ИСП РАН

Разработка и реализация метода генерации цепочек возвратно-ориентированного программирования

Алексей Вишняков

8 июня 2020 г.

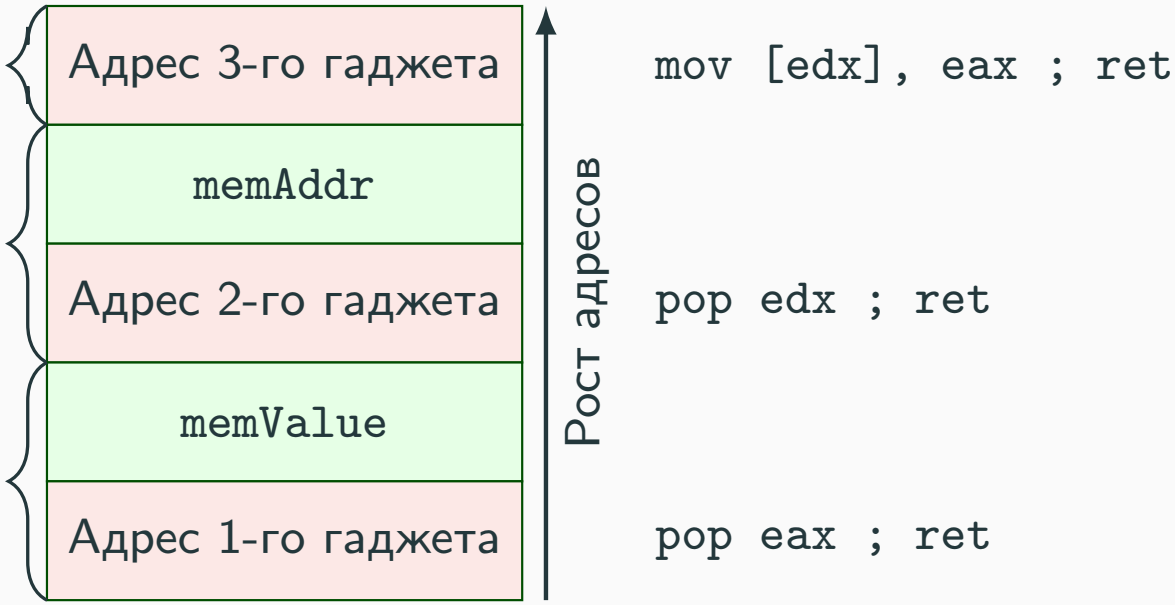
ИСП РАН

Актуальность и цель работы

- Ошибки — естественное свойство любой живой программы
- *Критические дефекты* — особенно опасные ошибки, которые могут быть использованы злоумышленником, чтобы добиться непредусмотренного поведения системы
- В ответ на усовершенствование методов защиты от использования дефектов появляются новые методы их обхода
- Методы возвратно-ориентированного программирования (ROP) позволяют обходить современные защитные механизмы
- Данная работа ставит своей целью разработку метода генерации ROP цепочек, который может быть использован для оценки качества реализованных защитных механизмов в рамках гранта РФФИ № 17-01-00600

Возвратно-ориентированное программирование

- **Возвратно-ориентированное программирование (ROP)** — атака повторного использования кода, которая позволяет обходить защиту, ограничивающую исполняемые области (DEP)
- **Гаджет** — последовательность инструкций, заканчивающаяся инструкцией передачи управления
- Гаджеты ищут в нерандомизированных областях
- Каждый гаджет выполняет некоторые вычисления и передает управление следующему гаджету



Постановка задачи

Необходимо разработать и реализовать метод генерации цепочек возвратно-ориентированного программирования

Метод должен:

- Генерировать цепочки гаджетов для архитектур x86 и MIPS
- Использовать как возвратно-ориентированные (ROP), так и переходо-ориентированные (JOP) гаджеты
  - `pop rax ; jmp rcx`
- Учитывать запрещенные символы
  - Например, в случае переполнения функцией `strcpy` цепочка не должна содержать нулевых байтов
- Генерировать цепочку гаджетов, выполняющую системный вызов

Существующие решения

- Генерация на основе шаблонов гаджетов
  - ROPgadget
  - Ropper
  - mona.py
- Построение графа зависимостей гаджета (ROPium)
- Генерация цепочек с использованием SMT-решателей
  - angrop
  - Exrop
- Генерация на основе семантических деревьев[1]
- Полноценно решает проблему запрещенных символов только ROPium
- Ни один из перечисленных инструментов не поддерживает MIPS

[1] E. J. Schwartz et al. Q: exploit hardening made easy

Поиск и определение семантики гаджетов

- Для поиска гаджетов используется ROPgadget
- Набор типов гаджетов (булевых постусловий) задает новую архитектуру набора команд (ISA)
  - `MoveRegG: OutReg` ← `InReg`
  - `LoadConstG: OutReg` ← `[SP + Offset]`
- Классификация гаджета выявляет:
  - Набор типов и параметров, которым он соответствует
  - Список «испорченных» регистров (значения которых не сохраняются после выполнения гаджета)
  - Информация о фрейме гаджета (размер фрейма, расположение адреса следующего гаджета)

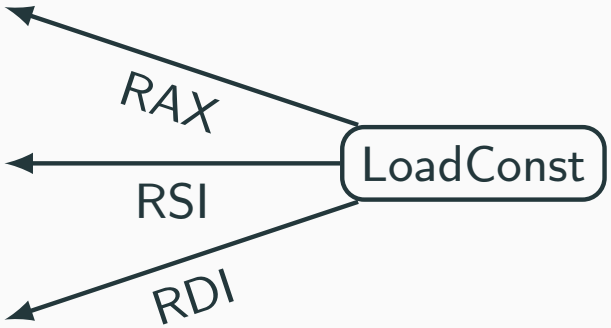
Описание архитектуры

- Разработанные алгоритмы генерации цепочек абстрагированы от архитектуры
- Для архитектуры описываются:
  - соглашения о вызовах (регистры-аргументы)
  - номера системных вызовов
  - используемые в цепочке регистры

Гаджеты загрузки

- Классификация ищет только гаджеты загрузки на один регистр
- Принципиальное отличие от Q[1] — мы объединяем гаджеты, загружающие сразу несколько регистров

```
LoadConst: POP RAX ; POP RDI ; POP RSI ; RET
```



