Анализ кода и информационная безопасность

Лекция 12

МГУ / ВМК / СП

1201

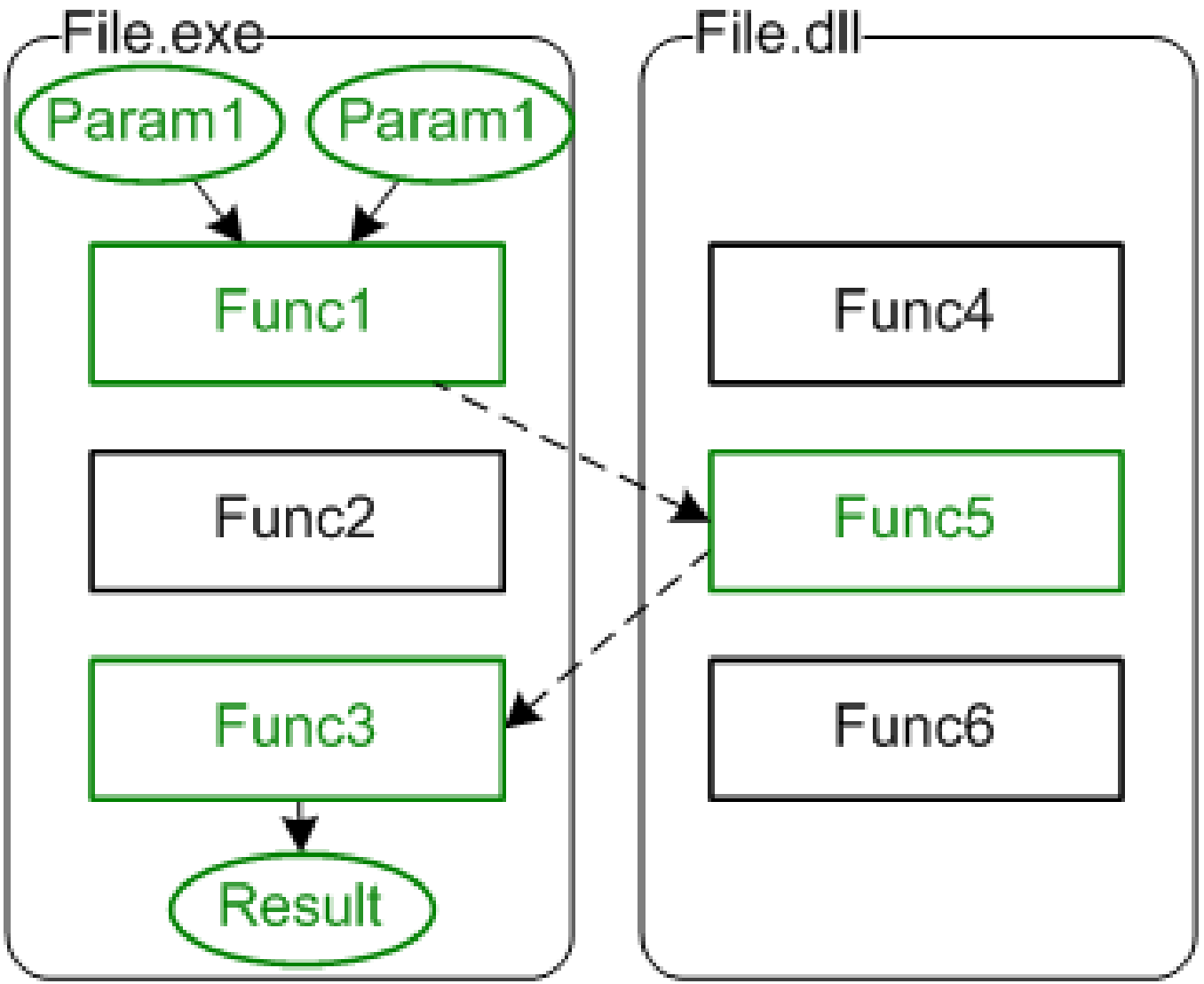
ВЫДЕЛЕНИЕ АЛГОРИТМА. СЛАЙСИНГ ПО ТРАССЕ.

Постановка задачи

- Есть набор исполняемых файлов, в которых наряду с другими реализован исследуемый алгоритм.
- Требуется выделить часть кода, непосредственно реализующая алгоритм и определить его входной и выходной интерфейс, так чтобы данный код мог использоваться сторонним приложением.

- Характерные примеры – алгоритмы шифрования и упаковки/распаковки данных.

