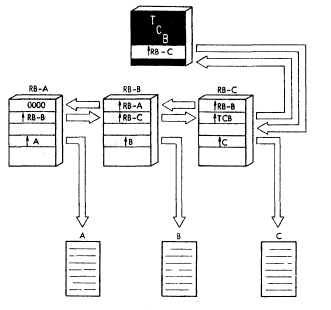
Summarize. Методы решения задачи про модули.

Обозначения: ЧП - чужая программа (исследуемая программа). Желтым выделены дополнения к документу. Номер в скобочках - номер "сессии" исправлений (1)

**Ростислав Ефремов. "Испорченный адрес"**

**Кратко.** Идея решения заключается в том, чтобы испортить адрес вызываемого модуля, и перехватывать ESTAE рутиной возникающие ошибки.

**Подробно.**При динамическом переходе адрес модуля, в который мы будем переходить, так или иначе программа получает через вызов макрокоманды **LOAD**. В ином случае, можно использовать макрокоманду **LINK** (переход с возвратом) или макрокоманд у **XCTL** (переход без возврата). Все три макрокоманды создают (или используют уже созданную) особую структуру для модуля в памяти: **LPRB** или **LRB**. Эта структура находится в очереди, прикрепленной к TCB в нашем address space(рис. 1). В ней находится адрес модуля в address space, его размер, точка входа, ссылка на следующий **LPRB**/**LRB**.

Рисунок

Теперь хочется прибавить 1 к адресам всех модулей, чтобы при переходе по адресу случался **ABEND S0C6** (invalid boundary). Приведу немного системной информации о соответствии SVC разным макрокомандам.

**LINK ~ SVC 6**

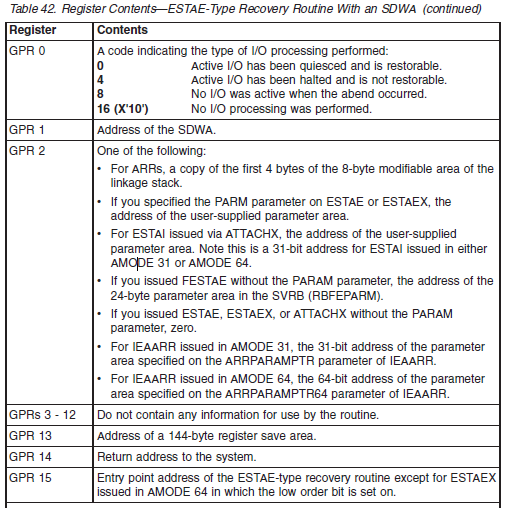
**XCTL ~ SVC 7**

**LOAD ~ SVC 8**

Сначала надо получить местоположение всех модулей для исправления. Как это сделать - еще нерешенный вопрос.(1)

Очевидно, надо перехватить SVC 8, чтобы выдавать некорректный адрес на точку входа в модуль. То есть надо будет вызвать старый SVC, затем испортить выдаваемый им адрес (прибавить единичку, чтобы он стал нечетным) и вернуть его как ни в чем не бывало. Перехватить или задать пользовательские SVC с номерами 0-199 нельзя. Что же делать? А давайте исправим в исследуемой программе все вызовы SVC 6, SVC 7, SVC 8 на SVC 216, SVC 217, SVC 218 и зададим пользовательские SVC с помощью макрокоманды SVCUPDTE. SVC 218 будет вызывать SVC 8 и прибавлять единичку к точке входа в модуль. SVC 216 и SVC 217 будут записывать в таблицу трассировки сообщение о переходе в другой модуль (путем вызова ABEND, чтобы попасть в нашу ESTAE рутину, описанную дальше) и продолжать исполнение SVC 6 и SVC 7.

Так же надо задать ESTAE рутину, которая будет ловить ABEND S0C6, так же делать записи в таблицу трассировки и восстанавливать исполнение. Адрес таблицы будем передавать через user-supplied parameter area, который задается при вызове макрокоманды ESTAE для создания обработчика ошибок.



