МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное

Учреждение высшего образования

**«Гжельский государственный университет»** (ГГУ)

Колледж ГГУ

Специальность 09.02.07 Информационные системы и программирования

**Реферат**

**По дисциплине «МДК 02.01. Моделирование и анализ программного обеспечения»**

**на тему «Дизассемблирование»**

ВЫПОЛНИЛА:

Студентка группы ИСП-О-17

Янгабозова Анастасия

ПРОВЕРИЛА:

Прокуронова А.Ю.

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

п. Электроизолятор

2019 г.

**Дизассемблирование** – преобразование программы на машинном языке к ее ассемблерному представлению. Декомпиляция – получение кода языка высокого уровня из программы на машинном языке или ассемблере.

**Декомпиляция** – достаточно сложный процесс. Это обусловлено следующими причинами:

− Процесс компиляции происходит с потерями. В машинном языке нет имен переменных и функций, и тип данных может быть определен только по производимым над ними операциям. Наблюдая пересылку 32-х бит данных, требуется значительная работа, чтобы определить, являются ли эти данные целым числом, дробью или указателем.

− Компиляция это операция типа множество-множество.Компиляция и декомпиляция могут быть выполнены множеством способов. Поэтому результат декомпиляции может значительно отличаться от исходного кода.

− Декомпиляторы в значительной степени зависимы от конкретного языка и библиотек. Обрабатывая исполняемый файл, созданный компилятором Delphi, декомпилятором, разработанным для C, можно получить фантастический результат.

− Необходимо точное дизассемблирование исполняемого файла. Любая ошибка или упущение на фазе дизассемблирования практически наверняка размножатся в результирующем коде.

**Зачем нужно дизассемблирование?** Цель инструментов дизассемблирования заключается в содействии исследованию функционирования программ, когда их исходные коды не доступны.

Наиболее распространенные цели дизассемблирования:

− анализ вредоносного программного обеспечения;

− анализ уязвимостей программного обеспечения с закрытым исходным кодом;

− анализ совместимости программного обеспечения с закрытым исходным кодом;

− валидация компилятора;

− отображение команд программы в процессе отладки.

**Базовый алгоритм дизассемблирования:**

Шаг 1. Первым шагом в процессе дизассемблирования является идентификация кодового сегмента. Так как команды обычно смешаны с данными, то дизассемблеру необходимо их разграничить.

Шаг 2. Получив адрес первой команды, необходимо прочитать значение, содержащееся по этому адресу (или смещению в файле) и выполнить табличное преобразование двоичного кода операции в соответствую ему мнемонику языка ассемблера.

Шаг 3. Как только команда была обнаружена и декодирована, ее ассемблерный эквивалент может быть добавлен к результирующему листингу. После этого необходимо выбрать одну из разновидностей синтаксиса языка ассемблера.

Шаг 4. Далее необходимо перейти к следующей команде и повторить предыдущие шаги до тех пор, пока каждая команда файла не будет дизассемблирована.

**Анализ вредоносного программного обеспечения.**

Разработчики вредоносного программного обеспечения вряд ли предоставят исходный код своего детища. А без доступа к исходному коду, будет довольно сложно определить поведение вредоносной программы. Существуют два основных метода исследования вредоносных программ - динамический и статический анализ. Динамический анализ заключается в исполнении программы в тщательно контролируемом окружении (песочнице) и записи каждого ее действия. Статический анализ основан на разборе исходного кода, который, в случае вредоносной программы, в основном состоит из дизассемблированных листингов.

**Анализ уязвимостей**.

Для простоты, можно разделить процесс аудита безопасности на три стадии: поиск уязвимостей, анализ уязвимостей, и разработка эксплойта. Одни и те же шаги предпринимаются вне зависимости от того, имеется ли у вас исходный код; однако, уровень трудоемкости резко возрастает, когда в вашем распоряжении есть лишь исполняемый файл. Первый шаг - исследование потенциально уязвимых условий в программе. Это зачастую достигается использованием динамических техник, таких как фаззинг, однако также может быть реализовано (обычно со значительно большими усилиями) посредством статического анализа. Как только проблема обнаружена, требуется определить, является ли она уязвимостью и если да, то при каких условиях.

**Листинг дизассемблирования** помогает понять, каким образом компилятор расположил переменные в памяти. Например, может быть полезно узнать, что объявленный программистом 70-байтный массив символов при распределении памяти компилятором был округлен в сторону 80 байт. Листинги дизассемблирования также предоставляют единственный способ понять, объявлены ли переменные глобально или внутри функций. Понимание реального расположения переменных в памяти жизненно важно при разработке эксплойтов.

**Анализ совместимости.**

Когда программы доступны только в виде исполняемых файлов, сторонним разработчикам крайне сложно обеспечить совместимость с ними своих программ, а также расширить их функциональность. Например, если производитель не предоставил драйвер для аппаратного устройства, то реверс инжиниринг – практически единственное средство для разработки альтернативных драйверов.

Валидация компилятора. Дизассемблирование может быть средством для проверки соответствия работы компилятора его спецификации. Также исследователя может заинтересовать наличие дополнительных возможностей, оптимизирующих результат компиляции. С точки зрения безопасности важно быть уверенным, что код, генерируемый компилятором, не содержит черных ходов.

**Отладка.**

К сожалению, дизассемблеры, встроенные в отладчики, зачастую малоэффективны (OllyDbg ─ исключение). Они неспособны к серийному дизассемблированию и иногда отказываются дизассемблировать, не будучи в состоянии определить границы функции. Поэтому, для лучшего контроля над процессом отладки, лучше использовать отладчик в сочетании с хорошим дизассемблером.

.