**Вояковская Наталья Николаевна (общая концепция). Ефремов Ростислав(конкретизация тех. деталей и динамический поиск модулей). Сапурина Лилия (статический анализ). "Испорченный branch"**

**Кратко.** Идея решения заключается в том, чтобы испортить коды команд перехода и коды SVC 6, SVC 7 SVC 8.

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

|  |  |  |
| --- | --- | --- |
| Команда | Адрес | Машинный код полностью |
| BR 2 | 4 | ... |
| ... |  |  |

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.

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.

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 упавшей команды перехода из PSW, предоставленного ESTAE рутине.

Адрес точки входа можно найти либо в одном из регистров на момент перехода, либо в списке параметров для SVC (либо точка входа по умолчанию берется из RB для соотв. модуля).

Есть еще проблема. ESTAE рутины хранятся в "стеке". То есть когда случается ошибка - вызывается ESTAE рутина, которая позже всех попала в стек. Очевидно, если ЧП заведет свою ESTAE рутину для S0C1 то она выполнится раньше нашей, а наша может вообще не выполниться (если ESTAE ЧП сделает rerty). Таким образом надо исправить в ЧП еще все svc которые отвечают за перехват ESTAE на наш svc в котором мы снимает нашу ESTAE рутину, котом активируем ESTAE ЧП а потом активируем нашу ESTAE. Мы остаемся на вершине стека.

**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. Исправляет окружение из SDWA и делает retry.

**Алгоритм retry рутины:**

1. если мы упали в команде, передающей исполнение без возврата - просто прыжок
2. если мы упали в команде, которая сохраняет psw - сохраняем psw который надо вручную и прыгаем по BR
3. если упали в link или load то надо вызвать link/load и вернуться в ЧП

|  |  |
| --- | --- |
| 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) |

**Очень подробно.**

* Постановка задачи

Необходимо построить граф переходов между модулями в результате динамического анализа исполнения ЧП.

Иными словами необходимо построить таблицу вида

|  |  |  |  |
| --- | --- | --- | --- |
| Имя вызывающего модуля | Адрес точки перехода | Имя вызванного модуля | Адрес точки назначения |
| OLOLOSHA1 | 0B10 | MAMANYA | 0BC4 |
| ... |  |  |  |

С помощью анализа исполнения некоего ЧП.

0)Пересоздать **RSA** (адрес **RSA** виден из **ЧП** в R1 после **LINK** и **LOAD**, если оно туда полезет - надо дать по рукам), очередь удаления

0.5) Поискать упавшую команду в RECB (хеш рутина...)

1)Если мы упали в каком-то **SVC**

1.1)Получить информацию из области параметров, на которую ссылается 15 регистр

1.2)Проверить **table of prepared modules** на наличие модуля связанного с командой

1.3)Если модуля нет то

2)Если команда не **SVC 6** - cделать запись в таблице трассировки, указанную в **USA**

3)Установим адрес Retry routine в зависимости от команды, которую исправляем

4) в **RSA** сохраним PSW из **SDWA**, список регистров, оригинал команды, переключим CRECB и передадим RSA через R3 в retry routine

5)SETRP и retry

**ESTAE routine**

IN: USA R0; OUT: RTOPM addr R0

1)Добавление в table of prepared modules

1)Статический анализ

2)Заполнение поля "current module name"

4)Генерация таблицы для SuperZAP

5)Вызов SuperZAP

**Препроцессинг модуля**

1)Чтение SYSIN и задание ICB, имен целевых библиотек, модуля, датасета для вывода трассировочной таблицы

**Разбор параметров**

1)Выделить память подвсе структуры)Заполнить **ICB** и **USA**

3)Разбор параметров из SYSIN

3)Препроцессинг главного модуля **ЧП**

4)Перехват **ESTAE**

5)Загрузка главного модуля **ЧП** в память

6)Исполнение **ЧП**

7)Анализ кода завершения и сохранение **trace table** на диск

**Инициализация**

1) Создание на основе RECB и имени модуля, датасета для SuperZAP, в котором будет таблица исправляемых команд. Возвращает DDNAME к этому датасету.

