АРХИТЕКТУРА СИСТЕМЫ ДЛЯ ПРОВЕДЕНИЯ ГЕНЕТИЧЕСКОГО РЕИНЖИНИРИНГА ПРОГРАММЫ С ПОДДЕРЖКОЙ ПОИСКА РАЗНОУРОВНЕВЫХ УЯЗВИМОСТЕЙ

Израилов К. Е.[1]

DOI: 10.21681/2311-3456-2025-1-108-116

Цель исследования: *повышение эффективности поиска уязвимостей в машинном коде программ путем его реверс-инжиниринга на базе генетического реинжиниринга, для чего предлагается архитектура соответствующей программной системы.*

Методы исследования: *обзор работ, системный анализ, структурный синтез архитектуры, аналитическое моделирование.*

Полученные результаты: *создана архитектура системы, представляющая собой совокупность последовательно выполняемых однотипных компонентов для деэволюции представлений исследуемой программы (ее машинного, ассемблерного и исходного кода, алгоритмов и пр.); на каждом из таких представлений осуществляется поиск соответствующих уязвимостей.*

Научная новизна *заключается в качественно новом развитии направления реверс-инжиниринга путем его интеллектуализации, для чего предлагается высокоуровневое описание архитектуры авторской системы генетического реинжиниринга, а также производится формализация функционирования ее элементов.*

Ключевые слова: *обратная разработка, обратный инжиниринг, генетический алгоритм, уязвимость, машинный код, архитектура, формализация.*

Введение

Уязвимости в программном обеспечении (далее – ПО) представляют существенную проблему для безопасности обрабатываемой информации [1]. Одним из наиболее эффективных путей противодействия является их поиск в программном коде с последующей нейтрализацией или отказом от использования небезопасного ПО [2, 3]. В случае отсутствия исходного кода программ применение ручного поиска уязвимостей в выполняемом (машинном, байт-, ином) коде имеет высокую трудоемкость и низкую оперативность, а автоматические средства – недостаточную результативность [4].

Уязвимости имеют наиболее явное отражение в тех представлениях программы (далее – Представление), в которых они были заложены и под которыми понимается состояние программы на некотором из этапов ее создания. В [5] были выделены следующие Представления – от наиболее высокоуровневых и человеко-ориентированных к низкоуровневым и машинно-ориентированным: идея, концептуальная модель, архитектура, алгоритмы, исходный код, ассемблерный код, машинный (далее – МК) или байт-код. Это принципиально усложняет задачу поиска всех уязвимостей только по одному («финишному») Представлению – выполняемому коду.

В этих обстоятельствах предпочтительным подходом может явиться предварительный реверс-инжиниринг (т.н. обратная разработка или реинжиниринг, далее – РИ) низкоуровневых Представлений к более высокоуровневым с применением соответствующих методов поиска уязвимостей на каждом из них. Данная задача РИ также считается сложной как с технической, так и с практической точек зрения и является отдельным научным направлением [6]. В связи с этим автором исследуется возможность деэволюции Представлений, основанная на применении искусственного интеллекта в части генетических алгоритмов (далее – ГА) для восстановления предыдущих Представлений по текущему (что позволяет говорить о полноценном *эволюционном реинжиниринге* программы). Суть такой генетической деэволюции заключается в итеративном решении оптимизационной задачи подбора кода программы (в текущей терминологии – искомой) в предыдущем Представлении, из которого был получен код заданной программы (в текущей терминологии – исследуемой) в текущем Представлении, что было многократно описано в авторских публикациях [7–10]. Согласно [9] любая программа представима в виде хромосомы их набора генов, каждый из которых в общем

1 Израилов Константин Евгеньевич, кандидат технических наук, доцент, старший научный сотрудник лаборатории проблем компьютерной безопасности Санкт-Петербургского Федерального исследовательского центра Российской академии наук, Санкт-Петербург, ORCHID: http://orcid.org/0000-0002-9412-5693. Scopus Author ID: 56122749800. E-mail: konstantin.izrailov@mail.ru

случае ответственен за одну из конструкций языка программирования (символ, токен, синтаксическое правило и т.п.). Тогда задача сводится к «умному» (с позиции качественной минимизации количества вариантов) подбору генов для получения искомой программы, преобразуемой (например, компиляцией из языка С в язык Ассемблер) в исследуемую. Для этого создается популяция программ (т.е. множество особей с различными хромосомами), селекция их экземпляров, преобразуемых в близкие к искомой программе (оценка близости осуществляется с помощью так называемой фитнес-функции), скрещивание генов полученных таким образом хромосом (т.е. две «наилучшие» программы дают некоторую третью, близкую к им обоим), мутация отдельных генов (т.е. случайное изменение конструкций программы для избежания локальных экстремумов) и повторение процесса с момента селекции; когда одна из программ оказывается в точности преобразуемой в исследуемую (т.е. найден глобальный экстремум), то это означает решение задачи деэволюции Представлений.

Ранее был получен ряд моделей, методов и алгоритмов, теоретически решающих данную задачу, а частные эксперименты показали подтверждение данной (авторской) концепции. Для уточнения возможности практического создания необходимых программных средств изначально требуется проектирование архитектуры системы для проведения генетического РИ программы с поддержкой поиска разноуровневых уязвимостей (далее – Система), что и будет предложено в настоящей статье.

Обзор работ

Проведем обзор работ, в которых освещаются принципы и функциональные возможности систем поиска уязвимостей, основанных на РИ Представлений программы от низкоуровневых – ассемблерных, к более высокоуровневым – исходному коду, алгоритмам и т.п.

В качестве классических и хорошо известных систем можно привести IDA Pro [11] и Ghidra [12], предназначенные для дизассемблирования МК программ и поддерживающие большое количество семейств процессоров. Также в состав первой системы входит плагин декомпиляции Hex-Rays [6], во второй же данный функционал является встроенным; впрочем, восстановление исходного кода ограничено заданным набором поддерживаемых процессоров. Также ассемблерное представление программы может быть отображено в виде блок-схем, которые не являются полноценными алгоритмами, поскольку в их элементах все также содержатся машинные инструкции, а граф управления строится исключительно по низкоуровневым конструкциям условного перехода (и их аналогам). Поиск уязвимостей в обеих системах может осуществляться либо вручную экспертом (т.е. высококвалифицированным специалистом по безопасности кода, далее – Эксперт), либо с применением внешних анализаторов через программные интерфейсы (на языках Python и Java для продуктов, соответственно).

