Научная статья
УДК 004.42
DOI:10.31854/1813-324X-2024-10-1-86-96

Методология проведения реверс-инжиниринга машинного кода. Часть 3. Динамическое исследование и документирование

Константин Евгеньевич Израилов, konstantin.izrailov@mail.ru

Санкт-Петербургский Федеральный исследовательский центр Российской академии наук,
Санкт-Петербург, 199178, Российская Федерация

Аннотация: *Изложены результаты создания единой методологии проведения реверс-инжиниринга машинного кода устройств. Данная, третья заключительная часть цикла статей посвящена динамическому исследованию кода с целью восстановления метаинформации о нем и дополнительному поиску уязвимостей, а также итоговому документированию результатов. Проводится обзор научных публикаций на тему существующих методов и средств динамического анализа машинного кода. Дается детальное описание и формализация шагов этапа, а также примеры их применения на практике. Полная схема предлагаемой методологии приводится в графическом виде с указанием получаемых основных и промежуточных результатов. Все шаги сведены в сводную таблицу, содержащей также некоторые их характеристик. Обсуждаются недостатки методологии и пути их устранения.*

Ключевые слова: *реверс-инжиниринг, обратная разработка, программная инженерия, динамический анализ, документирование, информационная безопасность, уязвимости, методология, схема*

Ссылка для цитирования: Израилов К.Е. Методология проведения реверс-инжиниринга машинного кода. Часть 3. Динамическое исследование и документирование // Труды учебных заведений связи. 2024. Т. 10. № 1. С. 86–96. DOI:10.31854/1813-324X-2024-10-1-86-96. EDN:NZUJAZ

Methodology for Reverse Engineering of Machine Code. Part 3. Dynamic Investigation and Documentation

Konstantin Izrailov, konstantin.izrailov@mail.ru

Saint-Petersburg Federal Research Center of the Russian Academy of Sciences,
St. Petersburg, 199178, Russian Federation

Abstract: *The results of creating a unified methodology for reverse engineering the machine code of devices are presented. This, the third and final part of the series of articles, is devoted to the dynamic examination of code in order to restore metainformation about it and additionally search for vulnerabilities, as well as the final documentation of the results. A review of scientific publications on the topic of existing methods and tools for dynamic analysis of machine code is carried out. A detailed description and formalization of the steps of the stage is given, as well as examples of their application in practice. A complete diagram of the proposed methodology is presented in graphical form, indicating the main and intermediate results obtained. All steps are summarized in a summary table, which also contains some of their characteristics. The shortcomings of the methodology and ways to eliminate them are discussed.*

Keywords: *reverse engineering, backwards engineering, software engineering, dynamic analysis, documentation, information security, vulnerabilities, methodology, diagram*

For citation: Izrailov K. Methodology for reverse engineering of machine code. Part 3. Dynamic investigation and documentation. *Proceedings of Telecommun. Univ.* 2024;10(1):86–96. DOI:10.31854/1813-324X-2024-10-1-86-96. EDN:NZUJAZ

Введение

Реверс-инжиниринг (далее – РИ) [1] машинного кода (далее – МК) является актуальной задачей, обусловленной необходимостью восстановления принципов и деталей работы программного обеспечения при отсутствии его исходного кода (далее – ИК). Однако, несмотря на востребованность в РИ, на текущий момент наблюдаются лишь частные решения по его приложению для реальных задач, что может обосновываться научной и практической сложностью данного процесса. Существует некоторое количество книг, монографий, статей и программных средств по анализу МК на предмет понимания логики его работы и частичного восстановления псевдокода (т. е. кода на языке программирования, близкого, но не обязательно совпадающего с ИК, из которого был получен МК). Восстановление алгоритмов псевдо или ИК считается типовой задачей, но, как правило, решаемой лишь для отдельных подпрограмм, а не для всей программы или же их совокупности. Архитектура программного обеспечения в принципе не имеет достаточно четкого определения и редко соотносится именно с МК, который по сравнению с ней является существенно более низкоуровневым представлением. Также не всегда рассматриваются взаимодействия между программами, не говоря уже о возможности применения интеллектуальных методов в интересах РИ. Основное же «упущение», по мнению автора, заключается в полном отсутствии какой-либо общей методологии РИ, обоснованной как с теоретической, так и с практической стороны.

В первых двух частях цикла статей давалось общее представление о предлагаемой автором методологии РИ (далее – Методология), вводилась онтологическая модель предметной области и описывались шаги 2-х следующих этапов: подготовка МК целой программной системы к РИ [2] и его статическое исследование (т. е. без выполнения программ) [3]. Таким образом, в данной завершающей части цикла статей будут описаны оставшиеся 2 этапа Методологии – динамическое исследование МК (т.е. с выполнением программ) с целью уточнения результатов статического исследования и их документирования; также будут даны ее итоговая схема и сводная таблица с систематизацией всех шагов.

Обзор работ

Проведем обзор работ, посвященных вопросу РИ в части существующих методов и средств для динамического анализа МК; как и в обзоре предыдущей статьи цикла, целью будет выделение шагов такого анализа, их принципов и особенностей. Все это позволит построить этапы Методологии, относящиеся к динамическому исследованию МК. Вопросы документирования результатов РИ будут оставлены без внимания, поскольку хотя и необходимы для логичного завершения процесса, однако выходят за рамки предметной области.