Выделение алгоритма



Цели выделения алгоритма

- Повторное использование
  - Реализация совместимой по данным программы
  - Изучение вредоносного кода (сетевые протоколы, упаковка данных)
  - Использование недокументированных функций библиотек
- Анализ свойств
  - Отсутствие закладок
  - Отсутствие утечек конфиденциальных данных
  - Криптографическая стойкость

Схема выделения алгоритма

1. Определение входного и выходного интерфейса (параметров)
2. Анализ зависимостей по данным и управлению между входными и выходными данными
3. Выделение кода, обрабатывающего входные данные и формирующего выходные данные
4. Определение структуры входных и выходных данных
5. Формирование «контрольной реализации» – работающего кода, который может использоваться сторонним приложением

Причины применения динамического анализа

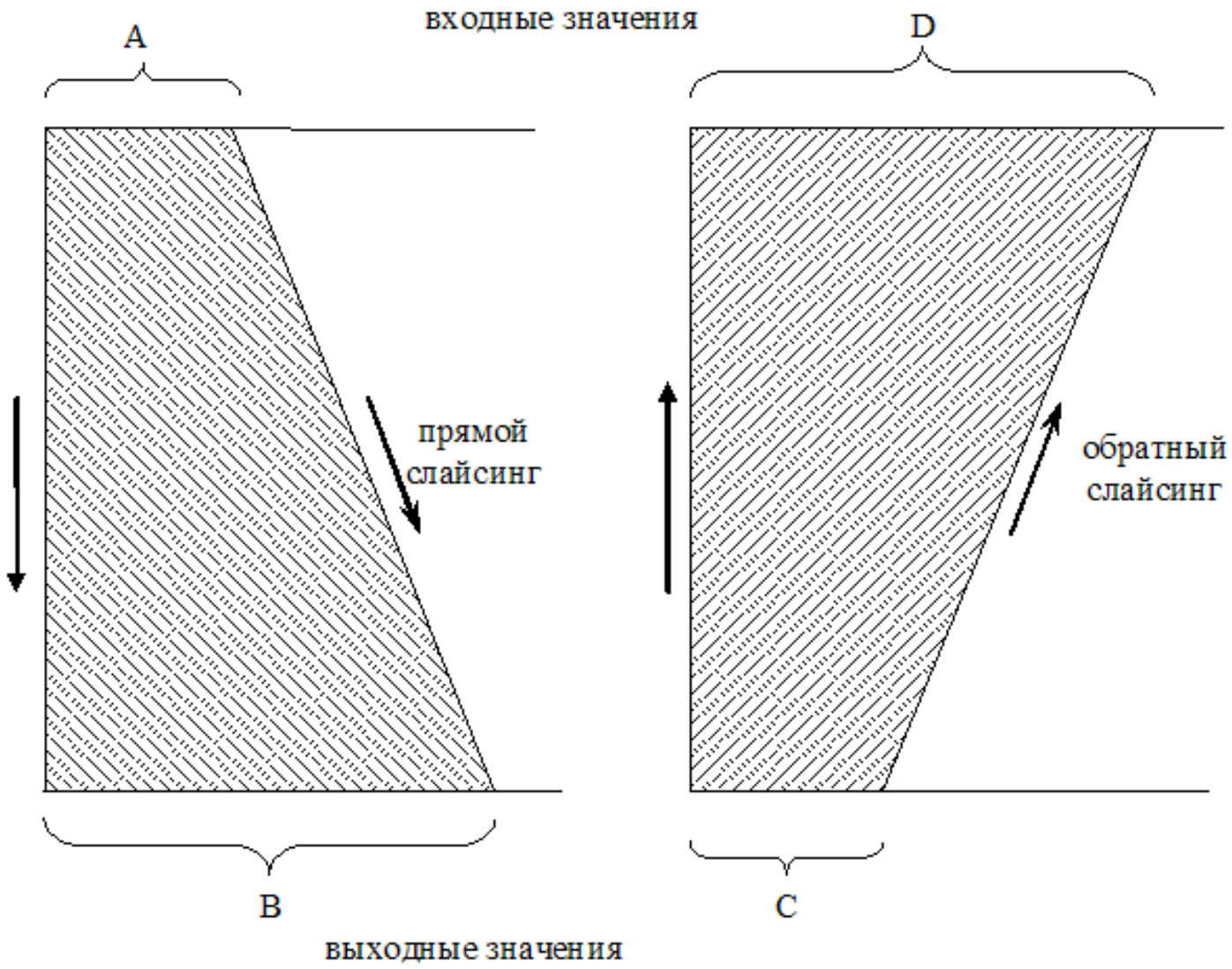
Статический анализ:

- Проблемы с анализом указателей
- Неустойчив к защите от анализа:
  - упаковка/шифрование кода
  - противодействие дизассемблированию/отладке
  - обфускация: запутывание потока управления, сокрытие вызовов функций, диспетчер

Выделение алгоритма из трассы программы

1. Получение трассы программы, содержащей выполнение алгоритма
2. Определение верхней и нижней границ выполнения алгоритма в трассе
3. Определение известной части входных и/или выходных данных
4. Итеративное применение алгоритмов прямого и обратного слайсинга между границами выполнения алгоритма

Применение слайсинга



Возникающие сложности

Получаемые в ходе слайсинга множества входных/выходных данных, как правило, являются надмножествами входных/выходных параметров алгоритма.

