Научная статья
УДК 004.056
https://doi.org/10.31854/1813-324X-2024-10-4-142-148

Интеллектуальный метод мутации входных корпусов с обратной связью

Николай Николаевич Самарин ✉, samarin_nik@mail.ru
Анастасия Викторовна Тулинова, yarmak.av@ibks.spbstu.ru

Научно-исследовательский институт «Квант»,
Москва, 125438, Российская Федерация

Аннотация

Фаззинг-тестирование представляется одним из эффективных способов повышения надежности программного обеспечения и входит в обязательный перечень мероприятий, проводимых на этапе квалификационного тестирования согласно национальному стандарту ГОСТ Р 56939-2016. Использование штатных мутаторов сводит реализацию данной задачи практически к полному перебору, что негативно сказывается на времени обнаружения некорректного поведения программы. В этой связи актуальным является вопрос рационализации подбора входных данных, при котором учитывается специфика корпуса данных, а также контекст, описывающий реакцию тестируемого программного обеспечения и позволяющий определить область и метод мутации на следующей итерации тестирования.

Цель *настоящей работы – повышение эффективности фаззинг-тестирования за счет интеллектуализации штатного мутатора с использованием аппарата нейронных сетей, предполагающей учет синтаксических и семантических особенностей входного корпуса и использующей обратную связь от тестируемой программы.*

Методы исследования. *При выполнении работы использовались методы анализа и синтеза, теории алгоритмов, дискретной и вычислительной математики, машинного обучения.*

Результаты. *Рассмотрены преимущества и недостатки штатного модуля модификации входных корпусов – мутатора – фаззера AFL. Приведено обоснование выбора искусственной нейронной сети на базе архитектуры долгой краткосрочной памяти в качестве механизма, реализующего интеллектуальное управление процессом генерации и преобразования входных корпусов. Описан предлагаемый метод мутации, подразумевающий интеграцию в работу штатного мутатора механизма принятия решения об объеме и формате необходимых мутаций для увеличения покрытия кода, а также последующее уточнение входных данных шелл-кодом для проверки работоспособности фрагмента, вызвавшего нештатное поведение программного обеспечения. Представлена схема работы модуля мутации, включающего в себя компонент преобразования входных корпусов для генерации трасс исполнения программы и компонент, направленный на подтверждение концепта и повторный вызов нештатного поведения программного обеспечения с использованием сформированного шелл-кода.*

Научная новизна. *В отличие от известных предлагаемый метод использует обратную связь, фиксирующую реакцию программного обеспечения, при формировании стратегии мутации данных, что определяет научную новизну полученных результатов.*

Значимость. *Предложенное решение позволяет уменьшить время тестирования программы при сохранении уровня покрытия кода. Полученные в работе результаты являются универсальными и, в перспективе, могут быть использованы при фаззинг-тестировании методами белого, черного и серого ящика.*

Ключевые слова: *кибербезопасность, фаззинг-тестирование, мутация входного корпуса, искусственные нейронные сети, сеть с долгой краткосрочной памятью*

Ссылка для цитирования: Самарин Н.Н., Тулинова А.В. Интеллектуальный метод мутации входных корпусов с обратной связью // Труды учебных заведений связи. 2024. Т. 10. № 4. С. 142–148. DOI:10.31854/1813-324X-2024-10-4-142-148. EDN:GKUGLY

Original research
https://doi.org/10.31854/1813-324X-2024-10-4-142-148

Intelligent Method for Mutation of Input Cases with Feedback

Nikolay N. Samarin ✉, samarin_nik@mail.ru
Anastasia V. Tulinova, yarmak.av@ibks.spbstu.ru

Research Institute «Kvant»,
Moscow, 125438, Russian Federation

Annotation

Relevance. *Fuzzing is one of the effective ways to improve the software reliability and is included in the mandatory list of research carried out at the stage of qualification testing according to national standard GOST R 56939-2016. The use of standard mutators reduces the fuzzing process to brute force, which negatively affects the time of incorrect program behavior detection. In this regard, it is important to rationalize the selection of input data, which takes into account the data corpus specifics, as well as the context describing the software response under test and allowing to determine the mutations at the next iteration of testing.*