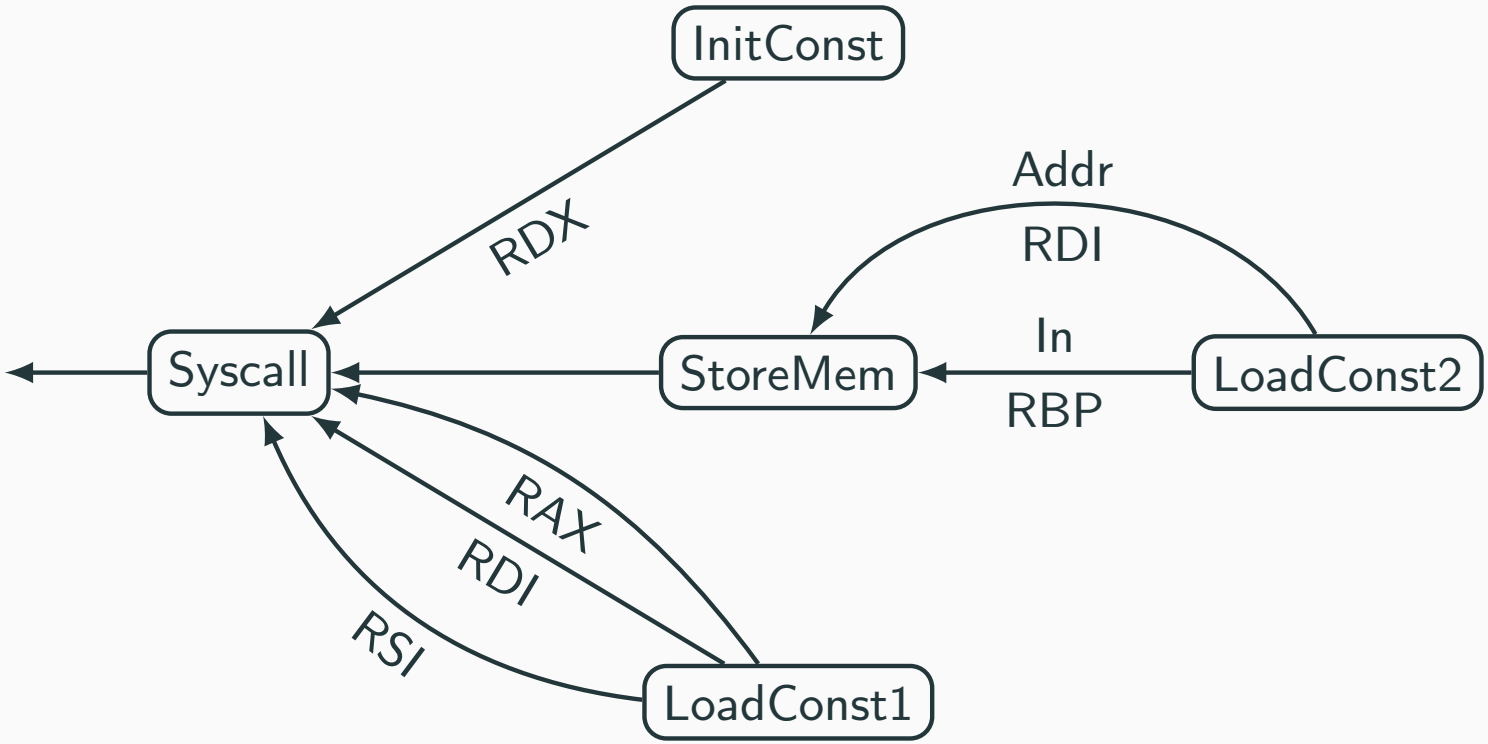
[1] E. J. Schwartz et al. Q: exploit hardening made easy

Предварительная обработка гаджетов

- Переходо-ориентированные (JOP) гаджеты объединяются с ROP гаджетами
  - Получается обычный ROP гаджет
  - `pop rax ; pop rcx ; ret ; pop rdx ; jmp rcx` — это `LoadConst: rax ← [SP], rdx ← [SP + 24]` с размером фрейма `FrameSize = 32` и адресом следующего гаджета по смещению 8 (`NextAddr = [SP + 8]`), адрес гаджета `pop rdx ; jmp rcx` необходимо разместить по смещению 16
- Генерация цепочек гаджетов — переборная задача
- Гаджеты фильтруются, т. е. оставляются только «лучшие» кандидаты
  - Сводится к задаче поиска максимумов на частично упорядоченном множестве
- Гаджеты сортируются по качеству
- Строятся индексы гаджетов для ускорения запросов на получение гаджетов некоторого типа с определенными значениями параметров

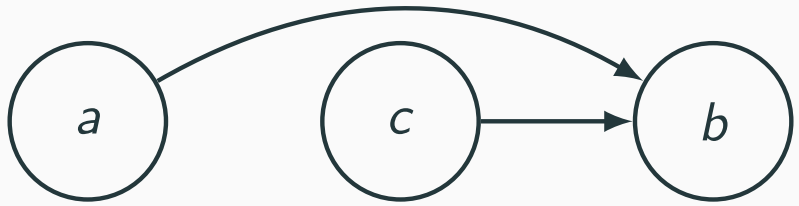
Ориентированный ациклический граф (DAG) гаджетов

```
LoadConst1: POP RAX ; POP RDI ; POP RSI ; RET
LoadConst2: POP RDI ; POP RBP ; RET
StoreMem  : MOV [RDI], RBP ; RET
InitConst : XOR RDX, RDX ; RET
```



Построение расписания для DAG гаджетов

- Расписание должно удовлетворять топологической сортировке DAG гаджетов
- Если выходной регистр гаджета a используется гаджетом b, то этот регистр не должен быть «испорчен» ни одним гаджетом в расписании между a и b

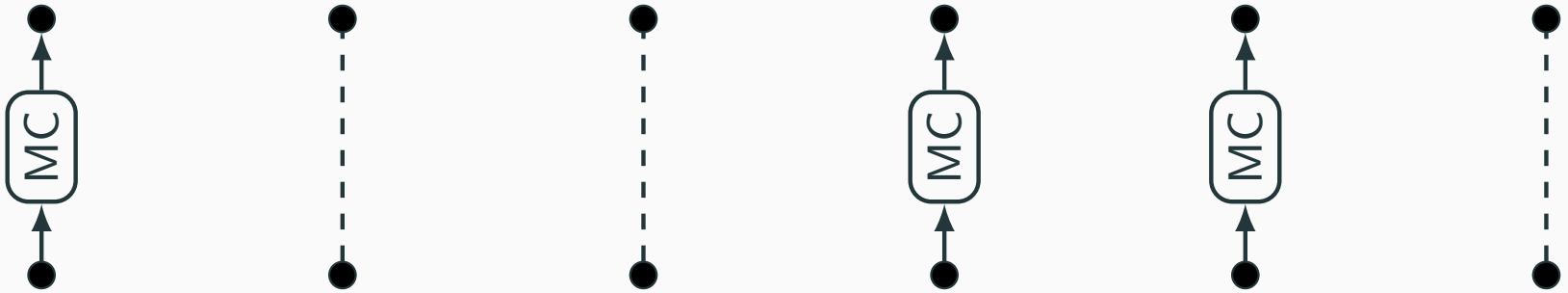


Перебор DAG гаджетов

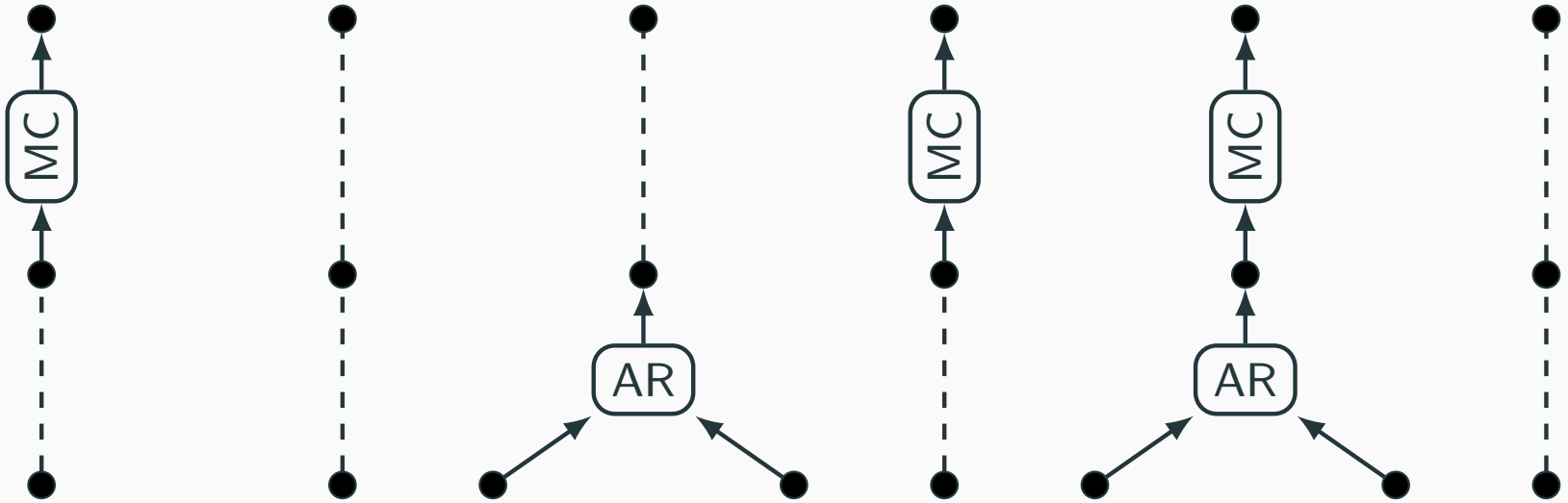
- Задача нахождения цепочек перемещения значения между двумя регистрами сводится к задаче поиска пути на графе
  - Вершины — регистры
  - Ребра — гаджеты
- Можно загрузить значение на один регистр и переместить его в другой
- Загрузить на регистр значение можно в результате арифметической операции
- Чтобы сохранить значение в память, можно предварительно инициализировать память константой и произвести арифметическую операцию над этой памятью
  - `mov dword[eax], 0 ; ret ; add [eax], ebx ; ret`

