Введение.

VM для эмуляции **API** используют адресные ловушки и шлюзы(**VM-call**). Ловушки вызываются(*execution flow*) до выборки(*execution fetch*) инструкции(*control flow*) на исполнение, *X* и *C-flow* расходятся. Шлюзы вызываются при выборке на исполнение – это обьекты в **CFG**, в частности не из текущего **ISA**[1] набора, например:

```
0f ff f0 (4b)[2]
lock mov ebx,(4b); jmp 0xffff(2b)[3]
```

Такие обьекты нарушают целостность *X*, *C* и *Data-flow*. Адресные ловушки нарушают **Cf**, так как не выполняется кодовая выборка. Шлюзы могут нарушать **Cf**, возвращая управление из процедур(эмулируя возврат). Ловушки и шлюзы нарушают целостность **Df**(*DFI*), так как вводятся данные без их выборки(***D-fetch***), в частности в контекст задачи(потока). Далее описан универсальный метод обнаружения и блокировки этих обьектов на основе проверки целостности **Df**(*PFI*).

Вводные понятия.

Секвенация.

Выделение ***секвенсором*** последовательности инструкций в **Cf**. Машинная трассировка в отличие от секвенации формирует **Xf**, который может отличаться от **Cf** из за адресных ловушек и эмуляции трассировки. Секвенация выполняется(*бинарная трансляция*) визорами(**DBI**[4]), посредством эмуляции или перемещения кодовой последовательности в буфера.

Атомы .

Обьекты **Cf** не **ISA** набора(могут вызываться через **ISA**), которые не могут быть секвенированы(разделены на последовательность инструкций), но могут оставаться при секвенации(если вызываются через **ISA**). Так ядерный интерфейс в пользовательском режиме это атом(например **sysenter**) – не может секвенироваться, но останется при секвенации(из **ISA**). Сервисный интерфейс в режиме совместимости[5] переключает режим и если секвенсор не использует такое переключение, то является атомом. Аналогично если секвенсор не раскрывает косвенные ветвления, то такая инструкция будет атомом. **VM-call** так же являются атомами и не могут быть секвенированы.

При секвенации адресных ловушек **VM-call** не произойдёт из за исполнения инструкции из адреса ловушки по другому адресу(как и при перемещении процедуры) или эмуляции этой инструкции.

Секвенация позволяет обнаружить **ISA** выборки данных(адресную трансляцию), тем самым проверить **DFI**[6] и обнаружить атомы(так как из атомов нет выборок), например:

```
push 0              ; [A]: 0
push esp            ; [B] = A
Call [*Proc]
```

1 Instruction Set Architecture.
2 *"Windows Offender: Reverse Engineering Windows Defender's Antivirus Emulator", Alexei Bulazel*
3 *"AVLeak: Fingerprinting Antivirus Emulators For Advanced Malware Evasion", Alexei Bulazel*
4 Dynamic Binary Instrumentation.
5 Compat.ibility Mode.
6 Data Flow Integrity.

```
Proc:
        mov R,[esp + 4]     ; R = [B]
        inc [R]             ; [R] + 1
        ret
```

При секвенации косвенных ветвлений секвенсор обнаружит выборку(***W-fetch***) inc [R] и нарушения **DFI** не будет, иначе атом(Proc) изменил память [A] без выборки:

```
        push 0              ; [A]: 0
        push esp            ; [B] = A
        Call [pProc]
                            ; [A]: 1
```

Граф указателей(**Pf**).

Указатели в **Pf** *наследуются* при загрузке или выгрузке ссылок[7], как и данные в **Df**. **Pf** формируется при секвенации, в минимальном **Pf** только маркируются регистры и адреса памяти маркерами ссылок(содержащими указатели), например:

SequenceDispatchISA(PUSH R):

```
                            ; #R индекс регистра, определён через diss.InputReg
                            ; или ISA-опкодом. Индексирует маркер указателя(ссылки).
                            ; M маркера адресов, R регистров.

        M{(T.Esp - 4)} = R{#R}
```

```
pProc:
        @Proc
                            ; Esp является ссылкой(L), так как содержит указатель.
        push 0              ; [A]: 0
                            ; R{#Esp}: L
        push esp            ; [B]: A
                            ; M{Esp} = R{#Esp}
                            ; (M{B} = R{#Esp})
        Call [pProc]
```

```
Proc:
        mov R,[esp + 4]     ; R{#R} = M{Esp + 4}
                            ; R{#R} = M{B}
                            ; PFG: R{#R} = R{#Esp}: L
        inc [R]             ; [R] + 1
        ret
```