Работа [4] посвящена поиску ошибок в МК, основанных на переполнении буфера и последующем выходе за его границы. Одной из причин этого указываются ошибки в работе цикла с доступом к элементу массива. Описывается процесс анализа МК для выявления таких потенциально опасных циклов по работе со строками, основанный на следующих критериях: индуктивность переменной, связанной с ячейкой памяти; обращение к памяти на каждой итерации; тождественность размера символа строки, ячейки памяти и инкрементации переменной цикла. Также определяется тип цикла по следующим критериям: а) для копирования строк – наличие двух буферов с доступом на каждой итерации (к первому – на чтение, ко второму – на запись); равенство элементов буферов с одинаковым индексом после каждой итерации; б) для вычисления длины строк – наличие одного буфера с доступом на чтение на каждой итерации; значение индуктивной переменной после окончания цикла совпадает с длиной строки в буфере. Таким образом возможно прогнозировать ошибки типа переполнения буфера. Очевидно, что для применения такого процесса на практике необходимо проведение динамического анализа или его альтернативы, близкой и к статическому анализу – символьному выполнению [5].

Работа [6] описывает разработку «расширяемого дизассемблера», суть которого заключается в формализованном платформенно-независимом описании инструкций МК программы на основании спецификации архитектуры процессора. Делается краткий обзор такого рода декодеров и их моделей представления инструкций, а именно следующих: libdisasm, IDA Pro, OllyDbg, LibVEX, Zynamics BinAudit, TSL и др. Описываются детали и особенности разработки дизассемблера, такие, как собственный язык описания архитектуры, работающий прототип для x86 и S/390 и т. п. Описанный подход может быть применен для создания унифицированных отладчиков МК.

В исследовании [7] приводятся различные средства проведения динамического анализа МК при таких операциях, как перехват вызовов, модификация

бинарного кода и т. п. Описываются следующие механизмы анализа: мониторинг вызываемых функций, анализ их параметров, отслеживание информационных потоков и последовательностей инструкций МК. Приводятся механизмы анти-отладки, состоящие из самомодификации кода, его упаковки, определения средств отладки, применения логических бомб и мониторинга производительности работы (которая, очевидно, при отладке снижается). Основная идея авторов в части глубокого динамического анализа заключается в обеспечении полного контроля за работой исследуемой программы в тестовой среде. Для этого, в частности, предлагается контролировать доступ к страницам памяти, вызовы системных функций и динамически сгенерированный код, а также перехватывать отдельные функции, их параметры и возвращаемые значения. Указывается возможность применения машинного обучения для классификации вредоносных программ по указанным ранее событиям, возникающим при выполнении МК.

В работе [8] исследуется возможность автоматизации деления МК на логические блоки, одной из целей чего является поиск уязвимостей. В интересах последнего предлагается применять (в том числе) такие виды динамического анализа, как Fuzzing – т. е. тестирование программы в некоторой среде путем подачи на ее вход большого набора данных с целью вызвать некорректное поведение исследуемого образца.

Диссертационное исследование [9] посвящено восстановлению алгоритмов программы путем ее динамического анализа; при этом, для представления алгоритмов выбирается машинно-независимая форма. Для решения указанной задачи предлагается применять трассировку выполнения МК. Указаны два проблемных вопроса такого рода динамических исследований – влияние анализа на поведение программы и ограниченность его охвата. В результате исследования автором получен метод для совмещения трасс выполнения программы с учетом возможной модификации ее МК, метод представления алгоритмов в машинно-независимом виде, соответствующая нотация представления инструкций МК, а также разработанный инструментарий. Расширение набора трасс, которые зависят от входных параметров программы, осуществляется за счет осведомленности эксперта-аналитика об объекте исследования.

Исследование [10] представляет инструмент (а точнее, целую инфраструктуру) VMAD (*аббр. от англ.* Virtual Machine for Advanced Dynamic analysis, *перев. на русс.* Виртуальная Машина для Расширенного Динамического анализа) для исследования МК для x86_64 архитектуры путем его выполнения. Продукт использует динамическую трансляцию, что существенно повышает производительность выполнения. Особое внимание в VMAD уделяется контролю доступа к критическим частям памяти, в том числе, при выполнении циклов. Продукт поддерживает загрузку отдельных модулей для разностороннего динамического анализа МК.

Согласно сделанным обзорам, существует некоторое количество публикаций, посвященных непосредственно методам и средствам динамического исследования МК и восстановлению из них информации, подходящей для экспертного анализа; при этом, в основном рассматриваются задачи автоматического анализа МК, декодирования его инструкций и получения из него алгоритмов (хотя и в платформенно-независимом представлении). Вопросы же глубокого экспертного анализа с целью получения такой высокоуровневой метаинформации о МК, как архитектура или концептуальная модель, оставлены практически без внимания. С этой позиции, описываемый далее Этап 3 Методологии, безусловно, будет иметь научную ценность, поскольку практически каждый его шаг может стать отдельным направлением научного исследования.

Схема Методологии

Онтологическая модель

В первой статье цикла была введена онтологическая модель предметной области, лежащая в основе Методологии [2]. Для более точного и корректного описания шагов текущих (3-го и 4-го) этапов Методологии повторно приведем ее основные сущности (которые далее будут писаться с большой буквы):

1) Машинный код – представление логики работы программ в виде инструкций процессора;

2) Программа – бинарный файл с МК и дополнительной информацией;

3) Программная система – совокупность взаимодействующих Программ;

4) Образ – представление Программной системы в виде монолитного файла;

5) Устройство хранения – устройство для получения и выполнения загруженного Образа;

6) Метаинформация – сведения о функционале МК (псевдокод, алгоритмы, архитектура, концептуальная модель и т. п.);

7) Реверс-инжиниринг – процесс восстановления Метаинформации о МК;

8) Уязвимость – отличие реализации МК от задуманной или заявленной, приводящее к возникновению информационных угроз.

