Динамический анализ бинарного кода

Падарян Вартан
vartan@ispras.ru

Информационная безопасность

- Information security → Computer security → Software security
- Компьютеры работают в сети
- Защита по периметру
- Тотальный анализ кода программ
  - статический
  - динамический
- Защита / Анализ

Цели

Защита информации

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

- Отсутствие ошибок → поиск ошибок во время разработки ПО и сертификации

Анализ

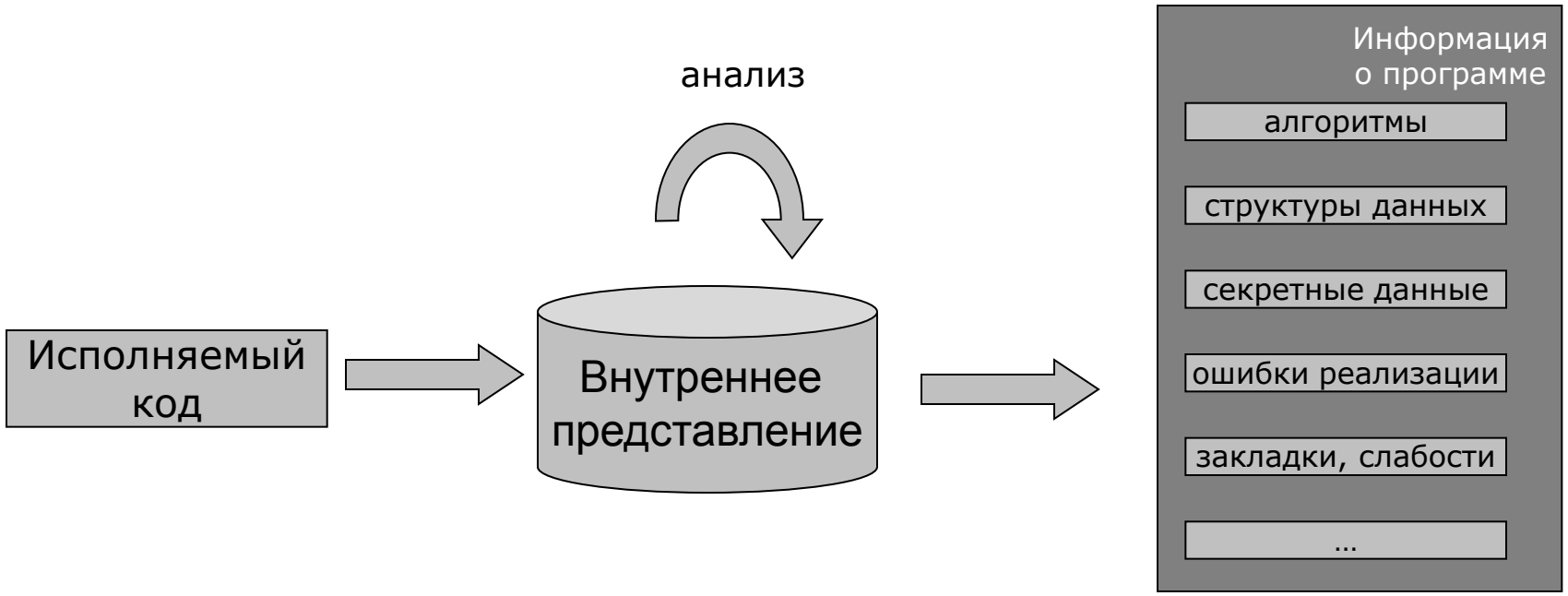
- Анализ реализации
- Несанкционированный доступ к данным
- Отказ в обслуживании

- Наличие ошибок → анализ ПО: поиск ошибок

Задачи

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

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



- Отсутствие исходных текстов.
- Противодействие со стороны анализируемой системы.
  - Антиотладочные приемы
  - Обфускация
  - Шифрование кода и данных
- Распределенность анализируемой системы по нескольким компьютерам.
- «Незамкнутость» анализируемой системы.