Причины

- Временные переменные, куда сохраняются данными неотличимы от выходных параметров
- Ложные зависимости, связанные с архитектурными особенностями (указатель стека, сегментные регистры)

Методы решения

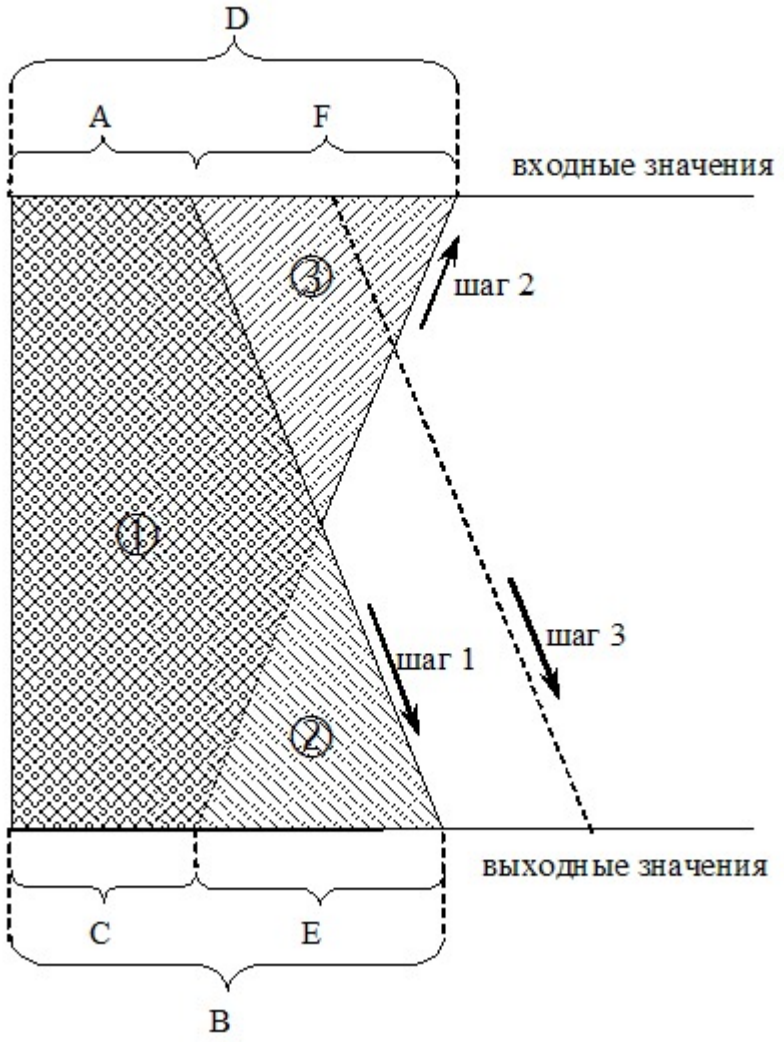
- Вспомогательные алгоритмы анализа (отсеивание неиспользуемых выходных данных)
- Фильтрация по регистрам
- Вручную

Итеративное применение слайсинга

Шаг 1. Получение выходных данных (B) по известной части входных (A)

Шаг 2. Удаление временных переменных (E) и получение входных дынных (D) по оставшимся выходным (C)

Шаг 3. Удаление ложных входных данных и повторение шага 1.