Инициализация регистров значениями

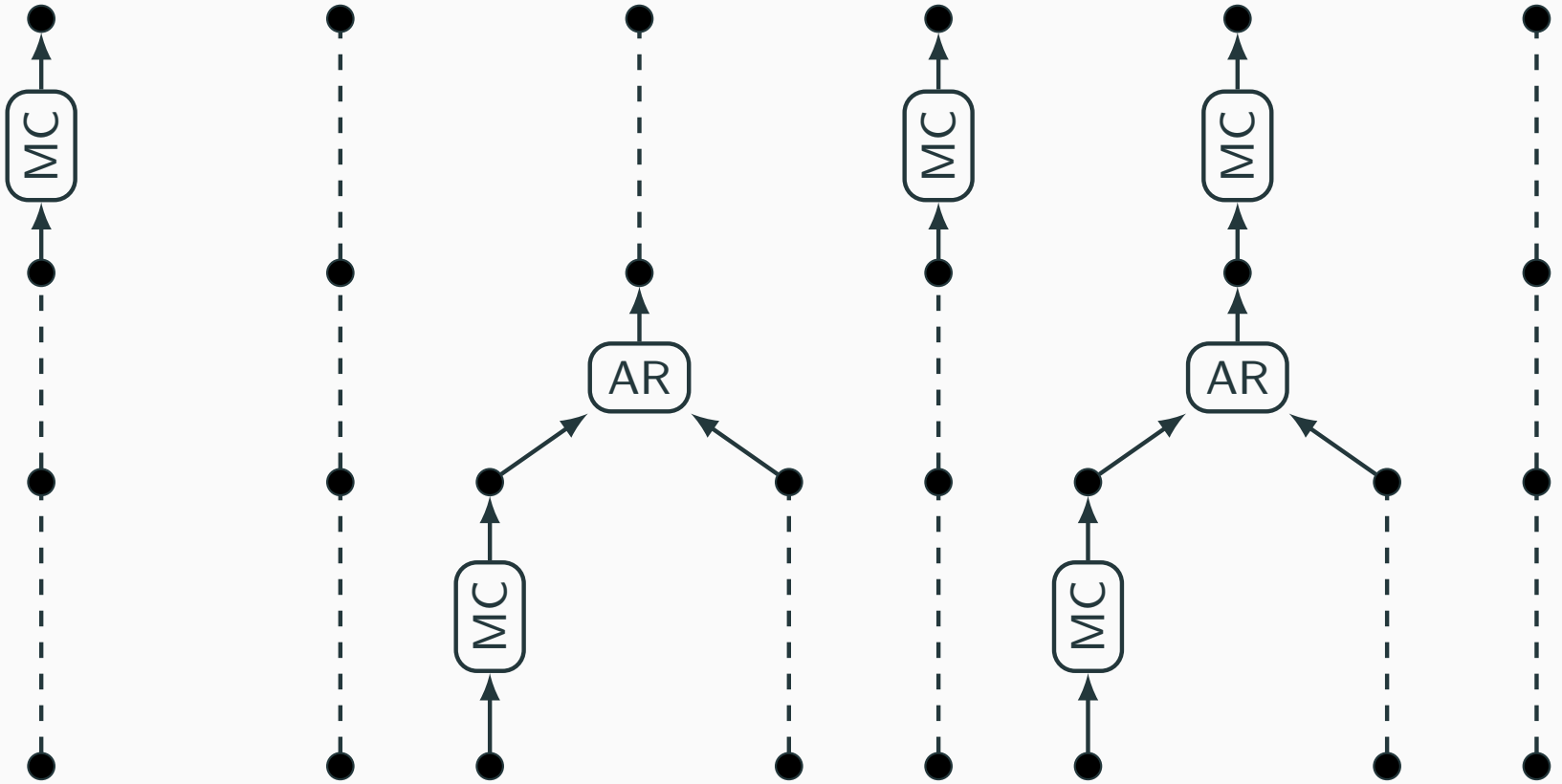
Инициализация регистров значениями



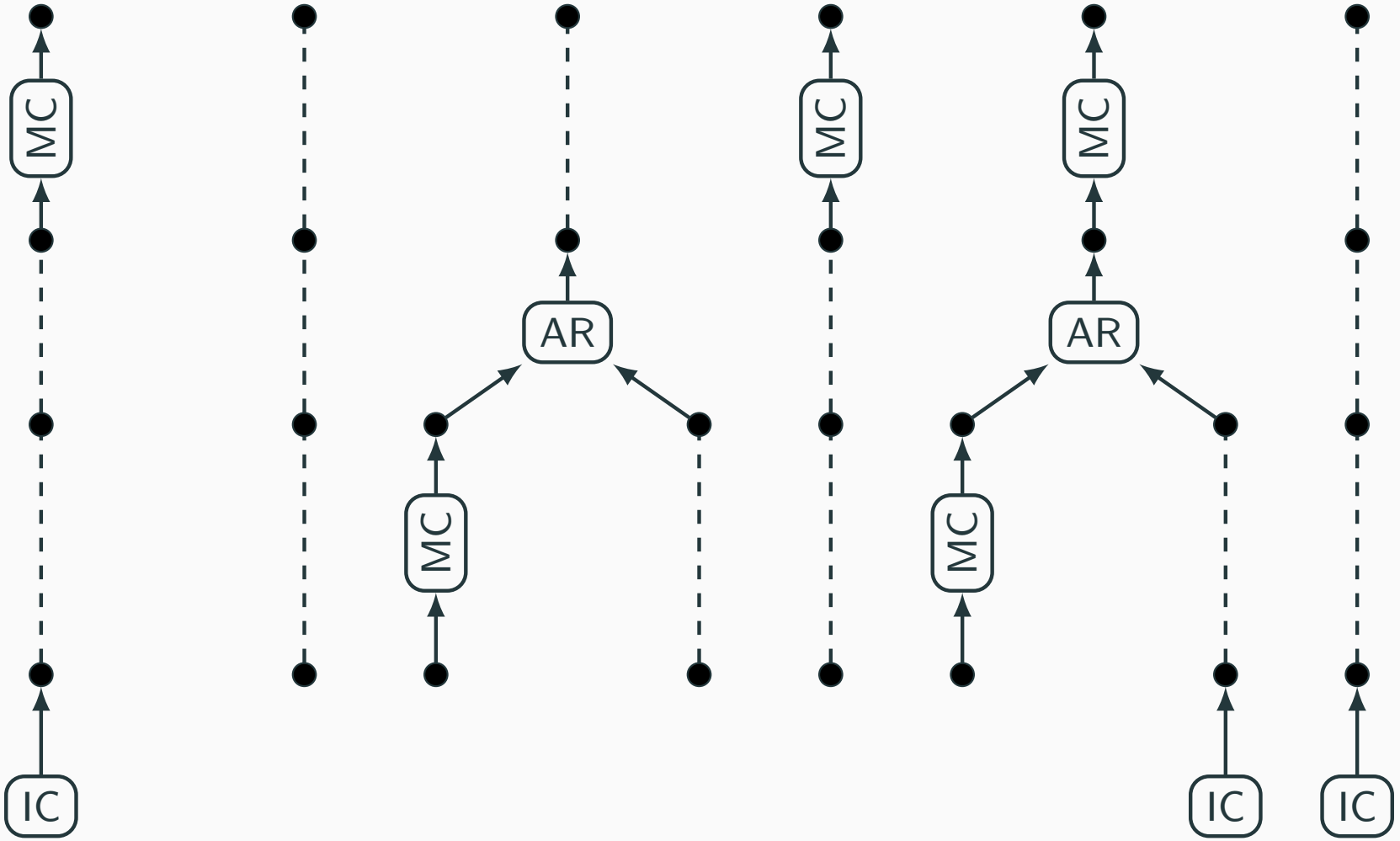
Инициализация регистров значениями



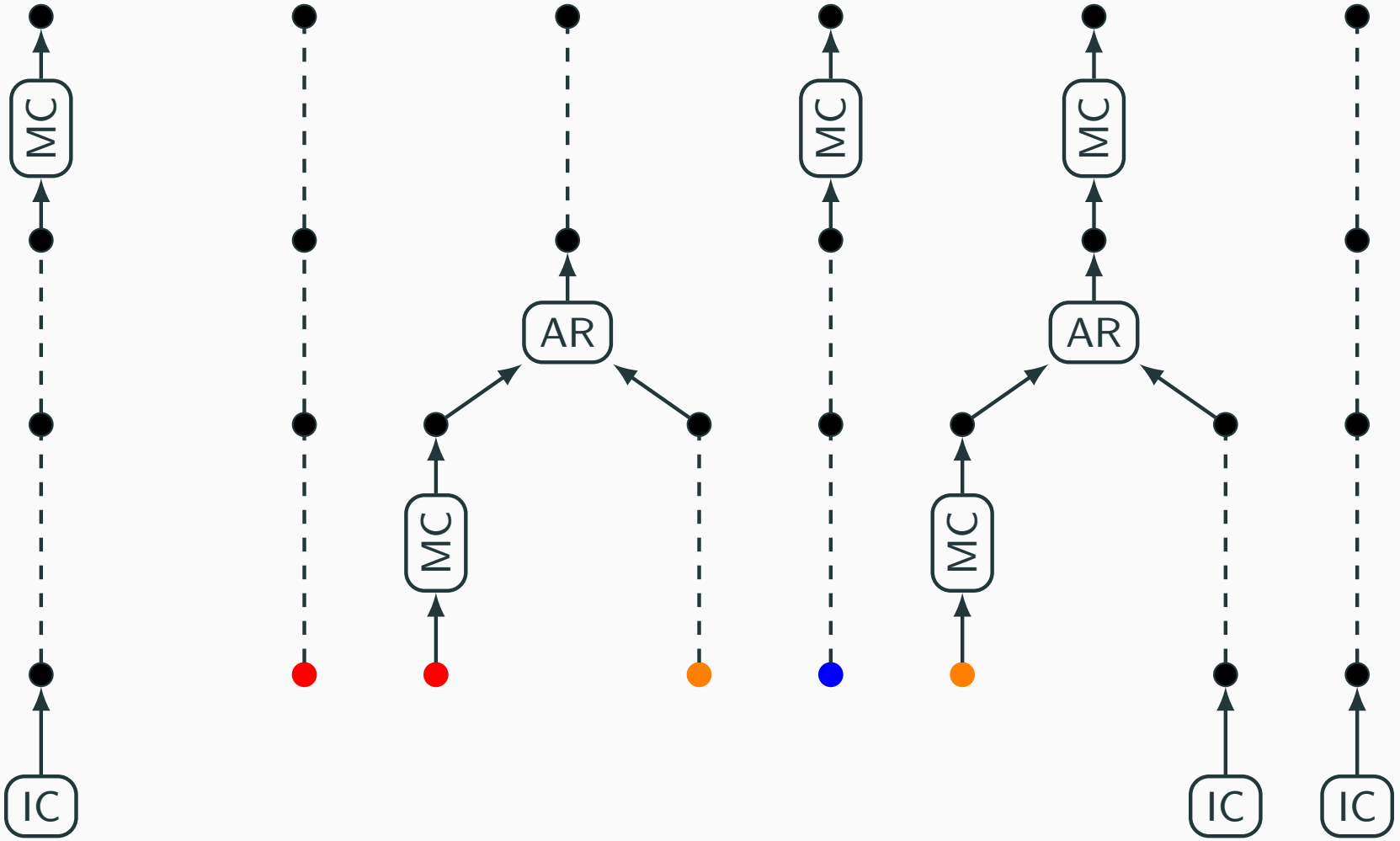
Инициализация регистров значениями



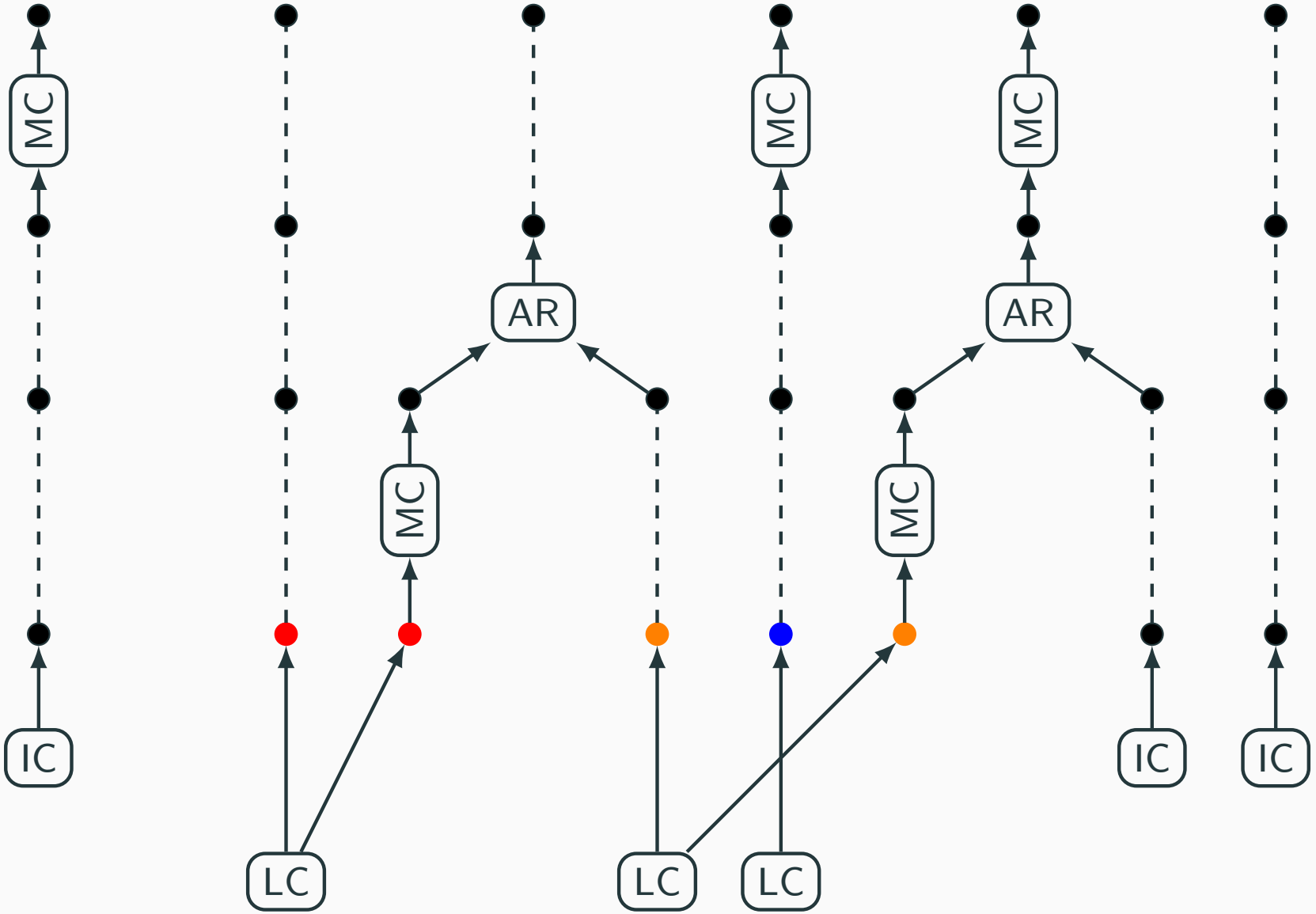
Инициализация регистров значениями



Инициализация регистров значениями



Инициализация регистров значениями



Учет запрещенных символов

- DAG гаджетов не должен загружать со стека значения, содержащие запрещенные символы
- Для вычисления операндов арифметических операций (lv и rv), которые не содержат запрещенных символов, используется метод динамического программирования
  $lv + rv = value$
  - Два состояния: присутствовал или отсутствовал перенос из предыдущего разряда

