mathcal{F}_2$		$\mathcal{F}_1$		$\mathcal{F}_2$	
t	$\varrho^T_{+,1}$	t	$\varrho^T_{+,2}$	t	$\varrho^T_{-,1}$	t	$\varrho^T_{-,2}$
CC	0,525	CC	0,411	CC	0,518	CC	0,402
CA	0,475	CA	0,361	CA	0,442	CA	0,348
AA	0,473	DC	0,342	AA	0,407	CD	0,335
DC	0,367	AA	0,335	CD	0,372	AA	0,317
AC	0,347	CD	0,324	AD	0,345	AD	0,314
CD	0,345	AD	0,307	DD	0,258	DD	0,304
AD	0,342	DA	0,303				
DA	0,306	AC	0,300				
DD	0,218	DD	0,299				

Рейтинг стойкости одиночных преобразований, представленный в таблицах 15 и 18 и полученный на основе статических характеристик программ, частично коррелирует с результатами оценки стойкости

этих преобразований, полученными в [20, 21] на основе динамических характеристик – характеристик символьного исполнения. В частности, результаты, полученные в настоящей работе, также как и результаты работ [20, 21] позволяют в качестве наиболее стойкого выделить преобразование ind, а в качестве наименее стойких – преобразования cff и sub (таблица 15). Относительно других одиночных преобразований (bcf, sbb, enc и fcw) корреляция явно не выражена: только в случае множества $\mathcal{F}_2$ номер строки в таблице 15, где расположено преобразование enc, совпадает с номером строки этого преобразования, когда оценка стойкости выполнена методами символьного исполнения. В то же время, среди обфусцирующих последовательностей, содержащих только одно преобразование из базового набора, наибольшей стойкостью в среднем обладают преобразования типа C, а наименьшей – преобразования типа D (таблица 18). Эти результаты качественно совпадают с усредненными результатами оценки стойкости типов преобразований, полученными в [20, 21]: набор типов одиночных преобразований, упорядоченный по убыванию усредненного рейтинга стойкости, полученный в настоящей работе, совпадает с аналогичным образом упорядоченным набором типов, полученным на основе результатов [20, 21] (таблица 18). Также стоит отметить, что референсное преобразование all во всех случаях обладает наибольшим рейтингом.

Таблица 20. Рейтинг стойкости типов *троек* обфусцирующих преобразований

С учетом порядка				Без учета порядка			
$\mathcal{F}_1$		$\mathcal{F}_2$		$\mathcal{F}_1$		$\mathcal{F}_2$	
t	$\varrho^T_{+,1}$	t	$\varrho^T_{+,2}$	t	$\varrho^T_{-,1}$	t	$\varrho^T_{-,2}$
CCC	0,692	CCC	0,491	CCC	0,692	CCC	0,491
CCA	0,637	CCA	0,456	CCA	0,582	CCA	0,417
CAA	0,592	DCC	0,390	CAA	0,485	CCD	0,383
CAC	0,470	CCD	0,375	CCD	0,452	CAA	0,340
CCD	0,461	CAA	0,371	CAD	0,394	CAD	0,328
CAD	0,458	DCA	0,345	AAD	0,353	CDD	0,312
DCC	0,442	CAD	0,343	CDD	0,278	AAD	0,300
DAA	0,430	CAC	0,337	ADD	0,265	ADD	0,299
DCA	0,420	CDA	0,322				
CDA	0,366	DAA	0,316				
DAC	0,340	DCD	0,312				
DCD	0,278	DDA	0,306				
DAD	0,277	DAC	0,297				
ADA	0,276	DAD	0,292				
ACA	0,272	ADA	0,284				
ACD	0,269	ACA	0,276				
DDA	0,252	ACD	0,271				

Таблица 21. Оценка степени зависимости стойкости от набора программ: $|n|$ и $|\theta(n)|$ – соответственно число базовых преобразований и число типов в наборе $n \in \mathcal{O}^i$, $|n| = |\theta(n)| = i$

Δ	$\|n\|$			$\|\theta(n)\|$		
	1	2	3	1	2	3
+	1,75	3	5	1	1,8	4,25
−		5,25			1,2	≈2,67

Стойкость обфусцирующего преобразования зависит от программы, к которой оно применяется, и порядка применения базовых преобразований. В частности, согласно таблице 15, на наборе $\mathcal{F}_1$ преобразования fcw и bcf более стойкие, чем sbb и enc соответственно. В то же время на наборе $\mathcal{F}_2$ для указанных пар наблюдается обратная ситуация. В таблице 16 эта зависимость проявляется в большей степени для пар базовых преобразований. Например, в левой половине таблицы 16 (оценка стойкости с учетом порядка преобразований) можно выделить только семь пар преобразований, рейтинг которых практически не зависит от набора данных; в правой половине таблицы 16 (оценка стойкости без учета порядка преобразований) можно выделить только шесть таких пар преобразований. В остальных случаях наблюдается зависимость от порядка применения преобразований и/или набора программ. Для троек преобразований (напомним, что в этом случае $\varrho_{+,i} = \varrho_{-,i}$), как следует из таблицы 20, в первой пятерке наиболее стойких троек независимо от рассматриваемого набора находятся четыре последовательности: (bcf,ind,fcw), (ind,fcw,sbb), (bcf,ind,fcw), (bcf,ind,sub). В остальных случаях для троек наблюдается зависимость от набора программ. Степень зависимости стойкости от набора программ отражена в таблице 21. Стоит отметить, что зависимость стойкости конкретных последовательностей преобразований от набора программ проявляется в меньшей степени, зависимость эффективности этих преобразований (ср. таблицы 13 и 21). Представляется, что применение одного деобфускатора D к программам из разных наборов делает эти наборы «менее различимыми» рассматриваемыми классификаторами из таблицы 3.