Графическое представление

Предлагаемая Методология состоит из 4 этапов, шаги 1-го и 2-го из которых были детально расписаны в предыдущих статьях цикла. Таким образом, в данной финальной статье цикла будет завершено представление Методологии путем описания ее шагов для 3 и 4 Этапов. Как и раньше, идентификаторы шагов имеют следующую запись: X. Y, где X –

номер этапа, Y – порядковый номер выполнения в рамках этапа. Полная схема Методологии в графическом виде представлена на рисунке 1. На схеме используются следующие обозначения: прямоугольник с белым фоном – результат применения шага; круг со светло-желтым фоном – идентификатор шага в текущей статье цикла; круг с темно-желтым фоном – идентификатор шага в предыдущей статье цикла; сплошная стрелка – основное действие шага над результатом предыдущего; пунктирная стрелка – дополнительное использование шагом результатов другого шага; прямоугольник с закругленными краями и надписью – группа логически связанных элементов (синий фон – для текущего этапа, серый фон – для предыдущего и последующих этапов); пунктирный прямоугольник с надписью – область шагов определенного этапа. Так, область «Этап 1. Подготовка объекта исследования» соответствует шагам (кроме 2.7 и 2.8), описанным в 1-й статье цикла, область «Этап 2, 3. Анализ объекта исследования» – шагам, описанным в предыдущей (для статического исследования) и текущей (для динамического исследования) статьях, область «Этап 4. Документирование» – шагам также для текущей статьи цикла.

Согласно схеме (см. рисунок 1) в данной части статьи будут расписаны Шаги с 3.1 по 4.4; они полностью определяют Этапы 3 и 4, поскольку предназначены для анализа МК динамическими способами, а также документирования результатов.

Этап 3. Динамическое исследование

Этап предназначен для выполнения дополнительного (или уточняющего) анализа МК путем его выполнения (как целиком, так и отдельных частей) в аппаратной или в виртуальной среде – что отражает динамичность исследования. Шаги являются крайне специфичными для конкретных Программ, в основном имеют ручное выполнение с высоким уровнем «творчества» эксперта и, поэтому, описываются без деталей. По возможности, будем давать примеры применения шагов, используя рассмотренный в предыдущих статьях цикла DXE-драйвер UEFI-фазы BDS (*аббр. от англ.* Boot Device Selection, *перев. на русс.* Выбор Загрузочного Устройства) [11] с именем BdsDxe.efi из материнской платы PRIME Z490-A производства ASUSTeK COMPUTER INC.

Шаг 3.1. Анализ выбранной подпрограммы для уточнения ее псевдокода

Шаг заключается в отслеживании и понимании влияния области МК на процессорные регистры и память (т. е. переменные), что позволит уточнить детали псевдокода [12]. Как результат, можно восстановить отдельные математические выражения, ручной статический анализ которых на Шаге 2.4 оказался затруднительным. Формальная запись шага ($Step_{3.1}$) имеет следующий вид:

$$PseudoCode'' = \\ = Step_{3.1}(Subroutine, GlobalDatas, Debugger),$$

где $PseudoCode''$ – псевдокод, уточненный в процессе динамического исследования на шаге; $Debugger$ – отладчик, используемый для анализа МК.

Примером шага является то, что если в процессе отладки некоторая переменная в выражении поочередно принимает значение 0 и 1, то потенциально переменная имеет bool-тип (т. е. бинарный), а к ней применяется унарный оператор отрицания: «*x = !x*» (в языке C++).

Шаг 3.2. Анализ выбранных подпрограмм для уточнения их алгоритмов

Шаг заключается в отслеживании и понимании действий и переходов в МК подпрограммы, что позволит уточнить детали ее алгоритма. Как результат, можно восстановить отдельные логические конструкции, ручной статический анализ которых на Шаге 2.5 оказался затруднительным. Формальная запись шага ($Step_{3.2}$) имеет следующий вид:

$$Algorithm'' = \\ = Step_{3.2}(Subroutine, GlobalDatas, Debugger),$$

где $Algorithm''$ – алгоритм, уточненный в процессе динамического исследования на шаге.

Примером шага является то, что если в процессе отладки при многократном повторении набора инструкций в МК некоторая область памяти будет увеличиваться на единицу, то с большой вероятностью псевдокод соответствует циклу с инкрементацией глобальной переменной.

Шаг 3.3. Анализ выбранных подпрограмм для уточнения схемы их внутрипрограммного взаимодействия

Шаг заключается в отслеживании и понимании переходов между подпрограммами, что позволит уточнить детали их информационного взаимодействия в рамках одной Программы. Как результат, можно восстановить отдельные вызовы функций и процедур (в том числе, выполняемые не напрямую, а через указатель на память), ручной статический анализ которых на Шаге 2.6 оказался затруднительным. Формальная запись шага ($Step_{3.3}$) имеет следующий вид:

$$(Scheme^{IntoProgramInteraction})'' = \\ = Step_{3.3}(Subroutines, Debugger),$$

где $(Scheme^{IntoProgramInteraction})''$ – схема внутрипрограммного взаимодействия, уточненная в процессе динамического исследования на шаге.

