УДК 004.051

В.С. Кантарович, ст. преп. (БГТУ, г. Минск);
П.П. Урбанович, проф., д-р техн. наук (БГТУ, г. Минск)

ОСОБЕННОСТИ РАЗРАБОТКИ И ПРИМЕНЕНИЯ GAN-МОДЕЛИ ДЛЯ ОБФУСКАЦИИ ПРОГРАММНОГО КОДА

Современные технологии машинного обучения и искусственного интеллекта являются мощным инструментом для создания эффективных механизмов в области безопасности и защиты данных. Однако, они также представляют собой серьезную угрозу, поскольку могут быть использованы злоумышленниками для осуществления атаки на конфиденциальность и целостность информации, в частности программного кода. Алгоритмы МО способны анализировать и обрабатывать большой объемов данных, в том числе – анализировать обфусцированный код: выявлять уязвимости, распознавать определенные паттерны проектирования и предсказывать методы защитных механизмов, восстанавливать функциональность кода. Это открывает возможности для реверсивной инженерии, позволяя злоумышленникам восстановить исходный код и использовать его в своих целях. Таким образом, актуальной является задача создания новых решений, обеспечивающих надежную защиту кода и сокрытие реализованного в нем алгоритма [1, 2].

Обфускация – это процесс преобразования программного кода в трудночитаемый формат, который сохраняет его функциональность, но затрудняет анализ. Нейронные сети могут генерировать и применять более эффективные методы сокрытия кода, повышая уровень его защиты, затрудняя реверсивную инженерию и работу существующих инструментов для анализа и восстановление интеллектуальной собственности [3].

Одним из эффективных подходов к решению задачи обфускации является использования генеративно-состязательных сетей (Generative Adversarial Network, *GAN*), которые могут применяться для создания обфусцированного кода, генерируя альтернативные версии программ. Сеть позволяет создавать уникальные структуры кода, которые сложно анализировать при помощи существующих инструментов для деобфускации. *GAN* состоит из двух нейронных сетей – генерирующей (генератор, *G*) и дискриминирующей (дискриминатор, *D*). Обе сети обучаются в состязательном режиме, где генератор пытается создавать новые данные, а дискриминатор – угадать, являются они реальными или поддельными [4].

В рамках данного исследования рассматривается применение *GAN* для обфускации программного кода на языке *JavaScript*.

Уровень безопасности программы оценивается эффективностью на основе таких метрик, как устойчивость к реверсивной инженерии, сохранение функциональности и производительность кода. Таким образом задачей *GAN*-модели является обучение генератора *G* на создание таких обфусцированных версий токенов кода, которые невозможно отличить от реального, используя *D*.

Архитектура *GAN* для обфускации имеет отличия от подобных решений для генерации и анализа изображений или текста. Кроме того, нужно учитывать особенности языка *JavaScript* (динамическая типизация, вложенные структуры, области видимости переменных), так как код, написанный на этом языке, предоставляется пользователям в явном виде. Следовательно, *GAN*-модель, построенная для обфускации *JavaScript*-кода, должна учитывать его структурированность, семантическую корректность и долгосрочные зависимости. Для этого используются методы токенизации, комбинации *LSTM* и *CNN*, *Attention*- механизм. Эти особенности архитектуры *GAN* делают ее адаптированной для решения задач по защите кода.

Поскольку *JavaScript*-код нельзя обрабатывать как обычный текст, то нужно учитывать его лексическую структуру и обрабатывать как структурированные токены. Для этого все пары файлов исходного и обфусцированного кода, на которых обучается модель, необходимо разделить на токены, используя специальную библиотеку *Pygments*, которая сгруппирует токены по типам (*KEYWORD_*, *NAME_*, *STRING_LITERAL*, *NUMBER_LITERAL*). Это поможет модели понимать структуру кода и обобщать паттерны. В *JavaScript* также возможны долгосрочные зависимости, где переменные используют значения, определенные ранее, должны учитываться синтаксические правила и области видимости переменных. Для этого используется *Bidirectional LSTM* для понимания контекста в обе стороны и *Attention*-механизм – для отслеживания различных зависимостей в коде.

