Научная статья
УДК 004.42
DOI:10.31854/1813-324X-2023-9-5-79-90

Методология реверс-инжиниринга машинного кода. Часть 1. Подготовка объекта исследования

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. В первой части цикла статей проводится обзор научных публикаций данной предметной области. В условиях отсутствия удовлетворительных решений предлагается авторская методология процесса, состоящая из 4-х следующих этапов: подготовительные мероприятия, статическое исследование, динамическое исследование и документирование. Приводится детальное описание шагов 1-го этапа, а также примеры их применения на практике с использованием типовых программных средств. Схема предлагаемой методологии представлена в графическом виде, а приведенные шаги имеют формальную запись. В следующих частях цикла статей будут описаны шаги остальных этапов и их систематизация в табличном виде с указанием входных и выходных объектов, а также формы выполнения шагов.*

Ключевые слова: *реверс-инжиниринг, обратная разработка, программная инженерия, информационная безопасность, уязвимости, методология, IDA Pro*

Ссылка для цитирования: Израилов К.Е. Методология 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

Methodology for Machine Code Reverse Engineering. Part 1. Preparation of the Research Object

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 devices machine code are presented. The first part of the series of articles reviews scientific publications in this subject area. In the absence of satisfactory solutions, the author's process methodology is proposed, consisting of the following 4 stages: preparatory activities, static research, dynamic research and documentation. A detailed description of the steps of the first stage is provided, as well as examples of their application in practice using standard software. The scheme of the proposed methodology is presented in graphical form, and the steps given are formally written. The next part of the series of articles will describe the steps of the remaining stages and their systematization in tabular form, indicating the input and output objects, as well as the form of steps execution.*

Keywords: *reverse engineering, software engineering, information security, vulnerabilities, methodology, IDA Pro*

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

1. Введение

Задача исследования машинного кода (далее – МК) программного обеспечения считается актуальной уже многие десятки лет. Основная цель такого исследования ставится как получение информации о функционале кода, отличие которого от заявленного может означать наличие в нем уязвимостей [1]. Также в ряде случаев необходима доработка программы или ее полная замена на аналог, а в случае отсутствия исходных кодов их восстановление из МК (а также алгоритмов и архитектуры) является практически единственным решением [2]. Сам процесс носит название «реверс-инжиниринга» (далее – РИ), заимствованное из английского «reverse engineering», что переводится, как интуитивно понятный термин – «обратная разработка».

Исходя из указанных потребностей, в РИ создано определенное количество методов и программных средств (например, IDA Pro, Ghidra, Radare2 и др.), позволяющих проводить исследование и восстанавливать необходимую информацию для отдельно взятых программ; при этом участие эксперта в процессе велико, поскольку какая-либо полностью специфицированная и автоматическая процедура отсутствует. Успешность же РИ оказывается полностью зависящей от знаний эксперта, его опыта, наличия под рукой средств, а также самого исследуемого экземпляра МК; при этом, если специфика программы или конечная цель исследования выходит за рамки осведомленности эксперта, то процесс проведения РИ может оказаться слабо прогнозируемым, неэффективным или же даже безрезультатным. Таким образом, в данной предметной области существует научная проблема, имеющая вид следующего противоречия. С одной стороны, в информационной среде присутствует огромное количество разнородного программного обеспечения с МК, выполняемым на различных устройствах, имеющим различный формат хранения, целевое назначение, интерфейсы взаимодействия, условия функционирования и т. п. – то есть РИ необходимо применять для всего многообразия программного обеспечения. С другой стороны, отдельно взятые эксперты по РИ (или даже экспертные группы) знакомы лишь с рядом методов, имеют в наличии и используют определенный набор средств, восстанавливают из МК лишь определенную часть информации – то есть в рамках одного исследования РИ может применяться для ограниченного числа программного обеспечения. Первым же шагом для разрешения противоречия должно стать определение методологии (далее – Методология), как логической организации методов и средств проведения РИ. А поскольку (что будет показано далее) подобная Методология, обладающая необходимой полнотой, отсутствует, то будет предложена новая, основанная на огромном научно-практическом опыте автора по проведению РИ.

2. Обзор работ

Проведем обзор работ, посвященных вопросу РИ в части существующих Методологий его проведения, а также общих используемых методов и инструментария; при этом вопросы применения отдельных программных средств затрагиваться не будут, поскольку они являются частными решениями по автоматизации методов.

В [3] рассматриваются 3 информационно-технических процесса, связанные с РИ и решающие следующие задачи. Во-первых, РИ существующих физических объектов помогает создать их цифровые модели, переведя процесс исследования и разработки в информационную плоскость. Так, например, получение 3D-модели технических устройств с последующей ее модификацией позволяют не только проектировать инновационный опытный образец, но и производить его тиражированный выпуск. Во-вторых, РИ программного обеспечения необходим для определения принципов работы последнего, его архитектуры и алгоритмов, модификация которых позволяет расширять функциональные возможности. Частными задачами в рамках РИ указана расшифровка форматов файлов, восстановление исходного кода или потерянной метаинформации (имен классов, функций и переменных), построение графов потока управления и т. п. И, в-третьих, рассматривается обратная сторона РИ, связанная с защитой собственных программных разработок от РИ. Для этого предлагается применять алгоритмы шифрования и обфускации. Также рассматривается внедрение в собственные продукты недекларированных возможностей с целью контроля использования программ и для усложнения их РИ, что, впрочем, идет в разрез с законодательством РФ. Упоминаются и более специфичные способы защиты кода от РИ, такие как его мутация, рандомизация и использование защищенных сред выполнения. Однако, несмотря на некоторую проработанность области РИ, в статье отсутствует какая-либо полноценная Методология, имеющая не только формализованный вид, но и содержащая хотя бы какую-то последовательность методов (или шагов) и инструментов.

Работа [4] посвящена вопросу создания центра реверс-инжиниринга (далее – ЦРИ), актуальность которого (по мнению авторов) обосновывается «технологической блокадой» России со стороны «США и их несамодостаточных стран-сателлитов», что требует создания собственных технических решений на основе существующих путем сбора и переработки информации о них. Подчеркивается важность не дублирования готовых решений, а создания на их основе новых инновационных с применением всего аппарата научно-практической мысли (например, теории решения изобретательских задач). Предложен алгоритм работы такого

ЦРИ, основной последовательностью шагов которого является следующая: составление, утверждение и уточнение технического задания; структурный и функционально-стоимостной анализ проектного проекта, а также патентные исследования; испытания натурного образца (в случае необходимости); выбор и копирование необходимых элементов; модернизация конструкции и ее патентная защита; формирование и передача финальных документов. В качестве недостатков предложенного решения отмечены трудности в определении слабых мест исходного изделия и способах оптимизации процесса из РИ.