Нам в ESTAE рутину поставляется целый SDWA, который содержит регистры на момент ошибки, а так же PSW. Адрес на момент вылета хранится в SDWATRAN, SDWAMOD1, SDWAPMKA (в SDWAPMKA надо занулить первый байт так как сам адрес не выровнен по слову, а байтовое поле перед ним как раз имеет выравнивание "слово"). Опытным путем было установлено, что если создавать ошибку нарушением выравнивания адреса модуля, ошибка случается не на команде перехода, а в точке назначения перехода. Так что выяснить "точку выхода" не получится, только "точку входа" и общую последовательность вызовов между модулями.

Дальше, когда по этому некорректному адресу будет попытка совершить переход, случится ABEND SC06, который ловится ESTAE рутиной (возможно ESPIE, но ее возможности еще не изучены). ABEND ловится, наш обработчик пишет в таблицу трассировки событие, задает параметры для восстановления исполнения с помощью SETRP и делает retry.

ERRC EQU \*

STM R14,R12,12(R13)

LR R3,R1

USING SDWA,R3

\*make trace record

L R4,SDWAPMKA

NIHH R4,X'0000'

SETRP RC=4,RETREGS=YES,WKAREA=(R3),FRESDWA=YES,RETADDR=(R4), X

REMREC=YES

LM R14,R12,12(R13)

BR R14

**Общий алгоритм:**

1. Загрузить все свои модули
2. С помощью SuperZAP заменить все вхождения SVC 6,SVC 7,SVC 8 на SVC 216,SVC 217,SVC 218 в ЧП
3. Загрузить ЧП в память.
4. Перехватить SVC 216,SVC 217,SVC 218, а так же ESTAE с параметром XCTL=YES, и user-supplied-area с адресом таблицы трассировки
5. Исполнить ЧП
6. Сохранить таблицу трассировки в датасет
7. Отменить все перехваты, и завершиться

**Алгоритм ESTAE рутины:**

1. Определить откуда мы попали в ESTAE: из SVC или из кода ЧП
2. Сделать запись в таблицу трассировки
3. Восстановить исполнение

**Алгоритм SVC 218:**

1. Вызвать SVC 8
2. Прибавить 1 к R0

**Алгоритм SVC 216, SVC 217:**

1. Сделать ABEND
2. Вызвать SVC 6/SVC7

|  |  |
| --- | --- |
| Преимущества | Недостатки |
| В код не вносятся ошибки | Невозможно определить "точку выхода" из модуля |
| Все адреса переходов определяются динамически | Необходимо перехватывать SVC |
|  | Необходимо использовать SuperZAP |
|  | Использование ESTAE плохо влияет на скорость работы. |
|  | Требуется синтаксический анализ |
|  | Чтобы определить имя след. модуля, придется искать диапазон адресов, куда мы попадем (1) |

**Вояковская Наталья Николаевна. "Испорченный branch"**

**Кратко.** Идея решения заключается в том, чтобы испортить коды команд перехода и коды SVC 6 и SVC 7.

**Подробно.** Сначала статически строим таблицу бранчей - мест, где мы будем падать. Отвечает за это модуль статического анализа листнинга.

|  |  |  |
| --- | --- | --- |
| Команда | Адрес | Тип |
| BR 2 | 4 | 0 |
| ... |  |  |

На основе таблицы, генерируем данные для программы SuperZAP и с ее помощью исправляем коды команд перехода и вызова SVC, чтобы программа на них падала с ABEND S0C1. Для этого надо получить местоположение всех модулей для исправления. Как это сделать - еще нерешенный вопрос.(1)

Построение списка модулей изначально предполагалось задачей статического анализа. Однако возможны неприятные моменты в духе: адрес DCB передается в регистре. В таком случае статический анализ становится весьма сложным в реализации. Придется обратиться к динамическому.

Новая идея состоит в том, что модуль будет обрабатываться ровно перед тем как его загрузят.

Изначально нам известен ровно один модуль ЧП - главный модуль. В нем находятся какие-то SVC 6, SVC 7, SVC 8. Давайте их испортим, чтобы динамически, обрабатывая ошибку, получить имена модулей и обработать их перед загрузкой.(2)

Обрабатывать ABEND будет ESTAE (или ESPIE) рутина. Обработка будет заключаться в том чтобы сделать запись в таблицу трассировки, проверив принадлежность адреса одному из модулей. И восстановить исполнение. Адрес точки выхода получается вычитанием длины команды перехода из PSW, предоставленного ESTAE рутине.

