ISSN 2181-3187

АНАЛИЗ МЕТОДОВ ОБРАТНОГО ИНЖИНИРИНГА

Турсунов Отабек Одилжон ўғли

Аннотация: В работе посвящена изучению обратного инжиниринга в контексте обеспечения безопасности программного обеспечения. Рассматриваются угрозы, которые потенциально могут быть решены обратным инжинирингом. Исследуются истоки происхождения обратного инжиниринга и сфер его применения. Ключевая цель работы заключалась в комплексном изучении методов обратного инжиниринга, их классификации, особенностей применения и роли в анализе программного обеспечения.

Основная часть

Обратный инжиниринг (Reverse Engineering - RE) представляет собой процесс анализа уже существующей системы с целью выявления её конструкции, состава, принципов действия или алгоритмов функционирования. Чаще всего он применяется в тех случаях, когда отсутствует доступ к технической документации, исходному коду или другим формальным источникам информации о разрабатываемом объекте. Цель обратного инжиниринга может заключаться как в восстановлении утерянной информации, так и в обеспечении совместимости, устранении уязвимостей или усовершенствовании оригинального продукта.

Современное понимание обратного инжиниринга охватывает широкий спектр объектов: от физических компонентов и электронных устройств до программных систем, сетевых протоколов и биологических структур.

ISSN 2181-3187

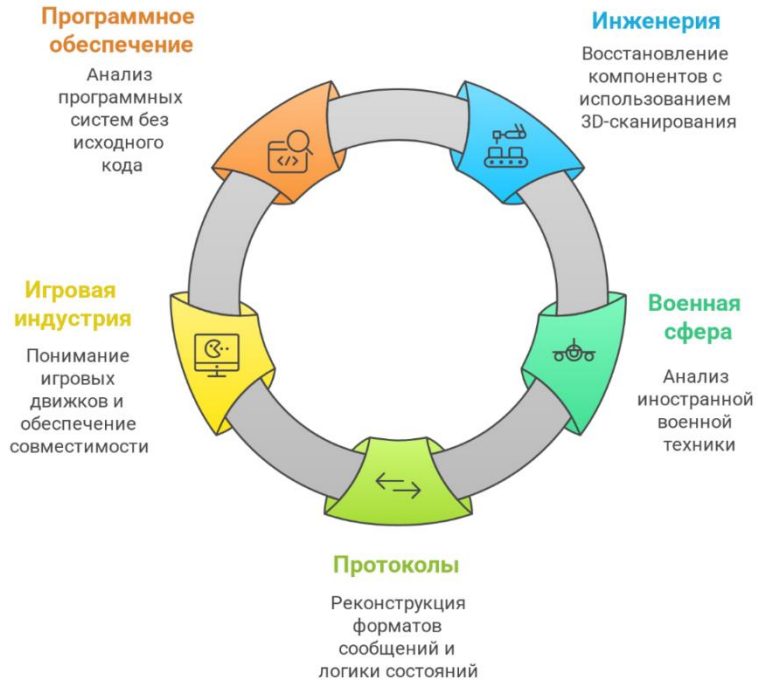

Рисунок 1.4 – обратный инжиниринг по отраслям

Процесс обратного инжиниринга обычно предполагает несколько последовательных этапов, направленных на всесторонний анализ объекта. Хотя конкретная методика может различаться в зависимости от предметной области, существует общий алгоритм, применимый к большинству случаев.

