УДК 004.932.2 DOI: 10.18698/0236-3933-2021-3-4-23

ПОИСК ОШИБОК В ПРОГРАММАХ ДЛЯ ОБРАБОТКИ ГРАФИЧЕСКИХ ИЗОБРАЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ МЕТОДА ФАЗЗИНГА

Г.С. Байдин baydin.g.s@bmstu.ru
М.В. Хизова mariakhizova@mail.ru

МГТУ им. Н.Э. Баумана, Москва, Российская Федерация

Аннотация

С увеличением числа программ для автоматизированной обработки графических изображений возникает необходимость в эффективных методах тестирования. Одним из таких методов является фаззинг, для которого необходимо определить наиболее эффективные алгоритмы по созданию тестовых данных в целях увеличения числа найденных ошибок и минимизации аппаратных ресурсов. Результатом проведенных исследований является сравнение алгоритмов создания тестовых данных для поиска ошибок в исполняемом коде программ, предназначенных для обработки графических изображений. Использование байесовских сетей для описания фаззинга позволяет определить связи между структурными компонентами при тестировании. По результатам сравнения алгоритмов фаззинга по созданию тестовых данных выявлены наиболее эффективные фаззеры, предназначенные для поиска ошибок в исполняемом коде программ по обработке графических изображений. Апробация работоспособности предложенных алгоритмов выполнена на ряде существующих уязвимостей, классифицированных как CVE (Common Vulnerabilities and Exposures). Обработка результатов экспериментов по созданию тестовых данных проведена с использованием среды имитационного моделирования, что позволяет пошагово анализировать процесс тестирования. Полученные результаты исследований, алгоритмы создания тестовых данных для поиска ошибок могут быть использованы на различных этапах тестирования программного обеспечения

Ключевые слова

Фаззинг, графические изображения, байесовские сети, фаззер Radamsa, AFL, AnyLogic

Поступила 04.11.2020
Принята 04.12.2020

Введение. В настоящий момент все большую актуальность приобретают вопросы обеспечения информационной безопасности. Активно разрабатываются и используются различные методы. Для анализа и поиска ошибок в программном обеспечении (ПО), связанных с некорректными входными данными, используется метод под названием фаззинг.

Фаззинг — это метод обнаружения ошибок в ПО с использованием подачи на вход заведомо некорректных данных и мониторингом исключительных ситуаций. Автоматизированные средства фаззинга называются фаззерами [1].

Существуют различные способы передачи входных параметров (данных) программе: файлы, сетевые сокеты, стандартные потоки ввода и вывода, переменные окружения, с помощью которых может быть найдена ошибка [2]. В настоящей работе рассмотрено обнаружение ошибок в исполняемом коде программ, предназначенных для обработки графических изображений.

Существует несколько классификаций фаззеров, но наиболее используемой является классификация по способу манипуляции с данными. В такой классификации фаззеры делятся на мутационные, которые изменяют существующие образцы данных, и генерационные, которые создают тестовые данные «с чистого листа», моделируя необходимый протокол или формат файла.

Эффективность данного метода неоднократно подтверждалась найденными ошибками в исполняемом коде программ. Например, фаззер American Fuzzy Loop (AFL) выявил серьезные программные ошибки в таких ПО, как Mongoose OS [3], MatrixSSL [4], Apache httpd [5], OpenSSH [6] и Mozilla NSS [7]. Фаззер Radamsa, в свою очередь, обнаружил серьезные ошибки в ПО Cisco ASA WebVPN/AnyConnect [8] и SAP NetWeaver [9, 10].

В настоящее время накоплены достаточно большие объемы информации, в том числе графических изображений, автоматизированная обработка которых осуществляется с использованием ПО [11]. Данное ПО должно обеспечивать необходимый уровень конфиденциальности обрабатываемых изображений.

Цель исследования — сравнение алгоритмов фаззинга создания тестовых данных для поиска ошибок в исполняемом коде программ, предназначенных для обработки графических изображений, с увеличением числа найденных ошибок и минимизацией аппаратных ресурсов.

Использование байесовских сетей для описания фаззинга программ обработки графических изображений позволяет определить связи между структурными компонентами тестирования с помощью меха-

низмов обучения на основе метода восхождения [12, 13]. Применение байесовских оценок на основе вероятностного вывода реализует эффективные численные методы оценки динамики трансформации ошибок в течение временного интервала, что позволяет отслеживать состояние жизнеспособности алгоритмов фаззинга. В [14] рассмотрена байесовская сеть, которая представляет собой направленный граф без циклов. Вершины смоделированы как случайные переменные X, $i = 1, ..., n$, с множествами всех допустимых значений D_n, $i \in D_n$, где условное распределение вероятностей $P(X_i \mid Parents(X_i))$ представляется как X — родительская вершина для Y:

$$P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i \mid Parents(X_i)), \qquad (1)$$

при этом (1) — полное совместное распределение вероятностей [15].

В целях моделирования процесса фаззинга программ обработки графических изображений применяют алгоритм восхождения [16]. Алгоритм восхождения на каждом шаге реализует максимально возможное усовершенствование целевой функции оценки качества с помощью различных изменений состава дуг, при этом структура байесовской сети инициализируется случайным образом. Алгоритм продолжает выполнение до тех пор, пока оценка качества увеличивается. Во избежание задержки в зоне локального максимума предлагается использовать гибридный алгоритм минимаксного восхождения (англ. *min-max hill climbing*) [17].

Указанный гибридный алгоритм является совокупностью двух алгоритмов: минимаксный предок–потомок (MMPC). Алгоритм MMPC выполняет обучение сети на основе метода локального обнаружения с построением структуры байесовской сети; эвристический поиск минимального и максимального узлов.

Алгоритм MMPC определяет базовую структуру байесовской сети путем нахождения всех узлов без учета ориентации в графе. Для этого необходимо найти множество, содержащее всех предков и потомков для некоторого узла X, подробно алгоритм приведен в [14].

В основе эвристического поиска минимального и максимального узлов лежит поиск методом восхождения, основанный на процессе вычисления метрики Байеса — Дирихле с целью получить апостериорное распределение структуры сети в неявном виде [18]. Поиск начинается с пустого графа, различные изменения состава дуг сети выполняются так, чтобы увеличить оценку, после чего поиск повторяется рекурсивным образом.

