Анализ кода и информационная безопасность

Лекция 09

МГУ / ВМК / СП

0901

НЕОБХОДИМОСТЬ АНАЛИЗА БИНАРНОГО КОДА

Исходный и бинарный код

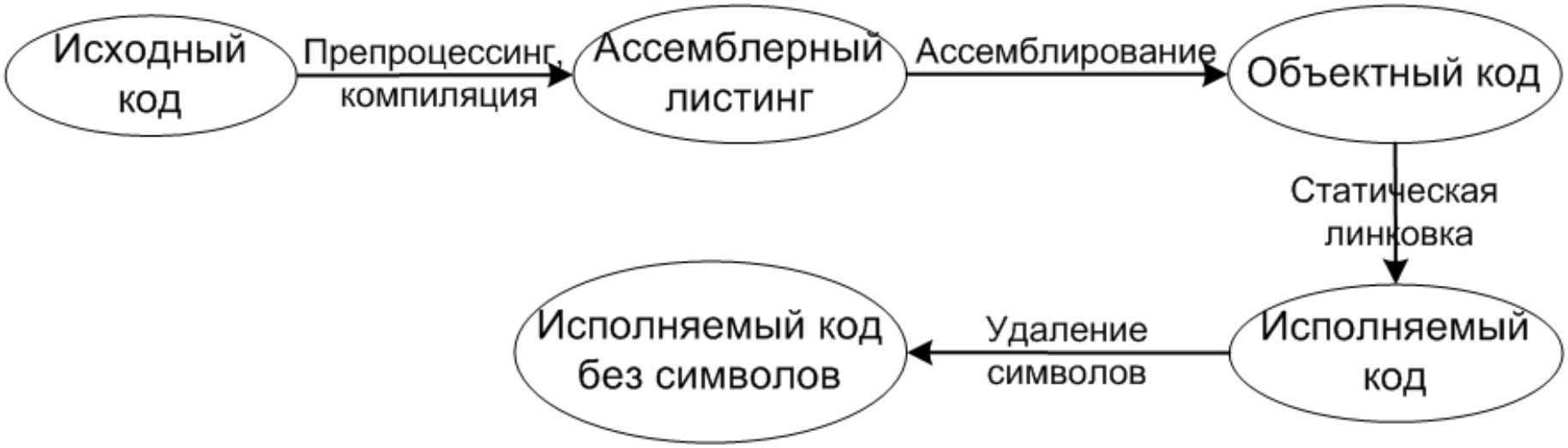
Исходный код

- Осмысленные имена.
- Комментарии.
- Объявления переменных.
- Система типов.
- Пакеты, классы и функции.
- Высокоуровневые конструкции:
  - ветвления;
  - циклы.

Бинарный код

- Адреса в памяти.
- Размещение переменных:
  - в статической памяти;
  - в куче;
  - на стеке;
  - на регистре;
  - может меняться.
- Нет системы типов.
- Статическое связывание.
- Передача управления.

Компиляция как процесс потери информации



Потерянная информация:

- Информация о типах (переменные, параметры, возвращаемые значения)
- Функции и переменные, их имена, соглашения о вызовах
- Циклы, ветвления
- Цели вызовов и переходов

Преимущества по сравнению с анализом исходного кода

- Независимость от исходного языка или их набора (зависимость от целевой архитектуры)
- «Вставки ассемблерного кода» становятся частью программы
- Не возникает проблем со сборкой исходного кода – компилятор, окружение
- Анализируются фактические библиотечные вызовы, а не «заглушки» реализованные вручную

Общие цели и задачи

Защита информации

- Поддержание работоспособности ПО.
- Соблюдение политик безопасности.
- Сертификация ПО, НДВ.

Анализ

- Анализ реализации.
- Несанкционированный доступ к данным.
- Отказ в обслуживании.

Типовые задачи

- Восстановление алгоритмов и форматов данных.
- Поиск ошибок.
- Сопоставление модели и ее реализации.

Оборудование

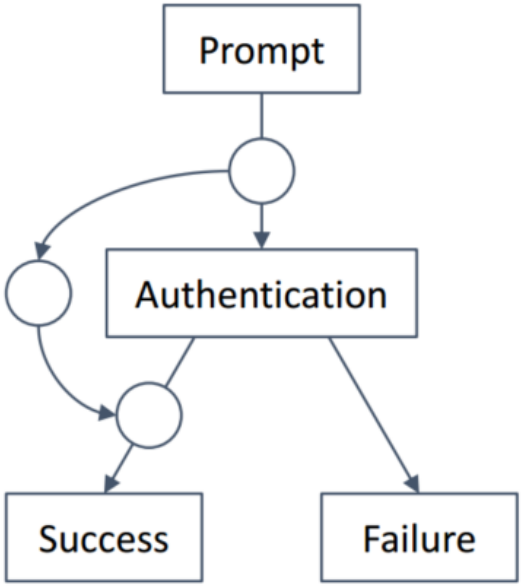
- Персональные компьютеры, сервера.
- Коммуникационное оборудование.
- Микроконтроллеры.
- Мобильные устройства.

Зачем анализировать бинарный код?