Целью исследования в [5] является оценка значимости патентной информации для обеспечения РИ, а задачей – выявление проблем, связанных с применением в РФ РИ к физическим объектам. В статье дискутируется терминология предметной области; так, под РИ (или реинжинирингом, обратной разработкой) понимается как процесс анализа продукта для сбора информации о нем, включая построение его 3D-модели, так и его воспроизведение. Приводятся два основных этапа РИ, а именно следующие: 1) сбор данных об объекте, его анализ и оценка правовых аспектов; 2) воспроизведение точной копии объекта, в том числе после его доработки. Подчеркивается обоснованность проведения РИ только при наличии материалов и средств для дальнейшего воспроизводства «клона». Делается вывод касательно важности поисково-исследовательской деятельности по сбору и анализу открытой патентной информации об объекте; приводится пример, что в патентах может быть найдено от 70 до 90 % технической информации об объекте.

В работе [6] описывается методика проведения РИ в программном продукте Siemens NX, позволяющая разрабатывать электронные макеты изделий перед его технологическим производством. В методике выделяются такие этапы, как импорт данных, совмещение и склейка фасетных тел, исправление ошибок геометрии, размещение и центрирование фасетного тела, получение сечений и/или цветовое выделение граней, создание модели и анализ ее точности, формирование «идеализированной» модели. Авторы указывают, что предложенная методика подойдет для всего спектра объектов любого масштаба и сложности.

Работа [7] обсуждает проблемы, существующие при практическом приложении РИ для производства ракетно-космической техники на основе существующих решений. К основным проблемам в части реализации на ЭВМ авторы относят следующие: дороговизна решения, высокая квалификация специалиста, слабая адаптивность моделей для всех разработчиков, рутинность процесса. Также отмечены второстепенные проблемы применения РИ для космических аппаратов, такие, как сложность управления проектом, нарушение геометрических связей, отсутствие стандартов РИ и несовместимость итоговых частей всей системы.

В статье [8] приводится мнение различных руководителей крупных IT-компаний касательно требований в вакансии к сотрудникам на позицию реверс-инженера, что, таким образом, может определять перечень знаний, умений, методов и средств, необходимых для проведения РИ. Все множество указанных требований может быть систематизировано в следующий перечень: мониторинг процессов в операционной системе (далее – ОС); практика проведения РИ; понимание основ работы аппаратного обеспечения и ОС; знание всего стека классических языков программирования (от ассемблерного до C/C++ и Python); работа с продуктом IDA Pro (включая плагин декомпиляции HexRays), WinDBG и OllyDbg, а также различными утилитами дизассемблирования и отладки.

Исследование [9] посвящено РИ сетевого протокола взаимодействия объектов. Для этого авторы делают обзор различных инструментов статического анализа трафика, что в итоге позволяет сформировать единый набор этапов проведения РИ, а именно следующих: первоначальная настройка параметров для выбора последующих методов, сбор и обработка данных, выделение в данных признаков для извлечения закономерностей внутри и между сетевыми сообщениями, идентификация сообщения для создания их семантических групп, определение формата сообщений для выделения их полей, раскрытие семантических значений полей, построение модели конечного автомата для состояний работы протокола, финальная пост-обработка результатов ручным или автоматическим способом.

Согласно обзору немногочисленного числа найденных релевантных работ, ни в одной из них не дается сколь-либо полноценного описания методологии РИ, а в самих исследованиях приводятся или достаточно общие этапы, или детализация их частных алгоритмов; при этом формализация РИ практически отсутствует. Также никак не рассматривается возможность применения для анализа МК машинного обучения, что с точки зрения автора является безусловным упущением [10, 11]. Таким образом, тема текущего авторского исследования является безусловно актуальной и новой.

3. Онтологическая модель

Перед описанием предлагаемой Методологии построим онтологическую модель проведения РИ, определяющую используемые далее основные понятия предметной области и их взаимосвязи; данная модель приведена на рисунке 1 (синим фоном отмечена основная деятельность РИ, серым – второстепенная деятельность, желтым – основной объект приложения, зеленым – основной результат, красным – побочные результаты).

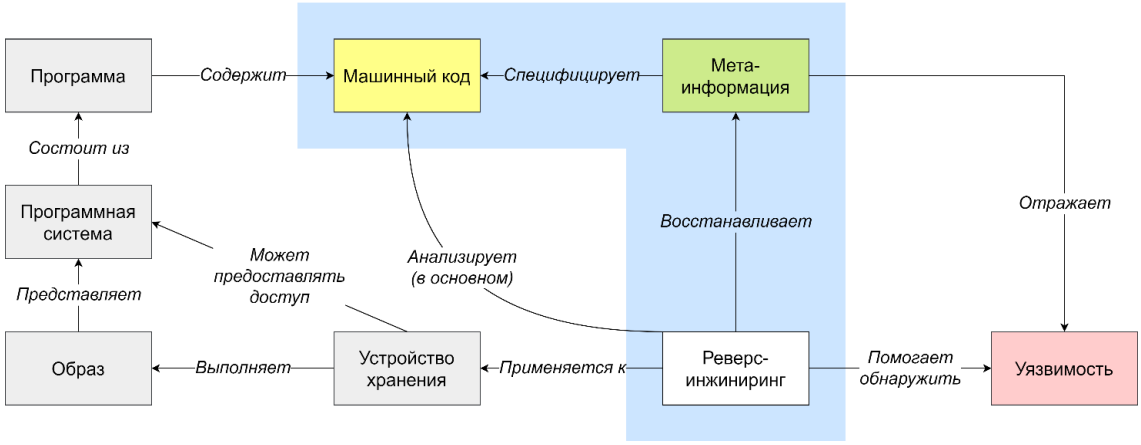

Рис. 1. Онтологическая модель проведения реверс-инжиниринга

Fig. 1. Ontological Model of the Reverse Engineering

Особенностью данной онтологической модели является то, что она сама по себе уже дает некоторое представление об основных принципах РИ.

На схеме присутствуют следующие элементы (их взаимосвязи отмечены курсивом):

1) Машинный код – низкоуровневое представление логики работы в виде инструкций, выполняемых процессором;

2) Программа – бинарный файл, *содержащий* экземпляр МК и информацию для его хранения и выполнения (например, имя файла, заголовок с информацией о процессоре выполнения и т. п.);

3) Программная система – логическая структура, *состоящая из* совокупности взаимосвязанных и взаимодействующих Программ (в вырожденном случае является единичной Программой), а также вспомогательных невыполняемых файлов;

4) Образ – *представление* Программной системы в виде монолитной бинарной сущности, как правило запакованной, сжатой или зашифрованной (например, в ISO-форме); с учетом специфики решаемых задач Образ может не всегда присутствовать в явном виде, как например в ОС Windows и Linux, которые собственными средствами *могут предоставлять доступ* к Программам;

5) Устройство хранения – техническое (реже, программное) устройство, *выполняющее* загружаемый в него Образ, который также может быть извлечен из устройства;

6) Метаинформация – сведения, *специфицирующие* МК и дающие полное человеко-ориентированное представление о его функционале (в основном включает псевдокод, алгоритмы, архитектуру, концептуальную модель, дополненные интерфейсами взаимодействия и т. п.) [12];

7) Реверс-инжиниринг – процесс восстановления Метаинформации о МК; в общем случае применяется к Устройству для извлечения и исследования Программ из Образа, хотя *основная* деятельность процесса состоит *в анализе* их МК [13];

8) Уязвимость – отличие итоговой реализации МК от задуманной или заявленной, что приводит к иному функционированию Программы, позволяющему реализовать угрозы (такое авторское введение понятия уязвимости раскрывается в [14, 15]); РИ *помогает обнаружить* Уязвимости, поскольку восстановленная в процессе этого Метаинформация *отражает* их признаки и более подходит для анализа экспертом.