В работе [13] представлена архитектура программного комплекса для динамического анализа МК, состоящая из собственно среды анализа, а также среды выполнения кода и средств разработки виртуальных машин. Комплекс производит запуск исследуемой программы в виртуальной среде и собирает информацию об ее работе (трассы выполнения, снимки памяти и т.п.). Строится статико-динамическое отображение программы, позволяющее получать блок-схемы МК и восстанавливать форматы данных. Одним из сценариев работы комплекса является символьное выполнение программы, позволяющее расширить покрытие кода и выявить условия, приводящие к реализации уязвимостей различного типа. Могут быть добавлены новые сценарии за счет имеющегося программного интерфейса.

В ранней авторской работе [14] описывается архитектура комплекса, предназначенная для преобразования МК в форму, подходящую Эксперту для анализа на предмет уязвимостей. Особенностями архитектуры является наличие модулей для получения метаинформации об уязвимостях, визуализации кода, оценки качества восстановленного кода и корректировки процесса декомпиляции Экспертом.

Работа [15] посвящена созданию средств автоматического поиска уязвимостей в МК, основанных на расширенном представлении его алгоритмов и архитектуры, заданном парадигмой из 7 положений. При этом каждое из средств может работать с единым отображением программы, но обнаруживать уязвимости своего класса. Описан типовой шаблон архитектуры таких средств, состоящих из алгоритмов поиска – сигнатурного, эвристического, интеллектуального и статического фаззинга; архитектура поддерживает подключение внешних модулей для алгоритмической обработки Представлений программы, спецификации уязвимостей и взаимодействия с Экспертом.

В исследовании [16] описывается программный комплекс для совместного применения РИ и алгоритмов машинного обучения с целью обнаружения вредоносного ПО для Android. В качестве признаков классификации выбраны такие, как разрешения приложений, вызовы системных функций, используемые сервисы и др. Для РИ применяется внешняя утилита Jadx-Gui [17], получающая исходный Java-код и файлы манифеста из APK-файлов (архивных исполняемых пакетов для Android).

В [18] приводится программный продукт SLaDe (близкий к полноценному комплексу), предназначенный для декомпиляции и отличающийся от аналогов генерацией более человеко-ориентированного исходного кода за счет интеллектуализации. Показано преимущество продукта по сравнению с Ghidra и моделью ChatGPT (в части улучшения читаемости кода) на функциях из датасета ExeBench [19].

Согласно проведенному обзору, существует определенное количество комплексов, использующих РИ для поиска уязвимостей и применяющих различные техники. Тем не менее, все они ограничены лишь рядом Представлений программы, языками программирования, платформами и способами последующего поиска, а также применяют частные архитектурные решения. Однако, комплексов или их аналогов, хотя бы близких к решению текущей задачи, обнаружено не было.

Функциональные требования

Исходя из предыдущих (более теоретико- и прототип-ориентированных) исследований разных авторов, были выдвинуты следующие условно пронумерованные требования к архитектуре Системы:

1. Преобразование Представлений от низкоуровневых к высокоуровневым;
2. Применение единого шаблона ГА для преобразования Представлений;
3. Независимость алгоритмов деэволюции от синтаксисов кода в Представлениях;
4. Возможность поиска уязвимостей в каждом из Представлений;
5. Автоматическое определение длины хромосомы – важнейшего элемента ГА [20], используемого при составлении программы в искомом Представлении и существенно ускоряющего решение задачи деэволюции;
6. Частичная оптимизация самого процесса ГА;
7. Поддержка неограниченного количества Представлений, применяемых в программном инжиниринге для последовательного преобразования между ними;
8. Возможность управления процессом Экспертом.

Все указанные функциональные требования могут быть удовлетворены следующими способами, системно связанными друг с другом. Последовательное преобразование между Представлениями программы реализуемо путем построения каскада из однотипных компонентов деэволюции, выход каждого из которых является входом для следующего (требование 1). Каждая такая деэволюция должна строиться на базе ГА без использования строго заданных алгоритмов преобразований или статистических закономерностей между конструкциями языков программирования; а исходя из сути ГА (как последовательного создания популяций особей-программ, заданных хромосомами, с выбором наилучших представителей, над которыми осуществляются операции скрещивания и мутации их хромосом), его ядро должно использовать сравнение программ в исследуемом и искомом Представлениях (требование 2). Поскольку Представления, входные и выходные для компонента деэволюции, задаются формальным синтаксисом, то реализация самого ядра преобразований может быть общей и располагаться в одном модуле (требование 3). Для каждого такого Представления должен существовать набор методов поиска уязвимостей, как общих – например, сигнатурных, так и специализированных – например, экспертных (требование 4). Определение длины хромосомы может осуществляться путем генерации множества программ в искомом Представлении и преобразования их в исследуемое Представление с определением длины последних, что даст статистическое соответствие между этими длинами (требование 5). Также возможна частичная оптимизация процесс ГА на основании обрабатываемого им Представления программы; например, выбором более «подходящей» первоначальной популяции, отличной от случайно сгенерированной, или уточнением алгоритмов ГА (требование 6). Построение стека из последовательности однотипных компонентов деэволюции гипотетически позволит производить преобразование между Представлениями программы любой длины и состава (требование 7). Для последовательного преобразования Представлений посредством стека компонентов деэволюции, а также поиска в них уязвимостей, целесообразно иметь отдельный компонент управления, взаимодействующий с Экспертом (требование 8).