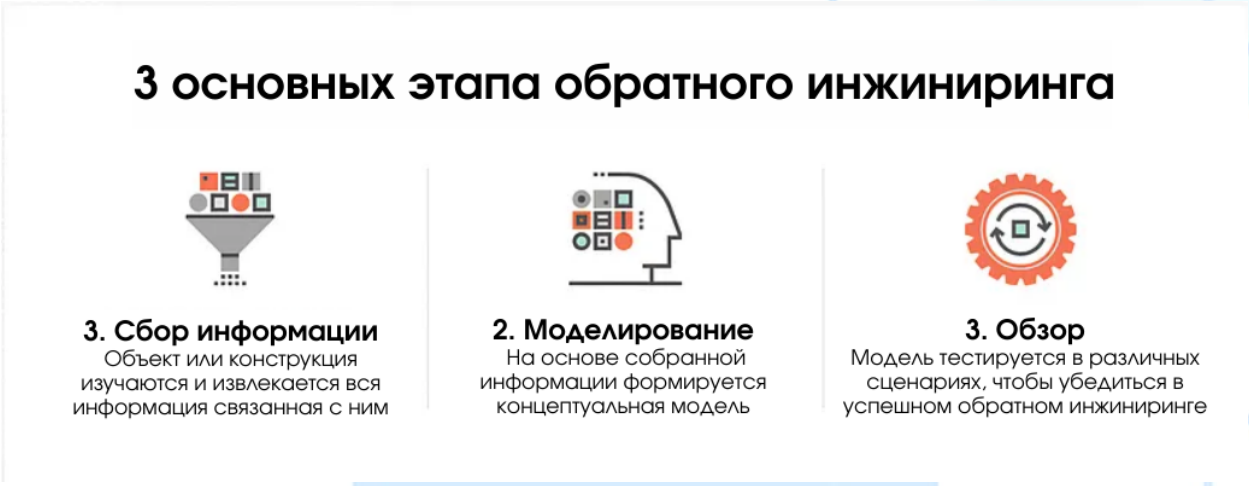

Рисунок 1.5 – этапы обратного инжиниринга

Как уже упоминалось ранее обратный инжиниринг является частью анализа программного обеспечения, применяемой в тех случаях, когда отсутствует исходный код. Его цель — восстановление логики и структуры работы программы на основе скомпилированного кода. Методы, используемые

ISSN 2181-3187

при этом, такие как дизассемблирование, декомпиляция и отладка, позволяют анализировать поведение программы с разных сторон: первые два относятся к статическому анализу, обеспечивая изучение кода без его выполнения, а отладка относится к динамическому анализу, позволяя наблюдать за программой в процессе её работ.

Методы дизассемблирования, декомпиляции и отладки позволяют извлекать техническую информацию из уже скомпилированного бинарного кода, именно поэтому данный процесс также называют бинарным обратным инжинирингом (binary reverse engineering). Хотя эти методы не предоставляют точную реконструкцию исходного кода, они дают возможность анализировать поведение программы, восстанавливать логические структуры и выявлять потенциальные уязвимости или вредоносные функции. Каждый из указанных методов предлагает собственный подход к анализу: дизассемблирование переводит машинный код в ассемблерные инструкции, декомпиляция восстанавливает приближённый к исходному код высокого уровня вид, а отладка позволяет отслеживать выполнение программы в реальном времени.

Эти методы не являются взаимоисключающими — напротив, в практике реверс-инжиниринга они часто применяются совместно, дополняя друг друга. Например, исследователь может использовать дизассемблирование для первичного изучения структуры кода, затем перейти к декомпиляции для лучшего понимания логики, а в завершение применить отладку для подтверждения своих гипотез во время исполнения программы.

Дизассемблирование представляет собой процесс преобразования машинного кода в язык ассемблера, выполняемый с помощью специальной программы — дизассемблера. Дизассемблер не восстанавливает исходный код в точности, а создает удобное для чтения человеком представление на низком уровне, ориентированное на последующий анализ. В результате дизассемблирования получается ассемблерный код, структурированный и удобный для восприятия человеком, но не обязательно пригодный для

ISSN 2181-3187

повторной сборки. Среди типичных сценариев использования можно выделить: восстановление утраченного исходного кода, поиск уязвимостей, анализ вредоносных программ, взлом программного обеспечения, анализ результата работы компилятора и его оптимизаций. Однако дизассемблирование имеет ряд ограничений. Ассемблерный код, получаемый из машинного, лишён комментариев, имён переменных, констант и других элементов, присутствующих в исходном коде.

Декомпиляция представляет собой один из ключевых методов обратного инжиниринга программного обеспечения, основной целью которого является преобразование исполняемого машинного кода обратно в представление, приближённое к исходному высокоуровневому языку программирования, например, C или Java. Это делает анализ кода более удобным, поскольку высокоуровневое представление ближе к привычным абстракциям и синтаксису, с которым работают разработчики. Программу, выполняющую процесс декомпиляции называют декомпайлером.

В отличие от дизассемблирования, где машинный код транслируется в ассемблерные инструкции, декомпиляция стремится восстановить логическую структуру исходной программы — переменные, управляющие конструкции, вызовы функций, циклы и условия. Таким образом, если дизассемблированный код отображает лишь техническую последовательность инструкций процессора, то декомпилированный код раскрывает семантику программы, что особенно ценно при анализе вредоносного ПО, утерянного исходного кода или при аудите безопасности.

Однако в процессе декомпиляции возникают определённые сложности. Во-первых, компиляция — это необратимый процесс, во время которого теряется множество исходных элементов: имена переменных, комментарии, структура классов и модулей. Во-вторых, компиляторы могут производить оптимизации, которые изменяют структуру программы, затрудняя восстановление изначального вида. Это делает декомпиляцию не точной

ISSN 2181-3187

реконструкцией, а попыткой восстановить максимально приближённую и читаемую версию исходного кода.