- Отсутствие исходного кода
  - Наследственное ПО
  - Проприетарные программы и библиотеки
  - Вредоносное ПО
- Бинарный код – то, что выполняется
  - Прошивки встроенных систем
  - Изменения вносимые при сборке : What You See Is Not What You eXecute (WYSINWYX)
- Условия анализа
  - Нет исходного кода и отладочной информации.
  - Противодействие анализу.
  - Анализируемая система может быть сложной и распределённой.
  - Анализ кода системных сервисов, драйверов, ядра ОС, firmware.

Backdoor: “Задняя дверь”

- Встроенная Linux система:
  - HTTP сервер для видеомониторинга с известным backdoor




```
LDR     R1, =a3sadmin ; "3sadmin"
MOV     R0, R7  ; s1
BL      strcmp
CMP     R0, #0
LDR     R1, =a27988303 ; "27988303"
MOV     R0, R4  ; s1
```

➔ Username: 3sadmin

➔ Password: 27988303

WYSINWYX: классический пример

- Windows
  - Процесс аутентификации(login) оставлял пароль пользователя в куче после завершенния
- Минимизации времени жизни:
  - очистка буфера
  - освобождение памяти
- Оптимизация со стороны компилятора
  - «Удаление бесполезного кода»

```
memset(password, '\0', len);
free(password);
```

$\Longrightarrow$

```
free(password);
```

WYSINWYX: вклад неопределённого поведения

- Проверка переполнения буфера

```
char *buf = ...;
char *buf_end = ...;
unsigned int len = ...;
if (buf + len >= buf_end)
    return;   /* len too large */
if (buf + len < buf)
    return;   /* overflow, buf+len wrapped around */
/* write to buf[0..len-1] */
```

  - Компилятор может удалить второй if, так как он не может выполниться без возникновения «неопределённого поведения»

WYSINWYX: совсем простой пример

- Неинициализированная переменная

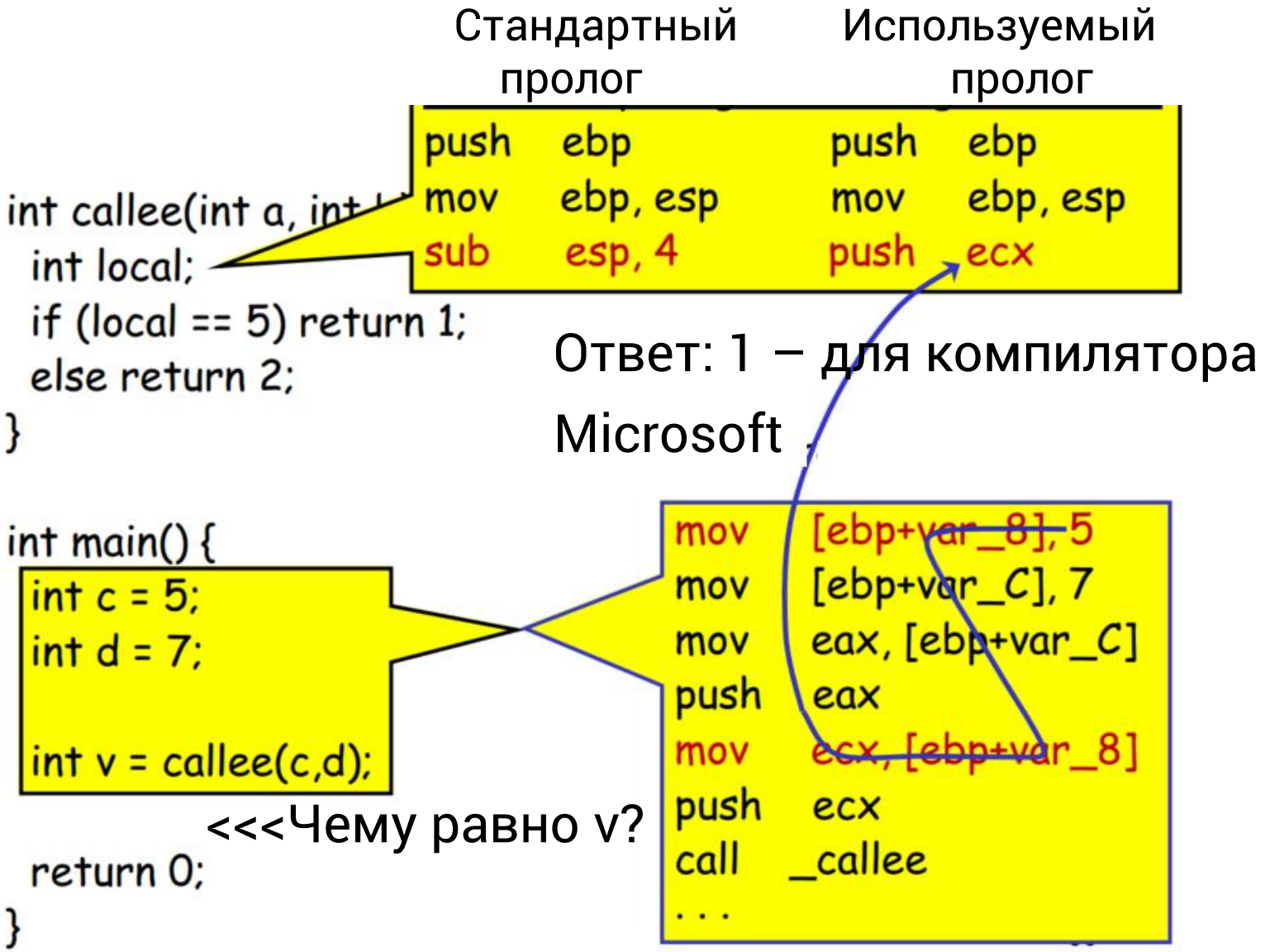
```
int callee(int a, int b) {
  int local;
  if (local == 5) return 1;
  else return 2;
}

int main() {
  int c = 5;
  int d = 7;

  int v = callee(c,d);
  return 0;
}
```

<<<Чему равно v?