**Генерация таблицы для SuperZAP**

1) Восстановление регистров

2) Прыжок в целевой адрес испорченного перехода

**Resuming Execution - RESEX(генерируется retry routine)**

IN:USA RO

RECB ADDR R1

IN:SAVEAREA R1

IN:SAVEAREA R1

RSA R0

1) Если это BR, BALR, BASR, B, BAL и т.д

1.1)Восстановить флаги ЧП (с помощью LOADPSW) (если BASR - сконструировать адрес возврата)

1.2)Сделать область памяти под storage key ЧП и перейти в storage key 0, поместить область в очередь на удаление

1.3)Скопировать в нее данные для восстановления регистров и перейти в storage key ЧП

1.4)Сконструировать RESEX и прыгнуть в него

2) Если это LOAD или LINK или ATTACH

2.1)Восстановить R0,R1,R15, восстановить флаги PSW и вызвать LOAD

2.2)перейти в storage key 0, поместить в R14 адрес возврата, восстановить R2,...R13, перейти в storage key ЧП и сделать BR14

3)Если это XCTL

3.1)Восстановить регистры R0,R1,R15, оттранслировать параметры XCTL в параметры для LOAD и CALL

3.2)Восстановить флаги PSW, вызвать LOAD, создать область памяти под регистры под ключом ЧП

3.3)перейти в storage key 0, заполнить регистры в соответствии со спецификацией XCTL, сконструировать RESEX, делающий CALL целевой метки, перейти в storage key ЧП, и прыгнуть в RESEX

RESEX конструируем под storage key ЧП!!!

добавить ATTACH

**Retry routine**

USA specified in param for estae R2

IN: USA R0 RTOPM R1;

IN:RECB R0

OUT:DDNAME R0

1) заполнение в RECB поля "offset" и "instruction"

**Статический анализ**

IN: USA R0 RTOPM R1;

LOAD

BR

LINK

LOAD

XCTL

**ЧП**

Теперь, когда очевидна схема работы программы и структуры данных, необходимо конкретизировать вопрос режимов работы. Программа может работать в разных режимах адресации. Программа может быть по-разному авторизована. И режим авторизации и режим адресации определяются значениями в PSW. Ровно по этой причине, при возвращении из retry routine необходимо делать load PSW или как минимум восстанавливать системные флаги. Важно, что придется по-разному обрабатывать ситуации, когда PSW для 64/31/24-битного режима.

Теперь почему нельзя прямо в ESTAE рутине сделать EXECUTE. ESTAE рутина требует возвращения в систему: выбора retry или percolate. Вызов EXECUTE бранча из рутины может привести к непредсказуемым последствиям.

Теперь опишем структуры данных. length - ВСЯ длина. POS - позиция последнего элемента списка относительно начала блока. LENGTH И POS находятся в заголовке. Кроме того, в заголовке может находиться поле NEXT, указывающее не следующий CB того же типа в списке.

CBHEAD DSECT

LENGTH DS F

POS DS F

CBHEAD# EQU \*-LENGTH

NEXT@ DS F

ID@ DS F

CBHEADE# EQU \*-LENGTH

1. **CBHEADER**

length

**CBHEADER**

position

next control block

id of block in TOPM

Данная структура предоставляет доступ к заголовку либо Control Block, либо Control Block Extended. LENGTH - размер заготовленной под таблицу памяти. POS - адрес в памяти относительно адреса блока + размер cbhead, куда нужно будет помещать новую запись таблиц. NEXT@ - адрес следующего блока того же типа. ID@ - код модуля к которому блок относится

1. **Trace table**

caller module name

**Trace table**

address of departure point

called module name

address of destination point

TRTBL DSECT

\*HEADER

CRMN DS CL64

ADEP@ DS FD

CDMN DS CL64

ADNP DS FD

TRTBL# EQU \*-CRMN

Данная таблица - то, что мы запишем по окончанию работы программы. Каждый вызов инструкции из **ICB** кроме **SVC 8** (LOAD) влечет занесение записи об этом событии в **trace table**.

Имя модуля ограничено 64 символами. Адрес может быть от 4 до 8 байт. **Размер неограничен.**

1. **RSA - Retry Supply Area**

PSW

**RSA - Retry Supply Area**

SAVEAREA

OLD REGISTERS

Данная структура данных нужна для обеспечения возврата в ЧП после ошибки из retry routine. Для разным режимов адресации и работы она различна - различается размер PSW и регистров. Таким образом, на PSW надо отвести 16 байт (максимально - режим адресации EA), на OLD REGISTERS надо отвести 144 байта (максимально - режим адресации EA).