Схема архитектуры

Концептуальная схема архитектуры Системы, соответствующая заданным требованиям и способам их удовлетворения, предлагаемая автором, состоит из последовательности компонентов деэволюции Представлений, соответствующей pipeline-дизайну [21] – т.е. когда данные из одного компонента поступают на вход другого. При этом логика работы каждого такого компонента не зависит от синтаксисов Представлений, а сами синтаксисы (и ряд других данных) являются его параметрами. Исходя из того, что компоненты и их модули фактически представляют собой некоторые обработчики данных – т.е. классические функции, их можно формализовать следующим образом:

$$R_{i-1} = Compoment^{Deevolution}(R_i P_i^D),$$

где $Component^{Deevolution}(\ldots)$ – компонент деэволюции i-го Представления (Deevolution – *перев. на русс.*

Деэволюция); P_i^D – параметры компонента (D – *аббр. от англ.* Deevolution); R_{i-1} и R_i – программа в искомом (предыдущем) и исследуемом (текущем) Представлениях, соответственно. Тогда вся совокупность компонентов деэволюции представима в следующем формальном виде:

$$R_1 = Compoment^{Deevolution}(R_2, P_2^D) \circ \ldots \circ Compoment^{Deevolution}(R_n, P_n^D),$$

где «∘» – оператор последовательного применения $Compoment^{Deevolution}(R,\ldots)$, выходные данные которого являются входными для следующего компонента, передаваемого в него через 1-й параметр (т.е. R_i).

Для поиска уязвимостей предназначен собственный компонент, принимающий на вход необходимое Представление, а также дополнительные параметры; его формальная запись следующая:

$$V_i = Compoment^{FindVulnerability}(R_i, P_i^{FV}),$$

где $Compoment^{FindVulnerability}(\ldots)$ – компонент поиска уязвимостей в i-ом Представлении (FindVulnerability – *перев. на русс.* Поиск Уязвимостей); V_i – уязвимости, найденные в i-ом Представлении; P_i^{FV} – параметры компонента (FV – *аббр. от англ.* FindVulnerability).

В результате работы компонента $Compoment^{FindVulnerability}(\ldots)$ происходит выявление уязвимостей в определенном Представлении, что также может осуществляться по обобщенным алгоритмам, параметрически настраиваемым (например, передачей в них сигнатур искомых объектов) [22, 23].

Компонент, контролирующий весь процесс, получает на вход некоторые управляющие команды от Эксперта, выполняя определенную логику запуска деэволюции и поиска уязвимостей. Формальная запись компонента будет опущена, поскольку он не обрабатывает какие-либо существенные информационные потоки, а лишь последовательно вызывает другие компоненты.

Более детальная компонентно-модульная схема архитектуры Системы, соответствующая заданным требованиям и способам реализации (отражающая только один компонент деэволюции кроме всего стека – т.н. архитектурный блок), представлена на Рисунке 1; использованы следующие обозначения: серый пунктирный прямоугольник – компонент, синий прямоугольник – модуль компонента, зеленый прямоугольник с прямыми краями – внешнее программное средство (вызываемое компонентом), зеленый

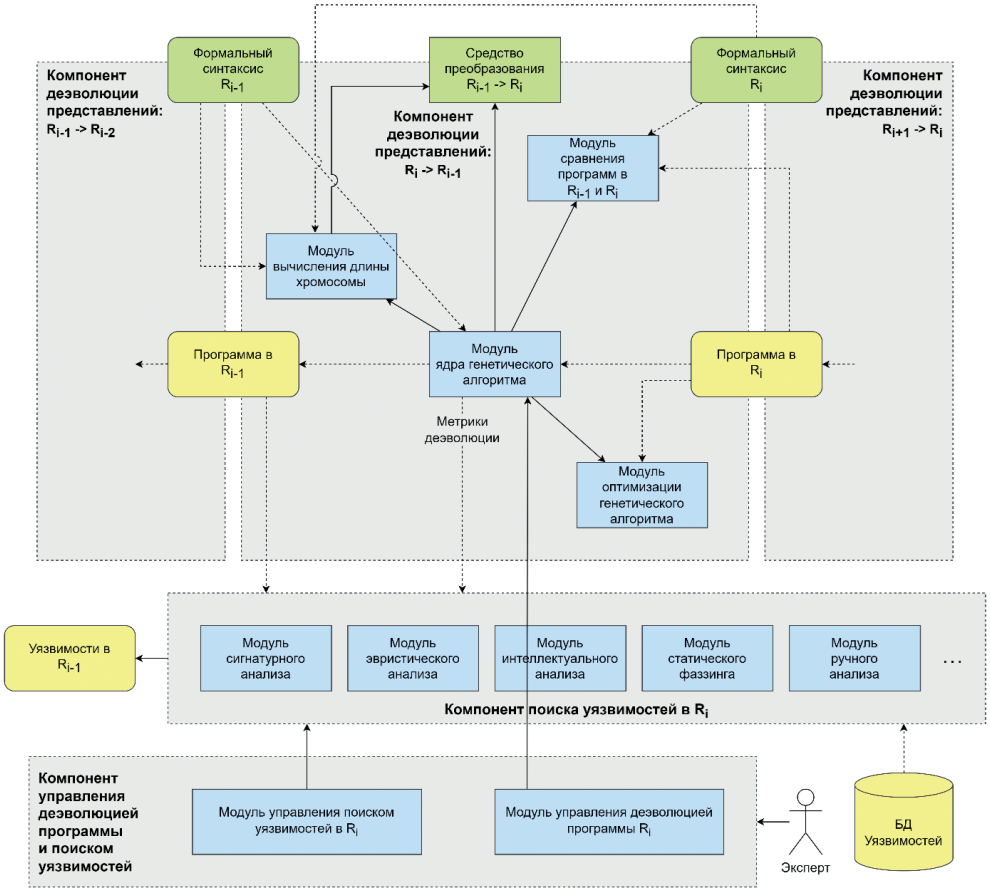

Рис. 1. Схема архитектуры системы для проведения генетического реинжиниринга программы с поддержкой поиска разноуровневых уязвимостей

прямоугольник со сглаженными краями – внешние данные (передаваемые в компоненты), желтый прямоугольник – генерируемые компонентами данные, сплошная линия – передача управления между модулями и компонентами (включая обмен данными), пунктирная линия – передача данных между модулями и компонентами, подписи к пунктирным стрелкам – передаваемые между модулями и компонентами данные.