WISINWYX: совсем простой пример (2)



Преимущества поиска уязвимостей в бинарном коде

- Многие эксплойты используют платформо-специфические особенности и особенности компиляторов:
  - Выравнивание памяти
  - Паддинг между полями структуры
  - Соседнее рамзмещение переменных в памяти
  - Использование регистров
  - Используемые оптимизации
  - Ошибки в компиляторе

0902

ЗАДАЧИ В АНАЛИЗЕ БИНАРНОГО КОДА. СТАТИЧЕСКИЙ ПОДХОД.

Общая схема анализа

1. Получить сам предмет изучения (бинарный код)
2. Восстановить абстракции необходимые для анализа (дизассемблирование)
3. Восстановить потоки управления и данных
4. Восстановление типов данных и высокоуровневых конструкций (декомпиляция)
5. Запуск специализированных алгоритмов анализов:
   - Восстановление алгоритма
   - Поиск ошибок
   - …

Получение предмета анализа

Предмет анализа

- Набор исполняемых файлов и библиотек
- Дамп памяти процесса
- Прошивка встроенной системы

Особенности

- Работа в виде нескольких взаимодействующих процессов/нитей
- Распределённая система (клиент/сервер)
- Распаковка исполняемого кода в процессе работы
- Запуск распаковки только при определённых условиях (защита от анализа)
- Получение прошивки: пакет обновления, отладочный интерфейс, …

Восстановление абстракций, необходимых для анализа

Абстракции:

- Инструкции, опкоды, операнды
- Функции
- Базовые блоки
- Переменные, параметры функций, их типы
- Высокоуровневые конструкции (if-then, switch-case, for,…)

Возникающие сложности:

- Разделение кода и данных (Архитектура Фон-Неймана)
- Переменный размер инструкций x86
- Переходы по вычисляемым адресам
- Функции без явных вызовов и возвратов
- Динамический код

Бинарный код: начальное представление

Исходный код

```
int
main(void)
{
    puts("Hello, world!");
    return 0;
}
```

Бинарный код

```
55 89 E5 83-E4 F0 83 EC
10 E8 EA FA-FF FF C7 04
24 60 30 40-00 E8 E6 FA
FF FF 31 C0-C9 C3 90 90
```

Выделение инструкций

Исходный код

```
int
main(void)
{
    puts("Hello, world!");
    return 0;
}
```

Бинарный код

```
55|89 E5|83 E4 F0|83 EC
10|E8 EA FA FF FF|C7 04
24 60 30 40 00|E8 E6 FA
FF FF|31 C0|C9|C3|90|90
```

Выделение опкодов и операндов

Исходный код

```
int
main(void)
{
    puts("Hello, world!");
    return 0;
}
```

Бинарный код

```
PUSH    EBP
MOV EBP, ESP
AND ESP, 0FFFFFFF0h
SUB ESP, 10h
CALL    401228h
MOV DWORD [ESP], 403060h
CALL    401230h     # _puts
XOR EAX, EAX
LEAVE
RET
NOP
NOP
```

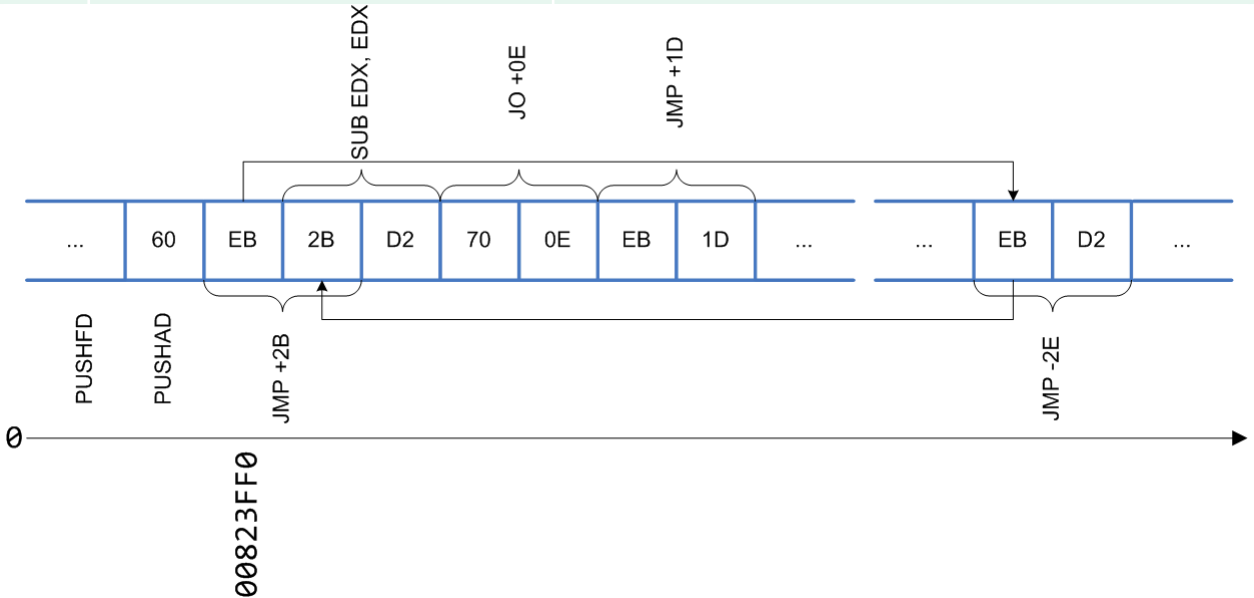
Восстановление потока управления и дизассемблирование