Purpose of the research *is to increase the efficiency of fuzzing by intellectualizing the standard mutator using neural networks, which takes into account the syntactic and semantic features of the input corpus and uses program feedback.*

Methods. *The methods of analysis and synthesis, theory of algorithms, discrete and computational mathematics, machine learning were used.*

Result. *The advantages and disadvantages of the standard module for AFL fuzzer's input corpus mutation are considered. The justification of neural network choice with LSTM-architecture as a mechanism that realizes the intelligent control of input corpora's' generation and transformation is given. The proposed mutation method is described, which implies the integration of decision making mechanism on the amount and format of necessary mutations to increase the code coverage into the standard mutator, as well as the subsequent refinement of input data by shellcode to check the operability of the fragment that caused abnormal software's behavior. The scheme of the mutation module is presented, which includes a component of input corporas conversion for generation of program execution traces and a component aimed at concept confirmation and re-call of abnormal software behavior using the generated shell-code.*

Novelty. *Unlike the known ones, the proposed method uses feedback, fixing the software reaction, when forming the data mutation strategy, which determines the scientific novelty of the obtained results.*

Significance. *The proposed solution allows reducing the program testing time while maintaining the code coverage. The results obtained in the research are universal and, in the future, can be used in white, black and gray box fuzzing methods.*

Keywords: *cybersecurity, fuzzing testing, input corpus mutation, artificial neural networks, long short-term memory network*

For citation: Samarin N.N., Tulinova A.V. Intelligent Method for Mutation of Input Cases with Feedback. *Proceedings of Telecommunication Universities.* 2024;10(4):142–148. (in Russ.) DOI:10.31854/1813-324X-2024-10-4-142-148. EDN:GKUGLY

Введение

Для автоматизации динамического анализа исполняемых файлов широко применяется фаззинг-тестирование. В зависимости от полноты знаний об исполняемом файле, варьируется тип тестирования: методом черного, серого или белого ящика [1]. Однако, даже в условиях исчерпывающей для проведения фаззинг-тестирования полноты знаний, одной из проблем является длительное время получения результата ввиду высокого показателя случайности мутации входных данных.

Наиболее широко используемым на сегодняшний день является фаззер AFL (*аббр. от англ.* American Fuzzy Lop) [1–3]. Как любое комплексное программное обеспечение (ПО), AFL состоит из набора модулей, которые могут быть изменены пользователем в зависимости от их цели тестирования [3]. Модуль, функционал которого рассмат-

ривается в рамках настоящего исследования, называется мутатором. Его задачей является подготовка нового входного корпуса на основе того, который подавался на вход ПО на прошлой итерации. Штатный мутатор не обладает возможностями анализа входных корпусов [2, 4]. В нем реализованы только базовые операции, такие как инвертирование бита, подмена строки или подстроки, сокращение длины корпуса или его увеличение произвольными данными и т. д.

Таким образом, применение штатного мутатора в ходе фаззинг-тестирования, по сути, мало отличается от метода перебора, что приводит к увеличению времени обнаружения случаев некорректного поведения тестируемого ПО [2]. Кроме того, полученный корпус, для которого выявлено некорректное поведение ПО, не может быть доработан до полноценного подтверждения концепта. Вместо этого пользователь должен самостоятельно реализовать его, опираясь на характеристики корпуса, такие как его размер, семантические особенности и пр.

Перспективным механизмом генерации входных корпусов на основе предыдущего опыта, то есть, с удержанием контекста, является искусственная нейронная сеть (ИНС) на базе архитектуры долгой краткосрочной памяти (LSTM, *аббр. от англ.* Long Short Term Memory). Данный тип ИНС имеет ключевую особенность – вентили забывания, которыми может быть задано окно удержания опыта. Реализация мутатора с использованием ИНС с долгой краткосрочной памятью обеспечивает возможность обработки обратной связи, поступающей в качестве реакции тестируемого ПО на входной корпус, и его преобразование – мутацию – в соответствии с полученной реакцией.