Создание DAG системного вызова (вызова функции)

- Поддерживаются целочисленные, строковые аргументы и массивы
- Во время обработки аргументов создаются необходимые DAG записи в память
- В соответствии с соглашением о вызове создается генератор DAG, устанавливающий требуемые регистры в заданные значения
- Данный DAG дополняется вершинами, осуществляющими системный вызов (передачу управления на функцию)

Результаты

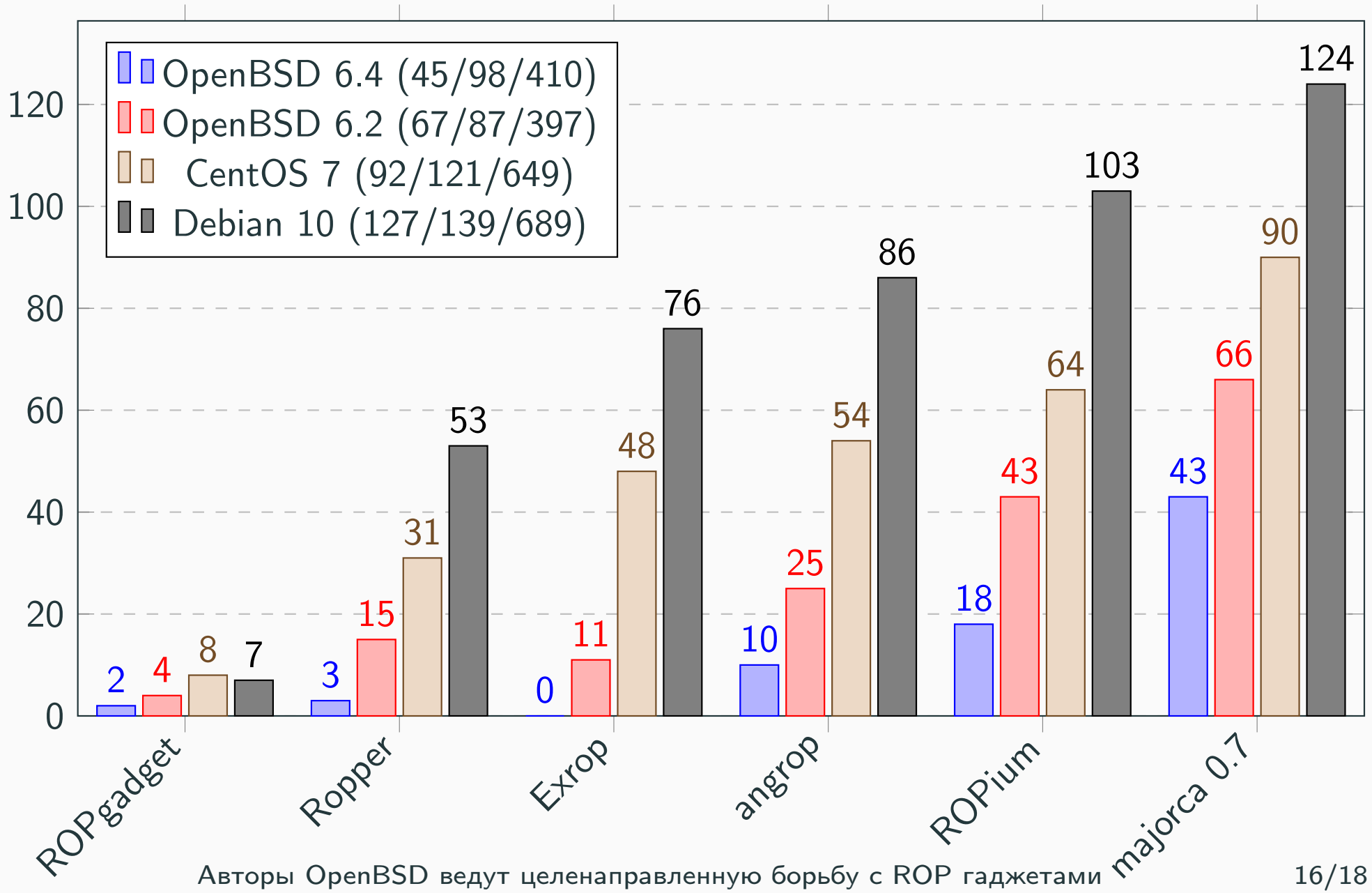
Был разработан метод генерации цепочек возвратно-ориентированного программирования для архитектур x86 и MIPS

- Метод позволяет осуществлять поиск ROP и JOP гаджетов, их классификацию и генерировать цепочку, выполняющую системный вызов
- Метод полностью решает проблему запрещенных символов

Метод был реализован в виде программного инструмента MAJORCA

- Реализованный инструмент показал лучшие результаты по сравнению с аналогичными инструментами, имеющими открытый исходный код
- Сравнение количества работоспособных цепочек системного вызова производилось с помощью реализованной тестовой системы ROP Benchmark

Результаты ROP Benchmark (лимит времени 1 час)



Публикации по теме магистерской диссертации

Вишняков, А. В. Классификация ROP гаджетов // *Труды института системного программирования РАН*. — 2016. — Т. 28, № 6. — С. 27–36. DOI: 10.15514/ISPRAS-2016-28(6)-2.

Метод анализа атак повторного использования кода / А. В. Вишняков, А. Р. Нурмухаметов, Ш. Ф. Курмангалеев, С. С. Гайсарян // *Труды института системного программирования РАН*. — 2018. — Т. 30, № 5. — С. 31–54. DOI: 10.15514/ISPRAS-2018-30(5)-2.

