14. *Pljonkin A., Petrov D., Sabantina L., Dakhkilgova K.* Nonclassical attack on a quantum keydistribution system, *Entropy,* 2021, Vol. 23, No. 5.
15. *Pljonkin A., Rumyantsev K., Kumar Singh P.* Synchronization in quantum key distribution systems, *Cryptography,* 2017, Vol. 1, No. 3, pp. 18.
16. *Gal'yardi R.M., Karp Sh.* Opticheskaya svyaz' [Optical communications]: trans. from engl, ed. by A.G. Sheremet'eva. Moscow: Svyaz', 1978, 424 p.
17. *Rumyantsev K.E., Rudinskiy E.A.* Dvukhetapnyy vremennoy algoritm sinkhronizatsii v sisteme kvantovogo raspredeleniya klyucha s avtomaticheskoy kompensatsiey polyarizatsionnykh iskazheniy [Two-stage time synchronization algorithm in a quantum key distribution system with automatic compensation of polarization distortions], *Izvestiya YuFU. Tekhnicheskie nauki* [Izvestiya SFedU. Engineering Sciences], 2017, No. 5 (190), pp. 75-89.
18. *Prudnikov V., Plenkin A., Yushitsyna V.* Kvantovo-kriptograficheskie seti [Quantum cryptographic networks], Litres, 2024.
19. *Rumyantsev K.E., Mironov Ya.K., Mironova P.D.* Sravnitel'nyy analiz vremennykh kharakteristik algoritmov obnaruzheniya sinkhroimpul'sa v sisteme kvantovogo raspredeleniya klyucha [Comparative analysis of temporal characteristics of sync pulse detection algorithms in a quantum key distribution system], *IV Vserossiyskaya nauchno-prakticheskaya konferentsiya "Digital Era", Groznyy, 01 marta 2024 goda* [IV All-Russian scientific and practical conference "Digital Era", Grozny, March 01, 2024]. Groznyy: Chechenskiy gosudarstvennyy universitet imeni Akhmata Abdulkhamidovicha Kadyrova, 2024, pp. 139-141.
20. *Miller A.V.* Sinkhronizatsiya vremeni v sputnikovom kvantovom raspredelenii klyuchey [Time synchronization in satellite quantum key distribution], *Problemy peredachi informatsii* [Problems of Information Transmission], 2023, Vol. 59, No. 4, pp. 13-27.
21. *Andreev S.A., Svistunova A.I.* Sistemy sinkhronizatsii dlya kvantovogo kanala svyazi v otkrytom prostranstve [Synchronization systems for a quantum communication channel in open space], *Nauka, tekhnika, pedagogika v vysshey shkole: Materialy Vserossiyskoy nauchno-prakticheskoy konferentsii, Moskva, 20–27 fevralya 2023 goda* [Science, technology, pedagogy in higher education: Proceedings of the All-Russian scientific and practical conference, Moscow, February 20-27, 2023]. Moscow: Moskovskiy Politekh, 2023, pp. 398-404.

Плёнкин Антон Павлович – Южный федеральный университет; e-mail: pljonkin@sfedu.ru; г. Таганрог, Россия; тел.: 89054592158; кафедра ИБТКС; к.т.н.; доцент.

Pljonkin Anton Pavlovich –Southern Federal University; e-mail: pljonkin@sfedu.ru; Taganrog, Russia; phone: +79054592158; the Department of Information Security of Telecommunication Systems; cand. of eng. sc.; associate professor.

УДК 004.089 DOI 10.18522/2311-3103-2025-3-264-273

П.Д. Борисов, Ю.В. Косолапов

О ФУНКЦИИ ПОХОЖЕСТИ ГРАФИЧЕСКИХ ПРЕДСТАВЛЕНИЙ ИСПОЛНЯЕМЫХ ФАЙЛОВ В МОДЕЛИ ОЦЕНКИ ОБФУСЦИРУЮЩИХ ПРЕОБРАЗОВАНИЙ

Обфускация программного кода используется с целью затруднения его анализа в модели, когда аналитик имеет полный доступ к программе. Обычно обфускация делится на криптографически стойкую и эвристически стойкую. В первом случае сложность анализа сопоставима с трудностью решения некоторой известной математической задачи. Во втором случае стойкость обосновывается, как правило, отсутствием известных на момент создания метода обфускации эффективных техник ее анализа. Криптографически стойкая обфускация пока не нашла применения на практике, в то время как эвристически стойкая широко применяется. Ранее авторами была предложена модель оценки эффективности и стойкости эвристических обфусцирующих преобразований, в основе которой лежит применение функции похожести. В настоящей работе с помощью методов машинного обучения строится такая функция похожести на основе сравнения графического представления исполняемых файлов программ. В частности, сравнение выполняется с помощью сверточной сети с четырьмя сверточными слоями, оптимизатором RMSprop, функцией потерь NLLLoss и двумя выходами полносвязного слоя. Предложенная функция применяется в

рамках реализации модели оценки эффективности и стойкости обфусцирующих преобразований. Кроме функции похожести, реализация модели также включает: базовый набор обфусцирующих преобразований, предоставляемых обфускатором Hikari; набор последовательностей обфусцирующих преобразований на основе базового набора; тестовое множество программ для обучения моделей, построенное на основе наборов CoreUtils, PolyBench и HashCat; аппроксимацию самой "понятной" версии программы с помощью наименьшей по размеру версии программы (ищется среди версий, полученных с помощью различных опций оптимизации компиляторов GCC, Clang и AOCC); схему деобфускации программ на основе оптимизирующего компилятора из состава LLVM. Результаты экспериментального исследования с реализованной моделью показали, что построенную функцию похожести применять в рамках модели оценки нецелесообразно из-за ее невысокой точности, но возможно ее применение при построении более сложных функций.