Процесс мутации входных корпусов

Фаззинг-тестирование с применением мутации – один из нескольких видов техник фаззинга. При таком тестировании фаззеру предоставляются сведения о полном наборе начальных входных корпусов: формат, максимальная и минимальная длина, наличия закономерностей или последовательностей в корпусе и т. д. На основе полученной информации мутатор – модуль преобразования – создает новые образцы входных корпусов, которые изменены в соответствии с указанными правилами мутации [3, 5]. Штатный мутатор поддерживает базовые преобразования, например, инвертирование произвольного бита входного корпуса, дополнение входного корпуса под последовательностью, перестановка слов и групп слов входного корпуса. Общий принцип работы мутатора представлен на рисунке 1.

Совокупное применение данных преобразований позволяет получить огромное количество вариантов входных корпусов, однако базовый мутатор не учитывает их синтаксические и семантические особенности, вследствие чего уровень покрытия на протяжении долгого времени может оставаться низким [4]. Прежде всего это связано с тем, что для запуска ПО чаще всего требуется ввести команду с заданными параметрами, которые стандартизированы разработчиком, а при получении в качестве аргументов сырых данных, ПО, вероятнее всего, будет выводить сообщение с поддерживаемыми командами.

В зависимости от целей фаззинг-тестирования создаются собственные мутаторы, расширяющие функционал штатных. Основной задачей мутатора является создание разнообразного множества входных корпусов с целью обеспечения прироста покрытия на каждой итерации фаззинг-тестирования.

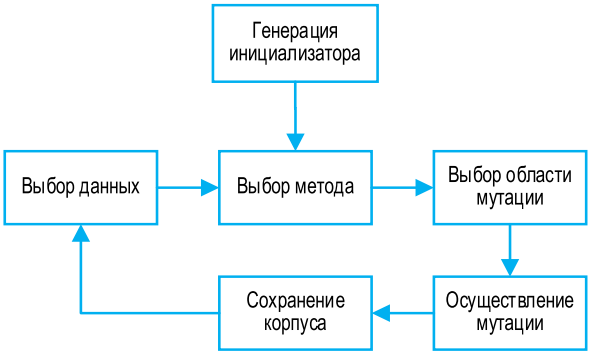

Рис. 1. Общий принцип работы мутатора

Fig. 1. The General Principle of Operation of the Mutator

Таким образом, для выполнения, например, фаззинг-тестирования сетевого сокета необходимо реализовать мутатор сетевого трафика. Мутатор сырых данных будет модифицировать сетевой пакет без учета требований к его формату, что приведет к отбрасыванию сетевого пакета в стеке TCP/IP до его попадания в сетевой сокет и ПО.

Несмотря на такие преимущества мутационных фаззеров, как простота настройки и скорость создания новых входных корпусов, они имеют два существенных недостатка, обусловленных самим механизмом мутации:

1) мутационные фаззеры могут пропустить комплексные уязвимости, требующие значительных изменений данных ввиду простоты выполняемых в отношении входных корпусов преобразований [3].

2) мутационные фаззеры не смогут воспроизвести все возможные формы протокола или формата файла в соответствии со спецификацией ввиду отсутствия правил для модификации файла или кадра протокола [1].

Данные недостатки могут быть компенсированы разработкой собственного мутатора с заданным набором правил и применением комплексных операций в отношении корпусов, однако такой мутатор может быть применим только в отношении конкретного экземпляра или нескольких схожих экземпляров ПО, он не обеспечивает полной унификации. Для унификации мутатора необходимо использование механизма, который обеспечивает мутацию на основе контекста (например, принимаемых на вход типов данных) и обратной связи – реакции программы на входной корпус.