Для формального описания нашей *GAN*-модели используются следующие элементы: $\boldsymbol{X}$ – пространство (множество) исходных *JavaScript*-кодов; $\boldsymbol{Y}$ – пространство (множество) обфусцированных исходных *JavaScript*-кодов; V – словарь токенов; T – максимальная длина последовательности токенов, полученная после токенизации *JavaScript* кода; $\hat{\mathbf{Y}}$ – пространство (множество) сгенерированных обфусцированных генератором *JavaScript*-кодов; $P_{data}(x)$ – распределение реальных оригинальных кодов, $x \in \boldsymbol{X}$; $P_{data}(y)$, – распределение реальных обфусцированных кодов, $y \in \boldsymbol{Y}$; $P_G(y)$ – распределение кодов,

сгенерированных сетью GAN; $G(x, \theta_G)$ – генератор, который преобразует векторы из скрытого пространства (представляет собой многомерное векторное пространство абстрактных важных характеристик данных) в последовательности токенов; параметры (в общем смысле) генератора G – θ_G; $D(y, \theta_D)$ – дискриминатор, который различает реальные и сгенерированные токены кода; параметры дискриминатора обозначим θ_D.

Каждое пространство исходных и обфусцированных *JavaScript*-кодов представляется как последовательность токенов и может быть выражен следующим образом, поэтому с учётом токенизации можем записать, что

$$x \in \boldsymbol{X}, \boldsymbol{V}: x = x_1, x_2, \ldots, x_s; y \in \boldsymbol{X}, \boldsymbol{V}: y = y_1, y_2, \ldots, y_s.$$

Генератор принимает входной код x и генерирует обфусцированных код $\hat{y}$:

$$\hat{y} = G(x; \theta_G).$$

Входными данными дискриминатора являются как реальные обфусцированные коды y, так и сгенерированные коды $\hat{y}$, а выходным – вероятность того, что входной код является реальным. Если значение вероятности близко к 1:

$$D(y, \hat{y}; \theta_D) \rightarrow P(y=\hat{y}) \rightarrow 1,$$

то дискриминатор «считает» код реальным, а не сгенерированным.

Генератор разработанной *GAN*-модели использует архитектуру *Sequence-to-Sequence* концепции *Encoder-Decoder* с *Binary Cross-Entropy* в качестве функции потерь и *Attention*-механизмом – для преобразования исходного *JavaScript*-кода в его обфусцированную версию, обеспечивая сохранение функциональности и семантики при изменении синтаксиса. Архитектура *Sequence-to-Sequence* используется, когда длины входной и выходной последовательностей различаются, и состоит из следующих компонентов: *Embedding*-слой, который преобразует токены исходного кода в векторные представления; *Encoder* (*Bidirectional LSTM*; *LSTM* – Long Short-Term Memory, долгая краткосрочная память – разновидность архитектуры рекуррентных нейронных сетей) – создает контекстное представление всего кода и использующий двунаправленную обработку для учета контекста в обе стороны (состоит из двух *LSTM*-слоев); *Attention*-механизм (*Multi-Head Attention*) позволяет генератору улучшать качество сгенерированных данных, акцентируя внимание на наиболее значимых частях исходного кода; *Decoder*, построенный также на основе двух слоев *LSTM*, генерирует обфусцированную последовательность токенов; выходной

слой – использует два полносвязных слоя для оценки логитов для каждого токена.

Дискриминатор разработанной *GAN*-модели использует гибридную архитектуру, комбинирующую сверточные нейронные сети (*CNN*) и *LSTM* с *Binary Cross-Entropy* в качестве функции потерь. Эта структура позволяет эффективно анализировать обфусцированный код, в которой *CNN* необходим для обнаружения локальных паттернов обфускации (переименование переменных, изменение синтаксиса, удаление пробелов), а *LSTM* – для понимания глобальных паттернов обфускации (согласованность переименований по всему файлу, структурные изменения кода, долгосрочные зависимости между частями кода). Дискриминатор состоит из следующих компонентов: *Embedding*-слой – преобразует токены обфусцированного кода в векторное представление; *CNN* блок с тремя сверточными слоями с различными фильтрами; *LSTM* блок с двумя слоями – анализирует глобальный контекст для понимания структуры кода; выходной классификатор, состоящий из трех полносвязных слоев – определяет, является код реальным обфусцированным или сгенерированным генератором.