Оценка эффективности и стойкости обфусцирующих преобразований; графическое представление исполняемых файлов; функция похожести.

P.D. Borisov, Yu.V. Kosolapov

ON THE SIMILARITY FUNCTION OF GRAPHIC REPRESENTATIONS OF EXECUTIVE FILES IN THE OBFUSCING TRANSFORMATION EVALUATION MODEL

Obfuscation of program code is used to complicate its analysis in a model when the analyst has full access to the program. Obfuscation is usually divided into cryptographically secure and heuristically resistant. In the first case, the complexity of the analysis is comparable to the difficulty of solving some known mathematical problem. In the second case, the resistance is usually justified by the lack of effective techniques for analyzing the obfuscation method known at the time of its creation. Cryptographically secure obfuscation has not yet found practical application, while heuristically resistant is widely used. Previously, the authors proposed a model for assessing the efficiency and resistance of heuristic obfuscating transformations based on the use of a similarity function. In this paper, such a similarity function is constructed using machine learning methods based on a comparison of the graphical representation of program executable files. In particular, the comparison is performed using a convolutional network with four convolutional layers, an RMSprop optimizer, an NLLLoss loss function, and two outputs of a fully connected layer. The proposed function is used in the implementation of a model for evaluating the efficiency and resistance of obfuscating transformations. In addition to the similarity function, the implementation of the model also includes: a basic set of obfuscating transformations provided by the Hikari obfuscator; a set of obfuscating transformation sequences based on the basic set; a test set of programs for training models based on the CoreUtils, PolyBench and HashCat program sets; approximation of the most "understandable" version of the program using the smallest version of the program (searched among the versions obtained using various optimization options of the GCC, Clang and AOCC compilers); a program deobfuscation scheme based on the optimizing compiler from LLVM. The results of an experimental study with the implemented model showed that it is impractical to use the constructed similarity function in the framework of the evaluation model due to its low accuracy, but it is possible to use it when constructing more complex functions.

Evaluation of the effectiveness and resilience of obfuscating transformations; graphical representation of executable files; similarity function.

Введение. Защита программного обеспечения (далее – ПО) от исследования, изменения данных и алгоритмов была и остается актуальной задачей. Целью такой защиты может быть сокрытие хранимых в ПО криптографических ключей, препятствие поиску и эксплуатации уязвимостей ПО, защита от нелегитимного использования лицензированного ПО (включая модификацию кода и компонентов), защита от мошенничества в компьютерных играх и т.п. Предполагается, что аналитик, исследующий программу, не ограничен по времени, в выборе средств и способов исследования, более того, аналитик может приобрести полную лицензированную версию ПО, обладающую всеми интересующими его данными и алгоритмами (модель угроз MATE, сокр. от англ. Man At The End). В такой модели одним из средств защиты является обфускация – изменение исходного кода или исполняемого образа программы, сохраняющее ее исходную функциональность, но затрудняющее ее анализ, понимание реализованных в ней алгоритмов, а также их модификацию.

С позиции надежности методы обфускации можно разделить на доказуемо надежные и эвристически надежные [1]. Первые гарантируют надежность запутывания ряда программ (отдельных классов программ [2–4]) по отношению к любому полиномиально ограниченному аналитику, а вторые нацелены на затруднение анализа существующими на текущий момент времени средствами. Пока на практике чаще применяются эвристические методы обфускации из-за их универсальности (применимости к любым алгоритмам). Кроме того, применение доказуемо надежных методов обфускации приводит в ряде случаев к сильному разрастанию даже небольших программ и росту времени выполнения обфусцированного кода [5]. С другой стороны, разработано большое количество методов обфускации (например, [6–8]), но эффективность таких методов часто или не оценивается, или не учитывает всех аспектов обратного проектирования программы аналитиком [9].

Поэтому важной и актуальной остается проблема автоматизированной (без привлечения аналитиков) количественной оценки эффективности и стойкости методов обфускации. Обфускация для компилируемых программ может применяться на разных уровнях (на уровне исходного кода, уровне промежуточного представления, уровне бинарного кода). Соответственно, в рамках модели MATE, когда аналитику доступны только исполняемые файлы программы, оценку эффективности и стойкости эвристических методов обфускации представляется уместным выполнять именно для исполняемого файла обфусцированной программы. Отметим, что попытки выполнить оценку на уровне бинарного кода уже предпринимались: для обфусцирующих преобразований, применяемых на уровне исходного кода – в работе [10], а для преобразований на уровне бинарного кода – в работе [11].

Известен ряд практических способов и метрик для оценки эффективности эвристических обфусцирующих преобразований, а также их стойкости к анализу и пониманию программ. Относительно недавно вышел обзор, в котором была рассмотрена 571 работа, посвященная защите ПО с помощью обфускации программного кода [9]. В обзоре отмечается, что в большинстве работ посвященных обфускации в первую очередь оценивается влияние на быстродействие программы (стоимость). Оценка эффективности обфускации производится значительно реже, при этом используются наборы тестовых программ, которые или не доступны публично, или не обладают достаточным разнообразием. Такие вопросы как "На сколько можно доверять обфускации?" и "Как строить стойкие методы обфускации?" остаются без ответа [3]. В частности, авторы [3], отметили, что данные вопросы не решены для компилируемых программ. К нерешенным также можно отнести вопрос "На сколько можно доверять средствам обфускации, реализующим существующие техники запутывания кода?".