Методы защиты от анализа

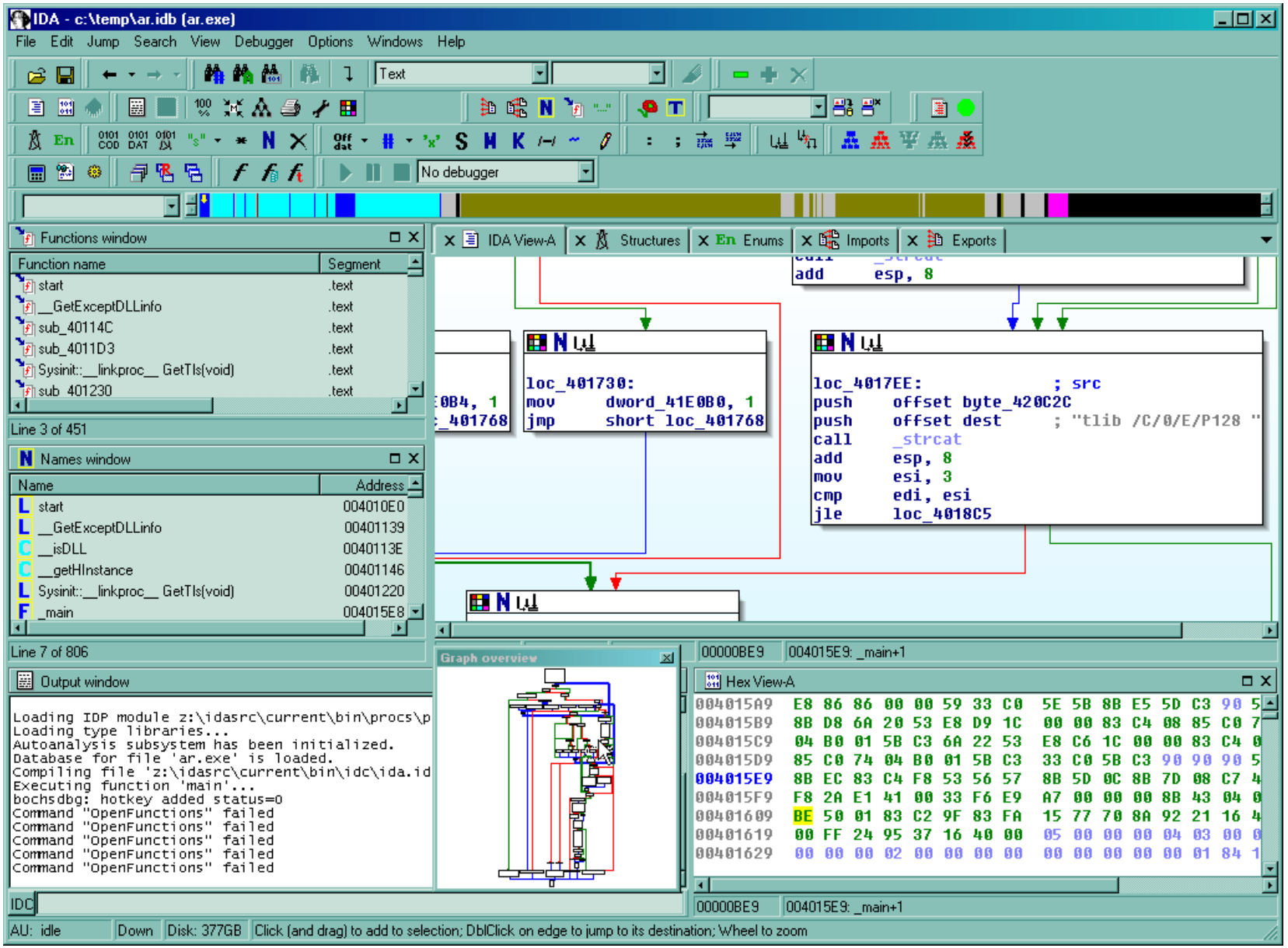
- Навесная
  - Упаковка/шифрование исполняемого файла
    - Файлов полученных в бинарном виде
    - Как последний этап защиты после компиляции из исходных кодов, с учетом дополнительной информации полученной при компиляции
- Встраиваемая
  - Запутывание программного кода во время компиляции
  - Встраивание различных приемов, защиты от тестирования отладчиками
- Виртуальная машина
  - Встраивание интерпретатора инструкций сторонней архитектуры и исполнение на нем частей программы