Множество узлов-кандидатов (как родительских, так и дочерних) инициализируется как родительские из-за невозможности явного определения родительских и дочерних узлов. Функция Дирихле определяет вектор параметров каждой переменной байесовской сети при фиксированном значении родителей:

$$p\left(Q_{i,j}^{G}\right)=\frac{\Gamma\left(\sum\limits_{s=1}^{r_i}\alpha_{i,j,s}\right)}{\prod\limits_{s=1}^{r_i}\Gamma(\alpha_{i,j,s})}\cdot 1_{\left\{Q_{i,j,1}^{G},\ldots,Q_{i,j,r_i-1}^{G}>0;\ \sum\limits_{k=1}^{r_i-1}Q_{i,j,k}^{G}<1\right\}}\prod_{k=1}^{r_i}\left(Q_{i,j,k}^{G}\right)^{\alpha_{i,j,k}-1}, \quad (2)$$

где параметр $\alpha_{i,j,k}>0$ — распределение Дирихле; $Q_{i,j}^{G}=$ $=\left(Q_{i,j,1}^{G},\ldots,Q_{i,j,r_i-1}^{G}\right)$ — вектор параметров, который соответствует одной переменной и одному значению ее родителя, $Q_{i,j,r_i}^{G}=1-\sum\limits_{k=1}^{r_i-1}Q_{i,j,k}^{G}>0.$

Свойство глобальной независимости переменных байесовской сети имеет вид

$$p\left(Q^{G}\mid G\right)=\prod_{i=1}^{n}p\left(Q_{i}^{G}\mid G\right). \quad (3)$$

Свойства локальной независимости переменных байесовской сети — вектора значений переменных байесовской сети, связанные с каждым значением родителей одной из переменных, представляются как

$$p\left(Q_{i}^{G}\mid G\right)=\prod_{j=1}^{q_i}p\left(Q_{i,j}^{G}\mid G\right). \quad (4)$$

В соответствии со свойством глобальной (3) и локальной (4) независимости переменных байесовской сети распределение Дирихле записывается в следующем виде:

$$p\left(Q^{G}\mid G\right)=\prod_{i=1}^{n}\prod_{j=1}^{q_i}\frac{\Gamma\left(\sum\limits_{k=1}^{r_i}\alpha_{i,j,k}\right)}{\prod\limits_{k=1}^{r_i}\Gamma(\alpha_{i,j,k})}\prod_{k=1}^{r_i}\left(Q_{i,j,k}^{G}\right)^{\alpha_{i,j,k}-1}. \quad (5)$$

Выражение, характеризующее апостериорное распределение вероятности для байесовской сети, согласно определению плотности распределения, имеет вид

$$p(D|G)=\prod_{i=1}^{n}\prod_{j=1}^{q_i}\frac{\Gamma\left(\sum\limits_{k=1}^{r_i}\alpha_{i,j,s}\right)}{\Gamma\left(\sum\limits_{s=1}^{r_i}N_{i,j,s}+\alpha_{i,j,s}\right)}\prod_{s=1}^{r_i}\frac{\Gamma(N_{i,j,s}+\alpha_{i,j,s})}{\Gamma(\alpha_{i,j,s})}. \quad (6)$$

Из формулы Байеса [19] следует, что

$$p(G|D)=\frac{P(D|G)P(G)}{P(D)}, \quad (7)$$

где от структуры сети зависит только числитель

$$p(G|D)\sim P(D|G)P(G), \quad (8)$$

априорная вероятность $P(G)$ будет идентична для всех структур байесовских сетей.

Выражение для метрики Байеса — Дирихле в логарифмической форме, учитывая формулу (6), можно записать так:

$$\ln P(D|G)=\sum_{i=1}^{n}\sum_{j=1}^{q_i}\ln\frac{\Gamma\left(\sum\limits_{k=1}^{r_i}\alpha_{i,j,s}\right)}{\Gamma\left(\sum\limits_{s=1}^{r_i}N_{i,j,s}+\alpha_{i,j,s}\right)}+\sum_{s=1}^{r_i}\frac{\Gamma(N_{i,j,s}+\alpha_{i,j,s})}{\Gamma(\alpha_{i,j,s})}. \quad (9)$$

Для получения эквивалентной метрики Байеса — Дирихле необходимо определить положительную меру на $\{1, ..., r_1\}\times ...\times\{1, ..., r_n\}$:

$$\alpha(x_1, ..., x_n)=\frac{1}{\prod\limits_{i=1}^{n}i}. \quad (10)$$

Исходя из (10), мера, эквивалентная метрике Байеса — Дирихле, описывается выражением

$$\ln P(D|G)=\sum_{i=1}^{n}\sum_{j=1}^{q_i}\ln\frac{\Gamma\left(\frac{1}{q_i}\right)}{\Gamma\left(\sum\limits_{s=1}^{r_i}N_{i,j,s}+\frac{1}{q_i}\right)}+\sum_{s=1}^{r_i}\frac{\Gamma\left(N_{i,j,s}+\frac{1}{q_i r_i}\right)}{\Gamma\left(\frac{1}{q_i r_i}\right)}. \quad (11)$$

Алгоритм эвристического поиска минимального и максимального узлов выполняется с помощью определенной процедуры, осуществляю-

щей проверку статистических критериев независимости для каждой переменной созданной выборки. Подробно алгоритм эвристического поиска минимального и максимального узлов рассмотрен в [14].

Проверка статистических критериев независимости для каждой переменной байесовской сети проводится с использованием критерия Пирсона (χ^2-критерия) [20]:

$$\chi^2 = \sum_{a,b,c} \frac{(N_{a,b,c} - E_{a,b,c})}{E_{a,b,c}}, \tag{12}$$

где

$$E_{a,b,c} = \frac{N_{ac}\ N_{bc}}{N_c} \tag{13}$$

— ожидаемое число выборок ($x=a$, $y=b$ и $z=c$) ($N_{a,b,c}$ — частота появления данных, $x=a$, $y=b$ и $z=c$).

При построении связей между узлами сети и определении их направлений в рамках алгоритма MMPC используется алгоритм «жадного» поиска на основе алгоритма восхождения [20].

Алгоритм «жадного» поиска заключается в определении сети с максимальным значением оценочной функции путем использования операторов добавления, удаления и изменения направленности связей, которые непосредственно влияют на значение данной функции. Критерием остановки алгоритма восхождения является достижение пика оценочной функции.

Шаг 1. Применение алгоритма MMPC для построения структуры сети.

Шаг 2. Вычисление графа с максимальным значением оценки на основе подхода Байеса — Дирихле.

