Call Map(CM).

Для получения управления после передачи его на не линейный код(из которого происходит передача управления на произвольны адрес) используется маршрутизация. Адрес ветвления заменяется на стаб, оригинальный адрес возврата(вектор) сохраняется в **TLS**. Возврат происходит на стаб, который передаёт управление на сохранённый в **TLS** адрес.

Эта модель маршрутизации верна в случае отсутствия рекурсивных вызовов, иначе возникают следующие проблемы:

1. Необходим массив векторов(стек) в **TLS**, так как адреса возврата могут быть разные.
2. При возврате часть векторов может быть утеряна. Необходимо связать вектор с записью в массиве векторов.
3. Вектор может быть не стековым адресом возврата, в таком случае он не связан со стеком потока.

Вектор должен отображаться на массив векторов по его значению. Тогда на основе адреса стаба можно получить исходный вектор. Используется линейный массив процедурных ветвлений(**CM**). Каждое ветвление выравнено на 8 байт и содержит общий адрес стаба. При возврате на одно из этих ветвлений в стек загружается текущий адрес*(Ip' = Ip + 5)*. Так как карта(**CM**) линейна, то смещение в ней(адрес возврата из процедурного ветвления) является индексом в массиве векторов:

```
M:      ...
        Align 8
Vn:
        Call S
        Align 8
Vn+1:
        Call S
        Align 8
        ...

S():
        P = POP()                   ; Адрес возврата из Call в CM.
        I = (P & NOT(7) – M)/8      ; Индекс в массиве(стеке) векторов.

        Ip = CmStack{I}             ; Оригинальный вектор.
```

CM-стек это массив, который индексируется черз **CM**. В него сохраняется по последнему(свободному) индексу оригинальный вектор. При возврате в стаб индекс последней записи устанавливается равным текущему.

Переполнение **CM**-стека(**CMS**).

Некоторые вызовы выполняются подобно условным ветвлениям, тоесть используют два вектора, на один из которых выполняется возврат. Это оставляет второй вектор в **CMS**. Примеры:

1. **NtContinue** возвращает управление из сервиса или передаёт управление на новый вектор(**CONTEXT.Ip**). В **CMS** будут соответственно две записи.

2. **NtCallbackReturn** возвращает управление из сервиса или возвращает управление в тень(shadow). В **CMS** останется запись с возвратом из сервиса(в сервисный шлюз).

3. **NtClose** разворачивает исключение или возвращает управление из сервиса.

При подобных рекурсиях **CMS** будет засоряться паразитными записями. Поэтому их необходимо удалять. Используется маркер паразитной записи — запись в **CMS** помечается(**IDLE**). При вызове любого сервиса и при возврате в стаб последняя **IDLE**-запись в стеке удаляется. Таким образом в случае 1(при двух записях)произойдёт следующее:

CMS:
..., P1(idle), P2

A. Возврат из сервиса(P1) приведёт к удалению P2 из за установки последнего индекса в стабе.
B. Возврат на новый вектор(P2) приведёт к удалению последней паразитной записи(**IDLE**).

В случае 2 запись в **CMS** одна:

CMS:
..., P(idle)

A. Возврат из сервиса приведёт к автоматическому удалению P(через механизм возврата - стаб).
B. После возврата в тень произойдёт возврат на вектор ниже в **CMS**, что приведёт к его очистке, либо произойдёт теневой вызов юзер-нотифи. При следующем вызове сервиса P будет удалён, так как помечен **IDLE**.

В случае 3 при обработке исключения и вызове любого сервиса последний вектор удалён не будет, так как не помечен **IDLE**.

Дефейны.

Описатель **CMS**:

```
TCM struct
        Ip2st       PVOID ?     ; Сохраняется адрес возврата в CM из стаба.
        Index       ULONG ?     ; Индекс последней свободной записи.
        Limit       ULONG ?     ; Максимальное число записей(для тени — 32).
        Stack       PCME ?      ; Массив записей CME.
        Map         PVOID ?     ; База карты CM.
TCM ends
```

Запись **CM(**в **CMS)**:

```
CME struct
        Ip          PVOID ?     ; Оригинальный вектор.
        Nt          NT <>       ; Описатель сервиса.
        Flags       DWORD ?     ; Флаги(маркер IDLE).
CME ends
```

Описатель сервиса необходим для сервисной post-нотифи. При возврате из сервиса его описатель получается из **CME.Nt**, где сохраняется при вызове сервиса(маршрутизации для

возврата из сервиса).

Алгоритмы.

Возврат на стаб через **CM**:

```
P = POP()                       ; Адрес возврата на CM-call.
I = ((P – Tcm.Map) & NOT(7))/8  ; CM-индекс.
CME = Tcm.Stack{I}              ; Запись в CMS соответствующая Ip'.
Ip = CME.Ip                     ; Оригинальный вектор.
if I                            ; CMS не пуст, проверяем предыдущую запись.
        CME = Tcm.Stack{I - 1}
        if CME.Flags.IDLE       ; Маркер IDLE, удаляем запись.
                I - 1
        fi
fi
Tcm.Index = I                   ; Устанавливаем новый предел CMS.
```

Вызов сервиса:

```
I = Tcm.Index
if I
        CME = Tcm.Stack{I – 1}  ; Проверяем последний элемент на маркер IDLE.
        if CME.Flags.IDLE       ; Удаляем паразитную запись.
                Tcm.Index - 1
        fi
fi
```

Маршрутизация через адрес возврата:

```
I = Tcm.Index
if Tcm.Limit == I               ; CMS переполнен.
        Fail
fi
CME = Tcm.Stack{I}
Sp = TSTATE.Rgp.rEsp

; Обмен векторами на стеке потока.

CME.Ip = [Sp]                   ; Сохраняем оригинальный вектор.
[Sp] = @Tcm.Map{I*8}            ; Загружаем вектор на CM-call.
Tcm.Index + 1                   ; Расширяем CMS.
```

Дополнительная маршрутизация из **NtContinue**:

```
I = Tcm.Index
if Tcm.Limit == I
        Fail
fi
CME = Tcm.Stack{I}

; Обмен векторами в контексте. Далее при вызове сервиса выполняется стековая
```

маршрутизация(добавление записи после текущей.

CME.Ip = CONTEXT.Ip
CONTEXT.Ip = @Tcm.Map{I*8}
CME.IDLE
Tcm.Index + 1

Поправка из **NtCallbackReturn**:

I = Tcm.Index
CME = Tcm.Stack{I}
CME.IDLE