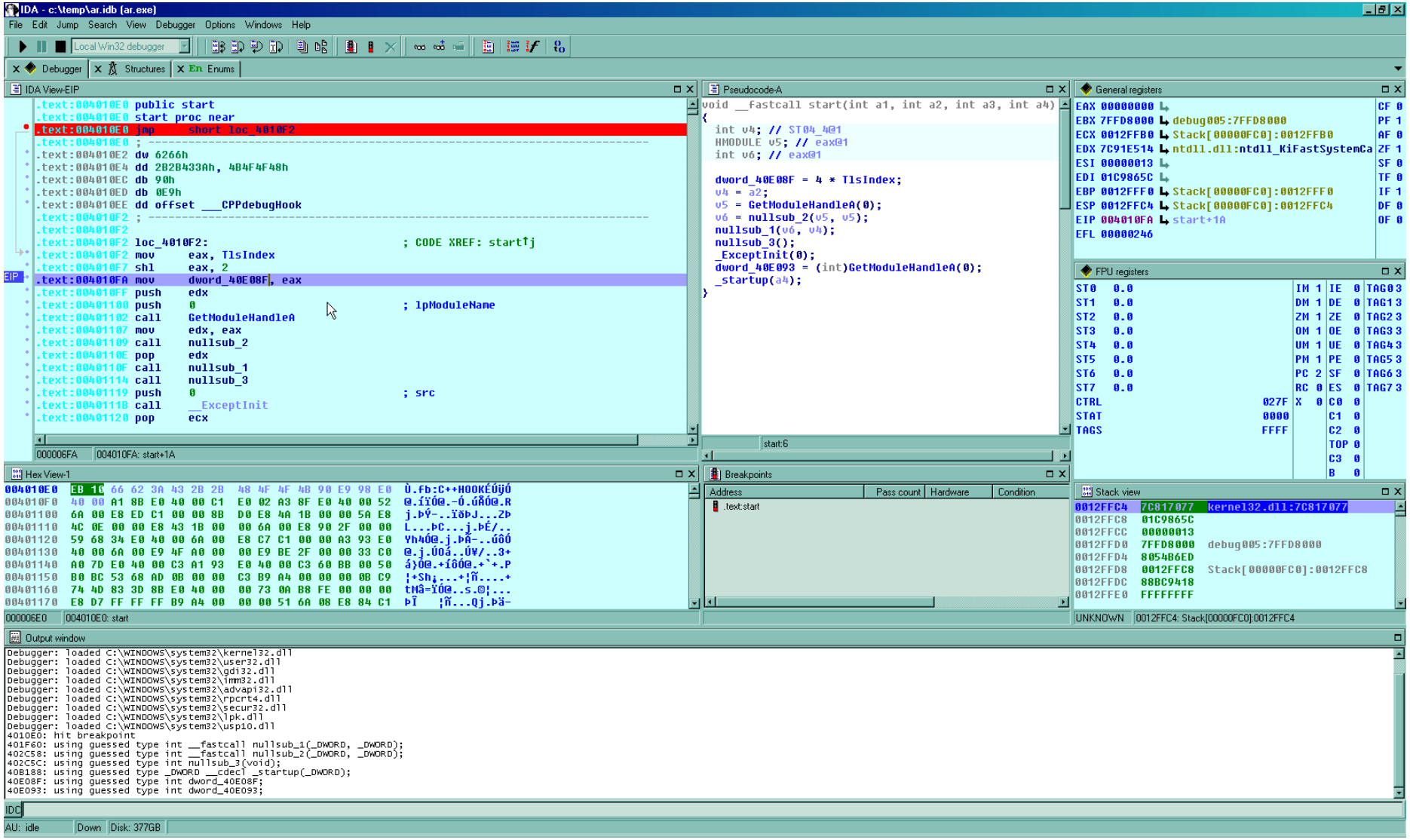
Средства анализа

- Статический анализ
  - Двоичный редактор
  - Дизассемблер (IDA Pro)
  - Декомпилятор (IDA Pro/HexRays)
  - Среда анализа (IDA Pro, CodeSurfer/x86)
  - Декомпилятор
- Динамический анализ
  - Отладчик (SoftICE, OllyDbg)
  - Трассировщик
- Bitblaze, CodeSurfer/x86, S2E …