RSA DSECT

PSW DS CL16

SAVEAREA DS CL144

OLDREG DS CL144

RSA# EQU \*-PSW

1. **USA - User-Supplied Area**

**TABLE OF PREPARED MODULE** address

**trace table** address

**USA - User-Supplied Area**

**RECB chain** address

**ICB** address

**RSA** address

**RETRY ROUTINE** address

**record of TOPM** address

**current RECB** address(3)

record of **RECB** address

**NAME OF BIN LIBRARY**

**length of NAME OF BIN LIBRARY**

**NAME OF LIST LIBRARY**

**length of NAME OF LIST LIBRARY**

record of **ICB**

**PARAMS** address

record of **PARAMS** address

**new RECB** address(3)

USA DSECT

TOPM@ DS F

RTOPM@ DS F

TRTBL@ DS F

RECB@ DS F

CRECB@ DS F

RRECB@ DS F

ICB@ DS F

RICB@ DS F

RSA@ DS F

PARAMS@ DS F

RPARAMS DS F

RETRY@ DS F

\_PDSN DS H

PDSN DS CL54

\_LPDSN DS H

LPDSN DS CL54

\_LPDSN DS H

USA# EQU \*-TOPM@

Данная структура данных нужна для передачи других структур в обработчик ошибок (нашу ESTAE рутину). **Размер ограничен.**

bin library - имя библиотеки с загрузочными модулями

list library - имя библиотеки с листингами

1. **table of prepared modules**

TOPM DSECT

\*HEADER

\_PMN DS H

PMN DS CL10

TOPM# EQU \*-\_PMN

Данная структура данных описывает список подготовленных препроцессингом к загрузке модулей **ЧП**. **Размер неограничен.**

module name (in PARM format)

**table of prepared modules**

1. **RECB - restore control block**

RECB DSECT

\*EXTENDED HEADER

OFFSET DS FD

INSTR DS FD

RECB# EQU \*-

Данная структура данных содержит список команд, которые были испорчены, а так же их оригинальный вариант. Максимальная длина команд из искомых - 47 бит. Выделим 64 бита. **Размер неограничен.**

**RECB - restore control block**

offset

instruction

1. **ICB - instruction control block**

Данная структура данных содержит в себе список команд, по которым должно генерироваться прерывание. Имя команды дополнено до 6 символов пробелами и начинается с 0. **Размер неограничен.**

ICB DSECT

\*HEADER

IC DS CL6

ICB# EQU \*-IC

**ICB - instruction control block**

list of 6-byte instr code

1. **DNCB - Data set name control block**

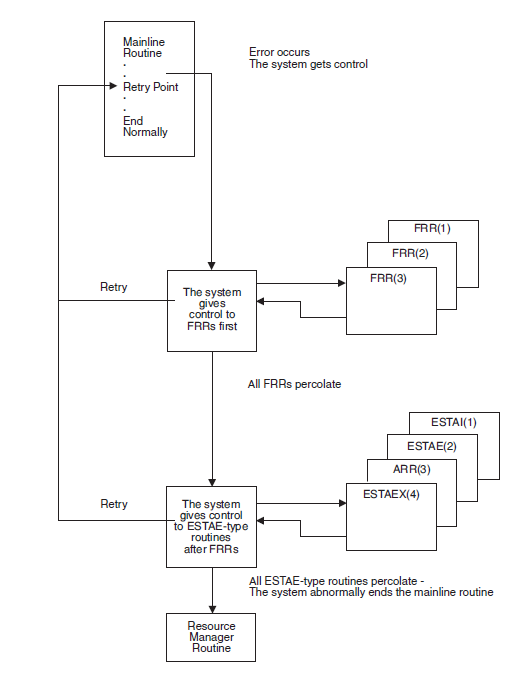
Данная структура используется для передачи имени дата сета и имени модуля в dynamic allocation.

**DNCB - data set name control block**

offset

instruction

Нужна таблица с датасетами которые могут понадобиться? Она аналогична TOPM и для записи или открытия датасетов мы будет передавать указатель на запись в ней.

**Проблема стековости ESTAE.**

Как известно, FRR и ESTAE рутины находятся в стеке. И вызывается при ошибке та, что находится в TOP. Она выбирает retry или percolate. Если percolate то мы спускаемся вниз по стеку, если retry, соответственно продолжается выполнение mainline. Мы задаем свою ESTAE routine на S0C1. Если ЧП вздумает назначить свою рутину, то наша окажется ниже по стеку или даже ее адрес затрется. Чтобы предотвратить это, перехватим SCV 135 и запишем вызов SVC 135 вместо всех вхождений SVC 60 в ЧП. SVC 235 будет брать и подсовывать ESTAE рутину ЧП под нашу, или перезаписывать ESTAE рутину ЧП под нашей.