Если упала команда бранчирования, точку выхода мы можем отыскать в psw, сделав EXECUTE команды. Если упала команда SVC 6 или SVC 7, точку выхода можно найти в заранее статически построенной таблице (будем вычленять EP из XCTL и LINK). Однако возможен неприятный случай, когда точка входа в модуль передается неявно через параметр DE в LINK или XCTL. В таком случае придется усложнить синтаксический анализ для отслеживания вызовов макрокоманды BLDL, которая генерирует информацию для поля DE.

**Общий алгоритм:**

1. Построить таблицу инструкции для главного модуля ЧП, которые нас интересуют с помощью статического анализа.
2. Сгенерировать таблицу для SuperZAP
3. Испортить инструкции программой SuperZAP
4. Установить ESTAE рутину для перехвата ABEND S0C1
5. Вызвать ЧП

**Алгоритм ESTAE рутины:**

1. Если мы упали в SVC, то надо проверить, есть ли модуль, связанный с командой в нашей таблице (параметры svc находим по адресу из 15 регистра на момент падения). Если нет - обработать его (статический анализ+SuperZAP) и добавить в таблицу.
2. Сделать запись в таблице трассировки, указанную в user-supplied area
3. Определяет какая команда должна была быть исполнена
4. Восстанавливает окружение на момент падения
5. Исполняет оригинальный вариант испорченной команды с помощью EXECUTE
6. Исправляет окружение из SDWA и делает retry в результат исполнения оригинальной команды.

|  |  |
| --- | --- |
| Преимущества | Недостатки |
| Ловятся "точки выхода" | Требуется синтаксический анализ |
|  | Проблема с определением точки входа, когда она генерируется BLDL |
|  | Необходимо использовать SuperZAP |
|  | Использование ESTAE плохо влияет на скорость работы. |
|  | В код вносятся ошибки |
|  | Чтобы определить имя след. модуля для BR или подобной команды, придется искать диапазон адресов, куда мы попадем (1) |

**Александр Щербаков. "SLIP TRAP"**

**Кратко:** Загружаем чужую программу в память, на все ее модули кидаем команду "SLIP" и ловим Branch Entry и SVC entry в GTF

**Подробно:** Заранее загружаем все модули чужой программы в память, кидаем на них SLIP TRAP. Исполняем ЧП. Анализируем записи в GTF. Все LOAD, LINK, XCTL грузят модули в родной address space, поэтому slip trap легко до них дотягивается, и можно проверить входит ли адрес бранчирования в определенный диапазон адресов или нет.

В процессе проверка возможностей Slip Trap по авторазметке памяти.(1)

**Общий алгоритм:**

1. Загружаем все модули ЧП в память
2. Генерируем команду SLIP
3. Отправляем ее в консоль и анализируем отклик
4. Исполняем ЧП
5. Анализ R15
6. Выборка и запись в датасет нужных записей о бранчировании или вызове SVC из General Trace Facility.

|  |  |
| --- | --- |
| Преимущества | Недостатки |
| Не требуется синтаксический анализ. | Возможно, SLIP TRAP будет работать медленнее ESTAE (или ESPIE) рутин. |
| Все отслеживается динамически | Чтобы определить цепочку модулей, нужно будет проверять принадлежность адресов разным диапазонам RB (1) |
| Ловятся "точки выхода" |  |
| В код не вносятся ошибка |  |
| Это просто |  |
| Не нужен список модулей?(1) |  |

**Что у нас есть:**

В активе: переход в режим супервизора; Dynamic Allocation; программа, которая умеет читать из MCS консоли; программа, которая вызывает SuperZAP; программа с ESTAE рутиной, которая умеет делать retry.

В процессе: синтаксический анализ.

Возможно понадобится: использование EXECUTE и LOAD PSW/STORE PSW; динамический перехват пользовательских SVC

Надо проверить: работу SLIP TRAP под Hercules, прокидывание DD имен нашей программы в ЧП