В работе [12] отмечено, что ответы на поставленные вопросы также зависят от цели аналитика и имеющихся у него средств анализа. Таким образом задача разработки и/или совершенствования способов количественной оценки эффективности и стойкости обфусцирующих преобразований, учитывающих статические и динамические характеристики программ, при наличии подходящих средств анализа у противника была и является актуальной. Способы оценки обфусцирующих преобразований должны учитывать широкое разнообразие техник, применяемых аналитиками при исследовании программ (например, дизассемблирование, отладку, анализ в виртуальной среде, символьное исполнение).

В настоящей работе используется предложенная ранее в [13] модель количественной оценки эффективности и стойкости обфусцирующих преобразований, для которой в рамках настоящей работы предложена новая функция похожести. В разделе 0 кратко приводится модель количественной оценки. В разделе 0 описывается новая функция похожести, а в разделе 0 представлены результаты экспериментальных исследований с новой функцией похожести.

Модель количественной оценки эффективности и стойкости обфусцирующих преобразований. Рассмотрим множество $\mathcal{P}_{\mathcal{P}}$ программ, полученных из программы P с помощью преобразований, сохраняющих семантику. Пусть $P_0 \in \mathcal{P}_{\mathcal{P}}$ – 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" [14]. С другой стороны, выбор P_0 может быть основан на сложности символьной интерпретации машинного кода [15, 16], когда в качестве P_0 может рассматриваться версия программы с наименьшим временем символьного исполнения. В [17] такой подход обоснован тем, что символьное исполнение можно рассматривать как модель динамического исследования программы аналитиком-человеком.

Понятность программы P будем рассматривать, как величину похожести программы P на P_0. Для этого рассмотрим функцию похожести

$$\delta: \mathcal{P} \times \mathcal{P} \to [0,1](\subseteq \mathrm{R}),$$

где *0* соответствует наименьшей степени похожести, а *1* – наибольшей. Также будем полагать, что $\delta(\mathrm{P}_1, \mathrm{P}_2) = \delta(\mathrm{P}_2, \mathrm{P}_1)$ для всех $(\mathrm{P}_1, \mathrm{P}_2) \in \mathcal{P} \times \mathcal{P}$ и $\delta(\mathrm{P}, \mathrm{P}) = 0$ для всех P из $\mathcal{P}$. В этом случае для фиксированных $\mathrm{P}_0 \in \mathcal{P}_{\mathcal{P}}$ и δ под *понятностью* программы $\mathrm{P} \in \mathcal{P}_{\mathcal{P}}$ можно подразумевать величину $\delta(\mathrm{P}, \mathrm{P}_0)$, как степень похожести на самую "понимаемую" версию программы.

Заметим, что поиск самой короткой программы или поиск программы с наименьшим временем символьного исполнения, являются вычислительно сложными задачами. Поэтому вместо P_0 предлагается использовать ее *аппроксимацию* $\mathrm{A}(P_0)$, найденную по P. Аналогичный подход применяется в [18], где при количественной оценке эффективности обфусцирующих преобразований вместо невычислимой Колмогоровской сложности программы применяется ее аппроксимация результатом сжатия программы. В качестве аппроксимации $\mathrm{A}(P_0)$ может быть выбрана, например, наименьшая по размеру версия программы P, полученная с помощью доступного набора оптимизирующих преобразований компиляторов. Представляется, что аппроксимацию также возможно построить на основе характеристик символьной интерпретации: например, в качестве $A(P_0)$ может быть выбрана версия программы (например, среди версий, полученных с помощью разных компиляторов и разных опций компиляции) с наименьшим временем символьной интерпретации.

Таким образом *понятностью* программы $\mathrm{P} \in \mathcal{P}_{\mathcal{P}}$ при фиксированных δ и $\mathrm{A}(\mathrm{P}_0) \in \mathcal{P}_{\mathcal{P}}$ назовем величину

$$\mathcal{C}(\mathrm{P}) = \delta\big(\mathrm{P}, \mathrm{A}(\mathrm{P}_0)\big) \in [0,1]. \quad (1)$$

Пусть $\mathcal{Q}$ – исследуемое множество последовательностей обфусцирующих преобразований, построенное на основе базового набора преобразований $\mathcal{O}$. С помощью характеристики (1) определим эффективность e обфусцирующего преобразования $O^t \in \mathcal{Q}$, примененного к программе P, а также стойкость r этого преобразования по отношению к деобфускатору D следующим образом:

$$\mathrm{e}(\mathrm{O}^{\mathrm{t}}, \mathrm{P}) = 1 - \mathcal{C}\big(\mathrm{O}^{\mathrm{t}}(\mathrm{P})\big) = 1 - \delta\big(\mathrm{O}^{\mathrm{t}}(\mathrm{P}), \mathrm{A}(\mathrm{P}_0)\big), \quad (2)$$

$$\mathrm{r}(\mathsf{D}, \mathrm{O}^{\mathrm{t}}, \mathrm{P}) = 1 - \mathcal{C}\left(\mathsf{D}\big(\mathrm{O}^{\mathrm{t}}(\mathrm{P})\big)\right) = 1 - \delta\left(\mathsf{D}\big(\mathrm{O}^{\mathrm{t}}(\mathrm{P})\big), \mathrm{A}(\mathrm{P}_0)\right). \quad (3)$$