Стойкость зависит от типов преобразования и порядка следования этих типов. Согласно таблицам 19 и 20, комбинации преобразований типа C и A позволяют строить стойкие обфусцирующие последовательности, причем стойкость зависит от порядка применения типов: последовательности, в которых сначала применяется преобразование типа C, а затем – типа A, более стойкие, чем последовательности с обратным порядком следования типов. Этот

эффект может быть объяснен тем, что преобразование типа A может изменить программу так, что применение преобразования C становится затруднительным. Результаты таблицы 20, в которых учитывается порядок применения преобразований, можно условно выделить две части: первые 7 строк, последние 7 строк. В первой части, как для $\mathcal{F}_1$, так и для $\mathcal{F}_2$, в каждом из типов на первом или втором месте есть преобразование типа C, причем нет типов, начинающихся типом A, и тип A всегда следует после типа C. Во второй части, как для $\mathcal{F}_1$, так и для $\mathcal{F}_2$, расположены типы, либо начинающиеся типом A, либо начинающиеся типом D, в которых тип A, при наличии, следует после D, либо начинающиеся с D и содержащие два преобразования типа D.

Увеличение числа преобразований в обфусцирующих последовательностях по-разному влияет на изменение стойкости. В частности, на наборах $\mathcal{F}_1$ и $\mathcal{F}_2$ добавление преобразования bcf уменьшает стойкость: при добавлении к преобразованию enc). На наборе $\mathcal{F}_1$ добавление преобразования cff к последовательностям X и (X,Y) приводит к уменьшению стойкости в 5-ти из 10-ти случаев; добавление преобразования sub к последовательностям X и (X,Y) в 9-ми из 15-ти случаев приводит к уменьшению стойкости; добавление преобразования sbb к последовательностям X и (X,Y) в 4-ех из 21-ого случая приводит к уменьшению стойкости. На множестве $\mathcal{F}_2$ в 5-ти из 10-ти случаев добавление преобразования cff к последовательностям X и (X,Y) приводит к уменьшению стойкости. Добавление преобразования sub к последовательностям X и (X,Y) в 11-ти из 15-ти случаев приводит к уменьшению стойкости. Добавление преобразования sbb к последовательностям X и (X,Y) в 9-ти из 21-го случая приводит к уменьшению стойкости.

6.6. Общие выводы. Можно заключить, что в среднем наиболее эффективными и стойкими являются обфусцирующие последовательности, начинающиеся преобразованием типа C, а наименее эффективными – последовательности, начинающиеся преобразованием типа D и при этом не содержание преобразований типа C, либо последовательности, начинающиеся преобразованием типа A. Стоит отметить, что не все преобразования типа C демонстрируют высокую эффективность и стойкость. Так, преобразование cff имеет низкий рейтинг как по эффективности, так и по стойкости: в таблицах 7 и 15 это преобразование занимает предпоследнюю или последнюю строки.

Последовательное добавление преобразований не всегда приводит к увеличению стойкости и эффективности обфусцирующих последовательностей. Результаты экспериментов показывают, что

последовательное добавление преобразований ind и fcw (тип C) к обфусцирующим последовательностям X и (X,Y) всегда приводит к увеличению эффективности и стойкости преобразований. Влияние преобразования ind на эффективность и стойкость, как представляется, связано с проблемами распознавания косвенных переходов автоматическими средствами статического анализа машинного кода, такими как дизассемблеры [58], что в свою очередь может приводить к ошибкам и/или неточностям восстановления, например, графа потока управления, на основе которого также принимается решение о степени похожести программ (специализированные характеристики в таблице 2). Трудность распознавания адреса косвенных переходов подтверждается теоретическими исследованиями: в [57] доказано, что точное определение адреса для косвенных переходов является NP-трудной задачей. Поэтому в [57] для затруднения восстановления графа потока управления предлагается способ на основе замены прямых переходов косвенными, когда адрес перехода вычисляется динамически. На практике сложность анализа кода, содержащего косвенные переходы, подтверждается также сложностью динамического анализа, а именно сложностью символьной интерпретацию кода, содержащего такие переходы [49]. В частности, такое подтверждение получено в [20, 21] (крайние правые два столбца в таблице 7).