адрес нашей ESTAE

1) деактивация нашей ESTAE рутины

2) активация ESTAE рутины ЧП

3) активация нашей рутины

**SVC 135**

адрес ESTAE ЧП

адрес нашей ESTAE

BR R2

...

ESTAE (SVC 60)

...

BR R2

**ЧП**

BR R2

...

ESTAE (SVC 135)

...

BR R2

**ЧП**

SUPERZAP

**link pack area, region, job pack area**

**Проблема определения целевого модуля при прыжке с помощью BR**

Когда мы определяем куда мы прыгнули с помощью BR мы целевой адрес проверяем на принадлежность всем диапазонам адресов для всех модулей в address space. В доках написано, что диапазоны и имена берем из LPRB или LRB. Однако LPRB и LRB это только в MFT системах. z/OS - это MVT система, так что там надо искать в CDE, что в принципе с точки зрения функционала тоже самое что LPRB, только в профиль. Диапазон адресов, в которых лежит подгруженный модуль, его имя, и даже конкретный CSECT оттуда узнать можно.

Каждый модуль в address space описывается contents directory entry (CDE) и в extent list (XL) в котором хранится размер модуля.

CDE это набор очередей. Есть очередь для link pack area и по одной для каждого region или job pack area. TCB для текущего job step, запросивший регион contents directory находится в system queue space. У CDE такой формат:

flags1 1b

next cde 3b

res 1b

RB pointer 3b

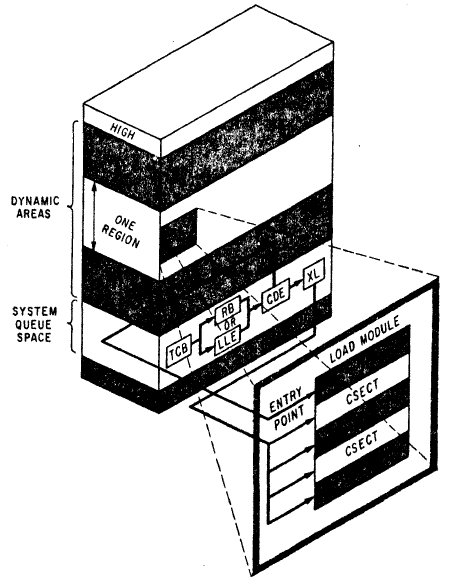
prog name 8b

count 1b

EP 3b

flags2 1b

XL addr 3b

****

flags2

Bit 0 - Reserved.

Bit 1 - Module is inactive.

Bit 2 - An extent list has been built for the module.

Bit 3 - This CDE contains a relocated alias entry point address.

Bit 4 - The module is refreshable.

Bit 5, 6, 7 - Reserved.

flags1

Bit 0 - Module was loaded by NIP.

Bit 1 - Module is in process of being loaded.

Bit 2 - Module is reenterable.

Bit 3 - Module is serially reusable .

Bit 4 - Module may not be reused.

Bit 5 - This CDE reflects an alias name (a minor CDE).

Bit 6 - Module is in job pack area.

Bit 7 - Module is not only-loadable

У XL следующий формат:

xl len 4b

num of csects

csect1 len 4b

csect1 loc 4b

...

Таким образом надо вытащить из TCB load list из него LRB/LPRB/PRB/LLE наш CDE а из него XL.

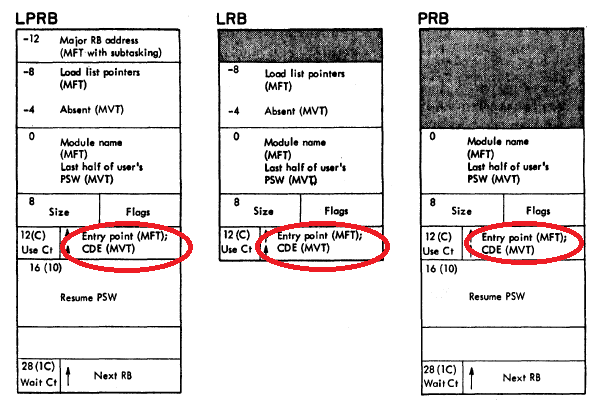
RES 1b

next(recent) LLE 3b

count

CDE pointer 3b

...