В рамках определений (2) и (3), задача выбора наиболее эффективного обфусцирующего преобразования для P решается так: $\mathrm{O}_1^{\mathrm{t}}$ эффективнее $\mathrm{O}_2^{\mathrm{t}}$, если

$$\mathrm{e}(\mathrm{O}_1^{\mathrm{t}}, \mathrm{P}) > \max\{1 - \mathcal{C}(\mathrm{P}), \mathrm{e}(\mathrm{O}_2^{\mathrm{t}}, \mathrm{P})\}.$$

Задача выбора среди этих же преобразований наиболее стойкого по отношению к деобфускатору D решается так: $\mathrm{O}_1^{\mathrm{t}}$ является D-устойчивее $\mathrm{O}_2^{\mathrm{t}}$, если

$$\mathrm{r}(\mathsf{D}, \mathrm{O}_1^{\mathrm{t}}, \mathrm{P}) > r(\mathsf{D}, \mathrm{O}_2^{\mathrm{t}}, \mathrm{P}).$$

Набор $\mathcal{M} = (\mathcal{O}, \mathcal{Q}, \delta, \mathsf{D}, \mathsf{A})$ с определенными в соответствии с (2) и (3) функциями e и r называется *моделью оценки эффективности и стойкости обфусцирующих преобразований*. Для реализации этой модели необходимо зафиксировать набор $\mathcal{O}$, множество $\mathcal{Q}$, выбрать функцию похожести δ исполняемых файлов программ, деобфускатор D и способ аппроксимирования A самой понятной программы $\mathrm{P}_0 \in \mathcal{P}_{\mathcal{P}}$ для каждого P.

Для зафиксированного алгоритма A понятность программы P зависит от выбора функции похожести δ, определенной на парах программ. По этой причине поиск *подходящих* и *эффективно вычислимых* функций похожести, а также установление корреляции

между различными функциями являются актуальными задачами. Далее предлагается функция похожести, построенная на основе сравнения графических представлений исполняемых файлов программ.

Функция похожести графических представлений файлов. В настоящем разделе строится функция похожести δ_{im} исполняемых файлов на основе их представления в виде изображений. Именно, при таком подходе исполняемый файл программы P представляется в виде двумерного изображения $Im(P)$ в оттенках серого цвета, в котором яркость серого пикселя определяется значением соответствующего байта в исполняемом коде: минимальное значение 0 соответствует черному цвету пикселя, а максимальное значение 255 – белому. Отметим, что графическое представление исполняемых файлов не является новым подходом в области информационной безопасности и часто используется при классификации вредоносного программного обеспечения (например, в [19–21] и [22]).

Опишем предлагаемый в настоящей работе способ построения функции похожести δ_{im} для исполняемых файлов. Сначала уточним способ представления файлов в графическом виде, используемый в работе. Исполняемый файл программы P размера b байтов преобразуется в квадратное изображение $Im(P)$ размера $b_i \times b_i$, где b_i – наибольшее целое число, для которого $b_i^2 \leq b$, после чего изображение приводится к размеру 512×512 (растягивается или сжимается). Последнее преобразование выполняется с целью обеспечения возможности сравнения исполняемых файлов разного размера. Таким образом, все исполняемые файлы преобразуются в изображения размера 512×512 в оттенках серого.

Способ сравнения исполняемых файлов с графическим представлением основан на применении сверточной сети, которая обучается на множестве пар сравниваемых программ, а результатом ее работы являются два числа от 0 до 1, в сумме дающие единицу. Из этих двух чисел первое характеризует уверенность сети в том, что пара программ функционально идентична, а второе характеризует уверенность сети в том, что сравниваемые программы функционально разные. Одной паре сравниваемых программ (P_1, P_2) из множества пар соответствует изображение $Im(P_1, P_2) = Im(P_1) - Im(P_2)$ размера 512×512, где побайтовое вычитание выполняется по модулю 256 (в кольце Z_{256}). Множество пар состоит из двух подмножеств: множества пар функционально одинаковых программ и множества пар функционального разных программ. Пары из первого множества имеют метку 0, а пары из второго – метку 1. Целью обучения сети является выделение (визуальных) признаков, позволяющих судить о функциональной идентичности двух сравниваемых программ. Таким образом, значением функции $\delta_{im}(P_1, P_2)$ для пары программы (P_1, P_2) является первое значение из двух, возвращаемых обученной сетью, то есть значением является число от 0 до 1 – степень уверенности сети в том, что сравниваемые программы функционально идентичны. Далее вместо $\delta_{im}(P_1, P_2)$ будет использоваться запись $\delta_{im}\big(Im(P_1, P_2)\big)$, подчеркивающая, что при вычислении похожести программ P_1 и P_2 используется разность их графических представлений.