A Method for Analyzing Code-Reuse Attacks / A. V Vishnyakov, A. R Nurmukhametov, Sh. F Kurmangaleev, S. S Gaisaryan // *Programming and Computer Software*. — 2019. — Vol. 45, no. 8. — Pp. 473–484. DOI: 10.1134/S0361768819080061. `https://vishnya.xyz/vishnyakov19-draft.pdf`.

Обзор методов автоматизированной генерации эксплойтов повторного использования кода / А. В. Вишняков, А. Р. Нурмухаметов // *Труды института системного программирования РАН*. — 2019. — Т. 31, № 6. — С. 99–124. DOI: 10.15514/ISPRAS-2019-31(6)-6.
`https://ispras.ru/preprints/docs/prep_32_2019.pdf`.

Свидетельства о государственной регистрации программ для ЭВМ

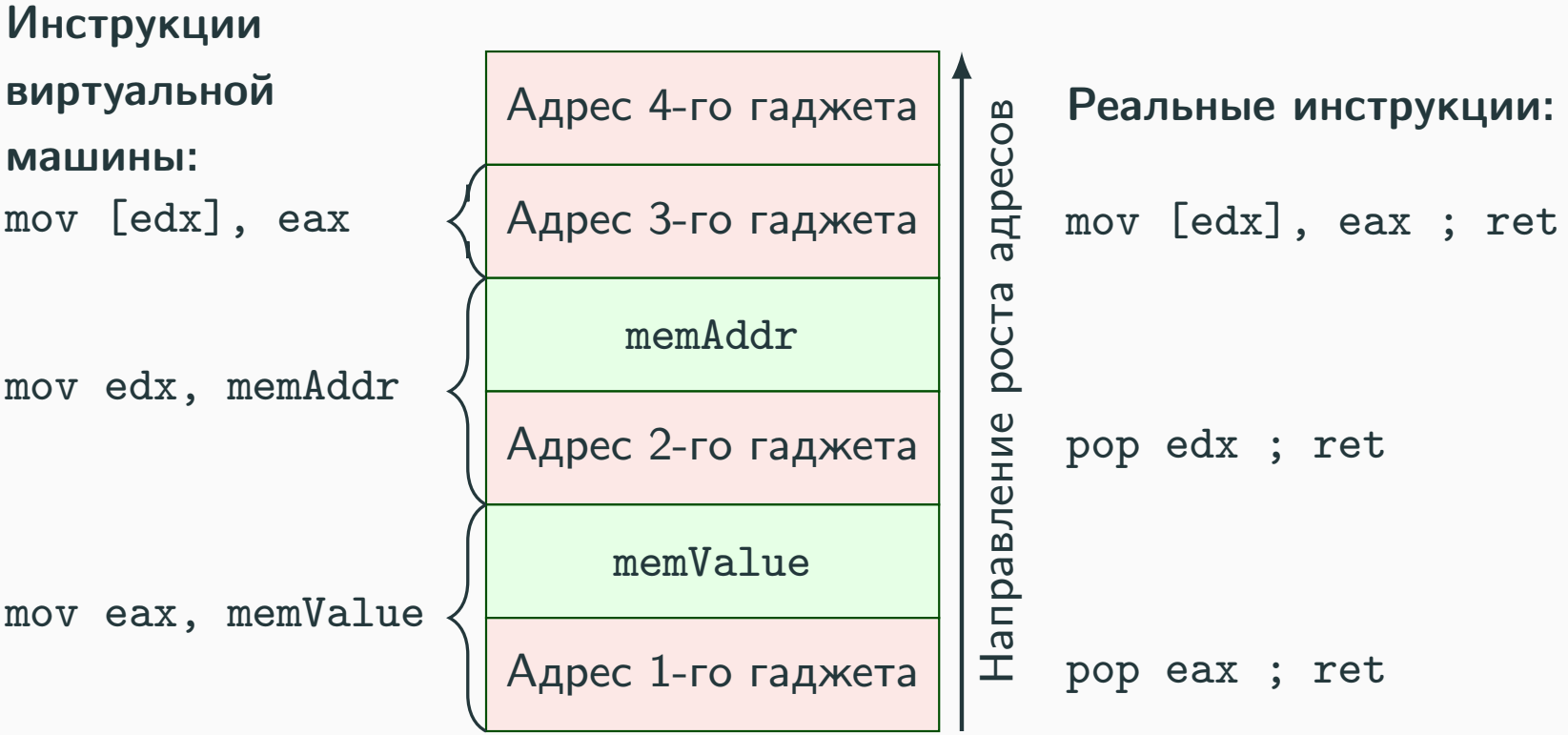
ПрЭВМ № 2019660612 «Инструмент классификации гаджетов возвратно-ориентированного программирования «GCF» / Ш. Ф. Курмангалеев, А. Р. Нурмухаметов, А. В. Вишняков, А. Н. Федотов. https://new.fips.ru/registers-doc-view/fips_servlet?DB=EVM&DocNumber=2019660612.

ПрЭВМ № 2019664648 «Инструмент мультиархитектурной генерации цепочек возвратно-ориентированного программирования «MAJORCA» / А. В. Вишняков, А. Р. Нурмухаметов, Ш. Ф. Курмангалеев, А. Н. Федотов. https://new.fips.ru/registers-doc-view/fips_servlet?DB=EVM&DocNumber=2019664648.

Спасибо за внимание

ROP цепочка — программа для виртуальной машины

- Набор гаджетов для каждого исполняемого файла свой
- ROP цепочка — программа для виртуальной машины, задаваемой исполняемым файлом
- Указатель стека выполняет роль счетчика инструкций
- **Коды операций** (адреса гаджетов) и их **операнды** размещаются на стеке



Поиск гаджетов

- Алгоритм Галилео
  - Найти все инструкции передачи управления в нерандомизированных исполняемых секциях программы
  - Для каждой найденной инструкции произвести дизассемблирование нескольких байтов, предшествующих инструкции
  - Добавить все корректно дизассемблированные последовательности инструкций в каталог гаджетов
- Для поиска гаджетов используется ROPgadget

ROPgadget: github.com/JonathanSalwan/ROPgadget

Ограничения на используемые гаджеты

С целью генерации ROP цепочек введем следующие ограничения на используемые гаджеты:

- Гаджет должен передавать управление следующему гаджету по адресу, размещенному на стеке или регистре
- Каждый гаджет должен увеличивать указатель стека на константное смещение
- Побочным эффектом гаджета не может быть чтение или запись в память, отличную от стека
  - `pop eax ; mov [ebx], ecx ; ret` — «плохой» гаджет загрузки константы в `eax`, но допустимый гаджет сохранения значения регистра `ecx` по адресу `ebx`
  - `push eax ; pop ebx ; ret` — допустимый гаджет перемещения значения регистра `eax` в `ebx`

Классификация гаджетов

- Классификация производится на основе анализа эффектов выполнения гаджета на различных входных данных
- Инструкции гаджета транслируются в промежуточное представление Pivot
- Инструкции промежуточного представления интерпретируются с использованием теневой памяти
  - Отслеживаются обращения к регистрам и памяти
  - Начальные значения регистров и областей памяти генерируются случайным образом
  - В результате будут получены начальные и конечные значения регистров и памяти
- В результате нескольких запусков интерпретации с отличными входными данными составляется список типов и параметров с истинными постусловиями для всех запусков
- Первые два запуска производятся на граничных значениях 0 и -1