7. Заключение. Заметим, что в [8] выделено всего 5 работ из 572, в которых предлагаются не только способы защиты, но и проводится анализ случаев, когда противник адаптирует средства анализа для обхода защиты. Малое число работ, учитывающих возможность адаптации противника, подчеркивает актуальность задачи количественной оценки эффективности и стойкости защиты при наличии подходящих средств анализа у противника. Предлагаемый в настоящей работе способ оценки эффективности и стойкости позволяет адаптировать оценку при появлении новых средств анализа. Это может быть выполнено за счет дообучения существующей функции похожести или построения новой функции.

Отметим, что предлагаемый в настоящей работе подход, может быть применен для оценки и эффективности, и стойкости. Представляется, что при подборе подходящей функции похожести этот метод оценки эффективности и стойкости может быть применен не только к бинарным исполняемым файлам, но и к файлам с исходным кодом и файлам с кодом промежуточного представления. Для оценки эффективности преобразований конкретных программ рекомендуется

использовать функции похожести, обладающие высокой точностью, например, функцию s_{NN} на основе нейронных сетей.

Дальнейшими направлениями исследований являются: 1) построение функций похожести с высокой точностью на наборах, отличных от наборов, использованных при обучении; 2) построение методами машинного обучения функций похожести, для которых возможно дообучение не только с помощью новых наборов данных, но и с помощью новых признаков; 3) построение функций похожести без использования методов машинного обучения, например, построение нечетких хеш-функций для исполняемых файлов.

Литература

1. Akhunzada A., Sookhak M, Anuar A.B., Gani A., Ahmed E., Shiraz M., Furnell S., Hayat A., Khan M.K. Man-At-The-End attacks: Analysis, taxonomy, human aspects, motivation and future directions // Journal of Network and Computer Applications. 2015. vol. 48. pp. 44–57.
2. Undrits R., Resende J. et. al. CyberSec4Europe D3.23: Cybersecurity Outlook 2. Research Report D3.23. 2022. pp. 1–82.
3. Biernacki L., Gallagher M., Xu Z., Aga M.T., Harris A., Wei S., Tiwari M., Kasikci B., Malik S., Austin T.. Software-driven security attacks: From vulnerability sources to durable hardware defenses // ACM Journal on Emerging Technologies in Computing Systems (JETC). 2021. vol. 17. no. 3. pp. 1–38.
4. Варновский Н.П., Захаров В.А., Кузюрин Н.Н., Шокуров А.В. Современное состояние исследований в области обфускации программ: определения стойкости обфускации // Труды Института системного программирования РАН. 2014. Т. 26. № 3. С. 167–198.
5. Barak B., Goldreich O., Impagliazzo R., Rudich S., Sahai A., Vadhan S., Yang K. On the (im) possibility of obfuscating programs // Journal of the ACM (JACM). 2012. vol. 59. no. 2. pp. 1–42.
6. Zobernig L. Mathematical Aspects of Program Obfuscation. Doctoral dissertation. ResearchSpace@ Auckland, 2020. URL: www.math.auckland.ac.nz/~ sgal018/Lukas-Zobernig-Thesis.pdf (дата обращения: 19.06.2023).
7. Garg S., Gentry C., Halevi S., Raykova M., Sahai A., Waters B. Candidate indistinguishability obfuscation and functional encryption for all circuits // SIAM Journal on Computing. 2016. vol. 45. no. 3. pp. 882–929.
8. Kochberger P., Schrittwieser S., Coppens B., De Sutter B. Evaluation Methodologies in Software Protection Research // arXiv preprint arXiv:2307.07300. 2023. pp. 1–67.
9. Zhou Y., Main A., Gu Y.X., Johnson H. Information hiding in software with mixed boolean-arithmetic transforms // Information Security Applications: 8th International Workshop on Information Security Applications. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007. vol. 4867. pp. 61–75.
10. Reichenwallner B., Meerwald-Stadler P. Efficient Deobfuscation of Linear Mixed Boolean- Arithmetic Expressions // Proceedings of the 2022 ACM Workshop on Research on offensive and defensive techniques in the context of Man At The End (MATE) attacks. 2022. pp. 19–28.
11. Xu D., Liu D., Feng W., Ming J., Zheng Q., Li J., Yu Q. Boosting SMT solver performance on mixed-bitwise-arithmetic expressions // Proceedings of the 42nd ACM SIGPLAN

International Conference on Programming Language Design and Implementation. 2021. pp. 651–664.