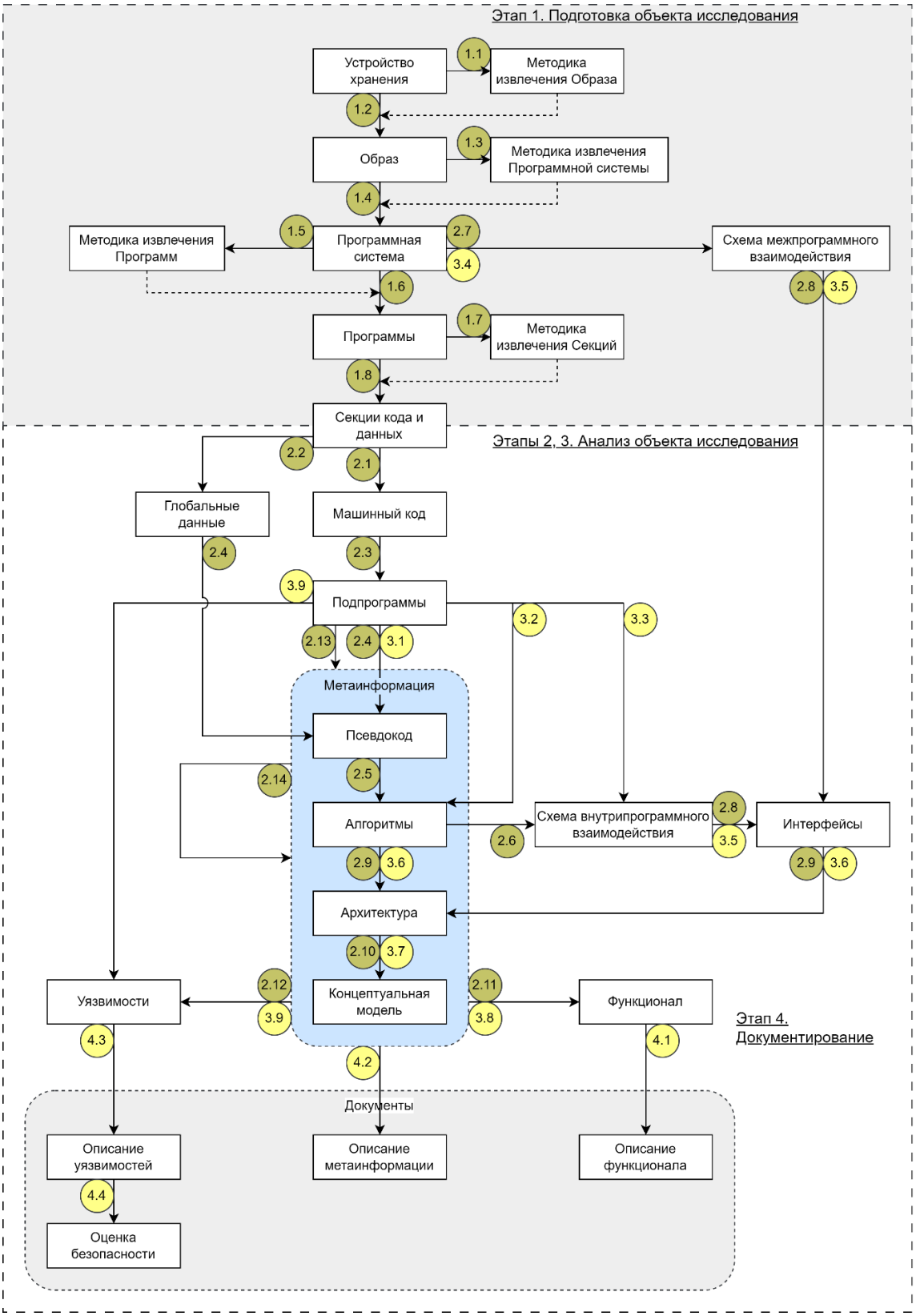

Рис. 1. Схема Методологии реверс-инжиниринга (Этапы 3 и 4)

Fig. 1. Reverse Engineering Methodology Diagram (Stages 3 and 4)

Примером шага является то, что если в одной подпрограмме поставлена точка останова, которая сработала при выполнении другой подпрограммы, то по стеку вызовов можно определить, как именно осуществился вызов первой подпрограммы из второй.

Шаг 3.4. Анализ подпрограмм для уточнения схемы их межпрограммного взаимодействия

Шаг заключается в отслеживании передачи информационных объектов между подпрограммами, расположенными в отдельных Программах целой Программной системы, что позволит уточнить детали их взаимодействия. Как результат, можно восстановить те типы взаимодействий, ручной статический анализ которых на Шаге 2.7 оказался затруднительным.

Формальная запись шага ($Step_{3.4}$) имеет следующий вид:

$$(Scheme^{InterProgramInteraction})'' = \\ = Step_{3.4}(Programms, Debugger),$$

где $(Scheme^{InterProgramInteraction})''$ – схема межпрограммного взаимодействия, уточненная в процессе динамического исследования на шаге.

Примером шага является то, что если в одной Программе поставлена точка останова, которая сработала при выполнении другой Программы, то по стеку вызовов (например, в Программном эмуляторе со встроенным отладчиком кода) можно определить, как именно осуществилась передача информации от первой Программы ко второй.

Шаг 3.5. Анализ схемы внутрипрограммного и межпрограммного взаимодействия для уточнения интерфейсов

Шаг заключается в отслеживании и понимании передачи информационных объектов между подпрограммами (как в рамках одной Программы, так и при наличии их системы), что позволит уточнить детали интерфейсов [13]. Как результат, можно восстановить новые межподпрограммные связи, ручной статический анализ которых на Шаге 2.8 оказался затруднительным. Формальная запись шага ($Step_{3.5}$) имеет следующий вид:

$$Interfaces'' = \\ = Step_{3.5}\begin{pmatrix}(Scheme^{IntoProgramInteraction})'', \\ (Scheme^{InterProgramInteraction})'', \\ Debugger\end{pmatrix},$$

где $Interfaces''$ – интерфейсы Программ, уточненные в процессе динамического исследования на шаге.

Примером шага является то, что если в точке входа в одном DXE-драйвере поставить точку останова, которая сработала при работе другого DXE-драйвера, а в стеке вызовов присутствует функция *StartImage()*, отвечающая за запуск загруженной Программы, то с большой вероятностью первый драйвер был вызван вторым напрямую (т. е. без использования стандартного в UEFI механизма запуска – т. н. DXE Dispatcher, *перев. на русс.* Диспетчер DXE).

Шаг 3.6. Анализ алгоритмов подпрограмм и интерфейсов для уточнения их общей архитектуры

Шаг заключается в отслеживании и понимании логики работы подпрограмм, а также их взаимном функционировании посредством интерфейсов (в независимости от расположения в Программах), что позволит уточнить детали архитектур – как каждой Программы, так и их Программной системы. Как результат, можно выделить новые модули архитектуры (по совместной работе подпрограмм над единой задачей), ручной статический анализ чего на Шаге 2.9 оказался затруднительным.