Алгоритм создания тестовых данных для поиска ошибок в исполняемом коде программ, предназначенных для обработки графических изображений, выполняется согласно гибридному алгоритму минимаксного восхождения, а также согласно алгоритму «жадного» поиска для построения связей между узлами сети и определения их направлений. Гибридный алгоритм минимаксного восхождения включает в себя эвристический поиск минимального и максимального узлов и MMPC, который выполняет обучение сети на основе метода локального обнаружения с построением структуры байесовской сети.

Использование данных алгоритмов наиболее эффективно при создании тестовых данных для поиска ошибок в исполняемом коде рассматриваемых программ. Указанные алгоритмы создания тестовых данных

для поиска ошибок могут быть использованы на различных этапах методик тестирования ПО.

Алгоритмы фаззинга программ для обработки изображений. Из многообразия имеющихся фаззеров для исследования выбраны AFL [21] и Radamsa [22], так как они предназначены для поиска ошибок в рассматриваемых программах, прошли успешную апробацию и с их помощью найдено 309 (AFL) и 94 (Radamsa) уязвимостей, добавленных в базу данных CVE.

Автоматизированное средство тестирования на основе AFL использует генетический алгоритм (англ. *genetic algorithm*), что позволяет эффективно увеличивать покрытие кода тестовыми данными [23]. Генетический алгоритм — это эвристический алгоритм поиска, применяемый в задачах оптимизации и моделирования путем случайного отбора, комбинирования и вариации искомых параметров с использованием механизмов, аналогичных естественному отбору в природе [24].

Алгоритм, реализованный в AFL, принимает на вход исходные данные для тестирования. Результатом работы являются, в случае их наличия, файлы, которые вызвали ошибку в исследуемой программе. Алгоритм состоит из следующих шагов.

Шаг 1. Загрузка начальных данных для тестирования в виде файлов в очередь.

Шаг 2. Выгрузка файла из очереди. Если очередь пуста, то алгоритм завершается.

Шаг 3. Уменьшение файла до наименьшего размера, при котором не происходит изменения поведения программы.

Шаг 4. Изменение файла с использованием алгоритма модификации данных.

Шаг 5. Отслеживание изменения состояния программы. Если файл привел к переходу в новое состояние, то помещается в очередь, если в программе обнаружена ошибка, то сохраняется. Переход к шагу 2.

Фаззер Radamsa использует исходные входные данные в виде файлов и на их основе создает тестовые данные по методу «черного ящика». Алгоритм фаззера принимает на вход исходные данные для тестирования (изображения) и число экземпляров тестовых данных. Результатом его работы являются, в случае их наличия, файлы, которые вызвали ошибку в исследуемой программе. Алгоритм состоит из следующих шагов.

Шаг 1. Загрузка начальных данных для тестирования в виде файлов в очередь.

Шаг 2. Выгрузка файла из очереди. Если очередь пуста, то алгоритм завершается.

Шаг 3. Изменение файла с использованием различных алгоритмов модификации данных и создание на его основе экземпляров тестовых данных. Если число экземпляров меньше или равно заранее заданному, то переход к шагу 4, если больше, то переход к шагу 5.

Шаг 4. Отслеживание изменения состояния программы. Если какой-либо экземпляр привел к переходу в новое состояние, то он помещается в очередь, если в программе обнаружена ошибка, то экземпляр помечается как вызвавший ошибку. Переход к шагу 2.

Шаг 5. Остановка генерации новых экземпляров.

Для проверки работоспособности алгоритмов AFL и Radamsa создана выборка, развернут тестовый стенд и подтверждены уязвимости, классифицированные как CVE-2017-12983 в ПО ImageMagick [25], CVE-2016-8684 в ПО GraphicsMagick [26], CVE-2016-10095 в библиотеке Libtiff [27], CVE-2012-2849 в GIF-декодере [28], CVE-2011-0205 в библиотеке ImageIO [29].

Фаззер Radamsa не только изменяет цвета пикселей на создаваемых тестовых изображениях, но и сдвигает пиксели на случайную величину, как это делает AFL.

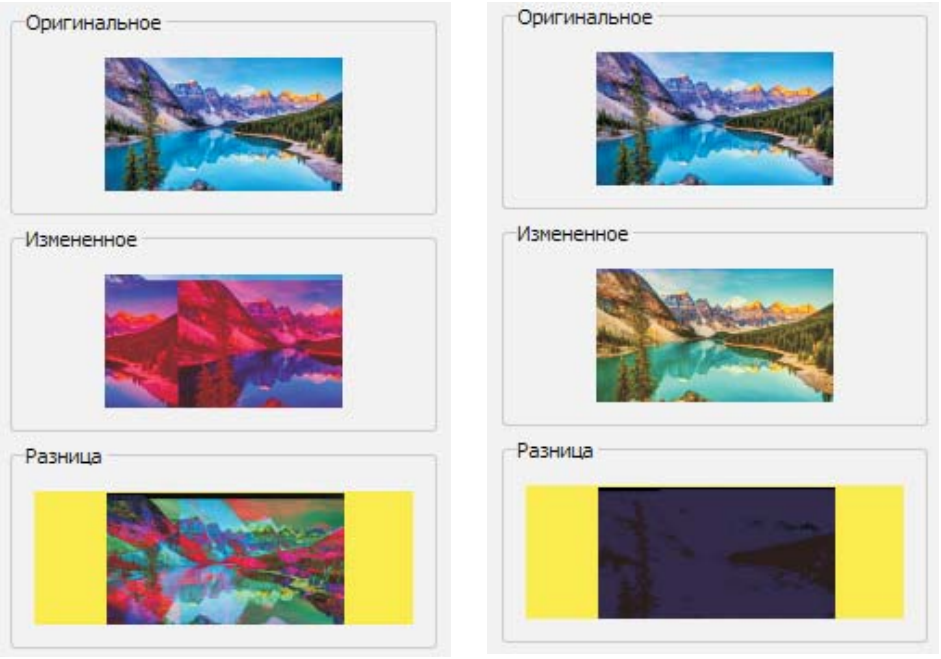
На рис. 1 приведены исходные изображения (оригиналы), выбранные для проведения экспериментов, а также тестовые изображения (измененные), созданные фаззерами Radamsa (*а*) и AFL (*б*) на их основе, и изображения, где показана разница между ними.

Сдвиг пикселей, осуществляемый фаззерами Radamsa и AFL, и изменение размера тестовых изображений, созданных на основе исходного, приведены в табл. 1.