Результаты классификации гаджета

- Типы и параметры гаджета
- Список «испорченных» регистров (значения которых не сохраняются после выполнения гаджета)
- Информация о фрейме гаджета
  - Размер фрейма
  - Расположение адреса следующего гаджета

```
0x400278 : Asm : PUSH RBX ; POP RAX ; RET
0x400278 : MoveRegG : EAX <- EBX, AX <- BX, AH <- BH, AL <- BL, RAX <- RBX
0x4000ef : Asm : MOV QWORD PTR [RAX], RBX ; RET
0x4000ef : StoreMemG : [RAX] <- RBX
0x4003e3 : Asm : XOR RAX, RAX ; JMP RCX
0x4003e3 : InitConstG : EAX <- 0x0, AX <- 0x0, AH <- 0x0, AL <- 0x0, RAX <- 0x0 :
           NextAddr=RCX, FrameSize=0
0x400411 : Asm : POP RAX ; JMP 0000000000400416h ; POP RDX ; RET
0x400411 : LoadConstG : EAX <- [RSP], EDX <- [RSP+8], AX <- [RSP], DX <- [RSP+8],
           AH <- [RSP+1], AL <- [RSP], DH <- [RSP+9], DL <- [RSP+8], RAX <- [RSP],
           RDX <- [RSP+8] : NextAddr=[RSP+16], FrameSize=24
0x400411 : ShiftStackG : RSP +<- 16
```

Входные данные MAJORCA

- MAJORCA получает на вход исполняемый бинарный файл
  - ELF
  - raw file с указанием архитектуры, разрядности и порядка байтов
- Можно задавать набор запрещенных символов
- MAJORCA запускает поиск и классификацию гаджетов
- Далее все классифицированные гаджеты загружаются из базы данных

Описание архитектуры

- Разработанные алгоритмы генерации цепочек абстрагированы от архитектуры
- Для архитектуры описываются соглашения о вызовах (регистры-аргументы), номера системных вызовов, используемые в цепочке регистры
- Все регистры представляются одним байтовым массивом
- Регистр задается смещением и размером в массиве

Гаджет

Один загруженный гаджет имеет:

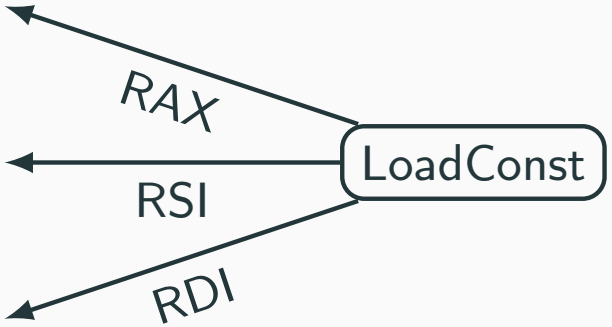
- Конкретный тип
- Значения параметров типа
- Адрес
- Информация о фрейме
- Набор испорченных регистров
  - Битовая маска «испорченных» байтов регистров
  - Смещение и размер берутся из описания регистра в архитектуре
  - Реализованы операции над множествами

Может быть несколько гаджетов с одним адресом

Гаджеты загрузки

- Классификация ищет только гаджеты загрузки на один регистр
  `LoadConstG: OutReg ← [SP + Offset]`
- Принципиальное отличие от Q — мы ищем гаджеты, загружающие сразу несколько регистров
- Гаджеты загрузки, лежащие по одному адресу, объединяются в один гаджет загрузки нескольких значений в несколько регистров `LoadConst: OutReg1 ← [SP + Offset1], OutReg2 ← [SP + Offset2], ...`

`LoadConst: POP RAX ; POP RDI ; POP RSI ; RET`



Гаджет загрузки нескольких регистров

Возможны копирования загружаемых регистров

Поэтому находятся все комбинации непересекающихся загружаемых регистров

```
POP RBX
MOV RAX, RBX
MOV RCX, RBX
POP RDX
MOV RBP, RDX
RET
```

Множества эквивалентности — `(RAX, RBX, RCX)` и `(RDX, RBP)`

Количество комбинаций — $3 * 2 = 6$

Будет создано 6 гаджетов

Ориентированный ациклический граф (DAG) гаджетов

- Позволяет описывать гаджеты загрузки нескольких значений со стека
- Вершины содержат реальные гаджеты с заданными значениями параметров
  - Параметры, размещаемые на стеке (смещение и значение)
- На ребрах хранятся имена входных параметров и регистры
- Зависимости отражаются специальными ребрами
  - Исходящая вершина должна быть вычислена раньше входящей
  - Регистр и параметр не заданы
- DAG гаджетов задается списком его исходящих ребер

Граф гаджетов перемещения значения между регистрами

Задача нахождения цепочек перемещения значения между двумя регистрами сводится к задаче поиска пути на графе

- Вершины — регистры
- Ребра — гаджеты с типами:
  1. MoveReg
  2. ArithmeticConst
  3. Neg

R. Hund et al. Return-oriented rootkits: bypassing kernel code integrity protection mechanisms

Инициализация регистров значениями

- Задача генерации DAG гаджетов, инициализирующего регистры значениями, является основополагающей
- Большинство других нагрузок (вызов функции, системный вызов, запись в память и т. д.) могут быть получены путем привязывания к полученному DAG одного гаджета

Алгоритм инициализации регистров значениями

1. Лениво порождаются различные DAG гаджетов, загружающие значения в запрошенные регистры
2. Проверяется, возможно ли построить расписание для полученного DAG
3. Если расписание построить удалось, то производится попытка установить стековые параметры вершин гаджетов загрузки с учетом запрещенных символов так, чтобы на выходных ребрах DAG были запрошенные значения
4. Возвращаются DAG гаджетов, для которых удалось построить расписание и установить стековые параметры