Формальная запись шага ($Step_{3.6}$) имеет следующий вид:

$$\overline{Architecture}'' = \\ = Step_{3.6}(Subroutines, Interfaces'', Debugger),$$

где $\overline{Architecture}''$ – архитектура Программной системы, уточненная в процессе динамического исследования на шаге.

В качестве примера можно произвести логирование вызываемых подпрограмм во внешний файл с отслеживанием фаз работы Программы (которые, например, могут отображаться в консоли). Таким образом, часть подпрограмм из лога по времени может быть сопоставлена с каждой фазой Программы, что позволит их объединить в логи.

Шаг 3.7. Анализ архитектуры для уточнения ее концептуальной модели

Шаг является вспомогательным, поскольку на основании архитектуры, уточненной на Шаге 3.6, необходимо уточнить концептуальную модель Программной системы. Формальная запись шага ($Step_{3.7}$) имеет следующий вид:

$$(Model^{Conceptual})'' = Step_{3.7}(\overline{Architecture}''),$$

где $(Model^{Conceptual})''$ – концептуальная модель Программной системы, уточненная в процессе динамического исследования на шаге.

Например, если в результате выполнения Шага 3.6 в архитектуру арифметического калькулятора (из предположения в Шаге 2.9) был добавлен модуль хранения результатов предыдущих вычислений, то в концептуальной модели должна появиться сущность – «История вычислений».

Шаг 3.8. Анализ восстановленной Метаинформации для уточнения функционала МК

Шаг является вспомогательным, поскольку, используя Метаинформацию, уточненную на этапе

динамического исследования, необходимо обновить описание общего функционала МК, созданного на этапе статического исследования.

Формальная запись шага ($Step_{3.8}$) имеет следующий вид (1), где $Functionality''$ – функционал, уточненный в процессе динамического исследования на шаге.

В качестве примера можно привести совокупность примеров из Шагов 3.1, 3.2, 3.6 и 3.7.

Шаг 3.9. Уточнение Уязвимостей по восстановленной Метаинформации

Шаг аналогичен Шагу 3.8 и заключается в повторном анализе Метаинформации с учетом уточнений, сделанных на фазе динамического анализа. Формальная запись шага ($Step_{3.9}$) имеет следующий вид (2), где $Vulnerabilities''$ – Уязвимости Программы, уточненные в процессе динамического исследования на шаге.

$$Functionality'' = Step_{3.8}\begin{pmatrix} PseudoCode'', Algorithm'', \overline{Architecture''}, \\ (Model^{Conceptual})'' \end{pmatrix}, \quad (1)$$

$$Vulnerabilities'' = Step_{3.9}\begin{pmatrix} PseudoCode'', Algorithm'', \overline{Architecture''}, \\ (Model^{Conceptual})'' \end{pmatrix}. \quad (2)$$

Этап 4. Документирование

Этап предназначен для оформления результатов проведенного исследования в документированном виде, готовом для возможного дальнейшего анализа. Примеры шагов являются интуитивно понятными и будут опущены.

Шаг 4.1. Описание функционала МК

Шаг заключается в документировании функционала, сформированного по восстановленной из МК Метаинформации в процессе статического и динамического исследований. Формальная запись шага ($Step_{4.1}$) имеет следующий вид:

$$Document^{Functionality} = Step_{4.1}(Metainfo),$$

где $Document^{Functionality}$ – итоговый документ с описанием функционала.

Шаг 4.2. Описание восстановленной Метаинформации

Шаг заключается в документировании всей Метаинформации, сформированной в процессе статического и динамического исследований. Формальная запись шага ($Step_{4.2}$) имеет следующий вид:

$$Document^{Metadatas} = Step_{4.2}(Metainfo),$$

где $Document^{Metadatas}$ – итоговый документ с описанием восстановленной Метаинформации.

Шаг 4.3. Описание найденных Уязвимостей

Шаг заключается в документировании Уязвимостей, сформированных в процессе статического и динамического исследований.

Формальная запись шага ($Step_{4.3}$) имеет следующий вид:

$$Document^{Vulnerabilities} = Step_{4.3}(Vulnerabilities),$$

где $Document^{Vulnerabilities}$ – итоговый документ с описанием найденных Уязвимостей.

Шаг 4.4. Оценка безопасности использования объекта исследования

Шаг заключается в создании документа с оценкой безопасности объекта РИ, которая сформирована путем анализа документа с найденными Уязвимостями [14]. Так, можно ввести средневзвешенный показатель, учитывающий опасность каждой категории найденных Уязвимостей; или же использовать экспертные суждения. Формальная запись шага ($Step_{4.4}$) имеет следующий вид:

$$Document^{SafetyAssessment} = \\ = Step_{4.4}(Document^{Vulnerabilities}),$$

где $Document^{SafetyAssessment}$ – итоговый документ с оценкой безопасности объекта РИ на основании найденных Уязвимостей.

Таблица шагов

Шаги Методологии могут быть сведены в табличный вид с указанием их этапа (в отдельных строках), идентификатора, названия, входных и выходных данных, сложности выполнения, а также формы выполнения (наиболее распространенной).

Такая систематизация приведена в таблице 1; для формы выполнения используется сокращение "Автомат." вместо "Автоматическая".

Анализ таблицы 1 позволяет сделать следующие выводы касательно всех 35 шагов Методологии.

Во-первых, все шаги имеют входные данные, полученные другими шагами (кроме первоначальных, применяемых для исходного объекта – Шага 1.1 и 1.2), а также все выходные данные шагов используются другими (кроме заключительных, формируемых итоговую документацию – Шаги 4.1, 4.2 и 4.4). Таким образом, можно говорить о корректности предложенной Методологии с позиции причинно-следственных связей.