4. Схема методологии

Предлагаемая Методология состоит из 4-х этапов, каждый из которых включает совокупность последовательно выполняемых шагов: *подготовительные мероприятия* (этап 1); *статическое исследование* (этап 2); *динамическое исследование* (этап 3); *документирование* (этап 4). При этом выходные результаты, полученные после применения одних шагов, являются входными данными для других шагов; исключением является 1-й шаг, имеющий на входе исходный МК, и шаги по описанию итоговых результатов, продуцирующие соответствующие финальные документы исследования. Шаги Методологии разбиты по этапам и имеют идентификаторы в следующем формате: *X.Y*, где *X* – номер этапа; *Y* – порядковый номер выполнения в рамках этапа.

Методология РИ в графическом виде имеет вид схемы, представленной на рисунке 2; на схеме используются следующие обозначения: прямоугольник с белым фоном – результат применения шага; круг с желтым фоном – идентификатор шага; сплошная стрелка – основное действие шага над результатом предыдущего; пунктирная стрелка – дополнительное использование шагов предыдущих результатов; пунктирный прямоугольник с белым фоном (и надписью «Этап 1. Подготовка объекта исследования») – шаги, описанные в текущей части цикла статей; пунктирный прямоугольник с серым фоном (и надписью «Этапы 2, 3. Анализ объекта исследования; Этап 4. Документирование») – шаги, которые будут описаны в следующих частях цикла.

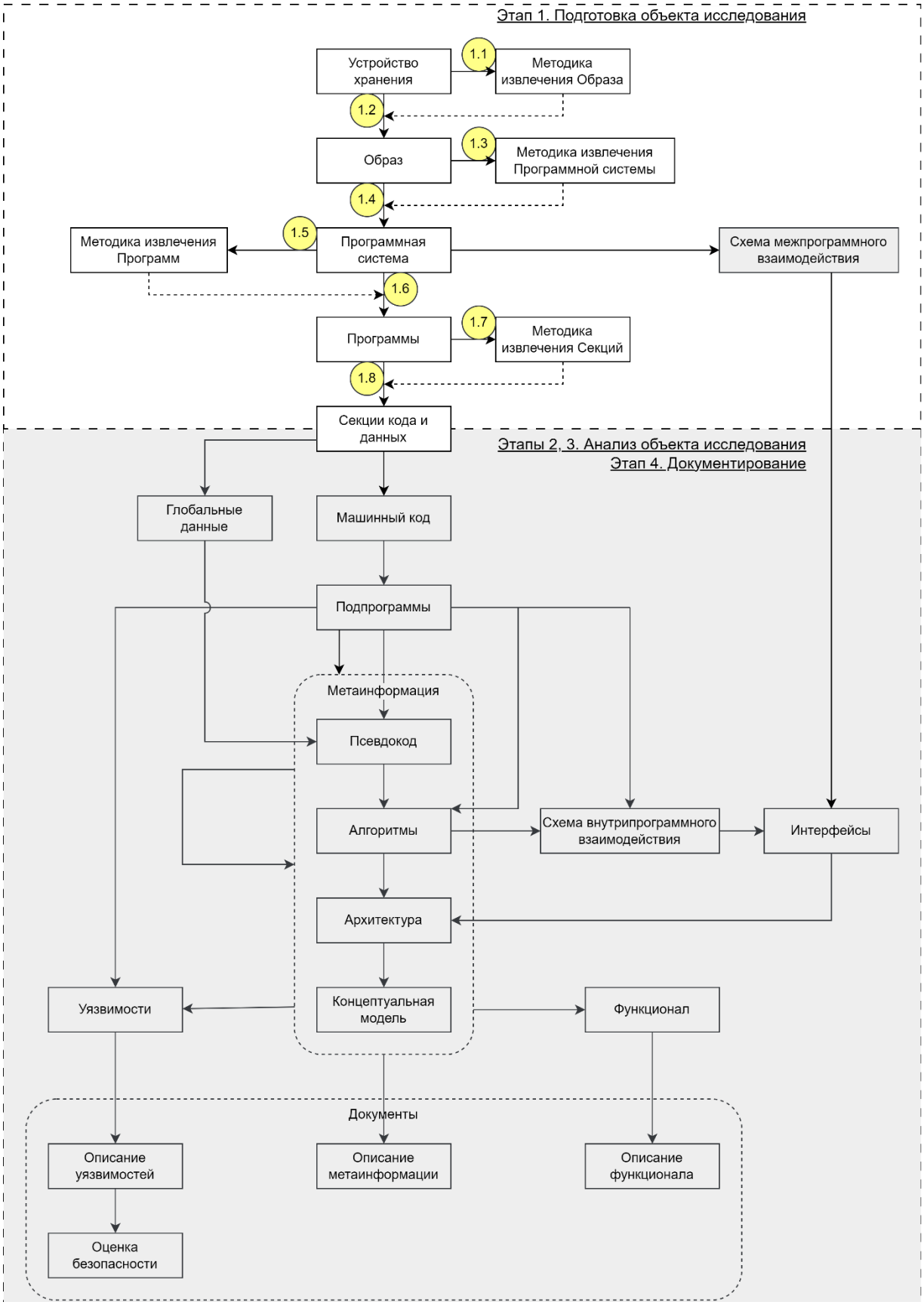

Рис. 2. Схема методологии реверс-инжиниринга (Этап 1)

Fig. 2. Reverse Engineering Methodology Diagram (Stage 1)

Согласно схеме (см. рисунок 2), в данной части статьи будут расписаны Шаги с 1.1 по 1.8; они полностью определяют 1-й этап РИ, предназначенный для извлечения из Устройства хранения основного объекта приложения РИ – программных секций с МК, а также дополняющих их секций с данными (которые носят вспомогательное назначение). Здесь и далее под секциями понимается деление файла программы на отдельные блоки данных определенного типа, таких, как следующие: «.text» – МК программы, «.data» – глобальные переменные, «rsrc» – ресурсы и др.

5. Этап 1. Подготовительные мероприятия

Шаги 1-го этапа Методологии, отвечающего за подготовку МК для непосредственного анализа, представлены далее.

Шаг 1.1. Сбор общей информации об устройстве хранения для разработки методики извлечения образа

Шаг является первоначальным для всей Методологии и текущего этапа, поскольку применяется к Устройству хранения, в котором содержится Образ с МК и всеми программно-логическими структурами. Способ получения такого Образа для дальнейшего анализа существенно зависит от конкретного типа устройства и механизмов хранения, что требует от экспертов изучение отдельных экземпляров устройств и создания соответствующих методик по работе с ними.

Формальная запись шага ($Step_{1.1}$) имеет следующий вид:

$$Method_{Image} = Step_{1.1}(Device),$$

где $Method_{Image}$ – методика получения Образа (*пер. на англ.* Image); $Device$ – исследуемое Устройство хранения.