12. Liu B., Shen J., Ming J., Zheng Q., Li J., Xu D. MBA-Blast: Unveiling and Simplifying Mixed Boolean-Arithmetic Obfuscation // Proceedings of the 30th USENIX Security Symposium. 2021. pp. 1701–1718.
13. Косолапов Ю.В. Об упрощении выражений со смешанной битовой и целочисленной арифметикой // Моделирование и анализ информационных систем. 2023. Т. 30. № 2. С. 140–159.
14. Ceccato M., Tonella P., Basile C., Falcarin P., Torchiano M., Coppens B., De Sutter B. Understanding the behaviour of hackers while performing attack tasks in a professional setting and in a public challenge // Empirical Software Engineering. 2019. no. 24. pp. 240–286.
15. Collberg C., Thomborson C., Low D. A taxonomy of obfuscating transformations. Computer Science Technical Reports 148. Department of Computer Science, The University of Auckland, New Zealand. 1997. pp. 1–36.
16. Mohsen R., Pinto A.M. Evaluating obfuscation security: A quantitative approach // In International Symposium on Foundations and Practice of Security, Springer International Publishing. 2015. pp. 174–192.
17. Banescu S., Ochoa M., Pretschner A. A framework for measuring software obfuscation resilience against automated attacks // In Proceedings of the 1st International Workshop on Software Protection (SPRO '15). IEEE Press, Piscataway, NJ, USA. 2015. pp. 45–51.
18. Holder W., McDonald J.T., Andel T.R. Evaluating optimal phase ordering in obfuscation executives // Proceedings of the 7th Software Security, Protection, and Reverse Engineering/Software Security and Protection Workshop. 2017. pp. 1–12.
19. Collberg C. The Tigress C Diversifier/Obfuscator. 2016. URL: tigress.cs.arizona.edu/ (дата обращения: 23.06.2023).
20. Kosolapov Y.V., Borisov P.D. Similarity features for the evaluation of obfuscation effectiveness // In 2020 International Conference on Decision Aid Sciences and Application (DASA). 2020. pp. 898–902.
21. Borisov P.D., Kosolapov Y.V. On the Characteristics of Symbolic Execution in the Problem of Assessing the Quality of Obfuscating Transformations // Aut. Control Comp. Sci. 2022. vol. 56(7). pp. 595–605.
22. Xiao Y, Guo Y., Wang Y. Metrics for code obfuscation based on symbolic execution and N-scope complexity // Chinese Journal of Network and Information Security. 2022. vol. 8. no. 6. pp. 123–134.
23. Crescenzo G.D. Cryptographic program obfuscation: Practical solutions and application-driven models // Versatile Cybersecurity. 2018. pp. 141–167.
24. Gulwani S., Polozov O., Singh R. Program synthesis // Foundations and Trends in Programming Languages. 2017. vol. 4. no. 1-2. pp. 1–119.
25. Borisov P.D., Kosolapov Y.V. On the Automatic Analysis of the Practical Resistance of Obfuscating Transformations // Aut. Control Comp. Sci. 2020. vol. 54. pp. 619–629.
26. Walenstein A., El-Ramly M., Cordy J.R., Evans W.S, Mahdavi K., Pizka M., Ramalingam G., von Gudenberg J.W. Similarity in Programs // Duplication, Redundancy, and Similarity in Software, Dagstuhl Seminar Proceedings. 2007. vol. 6301. pp. 1–8.
27. Ceccato M., Di Penta M., Nagra J., Falcarin P., Ricca F., Torchiano M., Tonella P. The effectiveness of source code obfuscation: An experimental assessment // 17th International Conference on Program Comprehension, IEEE. 2009. pp. 178–187.
28. 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. pp. 1040–1074.

29. Борисов П.Д., Косолапов Ю.В. Способ оценки похожести программ методами машинного обучения // Труды Института системного программирования РАН. 2022. Т. 34. № 5. С. 63–76.
30. Naville Z. Hikari–an improvement over Obfuscator-LLVM. 2017. URL: https://github.com/HikariObfuscator/Hikari (дата доступа: 14.11.2023).
31. Junod P., Rinaldini J., Wehrli J., Michielin J. Obfuscator-LLVM–software protection for the masses // In Proc. of IEEE/ACM 1st International Workshop on Software Protection. 2015. pp. 3–9.
32. Haq I.U., Caballero J. A survey of binary code similarity // ACM Computing Surveys (CSUR). 2021. vol. 54. no. 3. pp. 1–38.
33. Pagani F., Dell'Amico M., Balzarotti D. Beyond precision and recall: understanding uses (and misuses) of similarity hashes in binary analysis // In Proc. of the Eighth ACM Conference on Data and Application Security and Privacy. 2018. pp. 354–365.
34. Ding S.H., Fung B.C., Charland P. Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization // In 2019 IEEE Symposium on Security and Privacy (SP). IEEE. 2019. pp. 472–489.
35. Garba P., Favaro M. Saturn-software deobfuscation framework based on llvm // In Proceedings of the 3rd ACM Workshop on Software Protection. 2019. pp. 27–38.
36. Dinaburg A., Ruef A. Mcsema: Static translation of x86 instructions to LLVM. ReCon 2014 Conference, Montreal, Canada. 2014.
37. Eyrolles N. Obfuscation with Mixed Boolean-Arithmetic Expressions: reconstruction, analysis and simplification tools. Doctoral dissertation. Universite Paris-Saclay, 2017. URL: https://theses.hal.science/tel-01623849/document (дата обращения: 14.07.2023).
38. Liang M., Li Z., Zeng Q., Fang Z. Deobfuscation of virtualization-obfuscated code through symbolic execution and compilation optimization // In International Conference on Information and Communications Security. Springer International Publishing, 2018. pp. 313–324.
39. Panchenko M., Auler R., Sakka L., Ottoni G. Lightning BOLT: powerful, fast, and scalable binary optimization // In Proceedings of the 30th ACM SIGPLAN International Conference on Compiler Construction. 2021. pp. 119–130.
40. Moreira A.A., Ottoni G., Quintao Pereira F.M. Vespa: static profiling for binary optimization // Proceedings of the ACM on Programming Languages. 2021. vol. 5. pp. 1–28.
41. Viticchie A., Regano L., Torchiano M., Basile C., Ceccato M., Tonella P., Tiella R. Assessment of source code obfuscation techniques // 16th international working conference on source code analysis and manipulation (SCAM), IEEE. 2016. pp. 11–20.
42. GCC, the GNU Compiler Collection. URL: https://gcc.gnu.org/ (дата обращения: 14.07.2023).
43. Clang: a C language family frontend for LLVM. URL: https://clang.llvm.org/ (дата обращения: 14.07.2023).
44. AMD Optimizing C/C++ and Fortran Compilers (AOCC). URL: https://developer.amd.com/amd-aocc/ (дата обращения: 14.07.2023).
45. Coreutils – GNU core utilities. URL: https://www.gnu.org/software/coreutils/ (дата обращения: 14.07.2023).
46. PolyBench/C – the Polyhedral Benchmark suite. URL: https://web.cse.ohio-state.edu/ pouchet.2/software/polybench/ (дата обращения: 14.07.2023).
47. HashCat – advanced password recovery. URL: https://hashcat.net/hashcat/ (дата обращения: 14.07.2023).