Наиболее перспективны для решения такой задачи технологии машинного обучения и ИНС. Более того, обработку обратной связи с использованием ИНС возможно реализовать с использованием рекуррентного механизма в сетях с архитектурой LSTM.

Возможности искусственной нейронной сети с долгой краткосрочной памятью

Архитектура ИНС с долгой краткосрочной памятью основана на принципе удержания предыдущего опыта, полученного нейронной сетью, с помощью вентилей забывания. Такие сети являются разновидностью рекуррентных нейронных сетей, то есть, сетей, имеющих обратную связь [6]. Благодаря этому LSTM-сети решают проблему долговременной зависимости – запоминания в рамках достаточно больших окон [7]. Традиционно LSTM-сети используются в обработке естественного языка, распознавании речи, а также в генерации музыки и текста [6, 7].

Как любая рекуррентная сеть, LSTM-сеть состоит из цепочки повторяющихся модулей обычной нейронной сети, включающей в себя четыре слоя. Ключевым компонентом модуля LSTM является состояние ячейки – аналога конвейерной ленты, которая проходит через все модули сети. В тракте состояния ячейки данные передаются свободно и в отношении них могут быть применены различного рода линейные операции.

Для контроля состояния ячейки применяются фильтры – вентили забывания, традиционно представляющие собой слой сигмоидальной нейронной сети и операции поточечного умножения. Данный слой возвращает значение в интервале от 0 до 1 включительно, означающее, какой объем данных пропустить в следующий модуль. В LSTM-сети реализовано три таких фильтра, которые позволяют защищать данные от забывания и контролировать состояния ячейки.

В ходе обучения для минимизации ошибки на всем множестве тренировочных последовательностей применяется метод обратного распространения ошибки, развернутый во времени. Это, в свою очередь, позволяет корректировать веса пропорционально производной градиентного спуска в зависимости от величины ошибки.

Несмотря на один из важнейших недостатков применения градиентного спуска для обучения стандартных рекуррентных ИНС – градиенты ошибок уменьшаются с экспоненциальной скоростью по мере увеличения временной задержки между важными событиями – его применение для обучения LSTM-сетей достаточно эффективно [8]. Это обусловлено механизмом контроля состояния ячейки. Когда величины ошибки распространяются в обратном направлении от выходного слоя, ошибка оказывается «заперта» в памяти блока, что создает ситуацию «карусели ошибок», которая непрерывно передает ошибку обратно каждому из вентилей, пока они не будут натренированы отбрасывать значение.

Принцип работы сетей с долгой краткосрочной памятью состоит из четырех этапов [7, 9].

Этап 1. Данные, поступающие в модуль сети, передаются на слой фильтра забывания, определяющий объем информации, который пропускается в следующий слой [8]. Значение предыдущего выхода h_{t-1} и текущего входа x_t передаются в сигмоидальный слой и приобретают коэффициенты от 0 до 1. Значения с коэффициентами ближе к 0 забываются, ближе к 1 – передаются далее. Данный этап описывается формулой:

$$f_t = \sigma\left(W_f * [h_{t-1}, x_t] + b_f\right). \quad (1)$$

Этап 2. Определяется, какая информация будет храниться в состоянии ячейки [7]. Данный этап делится на два подэтапа. Сначала сигмоидальный слой – слой входного фильтра – определяет, какие значения должны быть обновлены:

$$i_t = \sigma(W_t * [h_{t-1}, x_t] + b_t). \quad (2)$$

После чего слой гиперболического тангенса строит вектор новых значений кандидатов $C_t^{'}$, которые могут быть добавлены в состояние ячейки:

$$C_t^{'} = \tanh(W_C * [h_{t-1}, x_t] + b_C). \quad (3)$$

Этап 3. Далее для того, чтобы изменить старое состояние ячейки C_{t-1} на C_t, необходимо умножить его на значение f_t и прибавить $i_t * C_t^{'}$ (4). Таким образом, применяется механизм забывания в отношении состояния ячейки [9, 10]. Фактически, полученные значения будут являться кандидатами, умноженными на число необходимых изменений:

$$C_t = f_t * C_{t-1} + i_t * C_t^{'}. \quad (4)$$

Этап 4. Последний этап заключается в определении объема выходной информации [10]. Значе-