****

Остается открытым вопрос о формате блоков когда для адресации модуля требуется 4 байта. CDE предоставляет для адресации entry point только три. Поэтому необходимо написать тест в котором load подгрузит модуль так чтобы его entry point был ниже 24-bit а другой выше и в нем мы выведем содержимое cde и xl. Главное чтобы они не изменили XL. Если есть XL то в него все помещается: 4 байта место, 4 байта размер. Ведь код не может лежать выше планки в 2 Gb.

Итак, лезем в TCB, находим очередь блоков, определяем те который PRB, LRB, LPRB, LLE. Идентифицируем себя и не трогаем, остальные пробегаем и вытаскиваем из XL диапазоны адресов и проверяем принадлежность целевого адреса хотя бы одному из них.

**Соглашения о коде.**

Можно писать как макросы, так и рутины. Рутины оформляются с помощью PROCBGN и PROCEND. Параметры в рутину передаем через R0,R1. Когда пишем макрос, то портящиеся регистры указываем а комментарии после названия. Макросы общего назначения будем складывать в EMCPROJ.MACLIB. Остальное в коде программы.

Имена строим из заглавных букв названия. Если будет с чем-то совпадать, добавляем вторую букву из первого слова и т.д.\*

Длины DSECT обозначаем как <имя DSECT>#

Длины данных в поля обозначаем как \_<имя поля> например CMN=Current Module Name, там хранится 'ololo', а в \_CMN должно храниться 5

Макросы рассчитываем на прием как регистров так и данных. Иногда целесообразно делать даже для числовых констант типа X'0005'.

Если есть DSECT, и он обозначает заголовок списка, то элемент списка будем обозначать как R<имяDSECT>

Формат входных параметров для программы SPYM.

BINARY=<имя библиотеки с бинарными модулями>

LISTING=<имя библиотеки с листингами>

MAIN=<имя главного модуля>

COMMANDS=<команды для перехвата>

COMMANDS=<команды для перехвата>

OUTPUT=<dataset для вывода собранной информации>

Входные параметры передаются через DD имя SYSIN в следующем формате:

Все модули, вызываемые из MAIN должны лежать в библиотеке BINARY. Все листинги для модулей должны лежать в библиотеке LISTING.

Пример:

//SYSIN DD \*

BINARY=EMCPROJ.LOADLIB

LISTING=EMCPROJ.INPUT

MAIN=SAMPLE

COMMANDS=LINK,BR,XCTL

COMMANDS=LOAD

OUTPUT=EMCPROJ.OUTPUT(SPYM)

/\*

План реализации:

Вспомогательные макросы и рутины

ПОДГОТОВЛЕН ЛИ МОДУЛЬ - ЕСТЬ ЛИ В TOPM (1.1)

ПОИСК В RECB КОМАНДЫ ПО СМЕЩЕНИЮ (1.2)

МАКРОС КОТОРЫЙ УМЕЕТ ВОЗВРАЩАТЬСЯ ПОСЛЕ КОМАНД С ВОЗВРАЩЕНИЕМ ИЛИ НЕВОЗВРАЩЕНИЕМ(2.3)

нужна макра, которая анализирует код dalloc и на его основе либо создает новый датасет для SuperZap либо выводит сообщение об соответствующей ошибке.

СДЕЛАТЬ: ПЕРЕДЕЛАТЬ ПЕРЕДАЧУ ПАРАМЕТРОВ В DALLOC. СДЕЛАТЬ ДЛЯ ЭТОГО СТРУКТУРУ.

СДЕЛАТЬ МАКРОС, ГОТОВЯЩИЙ ЭТУ СТРУКТУРУ НА ОСНОВЕ ПОЛНОГО ИМЕНИ ДАТАСЕТА.

научиться отлавливать и перехватывать создание epie и estae рутин внутри чп

Препроцессинг - ОТЛАДИТЬ ТЕКУЩУЮ ВЕРСИЮ

0)Добавление в table of prepared modules

1)Статический анализ

2)Заполнение поля "current module name"

4)Генерация таблицы для SuperZAP

5)Вызов SuperZAP

Инициализация

1)Выделить память подвсе структуры

2)Заполнить **ICB** и **USA**

3)Препроцессинг главного модуля **ЧП**

4)Перехват **ESTAE**

5)Загрузка главного модуля **ЧП** в память

6)Исполнение **ЧП**

7)Анализ кода завершения и сохранение **trace table** на диск

Первый этап!

Отслеживание переходов всех вида BR и BASR и svc6 и svc7

краткий вариант ESTAE рутины выделен серым цветом

retry routine будет работать для br,basr,svc6,svc7

пока датасет для superzap создается заранее вручную

и тут я понял что надо потестировать git