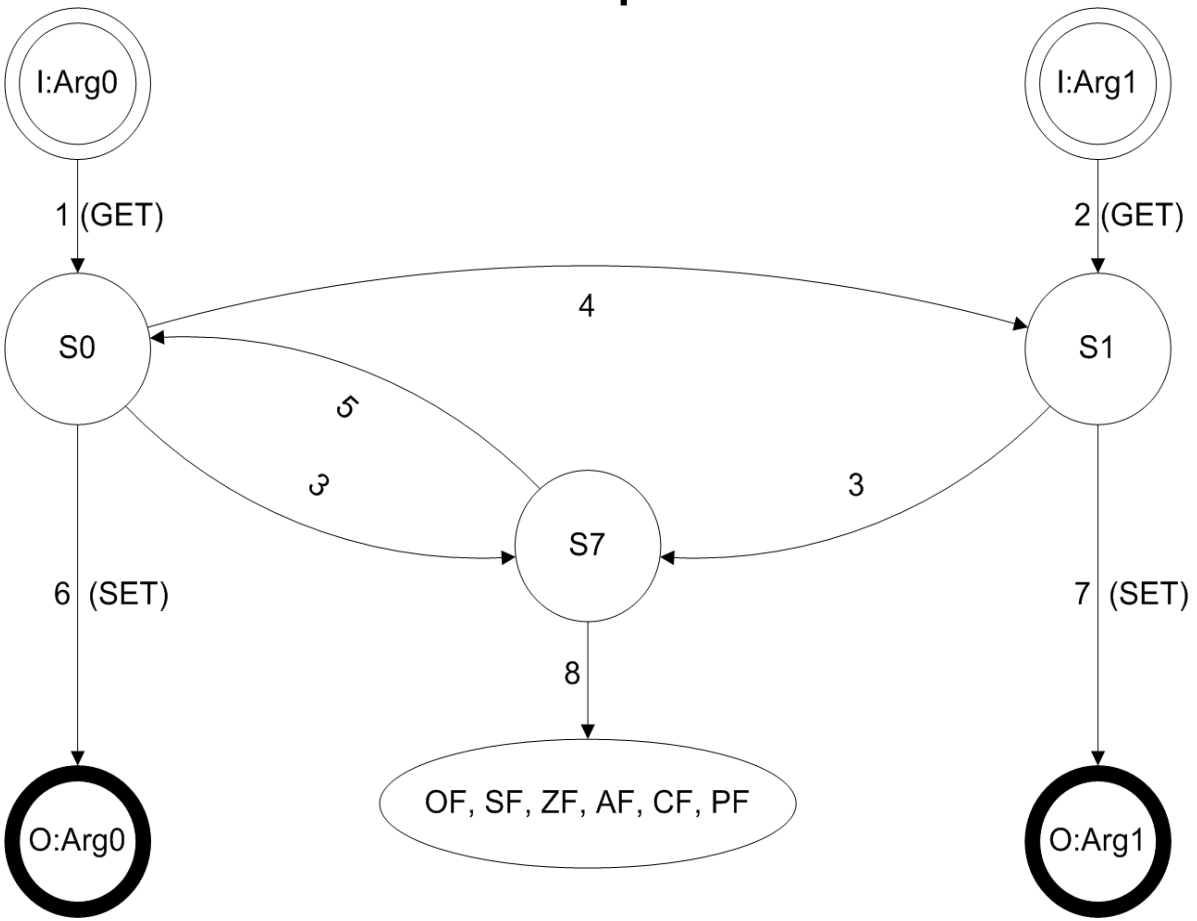


Особенности слайсинга по трассе исполнения

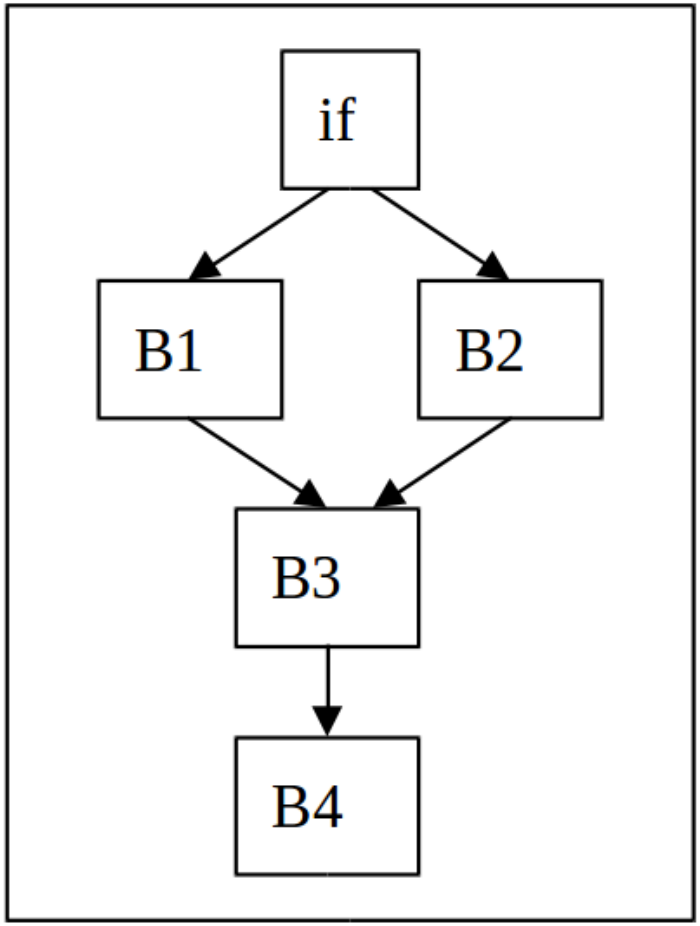
- Критерий слайсинга C(n, V)
  - n – шаг в трассе
  - V – множество ячеек памяти и регистров
- Неявные входные и выходные аргументы (PUSH, RET)
- Сложные зависимости между входными и выходными аргументами инструкций (XADD)
- Неполные зависимости по управлению из-за неполноты восстановления CFG
- Результат слайсинга – подтрасса, которую для получения алгоритма нужно «проецировать» на адресное пространство для получения алгоритма