48. small-programs. A set of small programs for experiments with obfuscations. URL: https://github.com/Boriskin61/small-programs (дата обращения: 22.07.2023).
49. Куц Д.О. Метод моделирования косвенной адресации в рамках динамической символьной интерпретации. 2023. URL: https://www.ispras.ru/dcouncil/docs/diss/2023/kuc/dissertacija-kuc.pdf (дата обращения: 03.09.2023).
50. Лебедев Р.К. Автоматическая генерация хэш-функций для обфускации программного кода // Прикладная дискретная математика. 2020. № 50. С. 102–117.
51. Лебедев В.В. Деобфускация Control Flow Flattening средствами символьного исполнения // Прикладная дискретная математика. Приложение. 2021. № 14. С. 134–138.
52. BinShamlan M.H., Bamatraf M.A., Zain A.A. The impact of control flow obfuscation technique on software protection against human attacks // In 2019 First International Conference of Intelligent Computing and Engineering (ICOICE), IEEE. 2019. pp. 1–5.
53. Xu D. Opaque Predicate: Attack and Defense in Obfuscated Binary Code. Doctoral dissertation, 2018. URL: https://etda.libraries.psu.edu/files/final_submissions/17513 (дата обращения: 22.09.2023).
54. Sun Y. Software Protection Algorithm based on Control Flow Obfuscation // International Journal of Performability Engineering. 2018. vol. 14. no. 9. pp. 2181–2188.
55. Kim J., Kang S., Cho E.S., Paik J.Y. LOM: lightweight classifier for obfuscation methods // In Information Security Applications: 22nd International Conference, WISA 2021. Springer International Publishing. 2021. pp. 3–15.
56. Zhao Y., Tang Z., Ye G., Peng D., Fang D., Chen X., Wang Z. Semantics-aware obfuscation scheme prediction for binary. Computers Security. 2020. no. 99. pp. 1–17.
57. Wang C., Hill J., Knight J., Davidson J. Software tamper resistance: Obstructing static analysis of programs. Technical report CS-2000-12. Department of Computer Science, University of Virginia, USA. 2000.
58. Dullien T., Rolles R. Graph-based comparison of executable objects (english version) // Proceedings of the Symposium sur la Securite des Technologies de 'Information et des Communications. 2005. vol. 5. no. 1.

Борисов Петр Дмитриевич — заведующий лабораторией, ФГАНУ НИИ "Спецвузавтоматика". Область научных интересов: исследование и анализ программного кода, методы практической обфускации и деобфускации, способы оценки качества обфусцирующих преобразований. Число научных публикаций — 10. borisovpetr@mail.ru; улица Города Волос, 6, 344011, Ростов-на-Дону, Россия; р.т.: +7(863)201-2817.

Косолапов Юрий Владимирович — канд. техн. наук, доцент кафедры, кафедра алгебры и дискретной математики, Институт математики, механики и компьютерных наук им. И.И. Воровича Южного федерального университета (ЮФУ). Область научных интересов: помехоустойчивые коды в криптографии и стеганографии, теоретическая и практическая обфускация кода. Число научных публикаций — 100. yvkosolapov@sfedu.ru; улица Мильчакова, 8а, 344090, Ростов-на-Дону, Россия; р.т.: +7(863)297-5111.