Модель *GAN* объединяет генератор и дискриминатор в одну модель *GAN = (G, D)*. Обе сети обучаются поочередно: дискриминатор получает реальный обфусцированный код (метка: 1) и сгенерированный от генератора (метка: 0) и обучается различать их; генератор стремится обмануть дискриминатор (генерирует код, который дискриминатор будет считать реальным) и использует *Teacher forcing* для соответствия реальным обфусцированным примерам.

При выполнении экспериментов для формирования набора данных с примерами *JavaScript*-кода использовано 50 пар, состоящих из файлов с исходным кодом и обфусцированными версиями. Исходный код включает различные функции, классы, операторы ветвления, циклы, математические операции и другое. Обфусцированные примеры были созданы с помощью онлайн-обфускаторов, таких как *JavaScript Obfuscator Tool*, *EvalPacker* и *JavaScript Obfuscator*.

После завершения обучения модели проведено тестирование для оценки процесса обфускации на небольшом *JavaScript*-файле *test_1* размером 98 байт. Среднее время выполнения необфусцированного кода составило 2.28 мс. Эффективность модели оценивалась по следующим критериям [5,6]: устойчивость к реверсивной инженерии R, сохранение функциональности F, производительность O_p. Разработанная модель показала следующие результаты: $R = 10\%$ – низкая степень защиты кода; $O_p = 35\%$ – снижение производительности обфус-

цированного кода по сравнению с оригиналом; $F = 100\%$ – полное сохранение функциональности программы после обфускации.

Разработанная модель пока демонстрирует ограниченную эффективность, ее дальнейшее совершенствование является перспективным. Модель сохраняет функциональность и контекст кода, обеспечивая согласованность переименований с использованием простых методов обфускации. Мы планируем увеличить размер обучающего датасета для лучшего обобщения изученных паттернов, усовершенствовать токенизацию и использовать абстрактные синтаксические деревья (AST) для глубокого понимания структуры кода. Целесообразно также комбинировать различные методы обфускации для повышения общей эффективности модели. Расширение возможностей модели являются ключевыми факторами для ее успешного внедрения в защиту кода.

ЛИТЕРАТУРА

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

2. Paul C., Whitman J., El-Karim A., Nandakumar P., Ortega F., Zheng L. De-obfuscation Techniques Using Neural Networks // ResearchGate. URL: https://www.researchgate.net/publication/391856178_De-obfuscation_Techniques_Using_Neural_Network (дата обращения: 28.11.2025).

3. Romanelli M., Chatzikokolakis K., Palamidessi C. Optimal Obfuscation Mechanisms via Machine Learning // arXiv preprint arXiv:1904.01059. 2019. DOI: 10.48550/arXiv.1904.01059.

4. Mirza M., Osindero S. Conditional Generative Adversarial Nets // arXiv preprint arXiv:1411.1784. 2014. DOI: 10.48550/arXiv.1411.1784.

5. Кантарович В. С., Урбанович П. П. Критерии эффективности алгоритмов обфускации и деобфускации программного кода и их сравнительный анализ на основе онлайн-обфускаторов // Беспилотные аппараты "БПЛА - 2025": сборник статей II международного форума по беспилотным аппаратам, Минск, 30 сентября – 2 октября 2025 г.: в 2 частях. Минск: БГТУ, 2025. Часть 1. С. 149–153.

6. Кантарович В.С., Урбанович П. П. Применение модели LSTM для обфускации программного кода // Передовые технологии и инновации в образовании и науке для улучшения качества жизни и стимулирования устойчивого экономического роста : сб. ст. VIII Международной научно-технической конференции «Минские научные чтения – 2025», Минск, 3 – 5 декабря 2025 г. : в 3 т. – Т. 2. Минск : БГТУ, 2025. С. 251 – 556.