Фаззеры Radamsa и AFL предоставляют большую возможность для создания тестовых данных и, соответственно, поиска ошибок, которые демонстрируются при построении модели.

Обработка результатов экспериментов по созданию тестовых данных с использованием имитационного моделирования. Обработка результатов экспериментов проводилась в среде *AnyLogic*.

Процесс построения модели в *AnyLogic* выполняется с использованием специального графического языка, описывающего свойства объектов и связи между ними. В рамках проведения исследований создана модель генерации графических изображений фаззеров Radamsa и AFL. На рис. 2 приведена модель в среде *AnyLogic* для сравнения алгоритмов фаззинга



а *б*

Рис. 1. Экземпляры тестовых данных, созданные фаззерами Radamsa (*а*) и AFL (*б*)

генерации тестовых данных графических изображений соответствующими фаззерами.

Таблица 1

Показатели исходного изображения и тестовых данных, созданных на его основе фаззерами Radamsa и AFL

Показатель	Radamsa	AFL
Свойства		
Разрешение (пикселей)	1280 × 720	1280 × 720
Размер (пикселей)	921,600 Кп	921,600 Кп
Размер файла 1	486 119/474,726 КБ	486 119/474,726 КБ
Размер файла 2	404 948/395,457 КБ	489 551/478,077 КБ
Формат файла 1	Joint Photographic Experts Group (jpg)	Joint Photographic Experts Group (jpg)
Формат файла 2	Joint Photographic Experts Group (jpg)	Joint Photographic Experts Group (jpg)
Каналы	3	3
Глубина канала	8U	8U

Окончание табл. 1

Показатель	Radamsa	AFL
Статистика		
Усредненная ошибка*	275,62772	102,03678
Минимальная ошибка	0	0
Максимальная ошибка*	255	106
Стандартное отклонение*	60,55504	16,38264
Среднеквадратическое отклонение (σ)*	282,20126	103,34358
Число ошибок (пикселей)*	921 429	915 399
Ошибка (% пикселей)*	99,98145	99,32715

* Превышает заданный порог

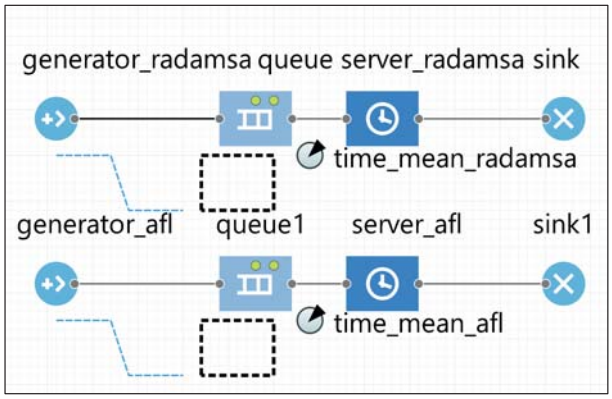


Рис. 2. Сравнение алгоритмов фаззинга генерации тестовых данных в среде *AnyLogic*

В целях последующего моделирования в среде *AnyLogic* процесса фаззинга используются следующие объекты и их характеристики.

1. Объект *Source* (или *generator*) создает новые экземпляры графических изображений. Временной интервал между запросами тестовых данных на обработку сервером задается согласно критерию Пирсона (12) (χ^2-критерий). Параметры *cpВремя_radamsa* и *cpВремя_afl* задаются согласно данным, полученным в ходе экспериментов с алгоритмами фаззинга программ для обработки изображений.

2. Объект *Queue* моделирует очередь сгенерированных экземпляров графических изображений, ожидающих освобождения сервера. Характеризуется параметром *емкостьБуфера*.

3. Объект *Delay* (или *server*) представляет в модели сервер, обрабатывающий сгенерированные экземпляры графических изображений. Время задержки сгенерированных экземпляров графических изображений также задается на основе (12). Параметры *time_mean_radamsa* и *time_mean_afl* определяются в ходе экспериментов алгоритмами фаззинга программ для обработки изображений.

4. Объект *Sink* используется в качестве конечной точки диаграммы процесса. Собирает данные о суммарном времени обработки сгенерированных экземпляров графических изображений (*сумВремяОбрЗапросов_radamsa*), среднем времени обработки одного сгенерированного экземпляра графического изображения (*срВремяОбрЗапроса_radamsa*), числе обработанных экземпляров графических изображений сервером (*колОбрЗапросов_radamsa*), средней длине очереди сгенерированных экземпляров графических изображений к серверу (*срДлинаОчереди_radamsa*) и о коэффициенте использования сервера (*коэфИспСервера_radamsa*) при поступлении с помощью переменных.

Далее с доверительной вероятностью $\alpha = 0,95$ и точностью $\varepsilon = 0,01$ определяется число итераций цикла модели

$$N = t_\alpha^2 \frac{p(1-p)}{\varepsilon^2} \approx 1,96^2 \frac{0,5^2}{0,01^2} \approx 9604, \qquad (14)$$

где $t_\alpha = 1,96$ — табулированный аргумент функции Лапласа; p — ожидаемая вероятность исхода события, в данном случае вероятность обработки запросов сервером. Выполнение 9604 итераций реализуется с помощью эксперимента *Варьирование параметров*. Расчет проведен для так называемого худшего случая, т. е. в предположении, что ожидаемая вероятность обработки запросов $p = 0,5$.

Результат моделирования при одной итерации приведен на рис. 3. Фаззер Radamsa генерирует новые экземпляры исходного графического изображения в 69 раз быстрее, чем AFL.