DOI 10.15622/ia.23.3.3

P. Borisov, Yu. Kosolapov

A METHOD TO QUANTITATIVE COMPARE OBFUSCATING TTRANSFORMATIONS

Borisov P., Kosolapov Yu. **A Method to Quantitative Compare Obfuscating Ttransformations.**

Abstract. The paper considers the problem of quantitative comparison of potency and resistance of practically applied obfuscating transformations of program code. A method is proposed to find the potency and resistance of transformations by calculating the "comprehensibility"of the obfuscated and deobfuscated versions of a program, respectively. As a measure of program comprehensibility, it is proposed to use the similarity of this program to the approximation of its "most comprehensible"version. Based on the proposed method a model to assess potency and resistance was built, the main elements of which are: a set of investigated obfuscating transformations, a similarity function, a method to approximate the most comprehensible version of the program and a deobfuscator. To implement this model 1) obfuscating transformations provided by Hikari obfuscator are chosen; 2) 8 similarity functions are constructed by machine learning methods using static characteristics of programs from CoreUtils, PolyBench and HashCat sets; 3) the smallest program version was chosen as an approximation of the most comprehensible program version (found among the versions obtained using optimization options of GCC, Clang and AOCC compilers); 4) a program deobfuscation scheme based on the optimizing compiler from LLVM was built and implemented. The results of the potency and resistance for sequences of transformations of lengths one, two and three were experimentally obtained. These results showed consistency with the results of independent potency and resistance evaluations obtained by other methods. In particular, it was found that the highest potency and resistance are demonstrated by sequences of transformations starting with transformations of the control flow graph, and the lowest resistance and potency are generally demonstrated by sequences that do not contain such transformations.

Keywords: obfuscation, executable code, efficiency, resistance, similarity.

References

1. Akhunzada A., Sookhak M, Anuar A.B., Gani A., Ahmed E., Shiraz M., Furnell S., Hayat A., Khan M.K. Man-At-The-End attacks: Analysis, taxonomy, human aspects, motivation and future directions. Journal of Network and Computer Applications. 2015. vol. 48. pp. 44–57.
2. Undrits R., Resende J. et. al. CyberSec4Europe D3.23: Cybersecurity Outlook 2. Research Report D3.23. 2022. pp. 1–82.
3. Biernacki L., Gallagher M., Xu Z., Aga M.T., Harris A., Wei S., Tiwari M., Kasikci B., Malik S., Austin T.. Software-driven security attacks: From vulnerability sources to durable hardware defenses. ACM Journal on Emerging Technologies in Computing Systems (JETC). 2021. vol. 17. no. 3. pp. 1–38.
4. Varnovsky N.P., Zakharov V.A., Kuzurin N.N., Shokurov V.A. [The current state of art in program obfuscations: definitions of obfuscation security]. Trudy Instituta sistemnogo programmirovaniya RAN – Proceedings of the Institute for System Programming of the RAS (Proceedings of ISP RAS). 2014. vol. 26. no. 3 pp. 167–198. (In Russ.).
5. Barak B., Goldreich O., Impagliazzo R., Rudich S., Sahai A., Vadhan S., Yang K. On the (im) possibility of obfuscating programs.Journal of the ACM (JACM). 2012. vol. 59. no. 2. pp. 1–42.

6. Zobernig L. Mathematical Aspects of Program Obfuscation. Doctoral dissertation. ResearchSpace@ Auckland, 2020. Available at: www.math.auckland.ac.nz/~ sgal018/Lukas-Zobernig-Thesis.pdf (accessed 19.06.2023).
7. Garg S., Gentry C., Halevi S., Raykova M., Sahai A., Waters B. Candidate indistinguishability obfuscation and functional encryption for all circuits. SIAM Journal on Computing. 2016. vol. 45. no. 3. pp. 882–929.
8. Kochberger P., Schrittwieser S., Coppens B., De Sutter B. Evaluation Methodologies in Software Protection Research. arXiv preprint arXiv:2307.07300. 2023. pp. 1–67.
9. Zhou Y., Main A., Gu Y.X., Johnson H. Information hiding in software with mixed boolean-arithmetic transforms. Information Security Applications: 8th International Workshop on Information Security Applications. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007. vol. 4867. pp. 61–75.
10. Reichenwallner B., Meerwald-Stadler P. Efficient Deobfuscation of Linear Mixed Boolean-Arithmetic Expressions. Proceedings of the 2022 ACM Workshop on Research on offensive and defensive techniques in the context of Man At The End (MATE) attacks. 2022. pp. 19–28.
11. Xu D., Liu D., Feng W., Ming J., Zheng Q., Li J., Yu Q. Boosting SMT solver performance on mixed-bitwise-arithmetic expressions. Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. 2021. pp. 651–664.
12. Liu B., Shen J., Ming J., Zheng Q., Li J., Xu D. MBA-Blast: Unveiling and Simplifying Mixed Boolean-Arithmetic Obfuscation. Proceedings of the 30th USENIX Security Symposium. 2021. pp. 1701–1718.
13. Kosolapov Yu.V. [On Simplifying Expressions with Mixed Boolean-Arithmetic]. Modelirovanie i analiz informacionnyh sistem – Modeling and Analysis of Information Systems. 2023. vol. 30. no. 2. pp. 140–159. (In Russ.).
14. Ceccato M., Tonella P., Basile C., Falcarin P., Torchiano M., Coppens B., De Sutter B. Understanding the behaviour of hackers while performing attack tasks in a professional setting and in a public challenge. Empirical Software Engineering. 2019. no. 24. pp. 240–286.
15. Collberg C., Thomborson C., Low D. A taxonomy of obfuscating transformations. Computer Science Technical Reports 148. Department of Computer Science, The University of Auckland, New Zealand. 1997. pp. 1–36.
16. Mohsen R., Pinto A.M. Evaluating obfuscation security: A quantitative approach. In International Symposium on Foundations and Practice of Security, Springer International Publishing. 2015. pp. 174–192.
17. Banescu S., Ochoa M., Pretschner A. A framework for measuring software obfuscation resilience against automated attacks. In Proceedings of the 1st International Workshop on Software Protection (SPRO '15). IEEE Press, Piscataway, NJ, USA. 2015. pp. 45–51.
18. Holder W., McDonald J.T., Andel T.R. Evaluating optimal phase ordering in obfuscation executives. Proceedings of the 7th Software Security, Protection, and Reverse Engineering/Software Security and Protection Workshop. 2017. pp. 1–12.
19. Collberg C. The Tigress C Diversifier/Obfuscator. 2016. Available at: tigress.cs.arizona.edu/ (accessed 23.06.2023).
20. Kosolapov Y.V., Borisov P.D. Similarity features for the evaluation of obfuscation effectiveness. In 2020 International Conference on Decision Aid Sciences and Application (DASA). 2020. pp. 898–902.
21. Borisov P.D., Kosolapov Y.V. On the Characteristics of Symbolic Execution in the Problem of Assessing the Quality of Obfuscating Transformations. Aut. Control Comp. Sci. 2022. vol. 56(7). pp. 595–605.