Пример разбора инструкции: XADD

Exchanges the first operand (destination operand) with the second operand (source operand), then loads the sum of the two values into the destination operand



Зависимости по управлению



Граф потока управления

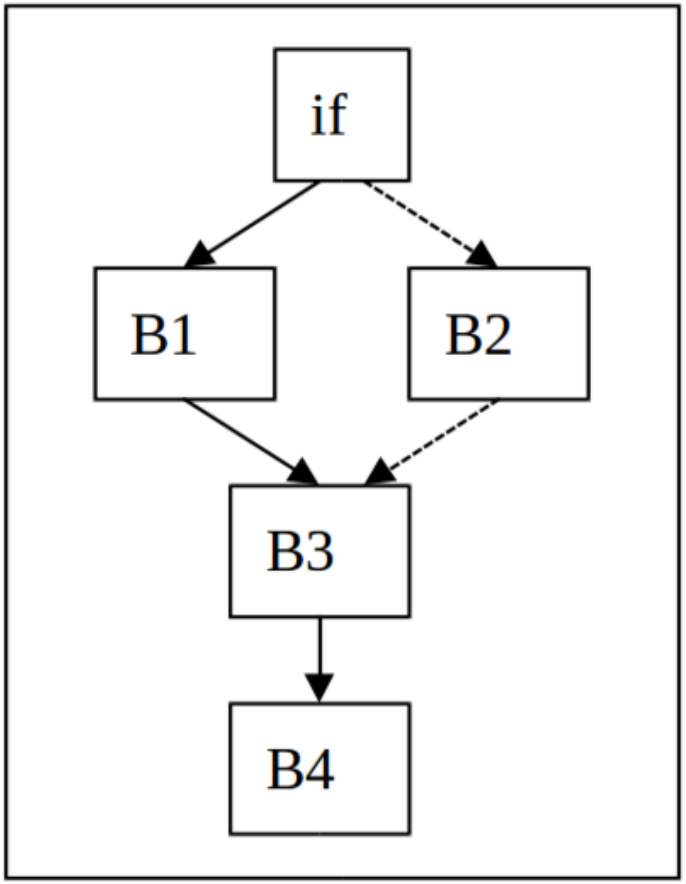
- Зависимость между инструкцией условного перехода и всеми инструкциями базовых блоков, входящих в ветви условного перехода до их слияния.
- На рисунке – зависимость между блоком if с одной стороны и инструкциями блоков B1 и B2 с другой

Определение границ влияния условного перехода

1. Узел d доминирует над узлом n, если любой путь из начального узла графа в узел n, проходит через d
2. Узел d постдоминирует над узлом n, если любой путь из конечного узла графа в узел n, проходит через d
3. Непосредственный доминатор – самый ‘близкий’ доминатор к доминируемому узлу.
4. Базовый блок, в котором сливается поток управления, разделённый в блоке if – непосредственный постдоминатор блока if.
5. Множество базовых блоков, инструкции которых зависят по управлению от блока if, это множество базовых блоков, до которых существуют пути из блока if, которые при этом не проходят его непосредственный постдоминатор.

Влияние неполноты восстановления CFG

- Отсутствуют нереализовавшиеся ветви (блок B2)
- В результате:
  - инструкции блока B1 не зависят по управлению от базового блока if



Случай реализации одной ветви

Влияние зависимостей по управлению на размер слайса

При обратном слайсе – слабое

- Добавление инструкции условного перехода и ячеек, влияющих на вычисление условий, если в слайс попала инструкция из границы влияния этого условного перехода

При прямом слайсе - сильное

- При попадании в слайс инструкции условного перехода добавляются *все инструкции* в границах влияния условного перехода и их выходные ячейки

Когда необходимо учитывать зависимости по управлению?