ТАБЛИЦА 1. Этапы и шаги Методологии

TABLE 1. The Methodology Stages and Steps

№	Шаг	Данные		Выполнение	
		входные	выходные	сложность	форма
1	*Подготовительные мероприятия*				
1.1	Сбор общей информации об устройстве хранения для разработки методики извлечения образа	Устройство хранения	Методика извлечения Образа	Низкая	Ручная
1.2	Применение к устройству хранения методики извлечения образа	Устройство хранения	Образ	Средняя	Автомат.
1.3	Сбор общей информации об образе для разработки методики извлечения программной системы	Образ	Методика извлечения Программной системы	Низкая	Ручная
1.4	Применения к образу методики извлечения программной системы	Образ	Программная система	Средняя	Ручная, Автомат.
1.5	Сбор общей информации о программной системе для разработки методики извлечения программ	Программная система	Методика извлечения Программ	Низкая	Ручная
1.6	Применение к программной системе методики извлечения программ	Программная система	Программы	Низкая	Ручная, Автомат.
1.7	Сбор общей информации о программах для разработки методики выделения секций	Программы	Методика извлечения секций	Низкая	Ручная
1.8	Применение к программам методики выделения секций	Программы	Секции кода и данных	Низкая	Автомат.
2	*Статическое исследование*				
2.1	Анализ МК для выделения точек входа, секций кода и данных	Секции кода и данных	Машинный код	Низкая	Автомат.
2.2	Анализ секции данных для выделения глобальных данных	Секции кода и данных	Глобальные данные	Средняя	Ручная, Автомат.
2.3	Анализ МК для выделения подпрограмм	Машинный код	Подпрограммы	Средняя	Ручная, Автомат.
2.4	Анализ МК выбранной подпрограммы для восстановления ее псевдокода	Глобальные данные	Псевдокод	Высокая	Ручная, Автомат.
2.5	Анализ псевдокода выбранной подпрограммы для восстановления ее алгоритма	Псевдокод	Алгоритмы	Средняя	Ручная, Автомат.
2.6	Анализ алгоритмов подпрограмм для создания схемы внутрипрограммного взаимодействия	Алгоритмы	Схема внутрипрограммного взаимодействия	Средняя	Ручная, Автомат.
2.7	Анализ Программной системы для создания схемы межпрограммного взаимодействия	Программная система	Схема межпрограммного взаимодействия	Средняя	Ручная, Автомат.
2.8	Анализ схемы внутрипрограммного и межпрограммного взаимодействия для выявления интерфейсов	Схема внутрипрограммного взаимодействия, Схема межпрограммного взаимодействия	Интерфейсы	Средняя	Ручная
2.9	Анализ алгоритмов подпрограмм и интерфейсов для восстановления их общей архитектуры	Интерфейсы	Архитектура	Средняя	Ручная
2.10	Анализ архитектуры для восстановления ее концептуальной модели	Архитектура	Концептуальная модель	Средняя	Ручная
2.11	Анализ восстановленной Метаинформации для описания функционала МК	Концептуальная модель	Функционал	Средняя	Ручная
2.12	Поиск Уязвимостей по восстановленной Метаинформации	Метаинформация (Псевдокод, Алгоритмы, Архитектура, Концептуальная модель)	Уязвимости	Высокая	Ручная, Автомат.

№	Шаг	Данные		Выполнение	
		входные	выходные	сложность	форма
2.13	Применение машинного обучения к подпрограммам для дополнительного восстановления Метаинформации	Подпрограммы	Метаинформация (Псевдокод, Алгоритмы, Архитектура, Концептуальная модель)	Высокая	Автомат.
2.14	Применение машинного обучения к восстановленной Метаинформации для ее уточнения	Метаинформация	Метаинформация	Высокая	Автомат.
3	*Динамическое исследование*				
3.1	Анализ выбранной подпрограммы для уточнения ее псевдокода	Подпрограммы	Псевдокод	Высокая	Ручная
3.2	Анализ выбранных подпрограмм для уточнения их алгоритмов	Подпрограммы	Алгоритмы	Средняя	Ручная
3.3	Анализ выбранных подпрограмм для уточнения схемы их внутрипрограммного взаимодействия	Подпрограммы	Схема внутрипрограммного взаимодействия	Средняя	Ручная
3.4	Анализ подпрограмм для уточнения схемы их межпрограммного взаимодействия	Программная система	Схема межпрограммного взаимодействия	Средняя	Ручная
3.5	Анализ схемы внутрипрограммного и межпрограммного взаимодействия для уточнения интерфейсов	Схема внутрипрограммного взаимодействия, Схема межпрограммного взаимодействия	Интерфейсы	Средняя	Ручная
3.6	Анализ алгоритмов подпрограмм и интерфейсов для уточнения их общей архитектуры	Алгоритмы	Архитектура	Средняя	Ручная
3.7	Анализ архитектуры для уточнения ее концептуальной модели	Архитектура	Концептуальная модель	Средняя	Ручная
3.8	Анализ восстановленной Метаинформации для уточнения функционала МК	Метаинформация	Функционал	Средняя	Ручная
3.9	Уточнение Уязвимостей по восстановленной Метаинформации	Метаинформация	Уязвимости	Низкая	Ручная
4	*Документирование*				
4.1	Описание функционала МК	Метаинформация	Документ с описанием функционала	Средняя	Ручная
4.2	Описание восстановленной Метаинформации	Метаинформация	Документ с описанием метаинформации	Средняя	Ручная
4.3	Описание найденных Уязвимостей	Метаинформация	Документ с описанием уязвимостей	Средняя	Ручная
4.4	Оценка безопасности использования объекта исследования	Уязвимости	Документ с оценкой Безопасности	Средняя	Ручная

Во-вторых, 5 шагов имеют высокую сложность выполнения, в основном, по следующим причинам – трудоемкость ручного восстановления псевдокода и поиска уязвимостей, а также применение машинного обучения, подготовка датасета и построение моделей для которого является нетривиальной задачей. При этом 8 шагов имеют низкую сложность выполнения из-за очевидности выполняемых действий и наличия готовых средств автоматизации.