Примером шага может быть сбор информации о материнской плате исследуемого компьютера (который в данном случае является Устройством хранения) с применением классической для этой задачи утилиты CPU-Z, пример графического окна которой показан на рисунке 3; так материнская плата имеет модель PRIME Z490-A производства ASUSTeK COMPUTER INC.

Затем, в соответствии с типом материнской платы, шаг может описать методику получения самого Образа. Так, в случае платы ASUS и ее UEFI-прошивки производства American Megatrends для этой задачи может быть применена утилита AfuWin, имеющая как графический, так и консольный вид.

Шаг 1.2. Применение к устройству хранения методики извлечения образа

Шаг заключается в применении методики (разработанной на Шаге 1.1), позволяющей получить Образ из конкретного экземпляра Устройства хранения.

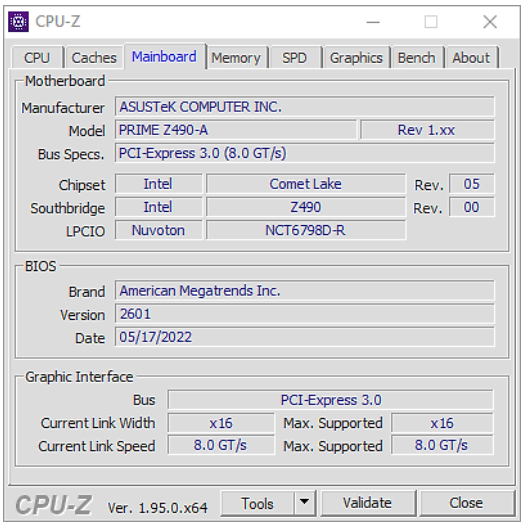

Рис. 3. Пример главного окна утилиты CPU-Z

Fig. 3. Example of the CPU-Z Utility Main Window

Формальная запись шага ($Step_{1.2}$) имеет следующий вид:

$$Image = Step_{1.2}(Method_{Image}, Device),$$

где $Image$ – Образ, полученный из исследуемого устройства ($Device$) с помощью специализированной для этого методики ($Method_{Image}$).

Примером шага может быть применение методики для получения Образа (в виде UEFI-прошивки) из компьютера с материнской платой ASUS, суть которой заключается в запуске утилиты AfuWin в графическом виде; главное окно утилиты представлено на рисунке 4.

Согласно рисунку 4, выбрано получение из материнской платы всех программных блоков, а нажатие кнопки Save приведет к сохранению Образа в бинарный файл на компьютере.

Шаг 1.3. Сбор общей информации об образе для разработки методики извлечения программной системы

Шаг применяется к полученному из Устройства хранения Образу и ставит своей задачей сбор информации о последнем для создания методики извлечения из него Программной системы, поскольку формат и внутренняя структура Образа может зависеть от производителя Устройства или же иметь различные механизмы защиты. Также, как было сказано ранее, в ряде случаев данный и последующий шаги можно пропустить, поскольку Программная система получается напрямую через Устройство; например, в случае Программ на персональном компьютере с типовой ОС.

Формальная запись шага ($Step_{1.3}$) имеет следующий вид:

$$Method_{ProgramSystem} = Step_{1.3}(Image),$$

где $Method_{ProgramSystem}$ – методика получения Программной системы (*пер. на англ.* Program Image); $Image$ – полученный ранее Образ.

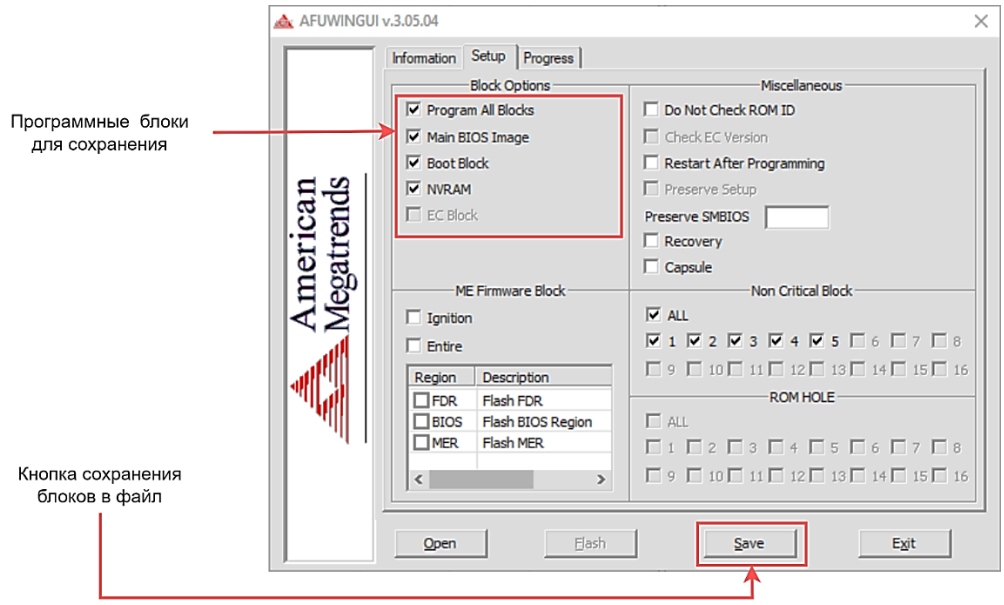

Рис. 4. Пример главного окна утилиты AfuWin

Fig. 4. Example of the AfuWin Utility Main Window

Примером шага может быть сбор информации об UEFI-прошивке (которая в данном случае является Образом) с применением достаточно редкого числа утилит, как стороннего, так и авторского производства. Так, для локализации файлов в Образе (которыми в случае UEFI-прошивки могут быть DXE-драйвера) возможно с применением утилиты UEFITool, пример главного окна которой (для версии NE alpha 67 от 20 июня 2023 г.) приведен на рисунке 5.