Загрузка значения на регистр *reg*

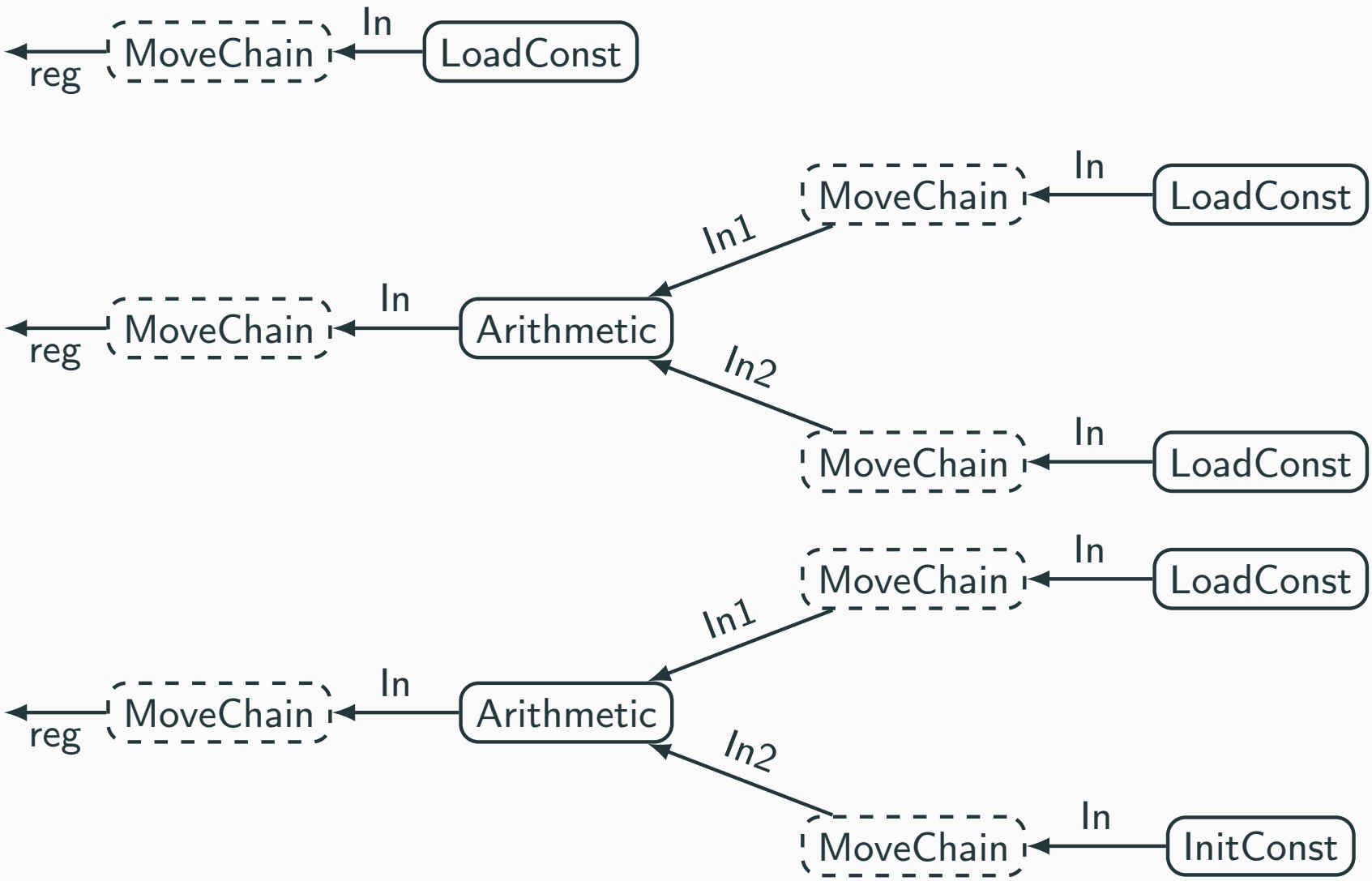


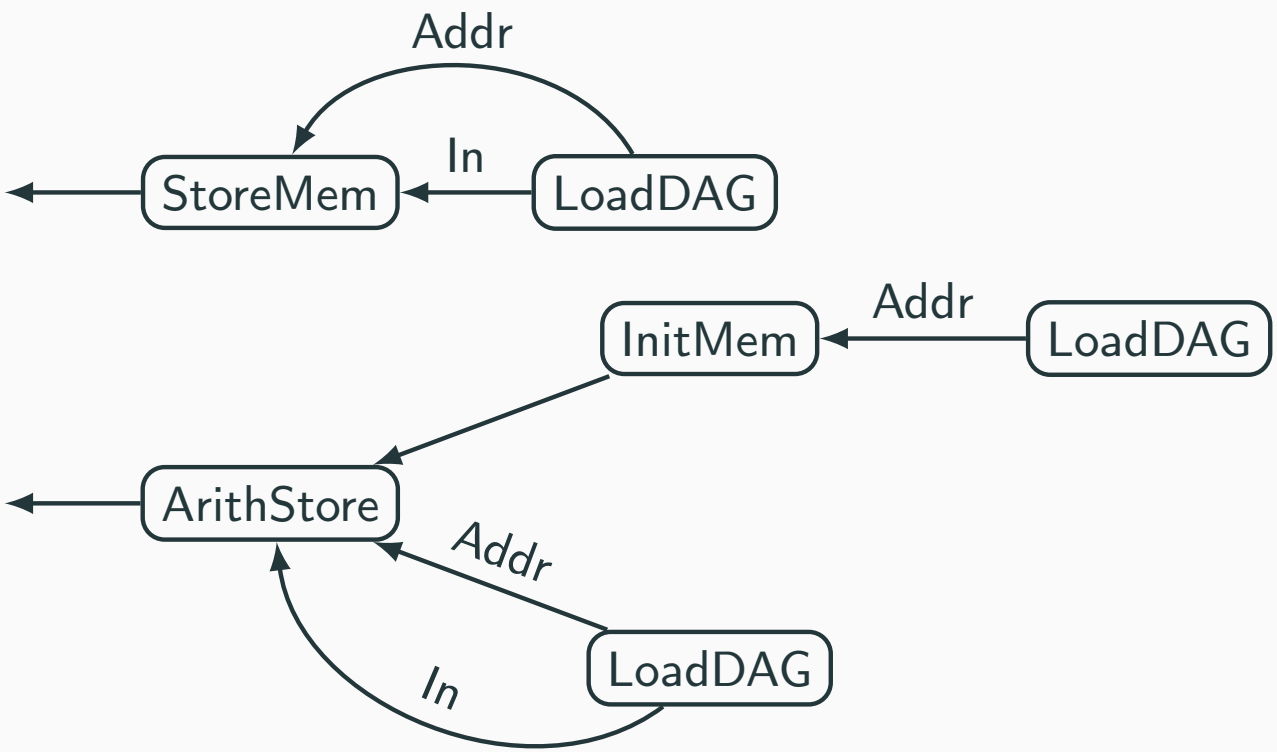
Схема перебора DAG гаджетов

- Выбираются регистры, которые будут перемещены `MoveChain`
- Выбираются регистры, к которым будут присоединены гаджеты Arithmetic
- Выбираются входы Arithmetic, которые будут перемещены `MoveChain`
- Выбираются регистры, которые будут инициализированы константой (InitConst)
- Выбирается разбиение оставшегося множества регистров
  - Используется алгоритм DLX[1]
- Регистры внутри одной части разбиения будут загружены одним гаджетом LoadConst

[1] D. E. Knuth. Dancing links

Запись значения в память

Алгоритм использует `LoadDAG` из предыдущего алгоритма для инициализации регистров



Создание ROP цепочки

- DAG гаджетов линеаризуется в список вершин путем построения расписания
- ROP цепочка генерируется из списка вершин с учетом размера фрейма, адреса следующего гаджета и стековых параметров

ZSNES 1.51 Linux x86 32-bit, запрещены символы: /, \, \0

```
from struct import pack
fill = b'A'  # fill character
chain  = pack('<I', 0x806719a)  # POP EAX ; POP EDI ; RET
chain += pack('<I', 0x91969dd1)
chain += pack('<I', 0x834a860)
chain += pack('<I', 0x807e192)  # NEG EAX ; POP EBX ; RET
chain += 4 * fill
chain += pack('<I', 0x808dbd5)  # MOV DWORD PTR [EDI], EAX ; RET
chain += pack('<I', 0x806719a)  # POP EAX ; POP EDI ; RET
chain += pack('<I', 0xff978cd1)
chain += pack('<I', 0x834a864)
chain += pack('<I', 0x807e192)  # NEG EAX ; POP EBX ; RET
chain += 4 * fill
chain += pack('<I', 0x808dbd5)  # MOV DWORD PTR [EDI], EAX ; RET
chain += pack('<I', 0x807945e)  # POP EBX ; POP EAX ; RET
chain += pack('<I', 0x834a860)
chain += pack('<I', 0xf7c6d8f3)
chain += pack('<I', 0x805318e)  # ADD EAX, 08392718h ; RET
chain += pack('<I', 0x80c6be5)  # XOR ECX, ECX ; RET
chain += pack('<I', 0x81449a0)  # XOR EDX, EDX ; RET
chain += pack('<I', 0x8066984)  # INT 80h  # execve('/bin/sh', 0, 0)
```

Сравнение с открытыми инструментами

ROPgadget: github.com/JonathanSalwan/ROPgadget

angrop: github.com/salls/angrop

ROPium: github.com/Boyan-MILANOV/ropium

Ropper: github.com/sashs/Ropper

Exrop: github.com/d4em0n/exrop

ROP Benchmark: github.com/ispras/rop-benchmark