В случае если зависимости по данным «трансформируются» в зависимости по управлению:

Табличные преобразования

```
char toUpper(char letter)
{
switch (letter)
   {
      case ‘a’: return ‘A’;
      …
      case  ‘z’: return ‘Z’;
      default: return letter;
   }
}
```

Циклы вычисления длины

```
int length(char* str)
{
   int len;
   for (len = 0; str[i] != 0;
        len++);
   return len;
}
```

1202

АНАЛИЗ ПОМЕЧЕННЫХ ДАННЫХ. ПОИСК УТЕЧЕК КОНФИДЕНЦИАЛЬНЫХ ДАННЫХ.

Анализ помеченных данных (Taint-анализ)

1. Динамический анализ потока данных, предложенный в 2000-х годах для поиска уязвимостей в бинарном коде
2. Для проведения анализа требуется задать:
   - Источники помеченных данных (потенциально опасных)
   - Правила распространения пометок
   - Правила проверки уязвимости (потенциально опасные инструкции)

Источники помеченных данных

Могут задаваться в зависимости от конкретной системы и модели угроз:

- Есть ли подключение по сети?
- Безопасны ли данные на диске?
- Может ли злоумышленник иметь физический доступ к системе?

В зависимости от модели угроз, источниками могут быть:

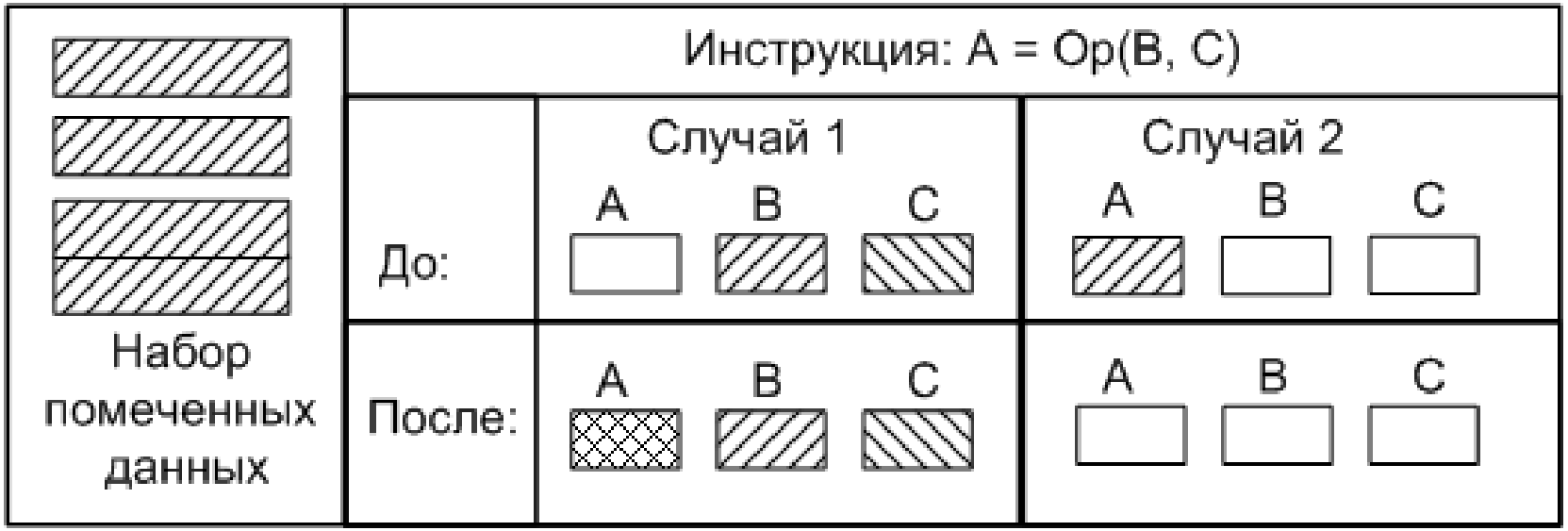
- Все или некоторые сетевые соединения (вызовы чтения данных из сокетов)
- Все или некоторые файлы на диске (вызовы чтения файлов)
- Пользовательский ввод (вызовы чтения из потока ввода)

Алгоритм распространения помеченных данных

В каждый момент времени есть набор помеченных данных – изначально инициализируется источником. После анализа каждого шага трассы набор обновляется.