Дадим описание элементов схемы (см. Рис. 1) и их частичную формальную запись с указанием основных параметров и возвращаемых данных.

Компонент деэволюции $Compoment^{Deevolution}$ (…) помимо программы в i-ом Представлении принимает на вход следующий параметр (указанный ранее):

$$P_i^D \equiv \langle S_i, S_{i-1}, Tool_i \rangle,$$

где S_i и S_{i-1} – формальный синтаксис i-го и $i-1$-го Представлений; $Tool_i$ – внешнее средство прямого преобразования программы из $i-1$-го Представления в i-ое Представление, т.е.:

$$R_i = Tool_i\ (R_{i-1}).$$

Примером средства преобразования является классический компилятор программ на языке С (т.е. в предыдущем Представлении), получающий ее в ассемблерном или бинарном виде (т.е. в текущем Представлении).

Компонент деэволюции состоит из набора взаимодействующих модулей, что определяет его следующим образом:

$$Compoment^{Deevolution} \equiv \langle Module_{Core}, Module_{ChromLength}, Module_{Comparison}, Module_{Optimization} \rangle,$$

где $Module_{Core}$ – модуль ядра ГА (*от англ.* Core, *перев. на русс.* Ядро), ответственного за решения задачи подбора программы в $i-1$-ом Представлении, преобразуемой в i-ое Представление; $Module_{ChromLength}$ – модуль вычисления длины хромосомы (*от англ.* Chromosome Length, *перев. на русс.* Длина Хромосомы) для записи программы в $i-1$-ом Представлении, которая может быть преобразована в i-ое Представление путем подбора конструкций программы (т.е. генов хромосомы особи популяции); $Module_{Comparison}$ – модуль нахождения близости двух программ (*от англ.* Comparison, *перев. на русс.* Сравнение) в одном Представлении (т.е. реализация фитнес-функции ГА); $Module_{Optimization}$ – модуль, оптимизирующий работу ГА (*от англ.* Optimization, *перев. на русс.* Оптимизация), т.е. уменьшающий количества создаваемых популяций (например, путем выбора изначальной популяции, близкой к искомой, улучшения точности фитнес-функции или уточнения настроек алгоритмов селекции, скрещивания и мутации).

Работа каждого из модулей компонента деэволюции может быть формально записана следующим образом:

1) модуль вычисления длины хромосомы:

$$Length_{i-1} = Module_{ChromLength}\ (S_i, S_{i-1}, \overline{Tool_i}),$$

где $Length_{i-1}$ – длина хромосомы для записи программы в $i-1$-ом Представлении; здесь и далее черта над параметром означает передачу управления (с обменом данными) – т.е. средство прямого преобразования $Tool_i$ не передается в модуль, а вызывается (с передачей в него программы в $i-1$-ом Представлении и возвратом из него программы в i-ом Представлении).

2) модуль сравнения программ:

$$Fitness = Module_{Comparison}\ (S_i, R1_i, R2_i),$$

где $Fitness$ – результат оценки близости двух программ в одном Представлении (т.е. S_i); $R1_i$ и $R2_i$ – первая и вторая программа в i-ом Представлении (согласно логике работы ГА, ими являются одна из особей популяции и исследуемая программа).

3) модуль оптимизации ГА:

$$GAS = Module_{Optimization}\ (R_i),$$

где GAS – набор настроек ГА (*аббр. от англ.* Genetic Algorithm Settings, *перев. на русс.* Настройки Генетического Алгоритма), определенных анализом исследуемой программы в i-ом Представлении.

4) модуль ядра ГА (в отличие от остальных модулей возвращающий кортеж данных):

$$\langle R_{i-1}, Metrics \rangle = Module_{Core}\ (\overline{R_i}, \overline{Length_{i-1}}, GAS, \overline{Module_{Comparison}}, \overline{Tools_i}),$$

где $Metrics$ – множество метрик деэволюции, отражающих признаки уязвимостей в программе (например, аномалии в процессе деэволюции Представлений в виде чрезмерно длительного подбора блоков конструкций программы могут служить о частичном внешнем разрушении ее структуры из-за внедрения уязвимости в более низкоуровневое Представление).

Компонент $Compoment^{FindVulnerability}$ () помимо программы в i-ом Представлении принимает на вход следующий параметр (указанный ранее):

$$P_i^{FV} \equiv \langle Metrics, DB^{Vulnerability} \rangle,$$

где $DB^{Vulnerability}$ – полная база уязвимостей с их разделением по Представлениям (включая характеристики, необходимые для поиска).

Компонент поиска уязвимостей состоит из набора модулей, что определяет его следующим образом:

$$Compoment^{FindVulnerability} \equiv \langle Module_{Analysis_1}, \ldots, \ldots, \langle Module_{Analysis_N} \rangle,$$

где $Module_{Analysis_i}$ – модуль анализа, работающий по i-ому принципу, такому, как сигнатурный анализ

(поиск уязвимостей по определенным шаблонам) [24], интеллектуальный анализ (применение искусственного интеллекта для обнаружения уязвимостей) [25], статический фаззинг (перебор параметров программы или одной из ее подпрограмм для выявления системных исключений [26]), ручной анализ (т.е. исследование программы в текущем Представлении Экспертом) и др. Модули могут работать параллельно, анализируя восстановленное Представление и формируя список найденных уязвимостей (в более общем случае – подозрительных мест, метрик безопасности и пр.). Также модули компонента могут иметь принципы работы, инвариантные к анализируемому преставлению, а специфика их работы в этом случае будет учитываться характеристиками уязвимостей, передаваемыми через параметры:

$$V_{i-1,k} = Module_{Analysis_k}(R_{i-1}, P_i^A),$$

где $V_{i-1,k}$ – множество уязвимостей, найденных в $i-1$-ом Представлении с помощью k-го модуля поиска, т.е. все уязвимости в i-ом Представлении:

$$V_i = \bigcup_k V_{i,k};$$

а P_i^A – параметр для работы модулей по анализу i-го Представления, который определяется следующим образом:

$$P_i^A \equiv \langle DB_i^{Vulnerability} \rangle,$$

где $DB_i^{Vulnerability}$ – база данных уязвимостей, относящихся к i-ому Представлению, где содержатся в том числе их характеристики, необходимые для поиска (например, сигнатуры [27]); данная частная база данных входит в более общую, передаваемую в компонент поиска уязвимостей, которая может быть записана, как:

$$DB^{Vulnerability} \equiv \bigcup_i DB_i^{Vulnerability}.$$

Компонент управления деэволюцией программы Compoment^Control (от англ. Управление) и поиском уязвимостей может быть формально записан следующим образом:

$$Compoment^{Control} \equiv$$
$$\equiv \langle Module_{ControlDeevolution}, Module_{ControlFindVulnerability} \rangle,$$