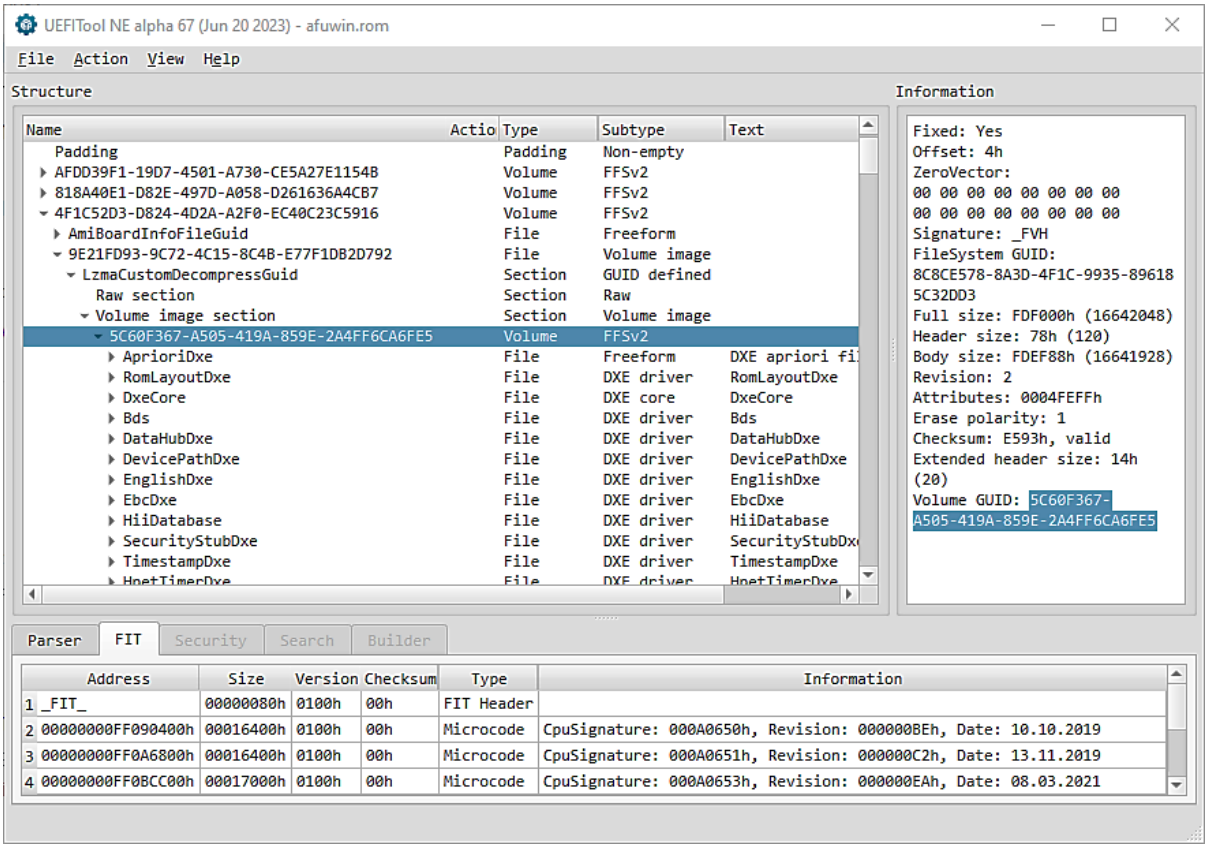

Рис. 5. Пример главного окна утилиты UEFI Tool (версии NE alpha 67)

Fig. 5. Example of the UEFI Tool Utility Main Window (version NE alpha 67)

Так, согласно рисунку 5, в Образе UEFI-прошивки присутствует том с GUID (*аббр. от англ.* Globally Unique Identifier, *пер. на русс.* Глобальный уникальный идентификатор) 5C60F367-A505-419A-859E-2A4FF6CA6FE5 и файловой системой FFSv2, содержащей набор драйверов – RomLayoutDxe, DxeCore, Bds и пр.

Затем, в соответствии с форматом Образа и способом его «раскрытия», результатом шага может стать описание методики извлечения Программной системы. В случае UEFI-прошивки для этой задачи подходит функционал UEFITool, позволяющий экспортировать бинарные данные всех логических элементов прошивки (как целых томов, так и отдельных драйверов) во внешние файлы.

Шаг 1.4. Применения к образу методики извлечения программной системы

Шаг заключается в применении методики (разработанной на Шаге 1.3), позволяющей получить Программную систему из конкретного экземпляра Образа.

Формальная запись шага ($Step_{1.4}$) имеет следующий вид:

$$ProgramSystem = Step_{1.4}(Method_{ProgramSystem}, Image),$$

где $ProgramSystem$ – Программная система, полученная из исследуемого Образа ($Image$) с помощью специализированной для этого методики ($Method_{ProgramSystem}$).

Примером шага может быть применение методики к UEFI-прошивке, суть которой заключается в ее открытии в утилите UEFITool, выборе нужного файлового тома и ручного применения к нему функционала утилиты по экспорту данных во внешние файлы. Как результат, можно получить бинарное представление Программной системы. В ряде случаев на данном шаге достаточно получить лишь список путей к файлам, поскольку их извлечение делается непосредственно из Образа, минуя работу с бинарным представлением Программной системы.

Шаг 1.5. Сбор общей информации о программной системе для разработки методики извлечения программ

Шаг применяется к полученной из Образа Программной системе и ставит своей задачей сбор информации о последней для создания методики извлечения из нее отдельных Программ, поскольку внутренняя структура системы и файлы с МК (из состава всех файлов) могут зависеть от ее формата и специфики исследуемого Устройства хранения.

Формальная запись шага ($Step_{1.5}$) имеет следующий вид:

$$Method_{Programs} = Step_{1.5}(ProgramSystem),$$

где $Method_{Programs}$ – методика получения Программ (*пер. на англ.* Programs); $ProgramSystem$ – полученная ранее Программная система.

Примером шага может быть сбор информации о файлах в томе UEF-прошивки для выделения DXE-драйверов (Программ, работающих на UEFI-фазе DXE, *аббр. от англ.* Driver Execution Environment, *пер. на русс*. Среда выполнения драйвера). В данном случае не все файлы в FFSv2 томах прошивок выполняемые, содержащие МК и, следовательно, выделение среди них Программ является отдельной задачей, которая может быть решена экспертно с применением упоминаемой утилиты UEFITool. Для этого можно воспользоваться стандартными названиями DXE-драйверов или изучением структуры файла, в котором будет присутствовать секция PE32, соответствующая PE-формату (*аббр. от англ.* Portable Executable, *пер. на русс.* Переносимый исполняемый) исполняемого файла для 32-битной разрядной системы. Одним из достаточно хорошо известных названий драйверов в UEFI является BDS (*аббр. от англ.* Boot Device Selection, *перев. на русс.* Выбор Загрузочного Устройства), реализующих одноименную фазу загрузки компьютера, а, следовательно, файл с таким именем скорее всего будет Программой. Пример отображения DXE-драйвера BDS с информацией о нем в утилите приведен на рисунке 6. Красным прямоугольником на белом фоне отмечена строка с названием DXE-драйвера, красным прямоугольником на синем фоне – его PE-секция с МК.

Все это позволяет сформировать итоговую методику получения Программ из Программной системы. Важно отметить, что в случае UEFI-прошивки на Шагах 1.3 и 1.4 достаточно лишь получить список файлов без непосредственного извлечения всей Программной системы в отдельный бинарный файл, поскольку получение конкретных Программ эффективнее делать непосредственно в утилитах по работе с UEFI-образом, такими, как UEFITool.

Шаг 1.6. Применение к программной системе методики извлечения программ

Шаг заключается в применении методики (разработанной на Шаге 1.5), позволяющей получить конкретные Программы из Программной системы. Формальная запись шага ($Step_{1.6}$) имеет следующий вид:

$$\begin{cases} Programs = \\ Step_{1.6}(Method_{Programs}, ProgramSystem), \\ Programs \equiv \{Program_i\} \end{cases}$$

где $Programs$ – множество Программ, полученных из исследуемой Программной системы ($Program\ System$) с помощью специализированной для этого методики ($Method_{Programs}$); $Program_i$ – i-я Программа из множества (для упрощения, индекс i далее опустим).