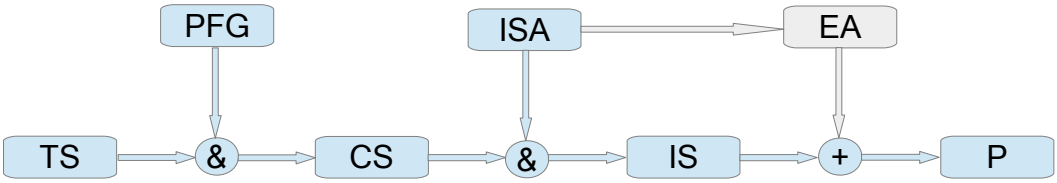
При секвенации возврат из атома может быть не на следующую за ним инструкцию, а выполнена эмуляция возврата из **API**. Тогда секвенация прекратится, поток выйдет из под визора. Что бы этого избежать необходимо установить ловушку(*S-route)* на адрес возврата, заменив его на адрес заглушки, начинающей секвенацию.

Приведение ссылки к указателю.

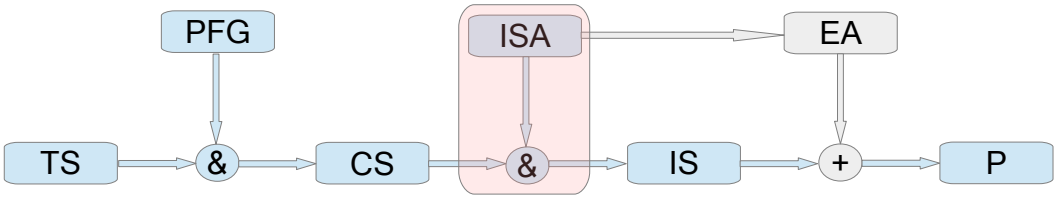
Контекст **TS** задачи(*потока*) - регистры и текущий стековый фрейм содержат ссылки, которые определяет **PFG**, это текущий набор(*Current Set*) ссылок **CS**. Кодировка

7 ***Ссылка*** – указатель содержащийся в регистрах или памяти.

инструкций **ISA** определяет какие из **CS** используются выполняемой инструкцией, это входной набор(*Input Set*) ссылок **IS**. В кодировке **ISA** *эффективный адрес* **EA**, если он есть, вычисляется из регистров **IS** и образуется указатель **P:**



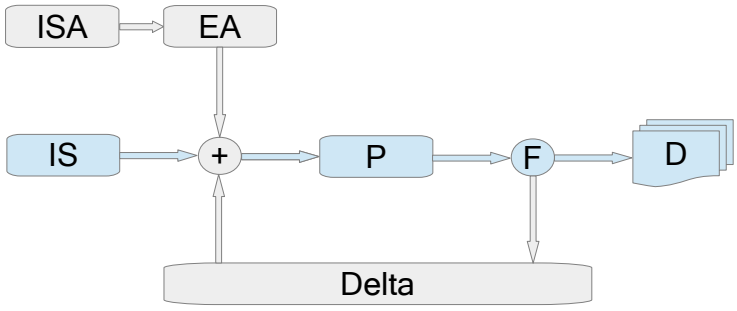
В примере для pProc(атом без секвенации) **IS** это параметр на стеке. Для атома приведение ссылки к указателю происходит без кодировки **IS** и **EA**, указатель **P** по которому произойдёт доступ к данным не соответствует **IS**:



Вызов атома Call [pProc] в **ISA** не содержит **IS**. Для инструкции inc [R] **IS**(R) кодируется полем *modr/m*(**EA**) инструкции. В **CS** регистр R является ссылкой и приведение ссылки при секвенации pProc происходит без ошибок **DFI**(для **Pf** это **PFI**).

Захват указателей(**IDP**).
Изменение указателя на область памяти для отображения на иную. При секвенации ссылка в **IS** или указатель через **EA** могут быть изменены(*захвачены*). Тогда указатель будет отличен от полученного адресной трансляцией без секвенации и выборка данных по указателю произойдёт в область памяти, не существующую без секвенации(*софтверный анклав*).

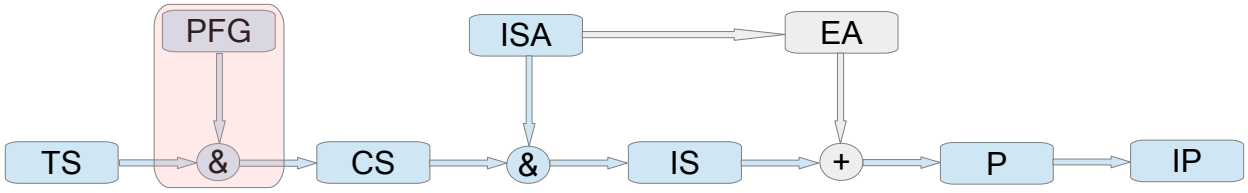
Указатель P вычисляемый при адресной трансляции на основе **EA**(кодируемого **ISA**) и ссылки в **CS** изменяется на значение **Delta** после события выборки **F**, данные выбираются из анклава:



Описание механизма.