И, в-третьих, 21 шаг должен выполняться полностью ручным способом, что подчеркивает общую трудоемкость Методологии. При этом полная автоматизация возможна лишь для 5 шагов.

Обсуждение

Приведем далее основные недостатки предложенной Методологии и способы если их не полного устранения, то смягчения.

1) Выполнение некоторых шагов может оказаться достаточно сложным и, при этом, без возможности применения средств автоматизации; например, данная ситуация характерна для выделения секций (Шаг 2.1), получения псевдокода (Шаг 2.4), интерфейсов (Шаг 2.8) и пр. Причиной этого является преднамеренное затруднение анализа разработчиком, например, при создании вре-

доносного программного обеспечения. Тем не менее, совместное использование этапов статического и динамического исследования, а также ручного труда опытных экспертов позволит получить необходимые результаты и в этом случае.

2) В реальной практике выполнение шагов по схеме может иметь не ниспадающий характер, а итеративный – когда после более позднего шага необходимо вернуться на уже выполненный; например, динамический анализ МК на Шаге 3.1 может показать, что в секции данных также присутствует МК, псевдокод которого необходимо восстановить на Шаге 2.4. Такие ситуации вполне реальны, однако будем их считать вариативностью Методологии и результатом ошибок на предыдущих шагах.

3) Часто требуется РИ не всего имеющегося в Образе МК, а лишь его части; например, отдельных механизмов или модулей. В этом случае, очевидно, восстановление всего МК будет чрезмерным и трудозатратным. Таким образом, при специализированных задачах РИ каждый шаг предварительно требует некоторой корректировки для выполнения только необходимой части работы. Например, если необходимо получить механизм работы BDS-фазы в UEFI-Образе, то скорее всего достаточно будет восстановить Метаинформацию только для файла *BdsDxe.efi*.

Заключение

В данной заключительной части цикла статей были описаны оставшиеся 2 из 4 этапов методологии РИ, которые не вошли в первые две части. Также была приведена полная схема методологии с указанием всех шагов и получаемых на них результатов. Итоговое сведение этапов и шагов в форме таблицы с рядом характеристик отображает их взаимосвязь и позволяет планировать способы реализации.

Новизна дополняет указанную в первых двух частях цикла тем, что помимо системности и масштабности охвата решения задачи реверс-инжиниринга, дается оценка сложности и формы выполнения всех 35 шагов, что подчеркивает детальность и глубину рассмотрения процесса. Теоретическая и практическая значимость дополнены приведением полной схемы РИ и наличием сводной таблицы этапов и шагов, позволяющей обоснованно проектировать процесс исследования МК в реальных проектах. Так (см. таблицу 1), исходя из сложности каждого шага Методологии, возможно оценить трудоемкость и всего процесса РИ, а учитывая форму выполнения шагов – возможную степень автоматизации.

Продолжением исследования должны стать бóльшая детализация шагов в части методик и средств их реализации, а также внедрение в методологию механизмов машинного обучения [15, 16].

Список источников

1. Hendrix T.D., Cross J.H., Barowski L.A., Mathias K.S. Tool support for reverse engineering multi-lingual software // Proceedings of the Fourth Working Conference on Reverse Engineering (Amsterdam, Netherlands, 06–08 October 1997). IEEE, 1997. PP. 136–143. DOI:10.1109/WCRE.1997.624584

2. Израилов К.Е. Методология проведения реверс-инжиниринга машинного кода. Часть 1. Подготовка объекта исследования // Труды учебных заведений связи. 2023. Т. 9. № 5. С. 79–90. DOI:10.31854/1813-324X-2023-9-5-79-90. EDN:ZXLTBA

3. Израилов К.Е. Методология проведения реверс-инжиниринга машинного кода. Часть 2. Статическое исследование // Труды учебных заведений связи. 2023. Т. 9. № 6. С. 68–82. DOI:10.31854/1813-324X-2023-9-6-68-82. EDN:SJSHCE

4. Каушан В.В. Поиск ошибок выхода за границы буфера в бинарном коде программ // Труды Института системного программирования РАН. 2016. Т. 28. № 5. С. 135–144. DOI:10.15514/ISPRAS-2016-28(5)-8. EDN:VBDRBC

5. Гузик В.Ф., Золотовский В.Е., Савельев П.В. Выполнение математических операций в системе символьных вычислений // Известия ТРТУ. 2007. № 3(75). С. 138–141. EDN:KTMRPB

6. Леошкевич И.О. Получение архитектурно-независимой семантики исполняемого кода // Безопасность информационных технологий. 2009. Т. 16. № 4. С. 120–124. EDN:WTKHGF

7. Переберина А.А., Костюшко А.В. Разработка инструментария для динамического анализа вредоносного программного обеспечения // Труды МФТИ. 2018. Т. 10. № 3(39). С. 24–44. EDN:YZAJAD

8. Ревнивых А.В., Велижанин А.С. Методика автоматизированного формирования структуры дизассемблированного листинга // Кибернетика и программирование. 2019. № 2. С. 1–16. DOI:10.25136/2306-4196.2019.2.28272. EDN:TGCZKJ

9. Соловьев М.А. Восстановление алгоритма по набору бинарных трасс. Дис. ... канд. физ.-мат. наук. М.: Институт системного программирования РАН, 2013. 123 с. EDN:SUSPIZ

10. Jimborean A., Herrmann M., Loechner V., Clauss P. VMAD: A virtual machine for advanced dynamic analysis of programs // Proceedings of International Symposium on Performance Analysis of Systems and Software (Austin, USA, 10–12 April 2011). IEEE, 2011. PP. 125–126. DOI:10.1109/ISPASS.2011.5762725

