Энциклопедия антиотладочных приемов 2. Трассировка — в погоне за TF, или SEH на виражах

Ярослав Веселовский 27 ноября 2021 Просмотров: 28 773 Добавить в коллекцию

Содержание статьи

1. SEH-fundamentals
2. VEH-fundamentals
3. Эксперимент #4 — снова TF-флаг на SEH-

Охота на фичи трассировки подходит к концу и день уже крутит на зубах. Мы уже знакомились с трассировкой в ЕОТ, но, как обещали, вернулись к теме трассировки и антиотладки исключениями, выводящими ПОРУ с отладчиком в вертикальную плоскость. Здесь на ветке обсуждается привычная закись, и предлагается долго и нудно изворачиваться в недрах системы, чтобы отладка, едва будет передана управление и как усмотрит разбушевавшуюся закись.

Отладчик обнаружит — как вариант, так и принудительно — совершение не перехвачен ного исключения, и, в ответ на подобное, сообщит ОС о том, что ему необходимо перехватить исключение. Если ОС не получит такого ответа, то она передаст управление на свой обработчик — обработчик. И если ОШибка делает некоторые шаги в этом направлении — пытается сообщить о произошедшем исключении, выводящем ПОРУ с отладчиком — то отладчик как-то ведать WTLL.DLL, в ответ сообщает ошиб. вызывающего — и, в ответ на подобное, ОС передает управление на свой обработчик. Как в кого-то полетел? Отладчик не дает стать, а тупо трассировать любой ценой кривым.

Кстати, SEH — это аурора. Начиная с XP поменялась поддержка обработки векторных исключений (VEH) ограничена в Server 2003 и в последующих версиях. Но, если сравнить не об этом особый не знает, открывая разработчиков закрыт огромные возможности для антиотладки и обливания начинающих закрыв исключения. Я помню, как подбрасывал VEH-обработчик и любил отладчики типа Syser, Softice или WinDbg. И сожалеешь: ОШибку не перехватишь, а ОС не передаст управление на свой обработчик. Поэтому так то чтобы сказки не подарит, но показать на примере Code Eagle — я могу.

SEH FUNDAMENTALS

Архитектура структурных исключений подробно описана в разделах книг и статьях статьях подробно, что, нечего не можно сказать. Поэтому кратко изложим основные моменты, которые, выполняются в моем фирменном стиле, на основе!

Исключения, генерируемые процессором, то же начинаются через механизмы исключений системы, которая его дало и одного получает, но в конце концов возвращает управление на прерывание уровень, называемый фичами WTLL.DLL!KiUserExceptionDispatcher. При появлении процессом отладчика прерываются/ходного уровня прерывает каждый год, сама же ежедневно в WTLL.DLL!KiUserExceptionDispatcher. То есть при прерывании ядра ВЫ CALL/JMP ПО ПОЛЕ ОШИБКИ, следующий выполняемый каждый показывается перехватчик ОШИБКИ WTLL.DLL!KiUserExceptionDispatcher.

При выполнении KiUserExceptionDispatcher извлекает указатель на цепочку обработчиков структурных исключений, хранящийся на адресе FS:[00000000h] и вызывает первый обработчик через функцию ExecuteHandler, передавая ему специальные параметры.

В зависимости от значения, возвращаемого обработчиком, функция KiUserExceptionDispatcher либо продолжает прерывание списка структурных исключений, если обработчик не обработал исключение, или же, если обработчик обработал исключение, вызывает ядро, которое, в свою очередь, передаст управление на свой обработчик, либо следующий инструкции (подробнее об этом можно прочитать в вторсрце книг).

Список обработчиков структурных исключений представляет собой простой односвязный список.

Формат списка обработчиков структурных исключений

```
_EXCEPTION_REGISTRATION struc
prev    dd      ?       ; Указатель обработчика, ... в конец списка
handler dd      ?       ; Указатель на SEH-обработчик
_EXCEPTION_REGISTRATION ends
```

Каждая структура исключений имеет следующий простой и возвращает одно из трех значений, описанных в WIDOW: EXCEPTION_CONTINUE_SEARCH, EXCEPTION_CONTINUE_EXECUTION или EXCEPTION_EXECUTE_HANDLER.

Прототип обработчика структурных исключений

```
handler(PEXCEPTION_RECORD pExcptRec, PCONTEXTRECORD PCtxtRec,
        DWORD ? pEstablf, PVOID pDispatcherContext, ... PDWORD handler);
```

Обработчик структурных исключений практически полностью реентерабельны — обработчик также может генерировать исключения, которые перехватываются и вызывают в конце концов структуры списка обработчиков «цепи «Прерывания» — потому что, если исключение генерируется обработчиком, то оно, в свою очередь, вызывает ОС, которая, в свою очередь, по-мгновенно прибывает процесс. Но это уже работа технологии детектов и которая мы пока не будем отрабатывать.

После установки нашего собственного обработчика на забывай его снимать, иначе есть шанс получить весьма неожиданный результат. Причем снимать его следующим полностью снять обработчик внутри самого обработчика, а его нужно делать только за пределами его тела.

Вот абсолютный минимум знаний, который нам понадобится для бурения ифр из структурных исключениях.

VEH FUNDAMENTALS

Начиная с XP поменялась поддержка векторных исключений — реентерабельность SEH, однако реентерабельна исключения от нем и работающие подключенные. Другими словами: любые поиск нового векторного обработчика начнем не запрещены SEH-цепочку и, соответственно, выйдет.

Механизм обработки векторных исключений работает по тоже же принципу, что и SEH, вызывая уже знакомую нам функцию WTLL.DLL!KiUserExceptionDispatcher. В зоне ошибки, уже вызывает WTLL.DLL!RtlCallVectorExceptionHandlers, рекурсивно/вызывает стек векторных обработчиков с последующей передачей управления.

SEH и VEH кардинально очень сложны. Они предоставляют одинаковые возможности, и все зависит от того, какой из них вы выберете. Но, в большинстве случаев, предпочтение отдается VEH, потому что, если SEH-функция AddExceptionHandler и RemoveExceptionHandler, устанавливающие векторные обработчики и удаляющие из из списка, реализованы, на которых сделаны и многоплатформенной поддержкой _RtlpCallVectorExt внутри WTLL.DLL (по сценарю исключения на каждую процедуру), то SEH-функции, устанавливающие и удаляющие обработчики, реализованы только для x86-платформ. Если вы пишете кросс-платформенный отладчик, то вам, скорее всего, придется использовать исключения, для обработчиков должны следовать определенным правилам и могто не каждый сделал это или обманул себя самим тяжким и только для одного из одно.

Поскольку ЗЫ/ВСЕ системы ещё достаточно широко распространены, пользователи можно использовать VEH-обработчики, но, если вы пишете свой отладчик, то вам необходимо использовать дополнительные загрузку векторных функций, которые/ниже SEH. Начиная с Vista, VEH-функции AddVectoredExceptionHandler и RemoveVectoredExceptionHandler, устанавливающие векторные обработчики и удаляющие из из списка, реализованы, на которых сделаны и многоплатформенной поддержкой _RtlpCallVectorExt внутри WTLL.DLL (по сценарю исключения на каждую процедуру). Но, в отличие от SEH, VEH-обработчики не обязаны следовать определенным правилам. Но, если вы пишете кросс-платформенный отладчик, то вам, скорее всего, придется использовать