ния предыдущего выхода h_{t-1} и текущего входа x_t проходят сигмоидальный слой, который определяет объем вывода из состояния ячейки:

$$o_t = \sigma(W_o\,[h_{t-1}, x_t] + b_o. \qquad (5)$$

После этого значения состояния ячейки проходят слой гиперболического тангенса для получения на выходе значений в диапазоне от 1 до −1, которые, в свою очередь, перемножаются с выходными значениями сигмоидального слоя:

$$h_t = o_t * \tanh(C_t)\,. \qquad (6)$$

Полученные значения h_t и C_t передаются в следующий модуль цепочки.

Описание метода мутации

В рамках предлагаемого метода предполагается модификация штатного мутатора механизмом, получающим на вход предыдущий корпус I_{t-1} и трассу исполнения T_t, которую этот корпус прошел в тестируемой программе. На основании значений кортежа входных данных (I_{t-1}, T_t) вычисляются необходимые для увеличения покрытия мутации M_1, входящие во множество возможных операций преобразования:

$$M_1 = [F_b, F_s,\ S_a, S_d], \qquad (7)$$

где $F_b(v) = \neg v$ – инвертирование значения произвольного бита v; $F_s(s) = \neg s$ – инвертирование значения произвольной подстроки s; $S_a(s,d) = s + d$ – добавление к исходной строке s другой подстроки d; $S_d(s,d) = s - d$ – удаление из исходной строки s подстроки d.

Объем необходимых мутаций при этом оценивается, исходя из длины трассы, фиксируемой в рамках заданного временного окна: если длина трассы не изменяется на протяжении N запусков, количество мутаций входного корпуса, осуществляемых в рамках одной итерации, увеличивается до тех пор, пока не будет зафиксирован прирост по покрытию. При обнаружении прироста количество мутаций снижается до исходного значения.

После того, как в ходе фаззинг-тестирования установлено нештатное завершение работы тестируемого ПО, мутация с целью увеличения покрытия прекращается. Вместо этого происходит загрузка второго блока входных корпусов, содержащих исполняемый код для создания подтверждения концепта. Входные корпуса второго блока состоят из двух частей: входного корпуса, вызвавшего нештатное завершение работы, и шелл-кода.

Мутация на данном этапе выполняется только в отношении шелл-кода в соответствии с заданными правилами M_2:

$$M_2 = [N_a, N_d, A_m], \qquad (8)$$

где $N_a(n)$ – увеличение NOP-дорожки (NOP, *аббр. от англ.* No OPeration) на значение n; $N_d(n)$ – уменьшение NOP-дорожки на значение n; $A_m(a_{\min}, a_{\max})$ – изменение адреса возврата на произвольный, в диапазоне $[a_{\min}, a_{\max}]$.

Данный этап выполняется до тех пор, пока шелл-код не будет выполнен, то есть пока не прекратится нештатное завершение тестируемого ПО. После этого фаззер переходит к новой итерации и продолжает искать новые трассы в ПО. Наглядно действие описанного метода представлено на рисунке 2.