22. Xiao Y, Guo Y., Wang Y. Metrics for code obfuscation based on symbolic execution and N-scope complexity. Chinese Journal of Network and Information Security. 2022. vol. 8. no. 6. pp. 123–134.
23. Crescenzo G.D. Cryptographic program obfuscation: Practical solutions and application-driven models. Versatile Cybersecurity. 2018. pp. 141–167.
24. Gulwani S., Polozov O., Singh R. Program synthesis. Foundations and Trends in Programming Languages. 2017. vol. 4. no. 1-2. pp. 1–119.
25. Borisov P.D., Kosolapov Y.V. On the Automatic Analysis of the Practical Resistance of Obfuscating Transformations. Aut. Control Comp. Sci. 2020. vol. 54. pp. 619–629.
26. Walenstein A., El-Ramly M., Cordy J.R., Evans W.S, Mahdavi K., Pizka M., Ramalingam G., von Gudenberg J.W. Similarity in Programs. Duplication, Redundancy, and Similarity in Software, Dagstuhl Seminar Proceedings. 2007. vol. 6301. pp. 1–8.
27. Ceccato M., Di Penta M., Nagra J., Falcarin P., Ricca F., Torchiano M., Tonella P. The effectiveness of source code obfuscation: An experimental assessment. 17th International Conference on Program Comprehension, IEEE. 2009. pp. 178–187.
28. 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. pp. 1040–1074.
29. Borisov P.D., Kosolapov Yu.V. [Method to Evaluate Program Similarity Using Machine Learning Methods]. Trudy Instituta sistemnogo programmirovaniya RAN – Proceedings of the Institute for System Programming of the RAS (Proceedings of ISP RAS). 2023. vol. 34. no. 5. pp. 63–76. (In Russ.).
30. Naville Z. Hikari–an improvement over Obfuscator-LLVM. 2017. Available at: https://github.com/HikariObfuscator/Hikari (accessed 14.11.2023).
31. Junod P., Rinaldini J., Wehrli J., Michielin J. Obfuscator-LLVM–software protection for the masses. In Proc. of IEEE/ACM 1st International Workshop on Software Protection. 2015. pp. 3–9.
32. Haq I.U., Caballero J. A survey of binary code similarity. ACM Computing Surveys (CSUR). 2021. vol. 54. no. 3. pp. 1–38.
33. Pagani F., Dell'Amico M., Balzarotti D. Beyond precision and recall: understanding uses (and misuses) of similarity hashes in binary analysis. In Proc. of the Eighth ACM Conference on Data and Application Security and Privacy. 2018. pp. 354–365.
34. Ding S.H., Fung B.C., Charland P. Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In 2019 IEEE Symposium on Security and Privacy (SP). IEEE. 2019. pp. 472–489.
35. Garba P., Favaro M. Saturn-software deobfuscation framework based on llvm. In Proceedings of the 3rd ACM Workshop on Software Protection. 2019. pp. 27–38.
36. Dinaburg A., Ruef A. Mcsema: Static translation of x86 instructions to LLVM. ReCon 2014 Conference, Montreal, Canada. 2014.
37. Eyrolles N. Obfuscation with Mixed Boolean-Arithmetic Expressions: reconstruction, analysis and simplification tools. Doctoral dissertation. Universite Paris-Saclay, 2017. Available at: https://theses.hal.science/tel-01623849/document (accessed 14.07.2023).
38. Liang M., Li Z., Zeng Q., Fang Z. Deobfuscation of virtualization-obfuscated code through symbolic execution and compilation optimization. In International Conference on Information and Communications Security. Springer International Publishing, 2018. pp. 313–324.
39. Panchenko M., Auler R., Sakka L., Ottoni G. Lightning BOLT: powerful, fast, and scalable binary optimization. In Proceedings of the 30th ACM SIGPLAN International Conference on Compiler Construction. 2021. pp. 119–130.