Для реализации предложенного способа за основу сети взята структура сверточной сети из [20] с четырьмя сверточными слоями (модель 2 в статье [20]), оптимизатором RMSprop, функцией потерь NLLLoss и двумя выходами полносвязного слоя. Формирование множества пар функционально одинаковых и разных программ реализовано на основе построенного в [13] множества исполняемых файлов. Именно, множество пар формируется на основе наборов программ CoreUtils[1], PolyBench[2] и HashCat[3] (всего 164 программы), собранных девятью компиляторами – GCC (версий 7.5.0, 8.4.0, 9.4.0, 10.3.0), Clang (версий 7.0.1, 8.0.1, 9.0.1, 10.0.0) и AOCC[4] (версии 3.0.0) – с пятью опциями оптимизации O0, O1, O2, O3 и Os. Таким образом, множество исполняемых файлов, постро-

[1] https://github.com/coreutils/coreutils (дата обращения: 13.04.2025).
[2] https://github.com/MatthiasJReisinger/PolyBenchC-4.2.1 (дата обращения: 13.04.2025).
[3] https://github.com/hashcat/hashcat-utils (дата обращения: 13.04.2025).
[4] https://www.amd.com/en/developer/aocc.html (дата обращения: 13.04.2025).

енное в [13], состоит из $164 \cdot 9 \cdot 5 = 7380$ файлов. Подмножество пар функционально одинаковых программ строилось по следующей схеме: 1) случайно выбиралась одна из 164-х программ, 2) для выбранной программы строились две случайные конфигурации (*компилятор*, *опция оптимизации*); шаги 1) и 2) повторялись для построения 10 тысяч уникальных пар. Подмножество пар функционально разных программ строилось путем построения 10 тысяч пар уникальных случайных конфигураций (*компилятор*, *опция оптимизации*, *программа*) с условием, чтобы в конфигурациях каждой пары не было одинаковых программ. Обучающая выборка содержала 50% пар из каждого подмножества, а контрольная и тестовая выборки – по 25% пар. Результаты обучения сети в течение 100 эпох показали точность (accuracy) 0.713. Попытки увеличить точность модификацией оптимизатора, функции потерь или увеличением числа эпох к положительному результату не привели.

Оценка обфусцирующих преобразований. В настоящей работе оценка обфусцирующих преобразований выполняется в соответствии с моделью, описанной в разделе 0, когда $\delta = \delta_{im}$, а остальные параметры модели выбраны такими же, как и в [13]. Именно, базовым набором обфусцирующих преобразований $\mathcal{O}$ здесь является набор из 7 преобразований, предоставляемых обфускатором Hikari[5].

В табл. 1 перечислены обозначения (столбец o) и описание этих преобразований, а также указан соответствующий тип $\theta(o)$ преобразований на основе определений, введенных для обфускатора Tigress в [15]. Набор Q представляет собой набор из 63-х последовательностей O^t, состоящих из одного, двух и трех разных преобразований, входящих в базовый набор (полный перечень обфусцирующих последовательностей можно найти в [13], табл. 6). В качестве деобфускатора используется предложенная в [16] и реализованная в [13] модель, основанная на оптимизирующем компиляторе, а в качестве аппроксимации A наиболее понятной версии P_0 программы P используется версия, полученная с помощью компилятора AOCC с опцией оптимизации Os (см. [13], раздел 6). Версию программы P, полученную с помощью компилятора AOCC с опцией оптимизации Os, будем обозначать $AOCC_{Os}(P_0)$. Таким образом, эффективность e и стойкость r обфусцирующей последовательности преобразований $O^t \in Q$ для программы P в рамках модели $\mathcal{M}$ вычисляется по формулам:

$$\mathrm{r}(O^t, P) = 1 - \delta_{im}\left(Im\left(\mathrm{D}\left(O^t(P)\right), AOCC_{Os}(P_0)\right)\right),$$

$$\mathrm{r}(O^t, P) = 1 - \delta_{im}\left(Im\left(\mathrm{D}\left(O^t(P)\right), AOCC_{Os}(P_0)\right)\right).$$

Таблица 1

Базовый набор обфусцирующих преобразований (столбец o($\in \mathcal{O}$) в таблице) обфускатора Hikari для программ на языке C

o($\in \mathcal{O}$)	**Описание**	$\theta(o)$
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

[5] https://github.com/HikariObfuscator/Hikari (дата обращения: 13.04.2025).

Для оценки эффективности и стойкости обфусцирующих преобразований выбраны два набора программ: набор из 164 программ, использовавшихся при построении функции δ_{im} (CoreUtils, PolyBench, HashCat), и набор Small-Programs[6] из 20 программ, которые не использовались при построении этой функции. Для каждого из наборов была вычислена средняя эффективность $\mathrm{e}(O^t,\cdot)$ и средняя стойкость $\mathrm{r}(O^t,\cdot)$преобразования $O^t \in Q$ (усреднение выполнялось по всем программам соответствующего набора). Также для каждого из наборов вычислена средняя эффективность $\mathrm{e}(\theta(O^t),\cdot)$ типа преобразования $\theta(O^t)$ и средняя стойкость $\mathrm{r}(\theta(O^t),\cdot)$ этого типа. Интуитивно ожидается, что преобразование, включающее все обфусцирующие преобразования (которая далее называется all), должно быть как наиболее эффективным, так и наиболее стойким. Поэтому в работе в качестве референсных значений вычисляются описанным выше способом эффективность и стойкость последовательности all.

В табл. 2 указаны последовательности, которые для обоих наборов программ попали в группу из $N(\in \{8,16,32\})$ лучших (худших) последовательностей, имеющих наибольшее (соответственно наименьшее) значение эффективности, стойкости или и того и другого одновременно.