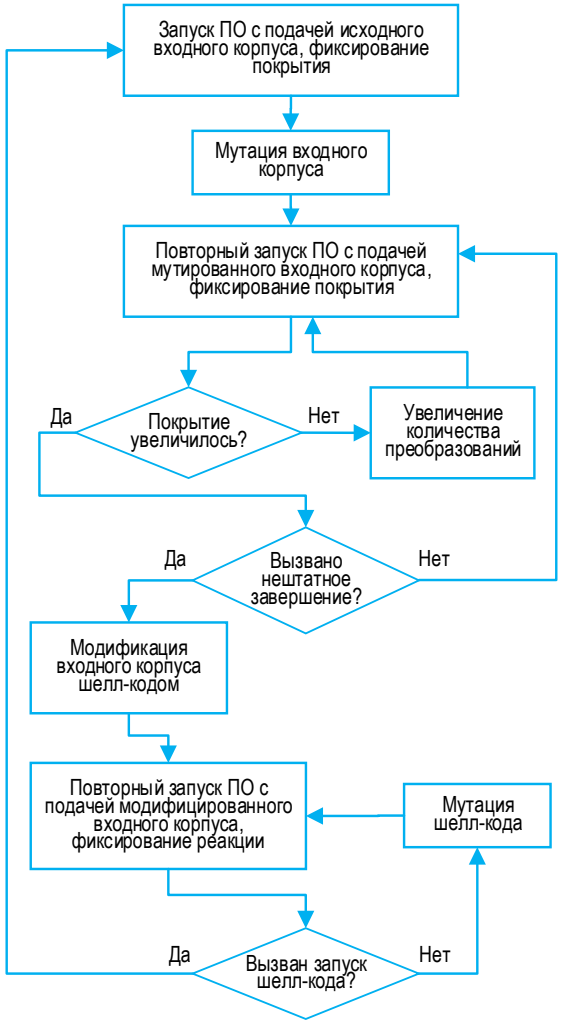

Рис. 2. Схема работы интеллектуального метода мутации входных данных

Fig. 2. The Schematic Diagram of the Input Mutation Intelligent Method Operation

Заключение

Фаззинг-тестирование является основным подходом при проведении динамического анализа программного обеспечения на предмет наличия уязвимостей, архитектурных или логических ошибок. Основным недостатком данного подхода является время его проведения ввиду хаотичности генера-

ции входных корпусов, подаваемых тестируемой программе [3, 4, 11]. В рамках проведенных исследований создан интеллектуальный метод мутации входных корпусов, который использует обратную связь, получаемую от объекта исследований и на ее основе принимает решение о необходимости увеличения количества модификаций. Метод включает в себя два этапа, имеющие различные цели.

На первом этапе фаззеру требуется вызвать нештатное завершения работы программы и найти как можно большее количество трасс для дальнейшего анализа. Возможные в программе трассы используются после работы второго этапа в качестве отправной точки для новой итерации.

На втором этапе, который наступает после обнаружения нештатного завершения работы программы, осуществляется дополнение входного корпуса шелл-кодом. На данном этапе целью является запуск этого шелл-кода в контексте тестируемой программы для подтверждения концепта выявленной уязвимости. Таким образом, на втором этапе мутации подвергается не исходная часть входного корпуса, а сам шелл-код.

Дальнейшие исследования в данной области должны быть направлены на создание интеллектуального метода мутации шелл-кода на основе реакции ПО. Такой метод позволит повысить производительность за счет формализации правил модификации шелл-кода и снижения количества некорректных с точки зрения синтаксиса и семантики исполняемого кода мутаций [12].

Список источников

1. Muduli S.K., Roy S. Satisfiability modulo fuzzing: a synergistic combination of SMT solving and fuzzing // Proceedings of the ACM on Programming Languages. 2022. Vol. 6. Iss. OOPSLA2. PP. 1236–1263. DOI:10.1145/3563332. EDN:VUQLTQ

2. Liu Z., Qian P., Yang J., Liu L., Xu X., He Q., et. al. Rethinking Smart Contract Fuzzing: Fuzzing with Invocation Ordering and I mportant Branch Revisiting // IEEE Transactions on Information Forensics and Security. 2023. Vol. 18. PP. 1237–1251. DOI:10.1109/tifs.2023.3237370. EDN:ZWKGJS