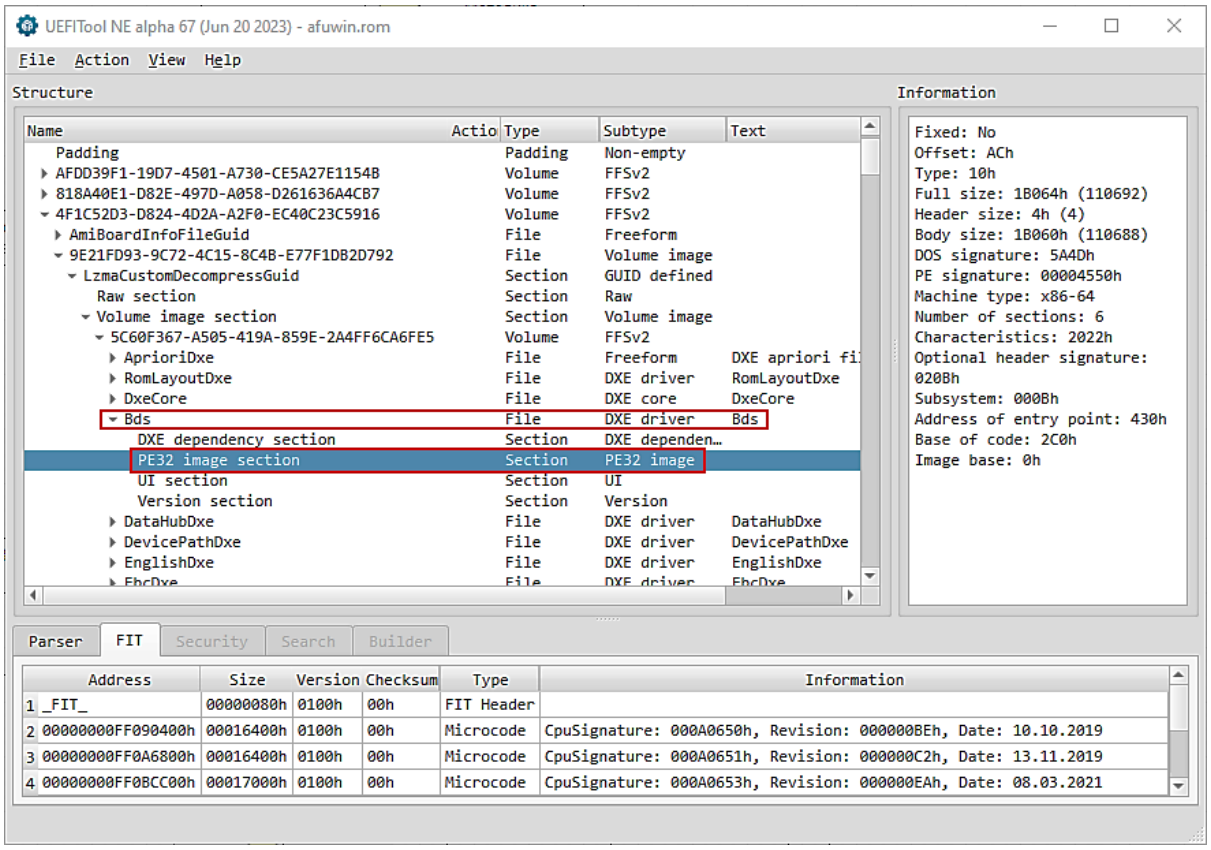

Рис. 6. Пример отображения информации о DXE-драйвере BDS в утилите UEFI Tool

Fig. 6. An Example of Displaying Information about the BDS DXE Driver in the UEFITool

Примером шага может быть применение методики к UEFI-прошивке, суть которой заключается в ее открытии в утилите UEFITool, выборе нужной Программы и применения к ней функционала утилиты по экспорту данных во внешний файл. Как результат, можно получить бинарное представление каждой Программы тома файловой системы; аналогичный функционал присутствует у авторской утилиты по работе UEFI-образом.

Шаг 1.7. Сбор общей информации о программах для разработки методики выделения секций

Шаг применяется к полученным из Программной системы Программам и ставит своей задачей сбор информации о каждой из последних для создания методики выделения секций с МК и используемыми им данными, поскольку внутренняя структура каждой Программы может зависеть от ее формата (примерами наиболее популярных являются PE для ОС семейства Windows и ELF для ОС семейства UNIX).

Формальная запись шага ($Step_{1.7}$) для каждой Программы имеет следующий вид:

$$Method_{Sections} = Step_{1.7}(Program),$$

где $Method_{Sections}$ – методика получения секций (*пер. на англ.* Sections); $Program$ – одна из полученных ранее Программ.

Примером шага может быть чтение первых байт Программы, в которых, как правило, расположена сигнатура, задающая формат файла: «MZ» (байты в шестнадцатеричном виде – 0x4D 0x5A) для PE-программы, «\x7fELF» (байты в шестнадцатеричном виде – 0x7f 0x45 0x4c 0x46) – для ELF-программ и др. Так, например, открытие DXE-драйвера для BDS в шестнадцатеричном онлайн-редакторе HexEd.it даст результат, представленный на рисунке 7. Первыми 2-мя байтами файла BdsDxe.efi являются 0x4D и 0x5A и, следовательно, файл представляет собой Программу в PE-формате.

Затем, в соответствии с форматом Программы, можно выбрать утилиту по работе с ней, которая позволяет извлекать программные секции – как правило она будет относиться к семейству дизассемблеров программ (т. е. с поддержкой их различных форматов). Одной из наиболее популярных такого рода утилит является продукт IDA Pro (*аббр. от англ.* Interactive DisAssembler, *пер. на русс.* Интерактивный ДизАссемблер), который позволяет не только определять форматы программ и процессора их выполнения, выделять в них секции, но и отображать МК, а также производить его извлечение автоматически с применением скриптов на С-подобном или Python языках [16]. Как альтернатива, существуют и специализированные программы по работе с заголовками файлов различных форматов.

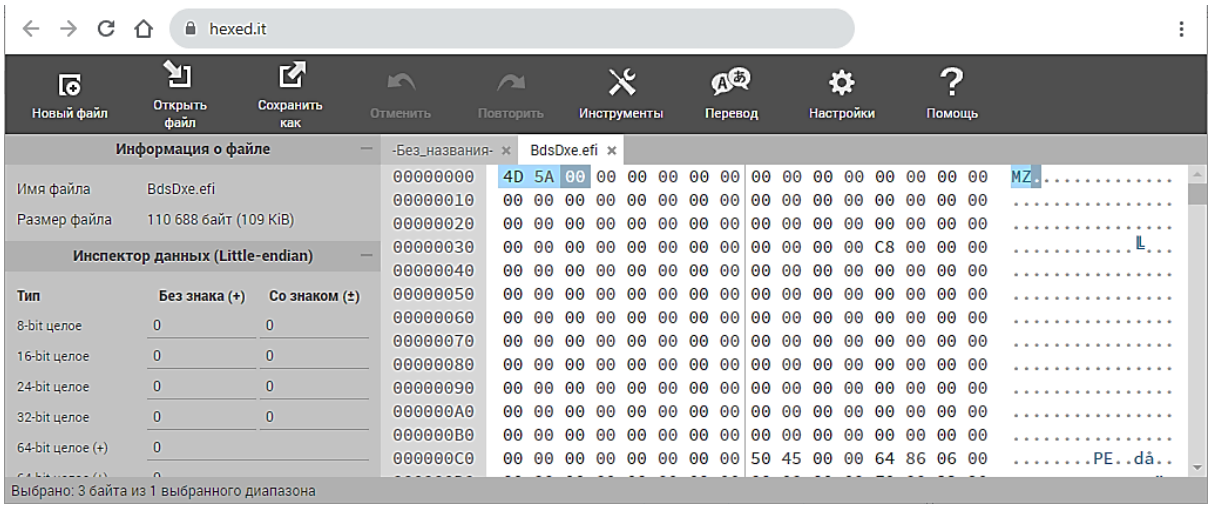

Рис. 7. Пример открытие DXE-драйвера для BDS в HexEd.it

Fig. 7. Example of Opening a DXE Driver for BDS in HexEd.it

Все это позволяет сформировать итоговую методику получения секций из каждой Программы в подходящей для этого утилите.