где $Module_{ControlDeevolution}$ и $Module_{ControlFindVulnerability}$ – модули управления процессом деэволюции и поиска уязвимости, соответственно.

В простейшем случае, компонент управления последовательно запускает компоненты деэволюции исследуемого Представления, применяя для восстанавливаемых (т.е. предыдущих) Представлений компонент поиска уязвимостей, обновляя тем самым их множество; следовательно, он может быть записан следующим образом:

$$\langle R, V \rangle = Compoment^{Control}(\overline{Component^{Deevolution}}, \overline{Component^{FindVulnerability}}),$$

где R – множество всех восстановленных Представлений, а V – множество всех найденных уязвимостей, т.е.:

$$\begin{cases} \mathrm{R} = \bigcup_i R_i \\ \mathrm{V} = \bigcup_j V_j \end{cases}.$$

Естественно, каждый компонент может быть дополнен другими необходимыми модулями без нарушения общей схемы функционирования Системы. Например, возможно расширение компонента деэволюции дополнительными оптимизирующими алгоритмами на базе машинного обучения, используя информацию о соответствии конструкций Представлений [28], получаемых при прямом преобразовании в процессе классической работы ГА; компонент поиска уязвимостей может совместно со статическими методами применять и динамический анализ в виде псевдо-выполнения на виртуальной машине кода программы для выявления ее опасных действий [29]; управление же процессом РИ может происходить не только линейно, но итеративно, корректируя деэволюцию на более ранних Представлениях исхода из качества деэволюции на более поздних.

Следуя схеме (см. Рис. 1) и формализации ее элементов можно утверждать о реализуемости архитектуры, по крайне мере, в виде программного прототипа.

Заключение

Работа относится к заключительной части большого авторского исследования по созданию методологии и технологии генетического РИ программ, главным предназначением которого является поиск в них разноуровневых уязвимостей.

Основным результатом текущего этапа исследования является создание архитектуры генетического РИ программы, состоящей из совокупности последовательно выполняемых однотипных компонентов деэволюции ее Представлений – т.е. их расширяемого стека, а также компонентов поиска уязвимостей и управления процессом.

Новизна результата состоит как в программном высокоуровневом описании принципиально новой системы РИ, так и в формализации элементов ее архитектуры.

Теоретическая значимость результата заключается в расширении шаблонов проектирования программных решений для РИ, а практическая значимость – в возможности непосредственной реализации и применения системы генетического РИ с последующим поиском уязвимостей, основанных на авторских моделях, методах, алгоритмах и архитектуре.

Продолжением исследования будет последовательная реализация и проведение экспериментов для следующих полноценных прототипов: компонента генетической деэволюции для получения программы на языке С из ее МК, компонента дальнейшей деэволюции в алгоритмы программы, компонента поиска уязвимостей (с внешними алгоритмами анализа Представлений), компонента управления процессом, а также всей такой двух-деэволюционной системы. Успешность проведенных экспериментов подтвердит обоснованность всех научно-технических результатов, полученных авторами ранее, работоспособность их практических прототипов, а также существенную значимость системы генетического РИ для области информационной безопасности ПО.

