ОГЛАВЛЕНИЕ

Стр.

ГЛАВА 1

Общие сведения о работе отладчика

1.1 Принципы работы отладчика

Отладчик — это программа, которая упрощает разработку программного обеспечения, предоставляя разработчику способы поиска ошибок. Обычно функционал отладчика предоставляет следующие возможнсти:

- Поставить точку останова. Например, пометить интсрукцию, дойдя до которой, программа должна остановить свое выполнение и передать управление отладчику.
- Трассировать программу. То есть, последовательно выполнять инструкции, и после каждой останавливать выполнение программы и передавать управление отладчику.
- Отобразить состояние регисторов процессора на момент останова.
- Отобразить состояние стека процесса на момент останова.

Возможности предоставляемые отладчиком могут быть использованы злоумышленниками для изучения уязвимостей программного обеспечения и обхода ограничений и защиты. Для лучшего понимания способов защиты от отладчика рассмотрим, как работает каждая из предоставляемых им возможностей более подробно.

Отладчик может самостоятельно запустить отлаживаемый процесс. В рассматриваемой системе Microsoft Windows для этого нужно вызвать функцию `CreateProcess`, с указанием ей в качестве параметра `fdwCreate` константы `DEBUG_PROCESS`. И также отладчик может подключиться к уже работающему процессу. Для этого ему необходимо получить идентификатор процесса при помощи функции `OpenProcess`, после чего вызвать `DebugActiveProcess` и таким образом подключиться к нему. Обычно отладчик открывает процесс с доступом на чтение и запись в виртуальную память процесса.

Затем отладчик в цикле обрабатывает события отладки, используя функцию `WaitForDebugEvent`. После завершения обработки очередного события отладки вызывает функцию `ContinueDebugEvent`. Общую схему работу отладчика можно представить следующим образом:

```
CreateProcess("FileName.exe", ..., DEBUG_PROCESS, ...);
for (;;) {
  WaitForDebugEvent(&dbgEv, INFINITE);
  switch(dbgEv.dwDebugEventCode)
  {
  case EXCEPTION_DEBUG_EVENT:
  ...
  }
  ContinueDebugEvent( dbgEv.dwProcessId,
                      dbgEv.dwThreadId,
                      dwContinueStatus );
}
```

1.2 Трассировка

Трассировка — последовательное выполенение программы, при котором после каждой инсрукции управление передается отладчику. В этом режиме программист может детально отследить изменения значений всех параметров процесса. Обеспечение режима пошагового выполнения программы предусмотренно на аппаратном уровне.

В процессоре (**TODO** каком) есть регистр флагов (EFLAGS), который состоит из 32 бит, каждый из которых отображет состояние процессора (рис. 1.1).

Восьмой из них является флагом трассировки (trap flag). Если этот флаг равен 1, то процессор будет выполнять прерывание типа 1 после каждой инструкции. При выполнении прерывания 1 процессор выполняет передачу управления в обработчик прерывния, который помещает состояние всех ре-