Шаг 1.8. Применение к программам методики выделения секций

Шаг является конечным для текущего этапа и заключается в применении методики (разработанной на Шаге 1.7), позволяющей получить секции с МК и данными в конкретной Программе. Формальная запись шага ($Step_{1.8}$) имеет следующий вид:

$$\begin{cases} Sections = Step_{1.8}(Method_{Sections}, Program) \\ Sections \equiv \langle Section_i \rangle \\ Class_{Section} \epsilon \{Section^{Code}, Section^{Data}\} \end{cases},$$

где $Sections$ – множество секций, полученных из исследуемой Программы ($Program$) с помощью специализированной для этого методики ($Method_{Sections}$); $Section_i$ – i-я секция из множества; $Class_{Section}$ – класс секции, которая содержит или код ($Section^{Code}$) или данные ($Section^{Data}$).

Примером шага может быть применение методики к DXE-драйверу с реализацией UEFI-фазы BDS – BdsDxe.efi, суть которой заключается в открытии файла в утилите IDA Pro и отображении окна Segments, пример которого приведен на рисунке 8. Так, Программа BdsDxe.efi была корректно открыта в утилите, а на экране отображены ее секции, одна из которых (с названием «.text») содержит МК.

Аналогичным образом применение утилиты PE Tools позволит вывести более детальную информацию о Программе, также включая и секции (пример вывода показан на рисунке 9).

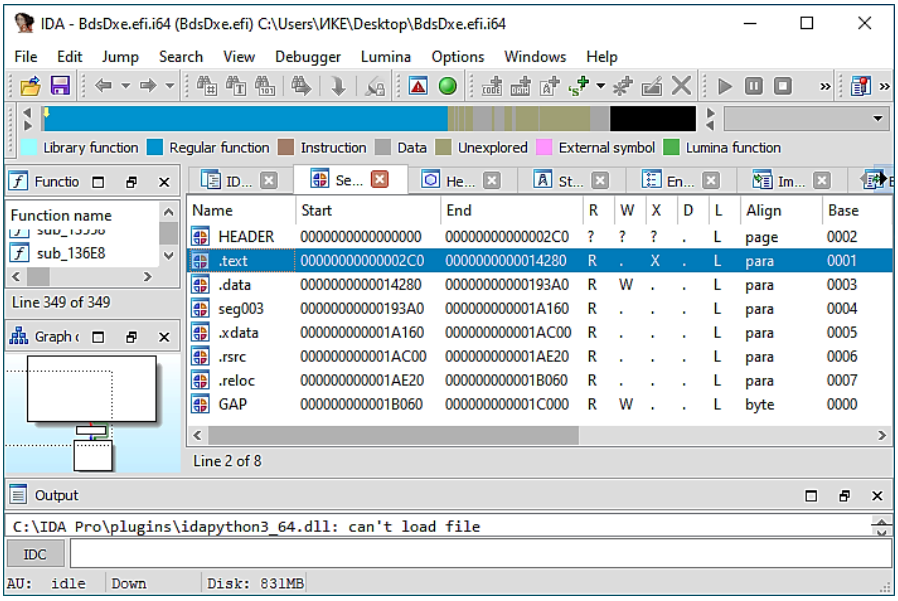

Рис. 8. Пример окна Segments утилиты IDA Pro

Fig. 8. Example of the IDA Pro Utility Segments Window

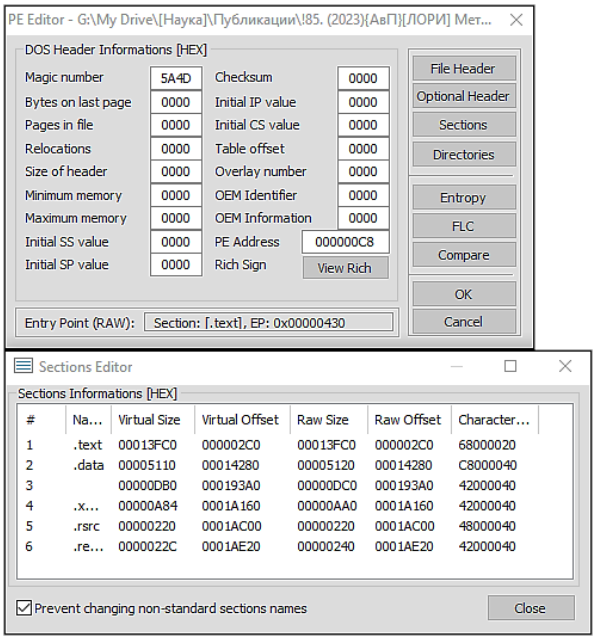

Рис. 9. Пример окон программы PE Tools

Fig. 9. Example of the PE Tools Utility Windows

Выделенные секции (называемые также сегментами) на рисунках 8 и 9 идентичны (за исключением специфичных для IDA Pro блоков данных HEADER и GAP, что подтверждает возможность применения обоих утилит.

Таким образом, после выполнения Шага 1.8 для анализа будут доступны программные секции, часть из которых содержит МК, а часть – используемые им данные, содержащие дополнительную информацию для работы шагов последующих этапов.

Заключение

Исследование посвящено построению единой Методологии проведения реверс-инжиниринга для машинного кода программ, основанной на существующих научных исследованиях предметной области и богатом авторском опыте. Обзор существующих работ показал практически полное отсутствие даже упоминания о каких-либо систематизированных и глубоко проработанных схемах в данной области. Как результат, предложена авторская методология, состоящая из 4-х этапов, шаги 1-го из которых описаны в первой (текущей) статье данного цикла, имеющая как схематичную форму, так и формализованную запись.

Новизной результата является системность подхода и масштаб охвата процесса по сравнению с существующим набором отдельно применяемых методов и средств, не всегда согласующихся друг с другом.

Теоретическая значимость результата заключается в создании полноценной схемы процесса (представляющей собой обобщенный алгоритм в виде операций и используемых данных), а практической – возможность реального применения от момента получения закрытого устройства до формирования итоговой документации с детальным описанием функционала машинного кода.

В следующих частях цикла статей будут расписаны шаги оставшихся 3-х этапов Методологии, сведенные также в единую таблицу с формой выполнения, а также входными и выходными информационными объектами.