40. Moreira A.A., Ottoni G., Quintao Pereira F.M. Vespa: static profiling for binary optimization. Proceedings of the ACM on Programming Languages. 2021. vol. 5. pp. 1–28.
41. Viticchie A., Regano L., Torchiano M., Basile C., Ceccato M., Tonella P., Tiella R. Assessment of source code obfuscation techniques. 16th international working conference on source code analysis and manipulation (SCAM), IEEE. 2016. pp. 11–20.
42. GCC, the GNU Compiler Collection. Available at: https://gcc.gnu.org/ (accessed 14.07.2023).
43. Clang: a C language family frontend for LLVM. Available at: https://clang.llvm.org/ (accessed 14.07.2023).
44. AMD Optimizing C/C++ and Fortran Compilers (AOCC). Available at: https://developer.amd.com/amd-aocc/ (accessed 14.07.2023).
45. Coreutils – GNU core utilities. Available at: https://www.gnu.org/software/coreutils/ (accessed 14.07.2023).
46. PolyBench/C – the Polyhedral Benchmark suite. Available at: https://web.cse.ohio-state.edu/ pouchet.2/software/polybench/ (accessed 14.07.2023).
47. HashCat – advanced password recovery. Available at: https://hashcat.net/hashcat/ (accessed 14.07.2023).
48. small-programs. A set of small programs for experiments with obfuscations. Available at: https://github.com/Boriskin61/small-programs (accessed 22.07.2023).
49. Kutz D.O. Method for modeling indirect addressing within dynamic symbolic interpretation. Doctorial dissertation, Moscow, 2023. Available at: https://www.ispras.ru/dcouncil/docs/diss/2023/kuc/dissertacija-kuc.pdf (accessed 03.09.2023).
50. Lebedev R.K. [Automatic generation of hash functions for program code obfuscation]. Prikladnaya Diskretnaya Matematika – Applied Discrete Mathematics. 2020. no. 50. pp. 102–117. (In Russ.).
51. Lebedev V.V. [Control Flow Flattening deobfuscation using symbolic execution]. Prikladnaya Diskretnaya Matematika – Applied Discrete Mathematics. 2021. no. 14. pp. 134–138. (In Russ.).
52. BinShamlan M.H., Bamatraf M.A., Zain A.A. The impact of control flow obfuscation technique on software protection against human attacks. In 2019 First International Conference of Intelligent Computing and Engineering (ICOICE), IEEE. 2019. pp. 1–5.
53. Xu D. Opaque Predicate: Attack and Defense in Obfuscated Binary Code. Doctoral dissertation, 2018. Available at: https://etda.libraries.psu.edu/files/final_submissions/17513 (accessed 22.09.2023).
54. Sun Y. Software Protection Algorithm based on Control Flow Obfuscation. International Journal of Performability Engineering. 2018. vol. 14. no. 9. pp. 2181–2188.
55. Kim J., Kang S., Cho E.S., Paik J.Y. LOM: lightweight classifier for obfuscation methods. In Information Security Applications: 22nd International Conference, WISA 2021. Springer International Publishing. 2021. pp. 3–15.
56. Zhao Y., Tang Z., Ye G., Peng D., Fang D., Chen X., Wang Z. Semantics-aware obfuscation scheme prediction for binary. Computers Security. 2020. no. 99. pp. 1–17.
57. Wang C., Hill J., Knight J., Davidson J. Software tamper resistance: Obstructing static analysis of programs. Technical report CS-2000-12. Department of Computer Science, University of Virginia, USA. 2000.
58. Dullien T., Rolles R. Graph-based comparison of executable objects (english version). Proceedings of the Symposium sur la Securite des Technologies de'Information et des Communications. 2005. vol. 5. no. 1.

Borisov Petr — Head of the laboratory, FSASE SRI "Specvuzavtomatika". Research interests: program code analysis, obfuscation and deobfuscation, methods for assessing the quality of obfuscating transformations. The number of publications — 10. borisovpetr@mail.ru; 6, Goroda of Volos St., 344011, Rostov-on-Don, Russia; office phone: +7(863)201-2817.

Kosolapov Yury — Ph.D., Associate professor of the department, Department of algebra and discrete mathematics, I.I. Vorovich Institute of Mathematics, Mechanics and Computer Sciences of the Southern Federal University (SFedU). Research interests: error-correcting codes in cryptography and steganography, theoretical and practical code obfuscation. The number of publications — 100. yvkosolapov@sfedu.ru; 8a, Milchakova St., 344090, Rostov-on-Don, Russia; office phone: +7(863)297-5111.