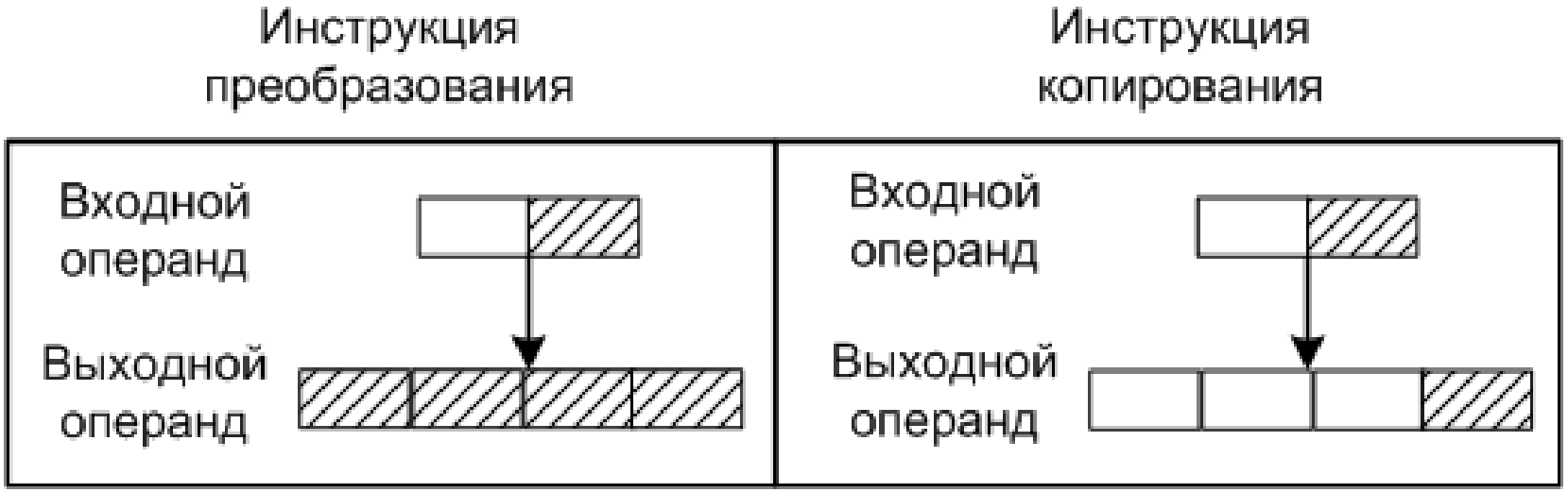
При анализе инструкции возможны 2 основных случая:

- один или более входных аргументов помечены
- все входные аргументы не помечены, а выходной - помечен



Точность распространения помеченных данных

- Типы зависимостей для распространения:
  - Зависимости по данным
  - Адресные зависимости
  - Зависимости по управлению
- Инструкции, несохраняющие зависимости: `XOR RAX,RAX`
- Уточнённое распространение зависимостей для инструкций копирования



Правила проверки уязвимости

1. Требуется задать набор инструкций и функций, исполнение и вызов которых могут приводить к уязвимости
2. Для каждой инструкции и функции определить условия, при которых уязвимость возникает – наличие пометки на одном или нескольких параметрах функции или аргументах инструкции

Пример:

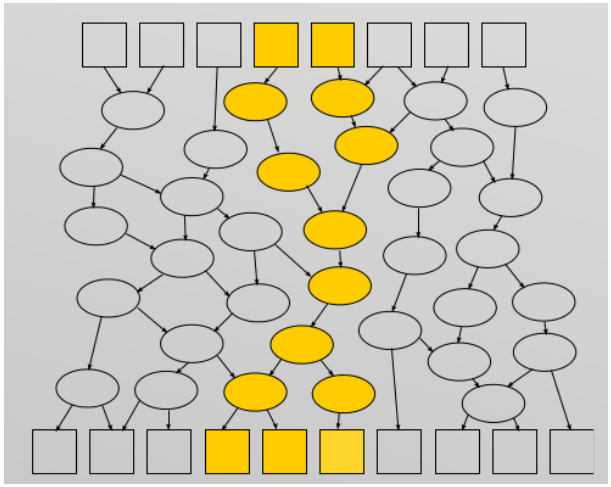
`JMP EAX` **или** `CALL EAX`, **где** `EAX` **- помечен**