Отладка в контексте обратного инжиниринга — это процесс динамического анализа исполняемой программы путём пошагового выполнения, установки точек останова и наблюдения за изменениями состояния памяти, регистров и управления потоком. Целью отладки в этом случае является не исправление кода, а изучение внутренней логики работы, обнаружение скрытых механизмов, оценка защитных приёмов и взаимодействия с операционной системой или внешними библиотеками. Она позволяет исследовать поведение программ в реальном времени, особенно в условиях отсутствия исходного кода или при наличии техник защиты от статического анализа. Программы, при помощи которых выполняется отладка называют отладчикам. Большинство отладчиков работают с одним определенным языком программирования, однако есть и те, что поддерживают сразу несколько типов.

Таблица 1

Сравнение методов обратного инжиниринга

Критерий	Декомпиляция	Дизассемблирование	Отладка
Уровень абстракции	Высокий (похож на исходный код).	Низкий (инструкции процессора).	Низкий/средний (регистры, память, иногда псевдокод).
Тип анализа	Статический	Статический	Динамический
Выходной результат	Псевдокод	Ассемблерный код	Просмотр данных регистров и памяти
Сложность анализа	Средняя (псевдокод	Высокая (требует знания архитектуры).	Средняя (зависит от

ISSN 2181-3187

	проще, но не всегда точен).		инструмента и целей).
Преимущества	-Упрощает понимание логики. - Восстанавливает структуры (циклы, условия). - Подходит для сложных программ.	Точный (прямо отражает машинный код). - Работает с любыми файлами. - Быстрый анализ.	- Позволяет наблюдать поведение в реальном времени. - Обходит обфускацию. - Показывает динамические данные.
Недостатки	- Псевдокод может быть неточным. - Медленнее дизассемблирования.	- Трудно читать без опыта. - Нет высокоуровневой логики. - Требует ручного анализа.	- Требует выполнения (риск для вредоносного ПО).
Потребление ресурсов	Низкая (200–500 МБ RAM).	Средняя–высокая (500 МБ–2 ГБ, особенно Ghidra)	Средняя (300 МБ–1 ГБ, зависит от отладчика)

В настоящее время процессы отладки, декомпиляции и дизассемблирования происходят при помощи использования соответствующих программ, многие из которых совмещают в себе обе или даже все методы обратного инжиниринга. Поэтому также будет логично провести сравнение

ISSN 2181-3187

методов обратного инжиниринга на примере популярных программных продуктов.

Таблица 2

Сравнение средств обратного инжиниринга

Критерий	**Ghidra**	**IDA Free**	**Hopper**	**x64dbg**	**Radare2**
Дизассемблер	✓	Главная функция	Главная функция	✓	Главная функция
Декомпилятор	Главная функция	✓ Через облако, для x64	✓	✗	✓
Отладчик	✓	✓	✓	Главная функция	✓
Операционные системы	Windows, Linux, macOS	Windows, Linux, macOS	macOS, Linux	Windows	Windows, Linux, macOS
Архитектуры	x86, x64, ARM, MIPS, PowerPC	x86, x64, ARM, MIPS, PowerPC	x86, x64, ARM	x86, x64	x86, x64, ARM, MIPS, PowerPC
Имеет GUI	✓	✓	✓	✓	✓ Cutter
Open-source	✓	✓	✗	✓	✓

Заключение

При выборе метода средства для обратного инжиниринга специалисты чаще всего ориентируются на имеющиеся опыт и знания: у более продвинутых в приоритете могут стоять инструменты дизассемблирования, в то время как начинающие чаще предпочитают использовать декомпиляторы в связи с

ISSN 2181-3187

отсутствием знаний языка ассемблера. Однако, всё же редко возникают случаи, когда специалисты ограничиваются всего одним методом, так как применение всех их в совокупности гораздо облегчает и ускоряет задачу.

Список использованной литературы

1. “Reverse-engineering” by Ben Lutkevich // Techtarget URL: https://www.techtarget.com/searchsoftwarequality/definition/reverse-engineering

2. Dilshan de silva, Piyumika Samarasekara, Ridmi Hettiarachchi “A Comparative Analysis of Static and Dynamic Code Analysis Techniques” - SLIIT, 2023

3. Y. Tang, M. Zhou, E. Zussman and R. Caudill, "Disassembly modeling, planning and application: a review," Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation.

4. Ying Cao, Runze Zhang, Ruigang Liang, and Kai Chen. 2024. “Evaluating the Effectiveness of Decompilers”. In Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2024).