Литература

1. *Леонов Н.В., Буйневич М.В. Проблемные вопросы поиска уязвимостей в программном обеспечении промышленных ИТ-устройств // Автоматизация в промышленности. 2023. № 12. С. 59–63.*
2. *Леонов Н.В. Противодействие уязвимостям программного обеспечения. Часть 1. Онтологическая модель // Вопросы кибербезопасности. 2024. № 2 (60). С. 87–92. DOI: 10.21681/2311-3456-2024-2-87-92.*
3. *Леонов Н.В. Противодействие уязвимостям программного обеспечения. Часть 2. Аналитическая модель и концептуальные решения // Вопросы кибербезопасности. 2024. № 3 (61). С. 90–95. DOI: 10.21681/2311-3456-2024-3-90-95.*
4. *Абитов Р.А., Павленко Е.Ю. Выявление уязвимостей в программном обеспечении для процессоров ARM с использованием символьного выполнения // Проблемы информационной безопасности. Компьютерные системы. 2021. № 3. С. 9–15.*
5. *Kotenko, I., Izrailov, K., Buinevich, M., Saenko I., Shorey R. Modeling the Development of Energy Network Software, Taking into Account the Detection and Elimination of Vulnerabilities // Energies. 2023. Vol. 16. Iss. 13. PP. 5111. DOI: 10.3390/en16135111.*
6. *Николаенко В.С. Сравнительный анализ обратной разработки проприетарных программ в зависимости от алгоритмического языка программирования // Вестник студенческого научного общества ГОУ ВПО «Донецкий национальный университет». 2022. Т. 1. № 14. С. 189–192.*
7. *Израилов К.Е. Концепция генетической деэволюции представлений программы. Часть 1 // Вопросы кибербезопасности. 2024. № 1 (59). С. 61–66. DOI: 10.21681/2311-3456-2024-1-61-66.*
8. *Израилов К.Е. Концепция генетической деэволюции представлений программы. Часть 2 // Вопросы кибербезопасности. 2024. № 2 (60). С. 81–86. DOI: 10.21681/2311-3456-2024-2-81-86.*
9. *Израилов К.Е. Концепция генетической декомпиляции машинного кода телекоммуникационных устройств // Труды учебных заведений связи. 2021. Т. 7. № 4. С. 10–17. DOI:10.31854/1813-324X-2021-7-4-95-109.*
10. *Израилов К.Е. Применение генетических алгоритмов для декомпиляции машинного кода // Защита информации. Инсайд. 2020. № 3 (93). С. 24–30.*
11. *Аёшин И.Т. Реверс-инжиниринг программного продукта с использованием IDA Pro // Актуальные проблемы авиации и космонавтики. 2018. Т. 3. № 4 (14). С. 808–809.*
12. *Воробьев А.М., Боцвин А.С., Нагибин Д.В. Анализ функциональных возможностей Ghidra - фреймворка для реверс-инжиниринга // Методы и технические средства обеспечения безопасности информации. 2019. № 28. С. 86–88.*
13. *Бугеря А.Б., Ефимов В.Ю., Кулагин И.И., Падарян В.А., Соловьев М.А., Тихонов А.Ю. Программный комплекс для выявления недекларированных возможностей в условиях отсутствия исходного кода // Труды Института системного программирования РАН. 2019. Т. 31. № 6.С. 33–64. DOI: 10.15514/ISPRAS-2019-31(6)-3.*
14. *Израилов К.Е., Покусов В.В. Архитектура программной платформы преобразования машинного кода в высокоуровневое представление для экспертного поиска уязвимостей // Электронный сетевой политематический журнал «Научные труды КубГТУ». 2021. № 6. С. 93–111.*
15. *Голубева Т.В., Тайлаков В.А., Василенко К.Д., Якубова Е.А. Исследование архитектуры прототипов средств для автоматического поиска уязвимостей в устройствах IOT и M2M // Вестник Алматинского университета энергетики и связи. 2022. № 2 (57). С. 122–134. DOI: 10.51775/2790-0886_2022_57_2_122.*
16. *Urooj B., Shah M.A., Maple C., Abbasi M.K., Riasat S. Malware Detection: A Framework for Reverse Engineered Android Applications Through Machine Learning Algorithms // IEEE Access. 2022. Vol. 10. PP. 89031-89050 2022. DOI: 10.1109/ACCESS.2022.3149053.*
17. *Mauthe N., Kargén U., Shahmehri N. A Large-Scale Empirical Study of Android App Decompilation // In proceedings of IEEE International Conference on Software Analysis, Evolution and Reengineering (Honolulu, HI, USA, 09-12 March 2021). 2021. PP. 400–410. DOI: 10.1109/SANER50967.2021.00044.*
18. *Armengol-Estapé J., Woodruff J., Cummins C., O'Boyle M.F.P. SLaDe: A Portable Small Language Model Decompiler for Optimized Assembly // In proceedings of IEEE/ACM International Symposium on Code Generation and Optimization (Edinburgh, United Kingdom, 02–06 March 2024). 2024. PP. 67-80. DOI: 10.1109/CGO57630.2024.10444788.*
19. *Armengol-Estapé J., Woodruff J., Brauckmann A., Magalhães J.W. de S. , O'Boyle M.F.P. ExeBench: an ML-scale dataset of executable C functions // In Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming (New York, USA, 13 June 2022). 2022. PP. 50–59. DOI: 10.1145/3520312.3534867.*
20. *Aliefa M.H., Suyanto S. Variable-Length Chromosome for Optimizing the Structure of Recurrent Neural Network // In proceedings of International Conference on Data Science and Its Applications (Bandung, Indonesia, 05-06 August 2020). 2020. PP. 1–5. DOI: 10.1109/ICoDSA50139.2020.9213012.*
21. *Jiang W., Sha E.H.-M., Zhuge Q., Yang L., Dong H., Chen X. On the Design of Minimal-Cost Pipeline Systems Satisfying Hard/Soft Real-Time Constraints // IEEE Transactions on Emerging Topics in Computing. Vol. 9. No. 1. PP. 24–34. DOI: 10.1109/TETC.2017.2788800.*
22. *Леонов Н.В., Буйневич М.В. Машинное обучение vs поиск уязвимостей в программном обеспечении: анализ применимости и синтез концептуальной системы // Труды учебных заведений связи. 2023. Т. 9. № 6. С. 83–94. DOI: 10.31854/1813-324X-2023-9-6-83-94.*

23. *Кубрин Г. С., Зегжда Д. П. Поиск уязвимостей программного обеспечения с применением ансабля алгоритмов анализа графов // Методы и технические средства обеспечения безопасности информации. 2023. № 32. С. 49–50.*
24. *Гетьман А. И., Горюнов М. Н., Мацкевич А. Г., Рыболовлев Д. А. Сравнение системы обнаружения вторжений на основе машинного обучения с сигнатурными средствами защиты информации // Труды Института системного программирования РАН. 2022. Т. 34. № 5. С. 111–126. DOI: 10.15514/ISPRAS-2022-34(5)-7.*
25. *Пидченко И. А., Выборнова О. Н. Применение машинного обучения совместно с эвристическим анализом для задач антивирусного сканирования // Математические методы в технике и технологиях – ММТТ. 2020. Т. 5. С. 96–99.*
26. *Самарин Н. Н. Метод поиска ошибок в программном коде на базе фаззинга «в памяти» // Проблемы информационной безопасности. Компьютерные системы. 2024. № 2 (59). С. 130–137. DOI: 10.48612/jisp/39tp-t61k-29uv.*
27. *Иванов В. А., Конышев М. Ю., Шаповалов С. Л. Имитационная и аналитическая модели для исследования сигнатур и обнаружения модифицированных компьютерных вирусов и вредоносного программного обеспечения в вычислительных системах и сетях специального назначения // Информационная безопасность – актуальная проблема современности. Совершенствование образовательных технологий подготовки специалистов в области информационной безопасности. 2021. № 1 (14). С. 11–15.*
28. *Грибков Н. А., Овасапян Т. Д., Москвин Д. А. Анализ восстановленного программного кода с использованием абстрактных синтаксических деревьев // Проблемы информационной безопасности. Компьютерные системы. 2023. № 2 (54). С. 47–60. DOI: 10.48612/jisp/ruar-u6he-kmd4.*
29. *Довгалюк П. М., Климушенкова М. А., Фурсова Н. И., Степанов В. М., Васильев И. А., Иванов А. А., Иванов А. В., Бакулин М. Г., Егоров Д. И. Natch: определение поверхности атаки программ с помощью отслеживания помеченных данных и интроспекции виртуальных машин // Труды Института системного программирования РАН. 2022. Т. 34. № 5. С. 89–110. DOI: 10.15514/ISPRAS-2022-34(5)-6.*