- Требуется отделять код от данных: нужны адреса входа в функции и адреса перехода
- Известен адрес начальной функции и адреса из таблицы экспорта
- В процессе дизассемблирования добавляются адреса вызовов и адреса переходов
- Проблемы:
  - вычисляемые адреса выходов и переходов -> часть кода не дизассемблирована
  - Отсутствие кода -> отсутствие рёбер -> неполный и неточный граф потока управления
  - Отсутствие рёбер -> отсутствие кода “наложенных инструкций”
  - Отсутствие кода распаковки -> отсутствие распакованного кода

Наложение инструкций

Адрес	Машинная команда	Ассемблерная инстру	
00823FEF	60	PUSHAD	
00823FF0	EB2B	JMP	0082401Dh
0082401D	EBD2	JMP	00823FF1h
00823FF1	2BD2	SUB	EDX, EDX
00823FF3	700E	JO	00824003h
00823FF5	EB1D	JMP	00824014h

Трасса выполнения



Что такое функция?

Функций в языках высокого уровня

- Имя
- Множество параметров
- Возвращаемое значение
- Тело функции
- Известные точки вызова

Функции в бинарном коде

- «Тело функции» может быть фрагментировано, общие части кода нескольких функций
- Любая передача управления – потенциальный вызов
- Функция возвращает управление на инструкцию, следующую за вызовом функции
- Существуют исключения: exit(), fork() и т.д.

Восстановление типов данных

- Данные в программе: входные/выходные данные, параметры функций, локальные/глобальные переменные
- Входные/выходные данные:
  - Пользовательский ввод/вывод
  - Файлы
  - Сетевые пакеты
- Тип данных:
  - Размер данных в байтах/битах
  - Целочисленность/дробность, знаковость данных
  - Структуры, массивы, указатели
  - Сложные (ссылочные) типы данных: списки, деревья, ...

Задачи декомпиляции

Восстановление программы на языке высокого уровня:

- Восстановление переменных и их типов данных
- Восстановление потока управления и управляющих конструкций (if, switch, for, …)
- Восстановление функций, их параметров и возвращаемых значений
- Восстановление классов, виртуальных таблиц и т.д.

Необходимо учитывать оптимизации:

- Разворачивание циклов
- Инлайнинг функций
- Оптимизацию возвратов из листовых функций и др.

Пример декомпиляции

Исходная программа	Восстановленная программа

int f(int* a, int n){
  int *p;
  int s=0;
  for(p=a; p<a+n; p++) {
  s += *p;
  }
  return s;
}
	int __cdecl sub_401290(int a1, int a2)
	{
	  unsigned int v2; // edx@1
	  int v3; // ecx@1
	  v3 = 0;
	  v2 = a1;
	  while (a1 + 4 * a2 > v2)
	  {
	  v3 += *(_DWORD *)v2;
	  v2 += 4;
	  }

Качество декомпиляции

Основные артефакты, ухудшающие читаемость кода:

•Неточность восстановления управляющих конструкций

- Операторы goto
- Операторы break в цикле
- Операторы continue в цикле
- Ассемблерные вставки

•Неточность восстановления типов данных

- Операции явного приведения типов
- Типы данных union
- Невосстановление структур, массивов, укзателей
- Адресная арифметика вместо доступа по индексу

Достоинства и недостатки статического подхода

- Достоинства
  - Статическое представление программы – естественное для человеческого мышления
  - Потенциально - исследование всей программы
  - Не требуется возможность запуска
- Недостатки
  - Косвенная адресация и динамическое изменение кода
  - Большое количество защит от анализа: обфускация, защита от дизассемблирования, динамическая распаковка
  - Как правило, анализируется отдельный модуль (IDA Pro)
  - Большинство видов анализа не входит в базовый набор - необходимость написания или покупки расширений (в случае IDA Pro – IDC-скрипты и модули-расширения, например, Zynamics, поглощенный Google)

Статический анализ и динамический анализ

Статический анализ

- Анализ программы «в целом».
- Программа не запускается.
- Общие выводы о свойствах программы.
- Естественное для человеческого мышления представление.

Динамический анализ

- Анализ программы в контексте заданных входных данных.
- Программа запускается
  - на реальной аппаратуре;
  - в симуляторе.
- Выводы об отдельных запусках.
- Больше информации.