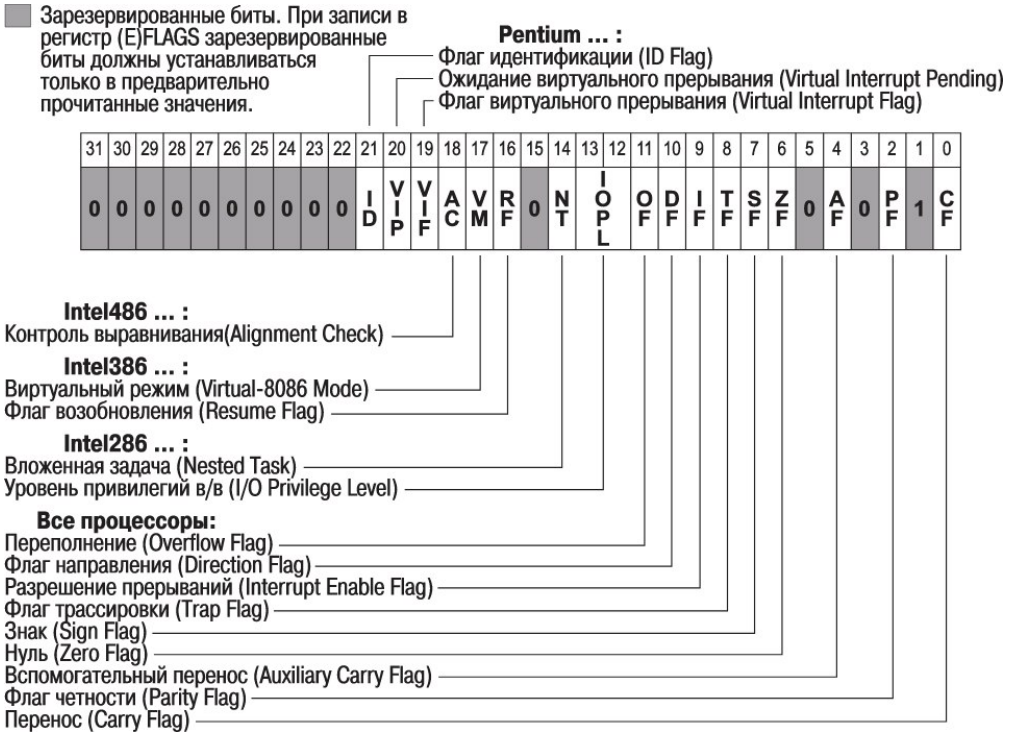

Рисунок 1.1 — Регистр флагов процессора

гистров процессора в стек после чего предает необходимую информацию отладчику. Обработчик прерывания в конце своей работы может либо оставить флаг трассировки равным 1, либо перевести его значение в 0.

Пример установки флага трассировки:

```
pushf                       ; Помещаем регистр флагов в стек
mov EBP, ESP                ; Сохраняем адрес вершины стека
or  WORD PTR[EBP], 0100h    ; Устанавливаем флаг TF
popf                        ; Восстанавливаем регистр флагов
```

Соответственно, чтобы снять флаг трассировки достаточно заменить инструкцию `OR` на инструкцию:

```
and  WORD PTR[EBP], FEFFh
```

Таким образом, можем заметить, что при проведении трассировки исходный код отлаживаемой программы никак не меняется.

1.3 Точки останова

Точка останова — место в коде программы, дойдя до которого процессор должен прервать выполнение программы и передать управление отладчику. После этого программист может просмотреть параметры состояния программы, поставить или убрать другие точки останова или запустить трассировку. Точки останова бывают двух типов: программные и аппаратные.

Рассмотрим принцип работы каждой из них.

1.3.1 Программные точки останова

Программные точки останова реализованы следующим образом. Когда программист ставит точку останова на какой-либо инструкции, отладчик запоминает данную инструкцию у себя в памяти, а затем заменяет данную инструкцию на

```
int 3 ; Генерация программного прерывания
```

Таким образом, когда процессор доходит до данной инструкции, возбуждается прерывание, управление переходит в обработчик прерывания, откуда затем информация передается в отладчик.

Инструкция `int 3` имеет специальный однобайтовый код операции (`0xCC`), в то время, как обычно прерывания имеют двухбайтовый код операции: `int x` $\rightarrow$ `0xCD x`. Это связано с тем, что может потребоваться заменить в коде однобайтовую операцию, например команду инкремента `inc`. В таком случае, если бы инструкция замены была больше одного байта, то повреждалась бы следующая инструкция. Это в свою очередь накладывает дополнительные расходы в ситуации, когда требуется из точки останова произвести трассировку.

Как видно, установка программной точки останова изменяет исходный код программы.