Для сбора статистики о длине очереди сгенерированных экземпляров графических изображений, ожидающих освобождения сервера, числе обработанных экземпляров графических изображений сервером, времени обработки одного сгенерированного экземпляра графического изображения сервером и коэффициенте использования сервера применены качественные и количественные характеристики. При проведении эксперимента по моделированию на 9604 итерациях использованы следующие параметры: *time_mean_radamsa* равен 10, *time_mean_afl* равен 619, *срВремя_radamsa*

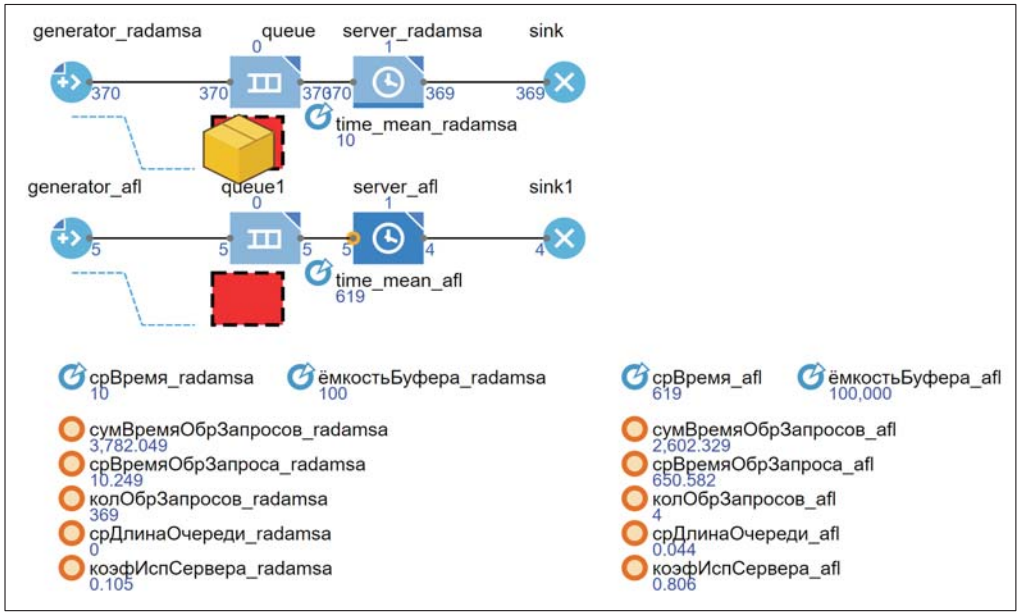


Рис. 3. Моделирование при условии одной итерации

равен 10, *срВремя_afl* равен 619, *емкостьБуфера_radamsa* равен 100, *емкостьБуфера_afl* равен 100 000.

За 3 ч 37 мин фаззером AFL создан 21 измененный экземпляр, таким образом, один экземпляр создается за 619 с. Фаззер Radamsa, в отличие от AFL, не имеет интерфейса, позволяющего наблюдать за ходом выполнения эксперимента генерации тестовых данных. Использование отсечек временных интервалов показало, что время работы фаззера Radamsa составило 16 мин 40 с, т. е. один экземпляр создается за 10 с. Установление значений емкости буфера в обоих случаях необходимо для реализации имитационного моделирования и определяется в соответствии со временем создания сгенерированных экземпляров графических изображений.

Результаты экспериментов, полученные с помощью моделирования создания графических изображений в среде *AnyLogic* (рис. 3 и 4) с использованием алгоритмов генерации тестовых данных в фаззерах Radamsa и AFL, приведены в табл. 2.

Обсуждение полученных результатов. В ходе экспериментов генерации тестовых данных установлено, что AFL больше задействует *server*, но медленнее генерирует новые экземпляры тестовых изображений из исходных. Средняя длина очереди запросов тестовых данных на обработку сервером Radamsa меньше, чем на обработку сервером AFL.

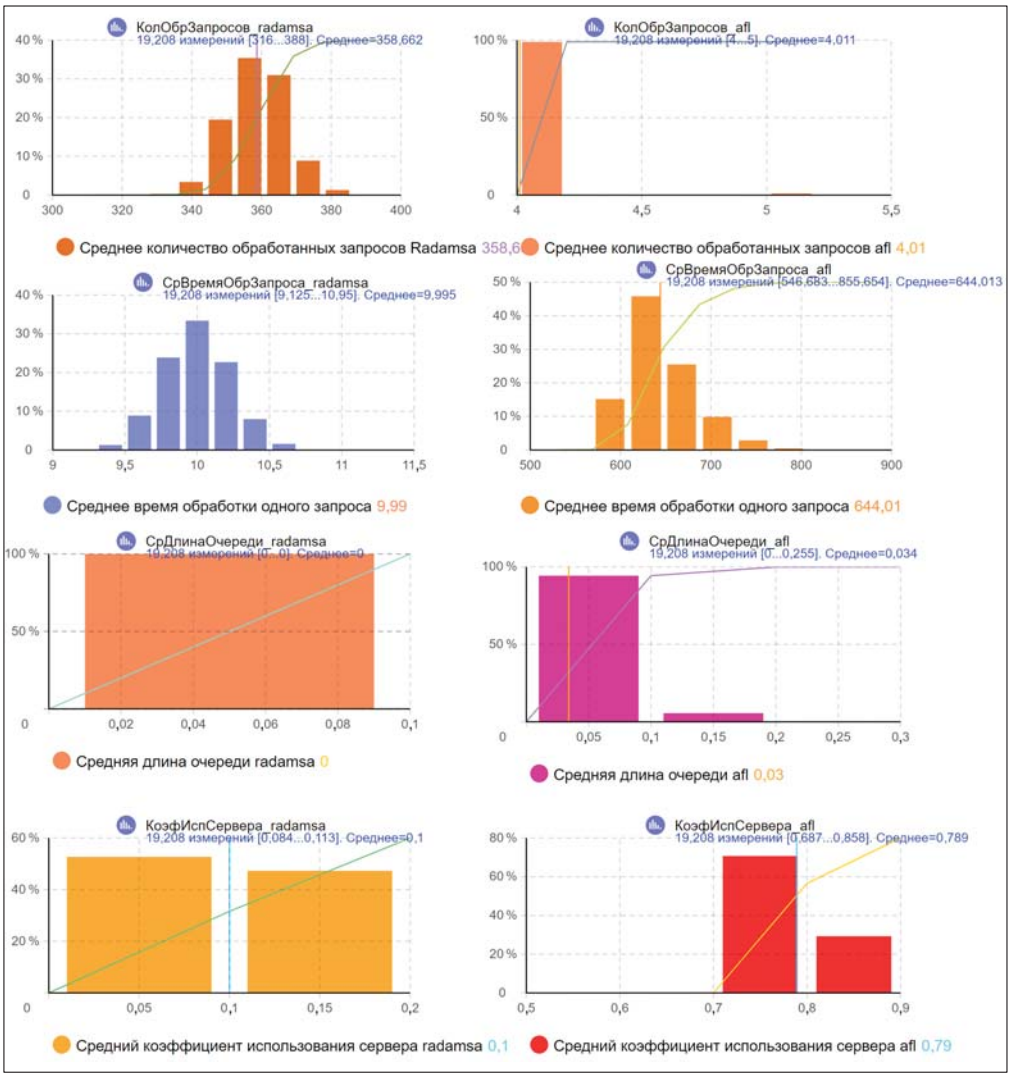


Рис. 4. Результаты моделирования при условии 9604 итераций в течение 3600 с

Таблица 2

Показатели работы алгоритмов создания графических изображений фаззеров Radamsa и AFL