В табл. 3 указаны типы последовательностей (полученные на основании табл. 1), которые для обоих наборов программ попали в группу из $N(\in \{3,7,15\})$ лучших (худших) типов последовательностей, имеющих наибольшее (соответственно наименьшее) значение эффективности, стойкости или и того и другого одновременно.

Таблица 2

Обфусцирующие последовательности, входящие в N лучших (T<N>) и худших (B<N>) последовательностей, как для первого, так и для второго набора программ ($N(\in \{8,16,32\})$)

	Относительно $\mathrm{e}(O^t,\cdot)$	**Относительно** $\mathrm{r}(O^t,\cdot)$	**Относительно** $\mathrm{e}(O^t,\cdot)$ **и** $\mathrm{r}(O^t,\cdot)$
T32	all, bcf-fcw, bcf-fcw-cff, bcf-ind, bcf-ind-fcw, bcf-ind-sbb, bcf-ind-sub, bcf-sbb, bcf-sub-sbb, cff-sub-sbb, fcw-cff-sbb,fcw-sbb, ind-cff-sub, ind-sub-sbb, sbb	all, bcf-cff-sub, bcf-ind, bcf-ind-fcw, bcf-ind-sbb, bcf-ind-sub, cff, enc-bcf-fcw, enc-bcf-ind, enc-fcw-cff, enc-ind-sub, fcw-sub-sbb, ind, ind-cff-sbb, ind-fcw-sbb, sbb, sub	all, bcf-ind, bcf-ind-fcw, bcf-ind-sbb, bcf-ind-sub, sbb
B32	bcf, bcf-cff, bcf-sub, enc, enc-cff, enc-fcw-cff, enc-ind, enc-ind-cff, enc-ind-fcw, enc-ind-sbb, enc-ind-sub, fcw-cff, fcw-sub, ind-fcw, ind-fcw-cff	bcf, bcf-cff, bcf-cff-sbb, bcf-fcw-sub, bcf-sbb, bcf-sub, cff-sbb, enc, enc-bcf, enc-bcf-cff, enc-bcf-sub, enc-cff, enc-cff-sbb, enc-fcw-sub, enc-ind-sbb, enc-sbb, enc-sub-sbb	bcf, bcf-cff, bcf-sub, enc, enc-cff, enc-ind-sbb
T16	all, bcf-ind-sbb, cff-ind-sub	all, bcf-ind-fcw, bcf-ind-sbb, bcf-ind-sub, enc-bcf-fcw, enc-fcw-cff, enc-ind-sub	all, bcf-ind-sbb, bcf-ind-sub
B16	enc, enc-cff, enc-ind, enc-ind-cff	bcf-cff-sbb, enc-bcf-cff, enc-cff-sbb	—
T8	all, bcf-ind-sbb	all, bcf-ind-sbb, enc-fcw-cff	all, bcf-ind-sbb
B8	—	bcf-cff-sbb	—

[6] https://github.com/Boriskin61/small-programs (дата обращения: 13.04.2025).

Таблица 3

Типы обфусцирующих последовательностей, входящие в N лучших (T<N>) и худших (B<N>) типов последовательностей, как для первого, так и для второго набора программ (N$(\in \{3, 7, 15\})$)

	Относительно $e(\theta(O^t),\cdot)$	**Относительно** $r(\theta(O^t),\cdot)$	**Относительно** $e(\theta(O^t),\cdot)$ **и** $r(\theta(O^t),\cdot)$
T15	all, C-D-A, C-C-A, C-C-D, A, A-C-A, A-A	all, C-C-A, C-C-D, A-C-D, A-C-A, A-D-A, C-A-A, D-A-C	all, C-C-A, C-C-D, A-C-A
B15	A-C, D-A-C, D-C, A-D, D	D-A, D-D-A, C-A, D-C, D-A-D, C-D	D-C
T7	all, C-D-A, C-C-A	all, C-C-D, D-A-C	all
B7	D-C	D-C, D-A-D	D-C
T3	all, C-C-A	all, D-A-C	all
B3	—	—	—

Из табл. 2 и 3 видно, что последовательность all всегда относится к наиболее эффективным и стойким последовательностям, что соответствует интуитивным ожиданиям. Последовательность bcf-ind-sbb, вошедшая в топ-8 (T8) наиболее стойких и эффективных обфусцирующих последовательностей для обоих наборов программ (согласно табл. 2), входит также в топ-32 (T32) по эффективности и топ-8 (T8) по стойкости для обоих наборов, согласно результатам работы [13] (см. табл. 6 и 14), полученным с помощью других функций похожести. Тип C-C-A, согласно табл. 3, входит в топ-15 (T15) наиболее эффективных и стойких типов, что коррелирует с результатами работы [13], где этот тип по эффективности и стойкости входит в число лучших (см. таблицы 12 и 20 в [13]). Меньше корреляция с результатами из работы [13] проявляется в части выделения наименее эффективных и стойких преобразований. Из таблицы 3 видно только, что почти все наиболее слабые типы последовательностей в своем составе содержат преобразование типа D. В [13] преобразование такого типа также обладает наименьшей эффективностью и стойкостью.

