

Защита потока данных(*Data Flow Guard*).

Защита потока управления(**CFG**) или данных(**DFG**) – методы обнаружения нарушения целостности кода или данных, обнаружение чужеродных объектов.

Нарушение целостности кода — появление в нём посторонних инструкций, код может сам изменяться или вызывать чужеродный код косвенным путём — посредством изменения памяти, например указателей в массивах методов или адресов возврата на стеке. В последнем случае защита целостности кода носит название защиты потока управления(**CFG**). Она реализуется различным путём: *NT-CFG*, *CFI*, *Intel-CET* etc, далее рассмотрена техника **DFG**, использование её для таких приложений, как обнаружение/обход **AVVM** и защита памяти.

DFG – есть защита потока данных(см. *Data Flow Graph*). По сравнению с **CFG** это более глубокая валидация кода. При **CFG** проверяются адреса инструкций или их связанность в графе(eg: *call-ret*). При **DFG** проверяется на целостность поток данных, используя контекст задачи(потока) и память. **DFG** позволяет обнаружить атомы.

Атом — объект(присутствующий в коде), изолированный от среды исполнения. Такие объекты выполняют теневые операции, не доступные текущей среде и соответственно нарушают **DFG**. Примеры атомов:

Наномиты — в коде ставится точка останова и через отладчик на этой точке останова изменяется контекст задачи/память. Это нарушение целостности потока данных — входные/выходные данные и операция, выполняемая инструкцией(которая является точкой останова) не связаны(eg: *Int3* не изменяет контекст задачи).

Ядерные сервисы — такие объекты передают управление на иной уровень привилегий, где выполняется обработка данных.

Объекты эмуляции — при обнаружении в коде маркера виртуальная машина(эмулятор) выполняет некоторую функцию.

Код, выполняющий атомы в общем случае изолирован от исходной среды – выполняется на ином уровне привилегий, в контексте гипервизора(**VM**) etc. Использование атомов позволяет обнаружить виртуализацию(**AV**-эмуляцию(**AVVM**)), отладочную активность etc.

AVVM использует атомы для эмуляции сложных осевых функций(**API**). В код функции вставляется вызов атома - маркер, при обнаружении которого **VM** получит событие выполнения соответствующего сервиса, обработает его и продолжит дальнейшую эмуляцию. Это позволяет отвязать работу, выполняемую кодом апи от самого кода и код от адресов. В этом случае перемещение тела апи или его пересборка(морфинг) оставляет вызов апи в коде.

Выборка данных(*Data Fetch*).

Для использования **DFG** необходимо отследить машинную выборку данных(**DF**). Это операции с памятью, выполняемые **CPU**. Выполнить это можно двумя путями:

1. Хардверный трек памяти, установка ловушек на память. В этом случае на блок памяти ставится ловушка, которая срабатывает при обращении к памяти.
2. Эмуляция каждой инструкции и декодировка адресов. Необходима полная трассировка/эмуляция приложения или вызываемой функции. Этот способ медленный и требует сложной системной обработки(события создания потоков, ядерные колбеки etc).

На основе информации про **DF**(адрес данных, контекст связанный с данными) может выполняться валидация **DFG**(проверка изменений памяти и результата операции в контексте).

В атомах DF не существует(по причине изоляции атомов от среды). Это даёт метод обнаружения и изоляции атомов. Заблокировав память(блокировка памяти есть установка ловушки на неё) от атома он не сможет выполняться корректно. При **DF** выборка перенаправляется на другую память или выполняется эмуляция. Эта техника именуется **IDP(Intercept Destruction of Pointers(захват разрушением указателей))**. Изначально применялась как руткит-техника: ссылка на данные делалась инвалидной(ставилась ловушка), **DF** отслеживалась и данные подменялись.

Защита памяти.

IDP помимо изоляции атомов как следствие даёт метод защиты памяти. В исходной области памяти данных не существует. Так как **DF** происходит только из текущей среды(процесса соответственно), то **DF** из иной среды(процесса или иного **CPL**(из ядра, если среда - юзермод)) не существует.

Динамическая защита памяти(**DYPE: DYnamic ProtEction**).

Учитывая выше описанное техника следующая.

Секции защищаемого модуля блокируются и ставится ловушка на **DF**. В результате:

1. Кода/данных в этих областях не существует. Изолированный из текущего процесса код(**Reader**) их прочитать не может. Нельзя снять дамп.
2. Атомы выполняться не могут – прекращение работы **AVVM** при выполнении осевых функций, использующих заблокированную память.

При срабатывании ловушки(**DF**) на чтении определяется метод адресации(инструкция раскодируется), выбираемые данные расшифровываются в буфер и на него перенаправляется **DF**(через изменение части контекста, из которой формируется адрес или через эмуляцию). Далее поток продолжает нормальное исполнение.

При срабатывании ловушки(**DF**) на выполнение — это запуск потоков, вызов колбеков и возврат из процедур запускается эмуляция. Код расшифровывается по одной инструкции и исполняется/эмулируется. Это происходит до перехода на код, за пределами заблокированной памяти, затем поток продолжает нормальное исполнение.

Валидация ридера(**Reader Check**).

DF возможна из стороннего кода(ридер), который не безопасен(**AV**) и загружен в текущий процесс. Проблемы не возникает если в модуле секции кода не содержат данных(констант). В таком случае достаточно полностью запретить **DF** – выборка кода выполняется **AV**. Если секции содержат константы, то необходимо определить какому ридеру разрешать **DF**. Решения возможны следующие:

1. Валидация ридера, определение что ридер небезопасен(списки **AV**). Сомнительный способ.

2. Отличить данные(константы) от кода. **DF** для данных разрешена, для кода запрещена. Универсально, но является сложной задачей.

Обход CFI.

Описанная техника позволяет обойти **CFG**, выполняемую через валидацию вызывающего кода(*Caller*): **CFI**. При **DF** могут быть возвращены виртуальные данные — сформирован корректный код вызова.

При возврате в защищаемую проекцию из сторонних процедур происходит срабатывание ловушки. Избежать такого срабатывания(и повысить эффективность **DYRE**) можно путём загрузки указателя на стаб, запускающий эмуляцию. Но такая операция не совместима с **CFI** и обойти можно используя описанный выше метод.

-

(c) Indy, 2016.