Показатель	Одна итерация алгоритма в фаззере		9604 итерации алгоритма в фаззере	
	Radamsa	AFL	Radamsa	AFL
Среднее число обработанных экземпляров графических изображений сервером	353	4	358,565 (min 324; max 392; σ = 8,435)	4,011 (min 4; max 5; σ = 0,106)

Окончание табл. 2

Показатель	Одна итерация алгоритма в фаззере		9604 итерации алгоритма в фаззере	
	Radamsa	AFL	Radamsa	AFL
Среднее время обработки одного сгенерированного экземпляра графического изображения	9,886	680,152	9,994 (min 9,102; max 10,875; σ = 0,233)	643,022 (min 552,553; max 858,742; σ = 37,035)
Средняя длина очереди сгенерированных экземпляров графических изображений к серверу	0	0,046	0 (min 0; max 0; σ = 0)	0,033 (min 0; max 0,271; σ = 0,033)
Коэффициент использования сервера	0,097	0,786	0,1 (min 0,086; max 0,113; σ = 0,003)	0,789 (min 0,703; max 0,855; σ = 0,018)

Из данных табл. 2 следует, что применение динамических байесовских сетей для моделирования процесса фаззинга сформировало единую вычислительную структуру, воспроизводящую функциональную модель процесса тестирования. Вычисление оценок на основе метрики Байеса — Дирихле значительно снизило время обработки тестовых данных для каждого графа. А именно, за одно и то же время фаззер Radamsa способен создать в среднем в 89 раз больше новых экземпляров, чем AFL. Время генерации одного нового изображения с помощью фаззера Radamsa в среднем в 64 раза больше времени создания экземпляра фаззером AFL.

Таким образом, обработка полученных результатов с использованием имитационного моделирования в среде *AnyLogic* показала, что фаззер Radamsa превосходит AFL в генерации тестовых данных по ряду параметров, указанных в табл. 2.

Заключение. Приведено сравнение алгоритмов фаззинга генерации тестовых данных графических изображений фаззеров Radamsa и AFL. По результатам исследования выявлено, что Radamsa, как и AFL, не только изменяет цвета тестовых графических изображений, но и сдвигает пиксели, как следствие, изменяя размер выходного изображения.

Определено, что фаззер Radamsa превосходит AFL по числу обработанных экземпляров графических изображений, времени обработки од-

ного сгенерированного экземпляра графического изображения, длине очереди сгенерированных экземпляров графических изображений и коэффициенту использования сервера.

Определены наиболее эффективные алгоритмы, используемые при создании тестовых данных для поиска ошибок в исполняемом коде программ обработки графических изображений.

ЛИТЕРАТУРА

[1] Саттон М., Грин А., Амини П. Fuzzing: исследование уязвимостей методом грубой силы. СПб., Символ-плюс, 2009.

[2] Application Programming Interface (API). *tadviser.ru: веб-сайт*. URL: https://www.tadviser.ru/index.php/Статья:Application_Programming_Interface_(API) (дата обращения: 12.06.2019).

[3] CSNC-2017-023: buffer overflow in Mongoose MQTT Broker. *seclists.org: веб-сайт*. URL: https://seclists.org/fulldisclosure/2017/Sep/52 (дата обращения: 12.06.2019).

[4] Craig Y. Flawed MatrixSSL code highlights need for better IoT update practices. *tripwire.com: веб-сайт*. URL: https://www.tripwire.com/state-of-security/security-data-protection/cyber-security/flawed-matrixssl-code-highlights-need-for-better-iot-update-practices (дата обращения: 12.06.2019).

[5] From fuzzing Apache httpd server to CVE-2017-7668 and a $1500 bounty. *animal0day.blogspot.com: веб-сайт*. URL: https://animal0day.blogspot.com/2017/07/from-fuzzing-apache-httpd-server-to-cve.html (дата обращения: 12.06.2019).

[6] [openssh-commits] [openssh] 01/01: upstream commit. *lists.mindrot.org: веб-сайт*. URL: https://lists.mindrot.org/pipermail/openssh-commits/2014-November/004134.html (дата обращения: 12.06.2019).

[7] Mozilla NSS: wrong calculation results in mp_div() and mp_exptmod(). *blog.fuzzing-project.org: веб-сайт*. URL: https://blog.fuzzing-project.org/37-Mozilla-NSS-Wrong-calculation-results-in-mp_div-and-mp_exptmod.html (дата обращения: 12.06.2019).

[8] CVE-2018-0101 detail. *nvd.nist.gov: веб-сайт*. URL: https://nvd.nist.gov/vuln/detail/CVE-2018-0101 (дата обращения: 13.06.2019).

[9] CVE-2017-9845 detail. *nvd.nist.gov: веб-сайт*. URL: https://nvd.nist.gov/vuln/detail/CVE-2017-9845 (дата обращения: 13.06.2019).

[10] CVE-2017-9843 detail. *nvd.nist.gov: веб-сайт*. URL: https://nvd.nist.gov/vuln/detail/CVE-2017-9843 (дата обращения: 13.06.2019).

[11] Байбикова Т.Н. Комплексы программ для цифровой обработки изображений. *Вестник Московского финансово-юридического университета*, 2016, № 2, с. 255–266.

[12] Азарнова Т.В., Баркалов С.А., Полухин П.В. Управление процессом тестирования веб-приложений методом фаззинга на основе динамических байесовских сетей. *Вестник Южно-Уральского государственного университета. Серия: Компьютерные технологии, управление, радиоэлектроника*, 2017, т. 17, № 2, с. 51–64. DOI: https://doi.org/10.14529/ctcr170205

[13] Масленников Е.Д., Сулимов В.Б. Предсказания на основе байесовских сетей доверия: алгоритм и программная реализация. *Вычислительные методы и программирование*, 2010, № 11, с. 94–107.

[14] Азарнова Т.В., Аснина Н.Г., Проскурин Д.К. и др. Формирование структуры байесовской сети процесса тестирования надежности информационных систем. *Вестник Воронежского государственного университета*, 2017, № 6, с. 45–51.