Заключение. Наблюдения, отмеченные в предыдущем разделе, а также невысокая точность построенной функции δ_{im} (0.713) позволяют использовать ее только в качестве вспомогательной при построении более сложных функций похожести, учитывающих и другие способы представления исполняемых файлов. Например, эта функция может быть применена для пополнения арсенала показателей похожести, на основе которых строятся функции похожести в [13]. Отметим, что вместо побайтовой разности $Im(P_1) - Im(P_2)$ может использоваться другое представление пары изображений, например, побитовое исключающее “или” изображений $Im(P_1) \oplus Im(P_2)$ или их горизонтальная склейка $Im(P_1) \parallel Im(P_2)$. Это может являться одним из направлений исследования возможности совершенствования функции похожести, в основе которой лежит сравнение графических представлений исполняемых файлов. Другим направлением исследования является применение известных сверточных нейронных сетей VGG16, InceptionV3, Efficientnetv2b0, Vision Transformers, которые показали высокую точность при классификации вредоносного программного обеспечения на основе графического представления исполняемых файлов [22].

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Varnovsky N. et al.* The current state of art in program obfuscations: definitions of obfuscation security // Proceedings of the Institute for system programming of the RAS. – 2014. – Vol. 26, No. 3.
2. *Garg S. et al.* Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits // 2013 IEEE 54th Annual Symposium on Foundations of Computer Science. – 2013.
3. *Xu H. et al.* Layered obfuscation: a taxonomy of software obfuscation techniques for layered security. – 2020. – Vol. 3.

4. *BinShamlan M. H. B.M.A..Z.A.A.* The impact of control flow obfuscation technique on software protection against human attacks // First International Conference of Intelligent Computing and Engineering (ICOICE). – 2019.
5. *Halevi S. et al.* Implementing BP-obfuscation using graph-induced encoding // Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. – 2017.
6. *Collberg C. T.C..L.D.* A taxonomy of obfuscating transformations. – 1997.
7. *Nagra J. C.C.* Surreptitious software: obfuscation, watermarking, and tamperproofing for software protection. – Pearson Education, 2009.
8. *Banescu S. P.A.* A tutorial on software obfuscation. – 2018. – Vol. 108.
9. *De Sutter B. et al.* Evaluation methodologies in software protection research. – 2024. – Vol. 57, No. 4.
10. *.Madou M. et al.* On the effectiveness of source code transformations for binary obfuscation // Proceedings of the International Conference on Software Engineering Research and Practice (SERP06). – 2006.
11. *Manikyam R. et al.* Comparing the effectiveness of commercial obfuscators against MATE attacks // Proceedings of the 6th Workshop on Software Security, Protection, and Reverse Engineering. – 2016.
12. *Schrittwieser S., Katzenbeisser S., Kinder J., Merzdovnik G., Weippl E*. Protecting software through obfuscation: Can it keep pace with progress in code analysis? // ACM Computing Surveys (CSUR). – 2016. – Vol. 49. – P. 1-37.
13. *Борисов П.Д., Косолапов Ю.В.* Способ количественного сравнения обфусцирующих преобразований // Информатика и автоматизация. – 2024. – Т. 23. – С. 684-726.
14. *Gulwani S., Polozov O., Singh R., others.* Program synthesis // Foundations and Trends® in Programming Languages. – 2017. – Vol. 4. – P. 1-119.
15. *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. – P. 1-12.
16. *Borisov P.D., Kosolapov Y.V.* On the Automatic Analysis of the Practical Resistance of Obfuscating Transformations // Automatic Control and Computer Sciences. – 2020. – Vol. 54. – P. 619-629.
17. *Borisov P.D., Kosolapov Y.V.* On the Characteristics of Symbolic Execution in the Problem of Assessing the Quality of Obfuscating Transformations // Automatic Control and Computer Sciences. – 2022. – Vol. 56. – P. 595-605.
18. *Mohsen R., Pinto A.* Evaluating Obfuscation Security: A Quantitative Approach October 2015.
19. *Lekssays A., Falah B., Abufardeh S.* A Novel Approach for Android Malware Detection and Classification using Convolutional Neural Networks // ICSOFT. – 2020. – P. 606-614.
20. *Kiger J., Ho S.S., Heydari V.* Malware binary image classification using convolutional neural networks // International Conference on Cyber Warfare and Security. – 2022. – Vol. 17. – P. 469-478.
21. *Jiang H., Polsani H., Liu Y.* DeepGray: Malware Classification Using Grayscale Images with Deep Learning // The International FLAIRS Conference Proceedings. – 2024. – Vol. 37.
22. *Ben Abdel Ouahab I., Bouhorma M., Boudhir A.A., El Aachak L.* Classification of grayscale malware images using the K-nearest neighbor algorithm // Innovations in Smart Cities Applications Edition 3: The Proceedings of the 4th International Conference on Smart City Applications 4. – 2020. – P. 1038-1050.

REFERENCES

1. *Varnovsky N. et al.* The current state of art in program obfuscations: definitions of obfuscation security, *Proceedings of the Institute for system programming of the RAS,* 2014, Vol. 26, No. 3.
2. *Garg S. et al.* Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits, *2013 IEEE 54th Annual Symposium on Foundations of Computer Science,* 2013.
3. *Xu H. et al.* Layered obfuscation: a taxonomy of software obfuscation techniques for layered security, Vol. 3, 2020.
4. *BinShamlan M. H. B.M.A..Z.A.A.* The impact of control flow obfuscation technique on software protection against human attacks, *First International Conference of Intelligent Computing and Engineering (ICOICE),* 2019.
5. *Halevi S. et al.* Implementing BP-obfuscation using graph-induced encoding, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security,* 2017.
6. *Collberg C. T.C..L.D.* A taxonomy of obfuscating transformations, 1997.
7. *Nagra J. C.C.* Surreptitious software: obfuscation, watermarking, and tamperproofing for software protection. Pearson Education, 2009.
8. *Banescu S. P.A.* A tutorial on software obfuscation, 2018, Vol. 108.
9. *De Sutter B. et al.* Evaluation methodologies in software protection research, 2024, Vol. 57, No. 4.
10. *.Madou M. et al.* On the effectiveness of source code transformations for binary obfuscation, *Proceedings of the International Conference on Software Engineering Research and Practice (SERP06),* 2006.