11. Черчесов А.Э. Фазы загрузки UEFI и способы контроля исполняемых образов // Вопросы защиты информации. 2018. № 2(121). С. 51–53. EDN:RSUTFZ

12. Safyallah H., Sartipi K. Dynamic Analysis of Software Systems using Execution Pattern Mining // Proceedings of the14th IEEE International Conference on Program Comprehension (Athens, Greece, 14–16 June 2006). IEEE, 2006. PP. 84–88. DOI:10.1109/ICPC.2006.19

13. Scherer K., Pfeffer T., Glesner S. I/O Interaction Analysis of Binary Code // Proceedings of the 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (Napoli, Italy, 12–14 June 2019). IEEE, 2019. PP. 225–230. DOI:10.1109/WETICE.2019.00056

14. Липаев В.В. Риски проектирования и производства мобильных программных продуктов // Труды Института системного программирования РАН. 2011. Т. 21. С. 167–182. EDN:OKGXND

15. 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. EDN:AIOFPM

16. Kotenko I., Izrailov K., Buinevich M. Static Analysis of Information Systems for IoT Cyber Security: A Survey of Machine Learning Approaches // Sensors. 2022. Vol. 22. Iss. 4. PP. 1335. DOI:10.3390/s22041335

References

1. Hendrix T.D., Cross J.H., Barowski L.A., Mathias K.S. Tool support for reverse engineering multi-lingual software. *Proceedings of the Fourth Working Conference on Reverse Engineering*, *06–08 October 1997, Amsterdam, Netherlands*. IEEE; 1997. p.136–143. DOI:10.1109/WCRE.1997.624584

2. Izrailov K. Methodology for Machine Code Reverse Engineering. Part 1. Preparation of the Research Object. *Proceedings of the Telecommun. Univ*. 2023;9(5):79–90. DOI:10.31854/1813-324X-2023-9-5-79-90. EDN:ZXLTBA

3. Izrailov K. Methodology for Machine Code Reverse Engineering. Part 2. Static Investigation. *Proceedings of the Telecommun. Univ*. 2023;9(6):68–82. DOI:10.31854/1813-324X-2023-9-6-68-82. EDN:SJSHCE

4. Kaushan V.V. Buffer overrun detection method in binary code. *Proceedings of ISP RAS*. 2016;28(5):135–144. DOI:10.15514/ISPRAS-2016-28(5)-8. EDN:VBDRBC

5. Guzik V.F., Zolotovskiy V.Ye., Savelev P.V. Performing mathematical operations in the symbolic computing system. *Izvestiya SFedu. Engineering sciences*. 2007;3(75):138–141. EDN:KTMRPB

6. Leoshkevich I.O. Obtaining architecture-independent semantics of executable code. *Proceedings of IT Security (Russia)*. 2009;16(4):120–124. EDN:WTKHGF

7. Pereberina A.A., Kostyushko A.V. Development of tools for dynamic malware analysis. *Proceedings of Moscow Institute of Physics and Technology*. 2018;10(3):24–44. EDN:YZAJAD

8. Revnivykh A.V., Velizhanin A.S. Methods for Automated Formation of a Disassembled Listing Structure. *Cybernetics and Programming*. 2019;2:1–16. DOI:10.25136/2306-4196.2019.2.28272. EDN:TGCZKJ

9. Solovev M.A. *Reconstructing the algorithm from a set of binary traces.* PhD Thesis. Moscow: Ivannikov Institute for System Programming of the Russian Academy of Sciences Publ.; 2013. 123 p. EDN:SUSPIZ

10. Jimborean A., Herrmann M., Loechner V., Clauss P. VMAD: A virtual machine for advanced dynamic analysis of programs. *Proceedings of International Symposium on Performance Analysis of Systems and Software, 10–12 April 2011, Austin, USA.* IEEE; 2011. p.125–126. DOI:10.1109/ISPASS.2011.5762725

11. Cherchesov A.E. UEFI boot phases and how to control executable images. *Voprosy zashchity informatsii.* 2018;2(121):51–53. EDN:RSUTFZ

12. Safyallah H., Sartipi K. Dynamic Analysis of Software Systems using Execution Pattern Mining. *Proceedings of the 14th IEEE International Conference on Program Comprehension, 14–16 June 2006, Athens, Greece*. IEEE; 2006. p.84–88. DOI:10.1109/ICPC.2006.19

13. Scherer K., Pfeffer T., Glesner S. I/O Interaction Analysis of Binary Code. *Proceedings of the 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, 12–14 June 2019, Napoli, Italy*. IEEE; 2019. p.225–230. DOI:10.1109/WETICE.2019.00056

14. Lipayev V.V. Risks of design and production of mobile software products. *Proceedings of ISP RAS*. 2011;21:167–182. EDN:OKGXND

15. Izrailov K. Enetic Decompilation Concept of the Telecommunication Devices Machine Code. *Proceedings of the Telecommun. Univ*. 2021;7(4):10–17. DOI:10.31854/1813-324X-2021-7-4-95-109. EDN:AIOFPM

16. Kotenko I., Izrailov K., Buinevich M. Static Analysis of Information Systems for IoT Cyber Security: A Survey of Machine Learning Approaches. *Sensors*. 2022;22(4):1335. DOI:10.3390/s22041335

Статья поступила в редакцию 29.11.2023; одобрена после рецензирования 02.02.2024; принята к публикации 05.02.2024.
The article was submitted 29.11.2023; approved after reviewing 02.02.2024; accepted for publication 05.02.2024.

Информация об авторе:

ИЗРАИЛОВ Константин Евгеньевич

кандидат технических наук, доцент, старший научный сотрудник лаборатории проблем компьютерной безопасности Санкт-Петербургского Федерального исследовательского центра Российской академии наук

https://orcid.org/0000-0002-9412-5693