[15] Торопова А.В. Подходы к диагностике согласованности данных в байесовских сетях доверия. *Труды СПИИРАН*, 2015, № 6, с. 156–178.

[16] Поиск восхождением к вершине (Hill Climbing). *drakon.su: веб-сайт*. URL: https://drakon.su/algoritmy/hill-climbing (дата обращения: 12.06.2019).

[17] Tsamardinos I., Brown L.E., Aliferis C.F. The max-min hill-climbing Bayesian network structure learning algorithm. *Mach. Learn.*, 2006, vol. 65, no. 1, pp. 31–78. DOI: https://doi.org/10.1007/s10994-006-6889-7

[18] Полухин П.В. Интеграция динамических байесовских сетей в процесс тестирования веб-приложений для выявления уязвимостей межсайтингового скриптинга. *Научное обозрение*, 2014, № 9, с. 414–422.

[19] Долгов А.И. О применимости формулы Байеса. *Вестник Донского государственного технического университета*, 2015, № 4, с. 107–115. DOI: https://doi.org/10.12737/16076

[20] Лемешко Б.Ю., Постовалов С.Н. О зависимости предельных распределений статистик Хи-квадрат Пирсона и отношения правдоподобия от способа группирования данных. *Заводская лаборатория. Диагностика материалов*, 1998, т. 64, № 5, с. 56–63.

[21] American fuzzy loop (2.52b). *lcamtuf.coredump.cx: веб-сайт*. URL: https://lcamtuf.coredump.cx/afl (дата обращения: 13.06.2019).

[22] A general-purpose fuzzer. *gitlab.com: веб-сайт*. URL: https://gitlab.com/akihe/radamsa (дата обращения: 13.06.2019).

[23] Shoshitaishvili Y., Wang R., Hauser C., et al. Firmalice — automatic detection of authentication bypass vulnerabilities in binary firmware. *Proc. NDSS*, 2015. DOI: https://doi.org/10.14722/ndss.2015.23294

[24] Генетический алгоритм. Просто о сложном. *habrahabr.ru: веб-сайт*. URL: https://habrahabr.ru/post/128704 (дата обращения: 13.06.2019).

[25] CVE-2017-12983. *nvd.nist.gov: веб-сайт*. URL: https://nvd.nist.gov/vuln/detail/CVE-2017-12983 (дата обращения: 13.06.2019).

[26] CVE-2016-8684. *nvd.nist.gov: веб-сайт.*
URL: https://nvd.nist.gov/vuln/detail/CVE-2016-8684 (дата обращения: 13.06.2019).

[27] CVE-2016-10095. *nvd.nist.gov: веб-сайт.*
URL: https://nvd.nist.gov/vuln/detail/CVE-2016-10095 (дата обращения: 13.06.2019).

[28] CVE-2012-2849. *nvd.nist.gov: веб-сайт.*
URL: https://nvd.nist.gov/vuln/detail/CVE-2012-2849 (дата обращения: 13.06.2019).

[29] CVE-2011-0205. *nvd.nist.gov: веб-сайт.*
URL: https://nvd.nist.gov/vuln/detail/CVE-2011-0205 (дата обращения: 13.06.2019).

Байдин Георгий Сергеевич — старший преподаватель кафедры «Информационная безопасность» МГТУ им. Н.Э. Баумана (Российская Федерация, 105005, Москва, 2-я Бауманская ул., д. 5, корп. 1).

Хизова Мария Владимировна — студентка кафедры «Информационная безопасность» МГТУ им. Н.Э. Баумана (Российская Федерация, 105005, Москва, 2-я Бауманская ул., д. 5, корп. 1).

Просьба ссылаться на эту статью следующим образом:

Байдин Г.С., Хизова М.В. Поиск ошибок в программах для обработки графических изображений с использованием метода фаззинга. *Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение*, 2021, № 3 (136), с. 4–23.
DOI: https://doi.org/10.18698/0236-3933-2021-3-4-23

FINDING ERRORS IN PROGRAMS FOR PROCESSING GRAPHIC IMAGES USING THE FUZZING METHOD

G.S. Baydin baydin.g.s@bmstu.ru
M.V. Khizova mariakhizova@mail.ru

Bauman Moscow State Technical University, Moscow, Russian Federation

Abstract

Increasing number of software for automated graphics processing requires effective testing methods. One of these methods is fuzzing, for which it is necessary to determine the most effective algorithms for creating test data in order to increase the number of errors found and minimize hardware resources. The comparison of algorithms for creating test data for finding errors in the executable code of programs designed for processing graphic images is the result of the performed research. Using Bayesian networks to describe fuzzing allows determining the relationships between structural components during testing. Based

Keywords

Fuzzing, graphic images, Bayesian networks, fuzzer Radamsa, AFL, AnyLogic

on the results of the comparison of fuzzing algorithms for creating test data, the most effective algorithms for finding errors in the executable code of programs for processing graphic images have been identified. The performance of the proposed algorithms was tested on a number of existing vulnerabilities classified as CVE (Common Vulnerabilities and Exposures). The processing of the results of experiments on the creation of test data was carried out using the simulation environment, allowing analyzing the testing process step by step. The obtained research results, algorithms for creating test data for finding errors can be used at various stages of software testing

Received 04.11.2020
Accepted 04.12.2020

REFERENCES

[1] Sutton M., Green A., Amini P. Fuzzing. Brute force vulnerability discovery. Addison-Wesley, 2007.

[2] Application Programming Interface (API). *tadviser.ru: website*.
Available at: https://www.tadviser.ru/index.php/Stat'ya:Application_Programming_Interface_(API) (accessed: 12.06.2019) (in Russ.).

[3] CSNC-2017-023: buffer overflow in Mongoose MQTT Broker. *seclists.org website*. Available at: https://seclists.org/fulldisclosure/2017/Sep/52 (accessed: 12.06.2019).

[4] Craig Y. Flawed MatrixSSL code highlights need for better IoT update practices. *tripwire.com: website*. Available at: https://www.tripwire.com/state-of-security/security-data-protection/cyber-security/flawed-matrixssl-code-highlights-need-for-better-iot-update-practices (accessed: 12.06.2019).

[5] From fuzzing Apache httpd server to CVE-2017-7668 and a $1500 bounty. *animal0day.blogspot.com: website*. Available at: https://animal0day.blogspot.com/2017/07/from-fuzzing-apache-httpd-server-to-cve.html (accessed: 12.06.2019).