3. Ерышов В.Г. Фаззинг тестирование. Классификация современных средств фаззинга // Сборник избранных статей по материалам научных конференций ГНИИ "Нацразвитие" Международные научные конференции (Санкт-Петербург, Россия, 26–31 августа 2021 года). СПб.: ГНИИ «Нацразвитие», 2021. С. 287–289. DOI:10.37539/AUG298.2021.94.77.007. EDN:QZILPI

4. Situ L.-Y., Zuo Z.-Q., Guan L., Wang L.-Z., Li X.-D., Shi J., et. al. Vulnerable Region-Aware Greybox Fuzzing // Journal of Computer Science and Technology. 2021. Vol. 36. Iss. 5. PP. 1212–1228. DOI:10.1007/s11390-021-1196-0. EDN:PAPPKT

5. Kim S.J., Shon T. Field classification-based novel fuzzing case generation for ICS protocols // The Journal of Supercomputing. 2018. Vol. 74. Iss. 9. PP. 4434–4450. DOI:10.1007/s11227-017-1980-3. EDN:TLEZVS

6. Wei W., Li X., Zhang B., Li L., Damaševičius R., Scherer R. LSTM-SN: complex text classifying with LSTM fusion social network // The Journal of Supercomputing. 2023. Vol. 79. Iss. 9. PP. 9558–9583. DOI:10.1007/s11227-022-05034-w. EDN:TSXGJI

7. Bayram F., Aupke P., Bestoun S.A., Kassler A., Theocharis A., Forsman J. DA-LSTM: A dynamic drift-adaptive learning framework for interval load forecasting with LSTM networks // Engineering Applications of Artificial Intelligence. 2023. Vol. 123. P. 106480. DOI:10.1016/j.engappai.2023.106480. EDN:WOXZSB

8. Pierre A.A., Akim S.A., Semenyo A.K., Babiga B. Peak Electrical Energy Consumption Prediction by ARIMA, LSTM, GRU, ARIMA-LSTM and ARIMA-GRU Approaches // Energies. 2023. Vol. 16. Iss. 12. P. 4739. DOI:10.3390/en16124739. EDN:RKROHA

9. Singh P., Kumar Ch., Kumar A. Next-LSTM: a novel LSTM-based image captioning technique // International Journal of System Assurance Engineering and Management. 2023. Vol. 14. Iss. 4. PP. 1492–1503. DOI:10.1007/s13198-023-01956-7. EDN:QUOVUU

10. Wen X., Li W. Wen X. Time Series Prediction Based on LSTM-Attention-LSTM Model // IEEE Access. 2023. Vol. 11. PP. 48322–48331. DOI:10.1109/access.2023.3276628. EDN:ZAABGZ

11. Ding T., Fu J., Shen R. Research on Multidimensional Mutation Strategy Method of Fuzzing Test // Proceedings of the 40th Chinese Control Conference (CCC, Shanghai, China 26–28 July 2021). Shanghai: IEEE, 2021. PP. 8639–8644. DOI:10.23919/CCC52363.2021.9550435. EDN:MJGBJG

12. Manès V.J.M., Han H., Han C., Cha S.K., Egele M., Woo M. The Art, Science, and Engineering of Fuzzing: A Survey // IEEE Transactions on Software Engineering. 2021. Vol. 47. Iss. 11. PP. 2312–2331. DOI:10.1109/TSE.2019.2946563. EDN:ZDDKFN

References

1. Muduli S.K., Roy S. Satisfiability modulo fuzzing: a synergistic combination of SMT solving and fuzzing. *Proceedings of the ACM on Programming Languages.* 2022;6(OOPSLA2):1236–1263. DOI:10.1145/3563332. EDN:VUQLTQ

2. Liu Z., Qian P., Yang J., Liu L., Xu X., He Q., et. al. Rethinking Smart Contract Fuzzing: Fuzzing with Invocation Ordering and Important Branch Revisiting. *IEEE Transactions on Information Forensics and Security.* 2023;18:1237–1251. DOI:10.1109/tifs.2023.3237370. EDN:ZWKGJS