Инструменты статического и динамического анализа

Статический анализ

- Двоичный редактор.
- Дизассемблер:
  - IDA Pro.
- Декомпилятор:
  - HexRays.
- Среда анализа:
  - IDA Pro + плагины;
  - CodeSurfer/x86.

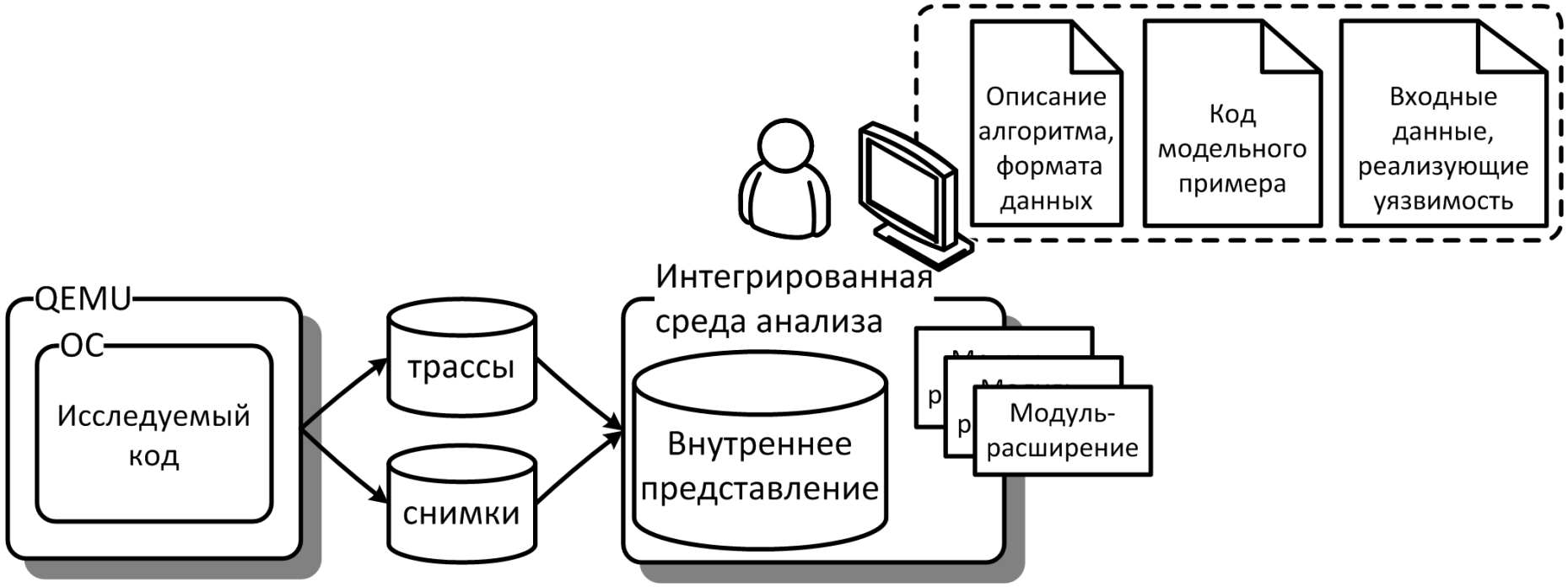
Динамический анализ

- Отладчик:
  - OllyDbg;
  - SoftICE.
- Симулятор с отладчиком:
  - QEMU + GDB;
  - VMware + Visual Studio.
- Трассировщик:
  - «онлайн»-анализ;
  - **«оффлайн»-анализ.**

0903

ДИНАМИЧЕСКИЙ ПОДХОД. АНАЛИЗ ТРАСС.