ARCHITECTURE OF THE SYSTEM FOR GENETIC REENGINEERING OF THE PROGRAM WITH SEARCH SUPPORT MULTI-LEVEL VULNERABILITIES

Izrailov K. E.[2]

Keywords: *reverse engineering, genetic algorithm, vulnerability, machine code, architecture, formalization.*

The goal of the investigation: *increasing the efficiency of searching for vulnerabilities in machine code of programs by reverse engineering it based on genetic reengineering, for which the architecture of the corresponding software system is proposed.*

Research methods: *works survey, system analysis, structural synthesis of architecture, analytical modeling.*

Result: *a system architecture has been created, which is a set of sequentially executed some-template components for the de-evolution of the representations of the program being investigated (its machine, assembler and source code, algorithms, etc.); on each of these representations, a search for corresponding vulnerabilities is carried out.*

The scientific novelty *consists in the qualitatively new development of the reverse engineering direction through its intellectualization, for which a high-level description of the author's genetic reengineering system architecture is proposed, and the formalization of the its elements functioning is also carried out.*

References

1. *Leonov N. V., Bujnevich M. V. Problemnye voprosy poiska ujazvimostej v programmnom obespechenii promyshlennyh IT-ustrojstv // Avtomatizacija v promyshlennosti. 2023. № 12. S. 59–63.*
2. *Leonov N. V. Protivodejstvie ujazvimostjam programmnogo obespechenija. Chast' 1. Ontologicheskaja model' // Voprosy kiberbezopasnosti. 2024. № 2 (60). S. 87–92. DOI: 10.21681/2311-3456-2024-2-87-92.*
3. *Leonov N. V. Protivodejstvie ujazvimostjam programmnogo obespechenija. Chast' 2. Analiticheskaja model' i konceptual'nye reshenija // Voprosy kiberbezopasnosti. 2024. № 3 (61). S. 90–95. DOI: 10.21681/2311-3456-2024-3-90-95.*
4. *Abitov R. A., Pavlenko E. Ju. Vyjavlenie ujazvimostej v programmnom obespechenii dlja processorov ARM s ispol'zovaniem simvol'nogo vypolnenija // Problemy informacionnoj bezopasnosti. Komp'juternye sistemy. 2021. № 3. S. 9–15.*
5. *Kotenko, I., Izrailov, K., Buinevich, M., Saenko I., Shorey R. Modeling the Development of Energy Network Software, Taking into Account the Detection and Elimination of Vulnerabilities // Energies. 2023. Vol. 16. Iss. 13. PP. 5111. DOI: 10.3390/en16135111*
6. *Nikolaenko V. S. Sravnitel'nyj analiz obratnoj razrabotki proprietarnyh programm v zavisimosti ot algoritmicheskogo jazyka programmirovanija // Vestnik studencheskogo nauchnogo obshhestva GOU VPO «Doneckij nacional'nyj universitet». 2022. T. 1. № 14. S. 189–192.*
7. *Izrailov K. E. Koncepcija geneticheskoj dejevoljucii predstavlenij programmy. Chast' 1 // Voprosy kiberbezopasnosti. 2024. № 1 (59). S. 61–66. DOI: 10.21681/2311-3456-2024-1-61-66*

2 Konstantin E. Izrailov, Ph.D., Docent, Senior Researcher of Laboratory of Computer Security Problems of St. Petersburg Federal Research Center of the Russian Academy of Sciences, Saint-Petersburg. ORCHID: http://orcid.org/0000-0002-9412-5693. Scopus Author ID: 56123238800. E-mail:konstantin.izrailov@mail.ru