11. *Manikyam R. et al.* Comparing the effectiveness of commercial obfuscators against MATE attacks, *Proceedings of the 6th Workshop on Software Security, Protection, and Reverse Engineering,* 2016.
12. *Schrittwieser S., Katzenbeisser S., Kinder J., Merzdovnik G., Weippl E*. Protecting software through obfuscation: Can it keep pace with progress in code analysis?, *ACM Computing Surveys (CSUR),* 2016, Vol. 49, pp. 1-37.
13. *Borisov P.D., Kosolapov Yu.V.* Sposob kolichestvennogo sravneniya obfustsiruyushchikh preobrazovaniy [Method for quantitative comparison of obfuscating transformations], *Informatika i avtomatizatsiya* [Computer Science and Automation], 2024, Vol. 23, pp. 684-726.
14. *Gulwani S., Polozov O., Singh R., others.* Program synthesis, *Foundations and Trends® in Programming Languages,* 2017, Vol. 4, pp. 1-119.
15. *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.
16. *Borisov P.D., Kosolapov Y.V.* On the Automatic Analysis of the Practical Resistance of Obfuscating Transformations, *Automatic Control and Computer Sciences,* 2020, Vol. 54, pp. 619-629.
17. *Borisov P.D., Kosolapov Y.V.* On the Characteristics of Symbolic Execution in the Problem of Assessing the Quality of Obfuscating Transformations, *Automatic Control and Computer Sciences,* 2022, Vol. 56, pp. 595-605.
18. *Mohsen R., Pinto A.* Evaluating Obfuscation Security: A Quantitative Approach October 2015.
19. *Lekssays A., Falah B., Abufardeh S.* A Novel Approach for Android Malware Detection and Classification using Convolutional Neural Networks, *ICSOFT,* 2020, pp. 606-614.
20. *Kiger J., Ho S.S., Heydari V.* Malware binary image classification using convolutional neural networks, *International Conference on Cyber Warfare and Security,* 2022, Vol. 17, pp. 469-478.
21. *Jiang H., Polsani H., Liu Y.* DeepGray: Malware Classification Using Grayscale Images with Deep Learning, *The International FLAIRS Conference Proceedings,* 2024, Vol. 37.
22. *Ben Abdel Ouahab I., Bouhorma M., Boudhir A.A., El Aachak L.* Classification of grayscale malware images using the K-nearest neighbor algorithm, *Innovations in Smart Cities Applications Edition 3: The Proceedings of the 4th International Conference on Smart City Applications 4,* 2020, pp. 1038-1050.

Борисов Петр Дмитриевич – ФГАНУ НИИ "Спецвузавтоматика"; e-mail: borisovpetr@mail.ru; г. Ростов-на-Дону, Россия; тел.: +7863201-2817; зав. лабораторией.

Косолапов Юрий Владимирович – Южный федеральный университет; e-mail: yvkosolapov@sfedu.ru; г. Ростов-на-Дону, Россия; тел.: +7863297-5111; кафедра алгебры и дискретной математики; к.т.н.; доцент.

Borisov Petr Dmitrievich – FSASE SRI "Specvuzavtomatika"; e-mail: borisovpetr@mail.ru; Rostov-on-Don, Russia; phone: +78632012817; head of the laboratory.

Kosolapov Yury Vladimirovich – Southern Federal University; e-mail: yvkosolapov@sfedu.ru; Rostov-on-Don, Russia; phone: +78632975111; the Department of Algebra and Discrete Mathematics; cand. of eng. sc.; associate professor

УДК 629.735.015 DOI 10.18522/2311-3103-2025-3-273-284

И.В. Борисов, А.С. Кузьменко, В.Е. Курьян, М.В. Курьян, Е.М. Левченко

ОПРЕДЕЛЕНИЕ ПОГРЕШНОСТЕЙ КООРДИНАТ ЦЕЛИ ПРИ МНОГОПОЗИЦИОННОЙ РАДИОЛОКАЦИИ С ИСПОЛЬЗОВАНИЕМ ГРУППЫ БЕСПИЛОТНЫХ ЛЕТАТЕЛЬНЫХ АППАРАТОВ

Предлагается и развивается алгебраический метод для определения координат целей и их погрешностей в составе группы беспилотных летательных аппаратов. Обоснованы основные допущения разрабатываемой модели функционирования группы беспилотных летательных аппаратов: скорости летательных аппаратов не превышают скорости звука в воздухе, а скорости целей, – не превосходят первую космическую. Представлены качественные оценки времени приёма радиолокационного сигнала для заданной пространственной погрешности координат цели, оценены требования к кварцевому генератору с целью обеспечения стабильности частоты. Сформулированы условия по количеству летательных аппаратов в группе, повышающих точность опреде-