Особенности анализа помеченных данных

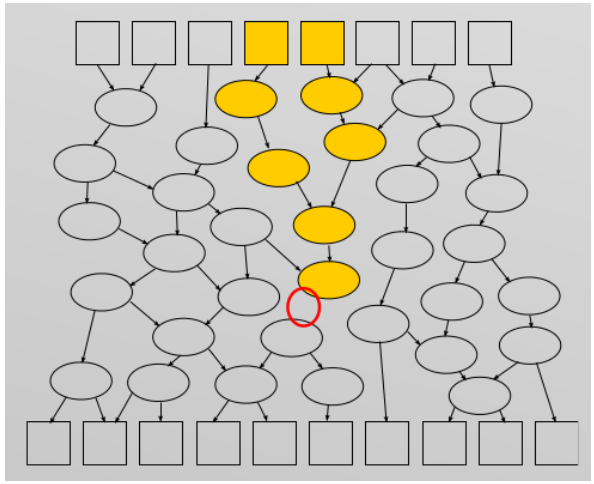
- Множество источников помеченных данных,
- Для каждого помеченного байта в каждой точке известно от каких байт каких источников он зависит
- Возможность задания гранулярности анализа – отслеживание на уровне отдельных байтов или битов
- Часто используется в онлайн режиме для предотвращения атак
- Наличие критериев остановки анализа (срабатывание проверки уязвимости) помимо исчерпания множества помеченных данных

Проблемы отслеживания зависимостей по данным

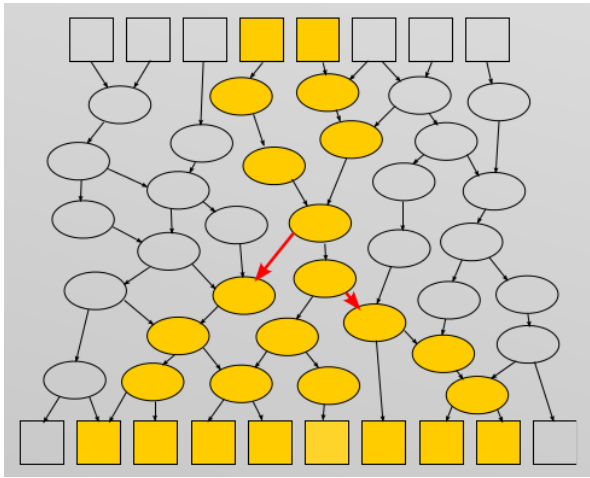
В зависимости от выбора типов отслеживаемых зависимостей может наблюдаться одна из двух ситуаций: неполнота и избыточность отобранной части программы



Истинные зависимости



Неполнота



Избыточность

Варианты использования Taint-анализа

- Поиск уязвимостей (TaintCheck[3])
- Обнаружение и анализ вредоносного ПО (TaintQemu)
- Отслеживание времени жизни чувствительных данных (TaintBochs)
- Тестирование ПО (TaintScope)
- Восстановление форматов данных (Tupni)
- Поиск утечек конфиденциальных данных (TaintDroid)

Поиск утечек конфиденциальных данных

Основная задача – предотвратить передачу данных, помеченных как «конфиденциальные» во вне системы (по сети) без их предварительного шифрования.

- Вопрос о корректности и стойкости алгоритма шифрования не рассматривается.
- Примерами «конфиденциальных» данных могут служить:
  - Пароли в оперативной памяти
  - Файлы с чувствительными данными на диске
  - Генерируемые ключи шифрования

Формальная постановка задачи

- Пусть G – граф зависимостей по данным.
- Пусть S – множество вершин, соответствующих источникам помеченных данных
- Пусть C – множество вершин, соответствующих вызовам функций шифрования
- Пусть O – множество вершин, соответствующих отправке данных в сеть.
- Требуется найти пути в графе G, такие, что они ведут из вершин множества S во множество O и при этом не проходят через вершины множества C.

Поиск утечек с помощью taint-анализа

Источники помеченных данных.

- Чтение памяти по заданным адресам (пароли)
- Вызовы функции чтения помеченных файлов
- Вызовы функций генерации ключей шифрования

Правила распространения.

- Дополнительно к базовым – прекращать распространение данных переданных в функцию шифрования

Проверка факта утечки.

- Передача помеченных данных в функцию отправки данных по сети

Литература

1. S. Cavadini. Secure Slices of Insecure Programs // ASIACCS '08 - 2004 г. – С. 112-122 Proceedings of the 2008 ACM symposium on Information, computer and communications security Pages 112-122
2. J. Caballero, N. Johnson, S. McCamant, D. Song. Binary Code Extraction and Interface Identification for Security Applications, NDSS, 2010
3. J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In Proceedings of NDSS '05, San Diego, California, USA, February 2005.

Литература к лекции
Дополнительные источники

1. L. Cavallaro, P. Saxena, R. Sekar. Anti-Taint-Analysis: Practical Evasion Techniques Against Information Flow Based Malware Defense, 2007
2. M. Kang, S. McCamant, P. Poosankam, D. Song. DTA++: Dynamic Taint Analysis with Targeted Control-Flow Propagation, 2011