Методика динамического оффлайн-анализа



Трасса выполнения

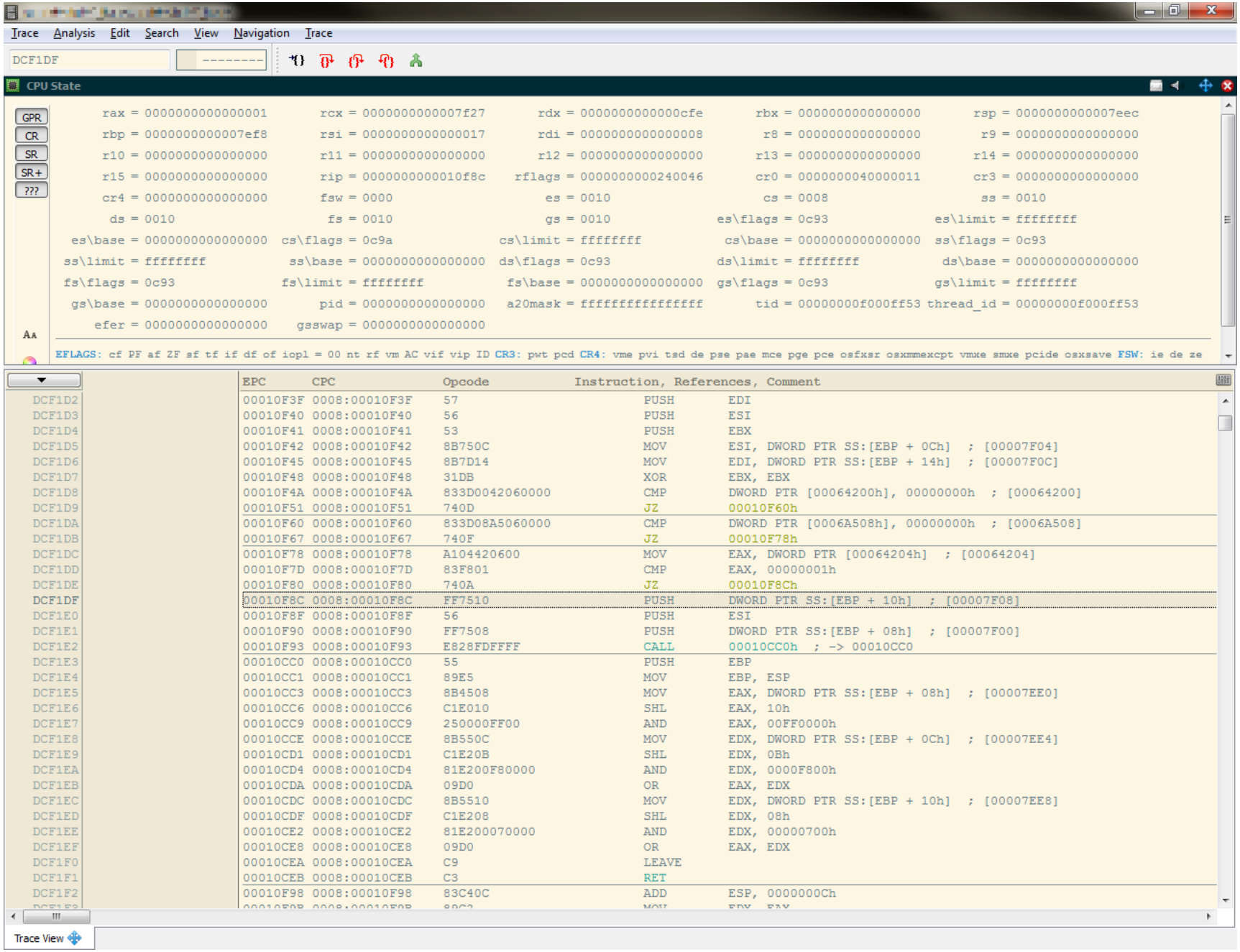
- Последовательность состояний машины во времени.
- Состояния всей системы, без пропусков.
- Шаг содержит:
  - код машинной команды;
  - значения основных регистров.
- Асинхронные события:
  - исключения;
  - прерывания.

Средства сбора трассы

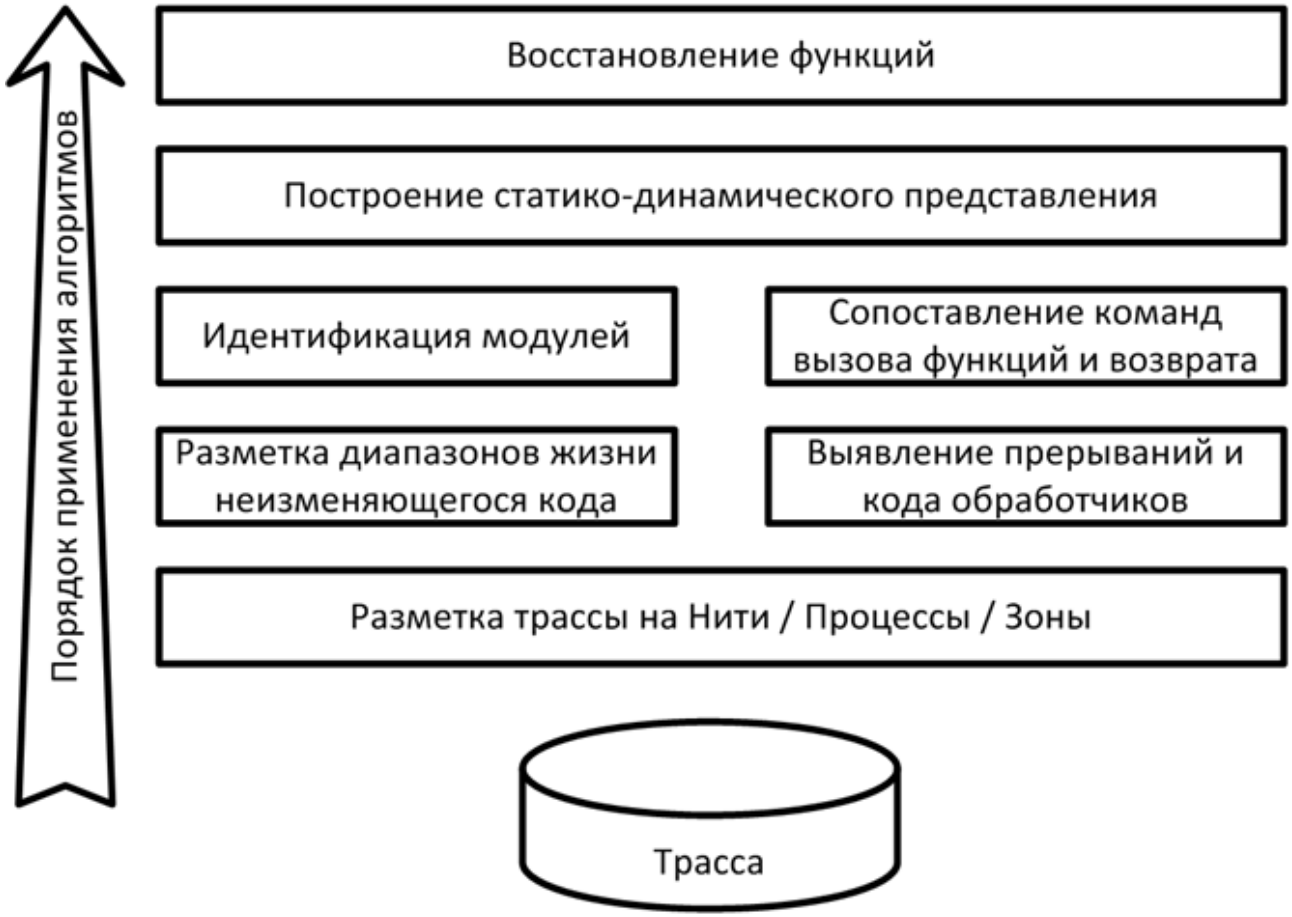
- Программный эмулятор (интерпретатор):
  - AMD SimNow;
  - Bochs.
- Бинарная трансляция:
  - QEMU;
  - VMware Workstation.
- Аппаратная виртуализация:
  - AMD SVM;
  - Intel VT-x.