3. Yeryshov V.G. Fuzzing testing. Classification of modern fuzzing tools. *Proceedings of International Scientific Conferences of the National Research Institute "National Development», 26–31 august 2021, St. Petersburg, Russia.* St. Petersburg: GNII "National Development" Publ.; 2021. p.287–289. DOI:10.37539/AUG298.2021.94.77.007. EDN:QZILPI

4. Situ L.-Y., Zuo Z.-Q., Guan L., Wang L.-Z., Li X.-D., Shi J., et. al. Vulnerable Region-Aware Greybox Fuzzing. *Journal of Computer Science and Technology.* 2021;36(5):1212–1228. DOI:10.1007/s11390-021-1196-0. EDN:PAPPKT

5. Kim S. J., Shon T. Field classification-based novel fuzzing case generation for ICS protocols. *The Journal of Supercomputing.* 2018;74(9):4434–4450. DOI:10.1007/s11227-017-1980-3. EDN:TLEZVS

6. Wei W., Li X., Zhang B., Li L., Damaševičius R., Scherer R. LSTM-SN: complex text classifying with LSTM fusion social network. *The Journal of Supercomputing.* 2023;79(9):9558–9583. DOI:10.1007/s11227-022-05034-w. EDN:TSXGJI

7. Bayram F., Aupke P., Bestoun S.A., Kassler A., Theocharis A., Forsman J. DA-LSTM: A dynamic drift-adaptive learning framework for interval load forecasting with LSTM networks. *Engineering Applications of Artificial Intelligence.* 2023; 123:106480. DOI:10.1016/j.engappai.2023.106480. EDN:WOXZSB

8. Pierre A.A., Akim S.A., Semenyo A.K., Babiga B. Peak Electrical Energy Consumption Prediction by ARIMA, LSTM, GRU, ARIMA-LSTM and ARIMA-GRU Approaches. *Energies*. 2023;16(12):4739. DOI:10.3390/en16124739. EDN:RKROHA

9. Singh P., Kumar Ch., Kumar A. Singh P. Next-LSTM: a novel LSTM-based image captioning technique. *International Journal of System Assurance Engineering and Management.* 2023;14(4):1492–1503. DOI:10.1007/s13198-023-01956-7. EDN:QUOVUU

10. Wen X., Li W. Wen X. Time Series Prediction Based on LSTM-Attention-LSTM Model. *IEEE Access.* 2023;11: 48322–48331. DOI:10.1109/access.2023.3276628. EDN:ZAABGZ

11. Ding T., Fu J., Shen R. Ding T. Research on Multidimensional Mutation Strategy Method of Fuzzing Test *Proceedings of the 40th Chinese Control Conference, CCC, 26–28 July 2021, Shanghai, China*. Shanghai: IEEE; 2021. p.8639–8644. DOI:10.23919/CCC52363.2021.9550435. EDN:MJGBJG

12. Manès V.J.M., Han H., Han C., Cha S.K., Egele M., Woo M. The Art, Science, and Engineering of Fuzzing: A Survey. *IEEE Transactions on Software Engineering*. 2021;47(11):2312–2331. DOI:10.1109/TSE.2019.2946563. EDN:ZDDKFN

Статья поступила в редакцию 27.05.2024; одобрена после рецензирования 10.08.2024; принята к публикации 17.08.2024.
The article was submitted 27.05.2024; approved after reviewing 10.08.2024; accepted for publication 17.08.2024.

Информация об авторах:

САМАРИН Николай Николаевич — кандидат технических наук, начальник научно-исследовательского отделения № 6 «Научно-исследовательского института «Квант»
https://orcid.org/0009-0007-4911-8471

ТУЛИНОВА Анастасия Викторовна — кандидат технических наук, инженер 1 категории научно-исследовательского отделения № 6 «Научно-исследовательского института «Квант»
https://orcid.org/0000-0002-7121-6031

Авторы сообщают об отсутствии конфликтов интересов.
The authors declare no conflicts of interests.