[6] [openssh-commits] [openssh] 01/01: upstream commit. *lists.mindrot.org: website*. Available at: https://lists.mindrot.org/pipermail/openssh-commits/2014-November/004134.html (accessed: 12.06.2019).

[7] Mozilla NSS: wrong calculation results in mp_div() and mp_exptmod(). *blog.fuzzing-project.org: веб-сайт*. Available at: https://blog.fuzzing-project.org/37-Mozilla-NSS-Wrong-calculation-results-in-mp_div-and-mp_exptmod.html (accessed: 12.06.2019).

[8] CVE-2018-0101 detail. *nvd.nist.gov: website*.
Available at: https://nvd.nist.gov/vuln/detail/CVE-2018-0101 (accessed: 13.06.2019).

[9] CVE-2017-9845 detail. *nvd.nist.gov: website*.
Available at: https://nvd.nist.gov/vuln/detail/CVE-2017-9845 (accessed: 13.06.2019).

[10] CVE-2017-9843 detail. *nvd.nist.gov: website*.
Available at: https://nvd.nist.gov/vuln/detail/CVE-2017-9843 (accessed: 13.06.2019).

[11] Baybikova T.N. Software pack for digital image processing. *Vestnik Moskovskogo finansovo-yuridicheskogo universiteta*, 2016, no. 2, pp. 255–266 (in Russ.).

[12] Azarnova T.V., Barkalov S.A., Polukhin P.V. Management of the process of web applications testing by the fuzzing method based on dynamic Bayesov networks. *Vestnik Yuzhno-Ural'skogo gosudarstvennogo universiteta. Seriya: Komp'yuternye tekhnologii, upravlenie, radioelektronika* [Bulletin of the South Ural State University. Series Computer Technology, Aotimatic Control, Radio Electronics], 2017, vol. 17, no. 2, pp. 51–64 (in Russ.). DOI: https://doi.org/10.14529/ctcr170205

[13] Maslennikov E.D., Sulimov V.B. Bayesian network prediction: algorithm and software implementation. *Vychislitel'nye metody i programmirovanie* [Numerical Methods and Programming], 2010, no. 11, pp. 94–107 (in Russ.).

[14] Azarnova T.V., Asnina N.G., Proskurin D.K., et al. Bayesian network structure formation of information systems reliability testing process. *Vestnik Voronezhskogo gosudarstvennogo universiteta* [Bulletin of Voronezh State Technical University], 2017, no. 6, pp. 45–51 (in Russ.).

[15] Toropova A.V. Approaches to the data coherence diagnosis in Bayesian belief network models. *Trudy SPIIRAN* [SPIIRAS Proceedings], 2015, no. 6, pp. 156–178 (in Russ.).

[16] Poisk voskhozhdeniem k vershine [Hill Climbing]. *drakon.su: website*.
Available at: https://drakon.su/algoritmy/hill-climbing (accessed: 12.06.2019) (in Russ.).

[17] Tsamardinos I., Brown L.E., Aliferis C.F. The max-min hill-climbing Bayesian network structure learning algorithm. *Mach. Learn.*, 2006, vol. 65, no. 1, pp. 31–78.
DOI: https://doi.org/10.1007/s10994-006-6889-7

[18] Polukhin P.V. Integration of dynamic Bayesian networks into the process of testing web applications for the purpose of determining the vulnerabilities of interwebsite scripting. *Nauchnoe obozrenie*, 2014, no. 9, pp. 414–422 (in Russ.).

[19] Dolgov A.I. On applicability of Bayes' formula. *Vestnik Donskogo gosudarstvennogo tekhnicheskogo universiteta* [Vestnik of Don State Technical University], 2015, no. 4, pp. 107–115 (in Russ.). DOI: https://doi.org/10.12737/16076

[20] Lemeshko B.Yu., Postovalov S.N. Limit distributions of the Pearson χ^2 and likelihood ratio statistics and their dependence on the mode of data grouping. *Industrial Laboratory*, 1998, vol. 64, no. 5, pp. 344–351.

[21] American fuzzy loop (2.52b). *lcamtuf.coredump.cx: website*.
Available at: https://lcamtuf.coredump.cx/afl (accessed: 13.06.2019).

[22] A general-purpose fuzzer. *gitlab.com: website*.
Available at: https://gitlab.com/akihe/radamsa (accessed: 13.06.2019).

[23] Shoshitaishvili Y., Wang R., Hauser C., et al. Firmalice — automatic detection of authentication bypass vulnerabilities in binary firmware. *Proc. NDSS*, 2015.
DOI: https://doi.org/10.14722/ndss.2015.23294

[24] Geneticheskiy algoritm. Prosto o slozhnom [Genetic algorithm. Simple answer on complex issue]. *habrahabr.ru: website*.
Available at: https://habrahabr.ru/post/128704 (accessed: 13.06.2019) (in Russ.).

[25] CVE-2017-12983. *nvd.nist.gov: website.*
Available at: https://nvd.nist.gov/vuln/detail/CVE-2017-12983 (accessed: 13.06.2019).

[26] CVE-2016-8684. *nvd.nist.gov: website.*
Available at: https://nvd.nist.gov/vuln/detail/CVE-2016-8684 (accessed: 13.06.2019).

[27] CVE-2016-10095. *nvd.nist.gov: website.*
Available at: https://nvd.nist.gov/vuln/detail/CVE-2016-10095 (accessed: 13.06.2019).

[28] CVE-2012-2849. *nvd.nist.gov: website.*
Available at: https://nvd.nist.gov/vuln/detail/CVE-2012-2849 (accessed: 13.06.2019).

[29] CVE-2011-0205. *nvd.nist.gov: website.*
Available at: https://nvd.nist.gov/vuln/detail/CVE-2011-0205 (accessed: 13.06.2019).

Baydin G.S. — Lecturer, Department of Information Security, Bauman Moscow State Technical University (2-ya Baumanskaya ul. 5/1, Moscow, 105005 Russian Federation).

Khizova M.V. — student, Department of Information Security, Bauman Moscow State Technical University (2-ya Baumanskaya ul. 5/1, Moscow, 105005 Russian Federation).

Please cite this article in English as:

Baydin G.S., Khizova M.V. Finding errors in programs for processing graphic images using the fuzzing method. *Herald of the Bauman Moscow State Technical University, Series Instrument Engineering*, 2021, no. 3 (136), pp. 4–23 (in Russ.).
DOI: https://doi.org/10.18698/0236-3933-2021-3-4-23