Особенности сбора трассы

- Большой объём трасс
- Большое замедление (несколько десятичных порядков).
- Сетевые программы перестают работать.
- Решение: трассировка в два прохода.
  - На первом проходе записывается легковесный журнал внешних событий.
  - На втором проходе строится полная трасса.


Trace Analysis Edit Search View Navigation Trace
DCF1DF
CPU State
GPR
CR
SR
SR+
???
rax = 0000000000000001 rcx = 0000000000007f27 rdx = 0000000000000cfe rbx = 0000000000000000 rsp = 00000000000007eec
rbp = 00000000000007ef8 rsi = 0000000000000017 rdi = 0000000000000008 r8 = 0000000000000000 r9 = 0000000000000000
r10 = 0000000000000000 r11 = 0000000000000000 r12 = 0000000000000000 r13 = 0000000000000000 r14 = 0000000000000000
r15 = 0000000000000000 rip = 000000000010f8c rflags = 0000000000240046 cr0 = 0000000040000011 cr3 = 0000000000000000
cr4 = 0000000000000000 fsw = 0000 es = 0010 cs = 0008 ss = 0010
ds = 0010 fs = 0010 gs = 0010 es\flags = 0c93 es\limit = ffffffff
es\base = 0000000000000000 cs\flags = 0c9a cs\limit = ffffffff cs\base = 0000000000000000 ss\flags = 0c93
ss\limit = ffffffff ss\base = 0000000000000000 ds\flags = 0c93 ds\limit = ffffffff ds\base = 0000000000000000
fs\flags = 0c93 fs\limit = ffffffff fs\base = 0000000000000000 gs\flags = 0c93 gs\limit = ffffffff
gs\base = 0000000000000000 pid = 0000000000000000 a20mask = ffffffffffffffff tid = 00000000f000ff53 thread_id = 00000000f000ff53
efer = 0000000000000000 gsswap = 0000000000000000
EFLAGS: cf PF af ZF sf tf if df of iopl = 00 nt rf vm AC vif vip ID CR3: pwt pcd CR4: vme pvi tsd de pse pae mce pge pce osfxsr osxmmexcpt vmxe smxe pcide osxsave FSW: ie de ze
EPC CPC Opcode Instruction, References, Comment
DCF1D2 00010F3F 0008:00010F3F 57 PUSH EDI
DCF1D3 00010F40 0008:00010F40 56 PUSH ESI
DCF1D4 00010F41 0008:00010F41 53 PUSH EBX
DCF1D5 00010F42 0008:00010F42 8B750C MOV ESI, DWORD PTR SS:[EBP + 0Ch] ; [00007F04]
DCF1D6 00010F45 0008:00010F45 8B7D14 MOV EDI, DWORD PTR SS:[EBP + 14h] ; [00007F0C]
DCF1D7 00010F48 0008:00010F48 31DB XOR EBX, EBX
DCF1D8 00010F4A 0008:00010F4A 833D0042060000 CMP DWORD PTR [00064200h], 00000000h ; [00064200]
DCF1D9 00010F51 0008:00010F51 740D JZ 00010F60h
DCF1DA 00010F60 0008:00010F60 833D08A5060000 CMP DWORD PTR [0006A508h], 00000000h ; [0006A508]
DCF1DB 00010F67 0008:00010F67 740F JZ 00010F78h
DCF1DC 00010F78 0008:00010F78 A104420600 MOV EAX, DWORD PTR [00064204h] ; [00064204]
DCF1DD 00010F7D 0008:00010F7D 83F801 CMP EAX, 00000001h
DCF1DE 00010F80 0008:00010F80 740A JZ 00010F8Ch
DCF1DF 00010F8C 0008:00010F8C FF7510 PUSH DWORD PTR SS:[EBP + 10h] ; [00007F08]
DCF1E0 00010F8F 0008:00010F8F 56 PUSH ESI
DCF1E1 00010F90 0008:00010F90 FF7508 PUSH DWORD PTR SS:[EBP + 08h] ; [00007F00]
DCF1E2 00010F93 0008:00010F93 E828FDFFFF CALL 00010CC0h ; -> 00010CC0
DCF1E3 00010CC0 0008:00010CC0 55 PUSH EBP
DCF1E4 00010CC1 0008:00010CC1 89E5 MOV EBP, ESP
DCF1E5 00010CC3 0008:00010CC3 8B4508 MOV EAX, DWORD PTR SS:[EBP + 08h] ; [00007EE0]
DCF1E6 00010CC6 0008:00010CC6 C1E010 SHL EAX, 10h
DCF1E7 00010CC9 0008:00010CC9 250000FF00 AND EAX, 00FF0000h
DCF1E8 00010CCE 0008:00010CCE 8B550C MOV EDX, DWORD PTR SS:[EBP + 0Ch] ; [00007EE4]
DCF1E9 00010CD1 0008:00010CD1 C1E20B SHL EDX, 0Bh
DCF1EA 00010CD4 0008:00010CD4 81E200F80000 AND EDX, 0000F800h
DCF1EB 00010CDA 0008:00010CDA 09D0 OR EAX, EDX
DCF1EC 00010CDC 0008:00010CDC 8B5510 MOV EDX, DWORD PTR SS:[EBP + 10h] ; [00007EE8]
DCF1ED 00010CDF 0008:00010CDF C1E208 SHL EDX, 08h
DCF1EE 00010CE2 0008:00010CE2 81E200070000 AND EDX, 00000700h
DCF1EF 00010CE8 0008:00010CE8 09D0 OR EAX, EDX
DCF1F0 00010CEA 0008:00010CEA C9 LEAVE
DCF1F1 00010CEB 0008:00010CEB C3 RET
DCF1F2 00010F98 0008:00010F98 83C40C ADD ESP, 0000000Ch
Trace View