Критерий обнаружения атома(также и **OP**-инжекта): доступ к данным без соответствия **IS** или размеру выборки. Для атома **IS** не известен(может быть определён после его исполнения), но известен для **ISA**(**EA**). Дополнительные критерии основаны на особенностях **ISA**: количество ссылок **CS** более одной и *дереференс* ссылки.

При передаче управления в **CFG** на адрес **IP** происходит *OP-инжект*[8], если указатель **P** не соотвествует **CS**(**IS**), так как ссылка не была загружена в **PFG**:



Доступ на запись памяти из атома.

На каждой итерации секвенсор должен проверять изменение памяти(запись из атома). Для **ISA** известна выборка данных(**EA**) - известна ссылка **CS**(всегда одна(**CS** = 1), за исключением нескольких инструкций(строковые)), что делает не нужным в данном случае **PFG** и определение **CS**(*diss.InputReg*). В **ISA** нет дереференсов, поэтому ссылки на стеке так же не нужно определять. Нижний адрес изменённой области должен соответствовать *линейному* адресу(**EA** при адресной трансляции), иначе это исполнение атома, либо *удалённая* запись в память. Дополнительной проверкой может быть:

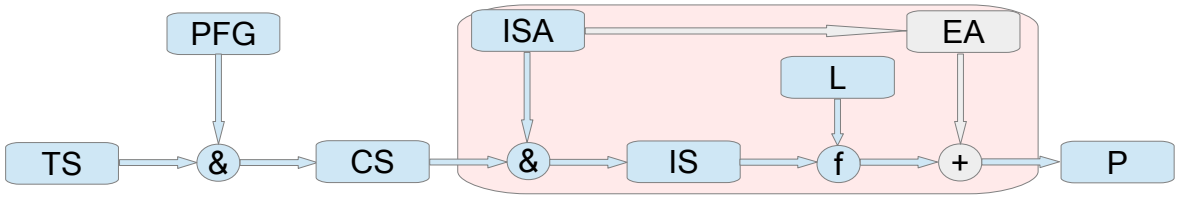
- Размер выборки, который может быть определён из **ISA**, либо использован максимальный размер текущего стекового фрейма, формированного при секвенации(*стек возвратов*, подобно **IA-CET**) или цепью **BP**-фреймов.
- Запись по разным областям(**CS** > 1 для атома). Такую запись можно обнаружить по изменению двух стековых фреймов.
- Не соответствие состояния задачи(контекста) изменяемым регистрам(*diss.OutputReg*).

Для проверки изменения памяти достаточно использовать только стек, так как секвенация может быть временной. Для обнаружения изменений в стеке нужна копия стека и размер выборки, либо цепь стековых фреймов(тогда ограничение размера выборки границей фрейма) и их *хэши*.

Проверка изменения памяти возможна только если размер атома равен размеру **ISA**, иначе секвенсор выйдет за пределы инструкции, либо атом не будет вызван(если состоит из нескольких **ISA** инструкций).

Доступ на чтение памяти из атома.

Необходимо выполнить *захват указателей*(ссылок), создав анклав для атома. Атом может использовать промежуточный буфер для доступа к **IS**, тогда произойдёт доступ к анклаву. Что бы избежать этого должен быть захвачен **CS**. Для атома при захвате **CS** не произойдёт восстановление ссылки **L** через функцию **f** от текущей ссылки в **IS**, так как IS не определён. Доступ к данным **D** произойдёт по исходной ссылке в **IS**(что приведёт к ошибке доступа в атоме):



8 OP-

CS(IS атома не известен) захвачен изменением, перед выполнением инструкции должен быть восстановлен IS и вновь захвачен на следующей итерации секвенсора. IS ядерных шлюзов(sysenter etc) стековый(аргументы в стеке), восстанавливается только ссылка на стек(SP). Возможна вариация без захвата CS.SP(стек необходим для обработки исключений) – захват ссылок в стеке, вместо ссылки на стек. В этом случае ссылки в стеке должны быть восстановлены до выполнения ядерного шлюза, а размер области со ссылками определяется числом сервисных аргументов(можно ограничить размером стекового фрейма). Восстановление ссылок в отличие от IDP необходимо при использовании указателя по значению, например в операциях сравнения.

Далее необходимо рассмотреть:

- Возможность обнаружения по отсутствию использования ссылок(*load - [use] - free*: между формирование ссылки и её уничтожением не было загрузки в CS.
- Деление области на части через ссылку в ней, ссылки разделяют аргументы ?
- Наследование указателей через перемещение данных в VM.
- Проблема длины атома.
- Обнаружение ловушек: разница в TS/MM после прямого вызова и секвенации.
- Хранение PFG/фреймов в первом BP-фрейме при секвенации.

Не закончен...