8. *Izrailov K. E. Koncepcija geneticheskoj dejevoljucii predstavlenij programmy. Chast' 2 // Voprosy kiberbezopasnosti. 2024. № 2 (60). S. 81–86. DOI: 10.21681/2311-3456-2024-2-81-86*
9. *Izrailov K. E. Koncepcija geneticheskoj dekompiljacii mashinnogo koda telekommunikacionnyh ustrojstv // Trudy uchebnyh zavedenij svjazi. 2021. T. 7. № 4. S. 10–17. DOI:10.31854/1813-324X-2021-7-4-95-109.*
10. *Izrailov K. E. Primenenie geneticheskih algoritmov dlja dekompiljacii mashinnogo koda // Zashhita informacii. Insajd. 2020. № 3 (93). S. 24–30.*
11. *Ajoshin I. T. Revers-inzhiniring programmnogo produkta s ispol'zovaniem IDA Pro // Aktual'nye problemy aviacii i kosmonavtiki. 2018. T. 3. № 4 (14). S. 808–809.*
12. *Vorob'ev A. M., Bocvin A. S., Nagibin D. V. Analiz funkcional'nyh vozmozhnostej Ghidra – frejmvorka dlja revers-inzhiniringa // Metody i tehnicheskie sredstva obespechenija bezopasnosti informacii. 2019. № 28. S. 86–88.*
13. *Bugerja A. B., Efimov V. Ju., Kulagin I. I., Padarjan V. A., Solov'ev M. A., Tihonov A. Ju. Programmnyj kompleks dlja vyjavlenija nedeklarirovannyh vozmozhnostej v uslovijah otsutstvija ishodnogo koda // Trudy Instituta sistemnogo programmirovanija RAN. 2019. T. 31. № 6.S. 33–64. DOI: 10.15514/ISPRAS-2019-31(6)-3.*
14. *Izrailov K. E., Pokusov V. V. Arhitektura programmnoj platformy preobrazovanija mashinnogo koda v vysokourovnevoe predstavlenie dlja jekspertnogo poiska ujazvimostej // Jelektronnyj setevoj politematicheskij zhurnal «Nauchnye trudy KubGTU». 2021. № 6. S. 93–111.*
15. *Golubeva T. V., Tajlakov V. A., Vasilenko K. D., Jakubova E. A. Issledovanie arhitektury prototipov sredstv dlja avtomaticheskogo poiska ujazvimostej v ustrojstvah IOT i M2M // Vestnik Almatinskogo universiteta jenergetiki i svjazi. 2022. № 2 (57). S. 122–134. DOI: 10.51775/2790-0886_2022_57_2_122.*
16. *Urooj B., Shah M. A., Maple C., Abbasi M. K., Riasat S. Malware Detection: A Framework for Reverse Engineered Android Applications Through Machine Learning Algorithms // IEEE Access. 2022. Vol. 10. PP. 89031–89050 2022. DOI: 10.1109/ACCESS.2022.3149053.*
17. *Mauthe N., Kargén U., Shahmehri N. A Large-Scale Empirical Study of Android App Decompilation // In proceedings of IEEE International Conference on Software Analysis, Evolution and Reengineering (Honolulu, HI, USA, 09-12 March 2021). 2021. PP. 400–410. DOI: 10.1109/SANER50967.2021.00044.*
18. *Armengol-Estapé J., Woodruff J., Cummins C., O'Boyle M. F. P. SLaDe: A Portable Small Language Model Decompiler for Optimized Assembly // In proceedings of IEEE/ACM International Symposium on Code Generation and Optimization (Edinburgh, United Kingdom, 02–06 March 2024). 2024. PP. 67–80. DOI: 10.1109/CGO57630.2024.10444788.*
19. *Armengol-Estapé J., Woodruff J., Brauckmann A., Magalhães J. W. de S. , O'Boyle M. F. P. ExeBench: an ML-scale dataset of executable C functions // In Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming (New York, USA, 13 June 2022). 2022. PP. 50–59. DOI: 10.1145/3520312.3534867.*
20. *Aliefa M. H., Suyanto S. Variable-Length Chromosome for Optimizing the Structure of Recurrent Neural Network // In proceedings of International Conference on Data Science and Its Applications (Bandung, Indonesia, 05-06 August 2020). 2020. PP. 1–5. DOI: 10.1109/ICoDSA50139.2020.9213012.*
21. *Jiang W., Sha E. H. -M., Zhuge Q., Yang L., Dong H., Chen X. On the Design of Minimal-Cost Pipeline Systems Satisfying Hard/Soft Real-Time Constraints // IEEE Transactions on Emerging Topics in Computing. Vol. 9. No. 1. PP. 24–34. DOI: 10.1109/TETC.2017.2788800.*
22. *Leonov N. V., Bujnevich M. V. Mashinnoe obuchenie vs poisk ujazvimostej v programmnom obespechenii: analiz primenimosti i sintez konceptual'noj sistemy // Trudy uchebnyh zavedenij svjazi. 2023. T. 9. № 6. S. 83–94. DOI: 10.31854/1813-324X-2023-9-6-83-94.*
23. *Kubrin G. S., Zegzhda D. P. Poisk ujazvimostej programmnogo obespechenija s primeneniem ansablja algoritmov analiza grafov // Metody i tehnicheskie sredstva obespechenija bezopasnosti informacii. 2023. № 32. S. 49–50.*
24. *Get'man A. I., Gorjunov M. N., Mackevich A. G., Rybolovlev D. A. Sravnenie sistemy obnaruzhenija vtorzhenij na osnove mashinnogo obuchenija s signaturnymi sredstvami zashhity informacii // Trudy Instituta sistemnogo programmirovanija RAN. 2022. T. 34. № 5. S. 111–126. DOI: 10.15514/ISPRAS-2022-34(5)-7.*
25. *Pidchenko I. A., Vybornova O. N. Primenenie mashinnogo obuchenija sovmestno s jevristicheskim analizom dlja zadach antivirusnogo skanirovanija // Matematicheskie metody v tehnike i tehnologijah – MMTT. 2020. T. 5. S. 96–99.*
26. *Samarin N. N. Metod poiska oshibok v programmnom kode na baze fazzinga «v pamjati» // Problemy informacionnoj bezopasnosti. Komp'juternye sistemy. 2024. № 2 (59). S. 130–137. DOI: 10.48612/jisp/39tp-t61k-29uv.*
27. *Ivanov V. A., Konyshev M. Ju., Shapovalov S. L. Imitacionnaja i analiticheskaja modeli dlja issledovanija signatur i obnaruzhenija modificirovannyh komp'juternyh virusov i vredonosnogo programmnogo obespechenija v vychislitel'nyh sistemah i setjah special'nogo naznachenija // Informacionnaja bezopasnost' – aktual'naja problema sovremennosti. Sovershenstvovanie obrazovatel'nyh tehnologij podgotovki specialistov v oblasti informacionnoj bezopasnosti. 2021. № 1 (14). S. 11–15.*
28. *Gribkov N. A., Ovasapjan T. D., Moskvin D. A. Analiz vosstanovlennogo programmnogo koda s ispol'zovaniem abstraktnyh sintaksicheskih derev'ev // Problemy informacionnoj bezopasnosti. Komp'juternye sistemy. 2023. № 2 (54). S. 47–60. DOI: 10.48612/jisp/ruar-u6he-kmd4.*
29. *Dovgaljuk P. M., Klimushenkova M. A., Fursova N. I., Stepanov V. M., Vasil'ev I. A., Ivanov A. A., Ivanov A. V., Bakulin M. G., Egorov D. I. Natch: opredelenie poverhnosti ataki programm s pomoshh'ju otslezhivanija pomechennyh dannyh i introspekcii virtual'nyh mashin // Trudy Instituta sistemnogo programmirovanija RAN. 2022. T. 34. № 5. S. 89–110. DOI: 10.15514/ISPRAS-2022-34(5)-6.*