Продолжение следует…

Список источников

1. Марков А.С., Цирлов В.Л. Опыт выявления уязвимостей в зарубежных программных продуктах // Вопросы кибербезопасности. 2013. № 1(1). С. 42–48.
2. Sabir U., Azam F., Haq S.U., Anwar M.W., Butt W.H., Amjad A. A Model Driven Reverse Engineering Framework for Generating High Level UML Models From Java Source Code // IEEE Access. 2019. Vol. 7. PP. 158931–158950. DOI:10.1109/ACCESS.2019.2950884
3. Баранова И.В., Батова М.М., Майоров С.В. Информационные инструменты реверсинжиниринга в стратегии деятельности инновационно-ориентированных структур // Теоретическая экономика. 2020. № 3(63). С. 28–35.
4. Передерий М.В. Реверс-инжиниринг в условиях инновационной инфраструктуры // Вестник Южно-Российского государственного технического университета (НПИ). Серия: Социально-экономические науки. 2015. № 5. С. 30–34.
5. Ивлиев Г.П., Эриванцева Т.Н. Патентная информация - источник ценных знаний для реинжиниринга // Право и цифровая экономика. 2022. № 3(17). С. 5–11. DOI:10.17803/2618-8198.2022.17.3.005-011
6. Нехорошев М.В. Методика реверс инжиниринга изделий в системе Siemens NX // Международная научно-техническая конференция «Проблемы и перспективы развития двигателестроения» (Самара, Россия, 23–25 июня 2021). Самара: Самарский национальный исследовательский университет имени академика С.П. Королева, 2021. Т. 1. С. 275–276.
7. Беляков А.А., Шулепов А.И. Проблемы практики реверс-инжиниринга космических аппаратов // Решетневские чтения: материалы XXV Международной научно-практической конференции, посвященной памяти генерального конструктора ракетно-космических систем академика М.Ф. Решетнева (Красноярск, Россия, 10–12 ноября 2021). Красноярск: Сибирский государственный университет науки и технологий имени академика М.Ф. Решетнева, 2021. Ч. 1. С. 8–9.
8. Штомпель И. Вакансия: реверс-инженер // Системный администратор. 2014. № 11(144). С. 85–87.
9. Kleber S., Maile L., Kargl F. Survey of Protocol Reverse Engineering Algorithms: Decomposition of Tools for Static Traffic Analysis // IEEE Communications Surveys & Tutorials. 2019. Vol. 21. Iss. 1. PP. 526–561. DOI:10.1109/COMST.2018.2867544
10. 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. P. 1335. DOI:10.3390/s22041335

11. Израилов К.Е. Концепция генетической декомпиляции машинного кода телекоммуникационных устройств // Труды учебных заведений связи. 2021. Т. 7. № 4. С. 95–109. DOI:10.31854/1813-324X-2021-7-4-95-109
12. 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
13. Долгова К.Н., Чернов А.В., Деревенец Е.О. Методы и алгоритмы восстановления программ на языке ассемблера в программы на языке высокого уровня // Проблемы информационной безопасности. Компьютерные системы. 2008. № 3. С. 54–68.
14. Израилов К.Е. Моделирование программы с уязвимостями с позиции эволюции ее представлений. Часть 1. Схема жизненного цикла // Труды учебных заведений связи. 2023. Т. 9. № 1. С. 75–93. DOI:10.31854/1813-324X-2023-9-1-75-93
15. Израилов К.Е. Моделирование программы с уязвимостями с позиции эволюции ее представлений. Часть 2. Аналитическая модель и эксперимент // Труды учебных заведений связи. 2023. Т. 9. № 2. С. 95–111. DOI:10.31854/1813-324X-2023-9-2-95-111
16. 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

References

1. Markov A., Tsirlov V. Experience in Identifying Vulnerabilities in Software. *Voprosy kiberbezopasnosti*. 2013;1(1):42–48.
2. Sabir U., Azam F., Haq S.U., Anwar M.W., Butt W.H., Amjad A. A Model Driven Reverse Engineering Framework for Generating High Level UML Models From Java Source Code. *IEEE Access.* 2019;7158931-158950. DOI:10.1109/ACCESS.2019.2950884
3. Baranova I.V., Batova M.M., Mayorov S.V. Reverse Engineering Information Tools in the Strategy of Innovative-Oriented Structures. *Teoreticheskaia ekonomika*. 2020;3(63):28–35.
4. Perederiy M.V. Reverse Engineering in the Conditions of Innovation Infrastructure. *Bulletin of the South-Russian State Technical University (NPI). Series: Socio-Economic Sciences*. 2015;5:30–34.
5. Ivliev G.P., Erivantseva T.N. Patent Information as a Source of Valuable Knowledge for Reengineering. *Law and Digital Economy*. 2022;3(17):5–11. DOI:10.17803/2618-8198.2022.17.3.005-011
6. Nekhoroshev M.V. Reverse Engineering of Products in Siemens NX. *Proceedings of the International Scientific and Technical Conference on Problems and Prospects of Engine Building Development, 23–25 June 2021, Samara, Russia, vol.1*. Samara: Samara National Research University Publ.; 2021. p.275–276.
7. Belyakov A.A., Shulepov A.I. Problems of Spacecraft Reverse Engineering Practice Proceedings of the XXV International Scientific and Practical Conference Dedicated to the Memory of the General Designer of Rocket and Space Systems Academician M.F. Reshetnev, 10–12 November 2021, Krasnoyarsk, Russia. Part 1. Krasnoyarsk: Siberian State University of Science and Technology Publ.; 2021. p.8–9.
8. Shtompel I. Vacancy: Reverse Engineer. *Sistemnyi administrator*. 2014;11(144):85–87.
9. Kleber S., Maile L., Kargl F. Survey of Protocol Reverse Engineering Algorithms: Decomposition of Tools for Static Traffic Analysis. *IEEE Communications Surveys & Tutorials*. 2019;21(1):526–561. DOI:10.1109/COMST.2018.2867544
10. 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
11. Izrailov K. The Genetic Decompilation Concept of the Telecommunication Devices Machine Code. *Proceedings of Telecommun. Univ*. 2021;7(4):95–109. DOI:10.31854/1813-324X-2021-7-4-95-109
12. 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;16(13):5111. DOI:10.3390/en16135111
13. Dolgova K.N., Chernov A.V., Derevenets E.O. Methods and Algorithms for Reconstructing Programs from Assembly to High Level Language. *Information Security Problems. Computer Systems*. 2008;3:54–68.
14. Izrailov K. Modeling a Program with Vulnerabilities in the Terms of Its Representations Evolution. Part 1. Life Cycle Scheme. *Proceedings of Telecommun. Univ*. 2023;9(1):75–93. DOI:10.31854/1813-324X-2023-9-1-75-93
15. Izrailov K. Modeling a Program with Vulnerabilities in the Terms of Its Representations Evolution. Part 2. Analytical Model and Experiment. *Proceedings of Telecommun. Univ*. 2023;9(2):95–111. DOI:10.31854/1813-324X-2023-9-2-95-111
16. 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

Статья поступила в редакцию 04.10.2023; одобрена после рецензирования 16.10.2023; принята к публикации 25.10.2023.
The article was submitted 04.10.2023; approved after reviewing 16.10.2023; accepted for publication 25.10.2023.

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