1.3.2 Аппаратные точки останова

В архитектуре x86 есть шесть регистров предназначенных специально для отладки. Именуются они DR0...DR7, при этом регистры DR4 и DR5 не используются. Данные регистры позволяют устанавливать точки останова с различными условиями. Также они ялвяются привилегированным ресурсом, следовательно инструкции, устанавливающие данные регистры, могут выполняться только с нолувого уровня защиты.

Регистры с DR0 по D3 содержат линейные адреса точек останова, каждая из которых связана с условием остановки. Условия определены в регистре DR7.

В регистре DR6 содержится статус отладки. Он позволяет отладчику ореде-лить, какие условия отладки возникли. Первые четыре бита указывают, какая из четырех точек останова в регистрах DR0...DR3 сработала. Бит 13 указывает, что следующая инструкция обращается к регистрам отладки. Бит 14 указывает на пошаговое выполнение (включает Trap Flag в регистре EFLAGS).

Регистр DR7 предназначен для управления процессом отладки, он позволяет выборочно включать условия остановки для точек останова в регистрах DR0...DR3. Есть два режима включения регистра: локальный (биты 0, 2, 4, 6) и глобальный (биты 1, 3, 5, 7). При локальном включении процессор сбрасывает условия остановки при каждом переключении задачи. При глобальном включении условия остановки не сбрасываются, а следовательно они используются для всех задач. Биты 17:16; 21:20; 25:24 и 29:28 позволяют установить следующие условия срабатывания точек останова:

- 00b — При выполнении инструкции.
- 01b — При записи данных.
- 10b — При обращении к порту ввода/вывода.
- 11b — Чтение и запись данных.

Как можно заметить, установка аппаратных точек останова никак не меняет исходный код программы.

1.4 Наиболее распространенные отладчики

Для лучшего противодействия отладчику полезно рассмотреть какие отладчики используются на сегодняшний день. Исходя из этой информации можно организовать простой способ защиты. Например, программа в процессе работы может просмотреть процессы запущенные в системе, и, если среди них встретится какой-либо знакомый отладчик, программа может изменить свое поведение или просто прервать выполнение.

1.4.1 SoftICE

Наиболее важным свойством отладчика SoftICE является то, что он работает на нулевом кольце защиты.

В архитектуе x86 есть три кольца защиты (ring-1, 2, 3). Данные кольца предназначены для ограничения взаимодействия выполняющихся программ между собой и с операционной системой. Как правило, на нулевом кольце защиты выполняется сама операционная система, которая одна имеет доступ ко всем привилегированным операциям. Рассматриваемый отладчик также располагается на нулевом кольце защиты, что позволяет ему отлаживать не только пользовательские приложения, но также драйвера и саму операционную систему.

Поддержка отладчика разработчиками прекратилась 11 июля 2007 года.

1.4.2 IDA Pro

IDA Pro — интерактивный дизассемблер и универсальный отладчик.

Дизассемблер — программный инструмент, позволяющий получить из машинного кода код на языке ассемблера. По принципу работы они делятся на пассивные и интерактивные. Автоматические предоставляют пользователю готовый листинг программы, а интерактивные позволяют на ходу изменять правила по которым производится трансляция.

Как дизассемблер IDA Pro способен создавать карты выполнения фрагментов программы, делая полученный код ассемблера еще более понятным человеком.В качестве отладчика IDA Pro охватывает все рассмотренные ранее возможности отладки, обеспечивает доступ ко всем сегментам пространства памяти процесса, а также обеспечивает подробную визуализацию.

Проект активно поддерживается и развивается.

1.4.3 WinDBG

WinDBG — отладчик предоставляемый фирмой Microsoft, предназначенный специально для работы в операционной среде Windows. Является более мощной альтернативой широко применяемому отадчику Visual Studio Debugger. Может использоваться как отладчик режима. Имеет поддержку сторонних расширений. На данный момент является одним из самых применяемых, благодаря своей универсальности.