Повышение уровня представления



Разметка трассы: процессы, потоки, зоны

- «К какому процессу/потоку/зоне относится данный шаг?»
- Разные операционные системы, полное отсутствие операционной системы.
- В общем случае: вычисляем определённое выражение на каждом шаге для PID, TID, ZID.

Разметка поколений кода

- Код по заданному адресу может меняться с течением времени:
  - динамическая загрузка библиотек;
  - самомодификация кода.
- В рамках одного поколения выполняемый код статичен.
- Множества чтения, записи, исполнения.
- Снимок выполнявшегося кода.

Сопоставление вызовов и возвратов

- В идеальном мире каждой команде вызова подпрограммы должна соответствовать команда возврата.
- В реальности:
  - возврат сразу из нескольких функций;
  - использование нестандартных команд;
  - прерывания.

Восстановление структуры кода

- Распознавание модулей:
  - производится в каждом поколении отдельно;
  - неизвестен адрес загрузки модуля;
  - формат модуля на диске и в памяти отличается;
  - релокации;
  - спекулятивное распознавание.
- Восстановление потоков управления.
- Распознавание функций.

Построение статического представления

- Отсутствует проблема разделения кода и данных
- Проблема неполного покрытия кода трассой
- Восстановление рёбер, незадействованных в трассе, если оба базовых блока известны
- Слияние представлений, полученных для связанных трасс:
  - несколько запусков на различных входных данных с общего начального состояния.

.

Выделение кода алгоритма

Задача

- выделить код, обрабатывающий входные данные;
- выделить код, формирующий выходные данные.

Подход

- Слайсинг трассы
- Критерий — шаг трассы, множество регистров и ячеек памяти.

Другие применения

- локализация алгоритма;
- восстановление буфера.

Наиболее важные методы анализа бинарного кода

- Дизассемблирование
- Восстановление функций
- Восстановление потока управления
- Выделение алгоритма с помощью слайсинга

Литература к лекции

Основные источники

1. В.А. Падарян, А.И. Гетьман, М.А. Соловьев, М.Г. Бакулин, А.И. Борзилов, В.В. Каушан, И.Н. Ледовских, Ю.В. Маркин, С.С. Панасенко. Методы и программные средства, поддерживающие комбинированный анализ бинарного кода. Труды Института системного программирования РАН Том 26. Выпуск 1. 2014 г. Стр. 251-276.
2. G. Balakrishnan and T. Reps. WYSINWYX: What You See Is Not What You eXecute. TOPLAS, 32(6), 2010.
3. Трошина Е. Н., Чернов А. В. Инструментальная среда восстановления исходного кода программы - декомпилятор TyDec // Прикладная информатика. 2010. №4.
4. C. Eagle. The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler // No Starch Press San Francisco, CA, USA ©2008

Литература к лекции
Дополнительные источники

1. X.Wang, N. Zeldovich, M. F. Kaashoek, A. Solar-Lezama. Towards optimization-safe systems: analyzing the impact of undefined behavior // In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, 2013, 260-275
2. D. Gopan, T. Reps. Low-level library analysis and summarization. In Proceedings of the 19th international conference on Computer aided verification, 2007, 68-81.