1.4.4 OllyDbg

OllyDbg — отладчик уровня приложений (ring-3). Помимо поддержки всех рассматриваемых ранее возможностей отладки, большая часть мощности OllyDbg заключается в расширениях, которые разрабатывают пользователи этого отладчика и выкладывают в сеть Internet. OllyDbg имеет бесплатное распространение, а также не требует установки.

Для тестирования разработанных методов в данной работе применяется именно этот отладчик.

1.5 PE формат

Исполняемые файлы в системе Windows имеет общую сигнатуру, называемую PE (Portable Executable). В PE формате содержится различная информация о исполняемом файле, как например: таблицы импорта и экспорта,

информация о различных секциях, точка входа и так далее. Структура PE формата представлена на рисунке 1.2.

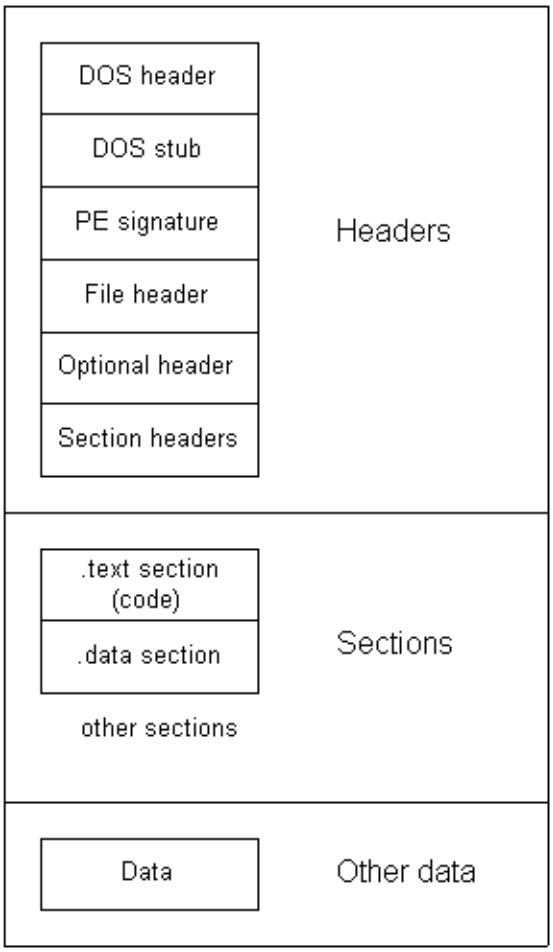

Рисунок 1.2 — Формат PE

Рассмотрим те части PE формата, которые будут использованы в данной работе. Первым идет MS-DOS заголовок, первыми двумя байтами которой является сигнатура "MZ". Большая часть полей данного заголовка предназаначены для запуска из-под DOS. Для дальнейшей работы, кроме проверки сигнатуры, потребуется поле `e_lfanew`, в котором содержится указатель на начало PE-заголовока.

PE-заголовок также имеет свою сигнатуру, которую необходимо проверить на корректность, а именно четыре байта "PE\0\0". В данном разделе хранится количество секций `NumberOfSection`, а также размер идущего за следующим опционального заголовка в байтах `SizeOfOptionalHeader`.

Далле расположен опциональный заголовок. Данный раздел содежит сме-

щение адреса входа относительно базового адреса загрузки файла — AddresOfEntryP
рекомендуемый базовый адрес загрузки файла ImageBase, количество элементов в таблице DATA_DIRECTORY — NumberOfRvaAndSizes. DATA_DIRECTORY представляет собой таблицу, каждый элемент которой представляет из себя структуру из двух полей, а именно виртульного адреса тех данных, на который указывает данный элемент, и их размер.

Далее идет таблица секций. Для каждой секции в этой таблице приведена следующая информация:

- Name — имя секции.
- VirtualAddress — виртуальный адрес секции в памяти.
- PointerToRawData — указатель на данные в файле.
- VirtualSize — размер, занимаемый секцией в памяти.
- Characteristics — свойства секции. Так, например, секция, которую можно запустить на выполнение должна обладать свойством IMAGE_SCN_CNT_CO