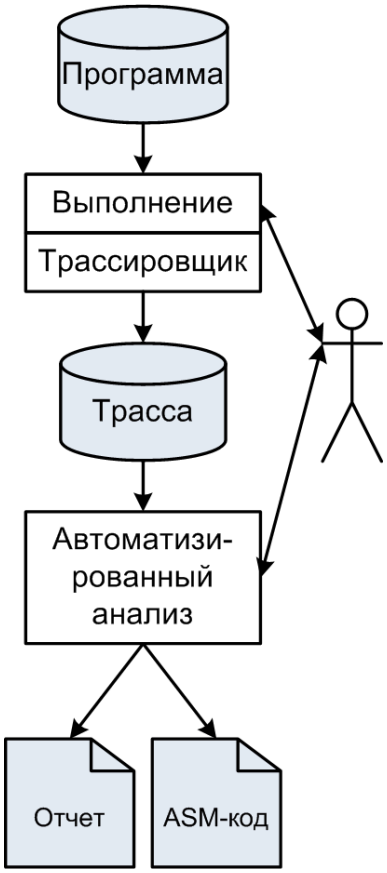
IDA Pro

- Интерактивность работы
- Встроенный язык программирования IDC
- Открытая и модульная архитектура
- Возможность работы практически со всеми популярными процессорами (список из 114 наименований)
- Возможность работы практически со всеми популярными форматами файлов (список из 37 наименований)
- Развитая система навигации
- Система типов и параметров функций
- Поддержка работы со структурами данных высокого уровня: массивами, структурами, перечисляемыми типами
- Отладчик для Win32
- Распознавание стандартных библиотечных функций (технология FLIRT)
- Декомпилятор Hex Rays


IDA - c:\temp\ar.idb (ar.exe)
File Edit Jump Search View Debugger Options Windows Help
Text
No debugger
Functions window
Function name
Segment
start .text
__GetExceptDLLinfo .text
sub_40114C .text
sub_4011D3 .text
Sysinit::__linkproc__ GetTls(void) .text
sub_401230 .text
Line 3 of 451
Names window
Name
Address
start 004010E0
__GetExceptDLLinfo 00401139
__isDLL 0040113E
__getHInstance 00401146
Sysinit::__linkproc__ GetTls(void) 00401220
_main 004015E8
Line 7 of 806
IDA View-A
Structures
Enums
Imports
Exports
add esp, 8
loc_401730:
mov dword_41E0B0, 1
jmp short loc_401768
loc_4017EE: ; src
push offset byte_420C2C
push offset dest ; "tlib /C/0/E/P128 "
call _strcat
add esp, 8
mov esi, 3
cmp edi, esi
jle loc_4018C5
Graph overview
00000BE9 004015E9: _main+1
Output window
Loading IDP module z:\idasrc\current\bin\procs\p
Loading type libraries...
Autoanalysis subsystem has been initialized.
Database for file 'ar.exe' is loaded.
Compiling file 'z:\idasrc\current\bin\idc\ida.id
Executing function 'main'...
bochsdbg: hotkey added status=0
Command "OpenFunctions" failed
Command "OpenFunctions" failed
Command "OpenFunctions" failed
Command "OpenFunctions" failed
Command "OpenFunctions" failed
Hex View-A
004015A9 E8 86 86 00 00 59 33 C0 5E 5B 8B E5 5D C3 90 5
004015B9 8B D8 6A 20 53 E8 D9 1C 00 00 83 C4 08 85 C0 7
004015C9 04 B0 01 5B C3 6A 22 53 E8 C6 1C 00 00 83 C4 0
004015D9 85 C0 74 04 B0 01 5B C3 33 C0 5B C3 90 90 90 5
004015E9 8B EC 83 C4 F8 53 56 57 8B 5D 0C 8B 7D 08 C7 4
004015F9 F8 2A E1 41 00 33 F6 E9 A7 00 00 00 8B 43 04 0
00401609 BE 50 01 83 C2 9F 83 FA 15 77 70 8A 92 21 16 4
00401619 00 FF 24 95 37 16 40 00 05 00 00 00 04 03 00 0
00401629 00 00 00 02 00 00 00 00 00 00 00 00 00 01 84 1
00000BE9 004015E9: _main+1
IDC
AU: idle
Down
Disk: 377GB
Click (and drag) to add to selection; DblClick on edge to jump to its destination; Wheel to zoom

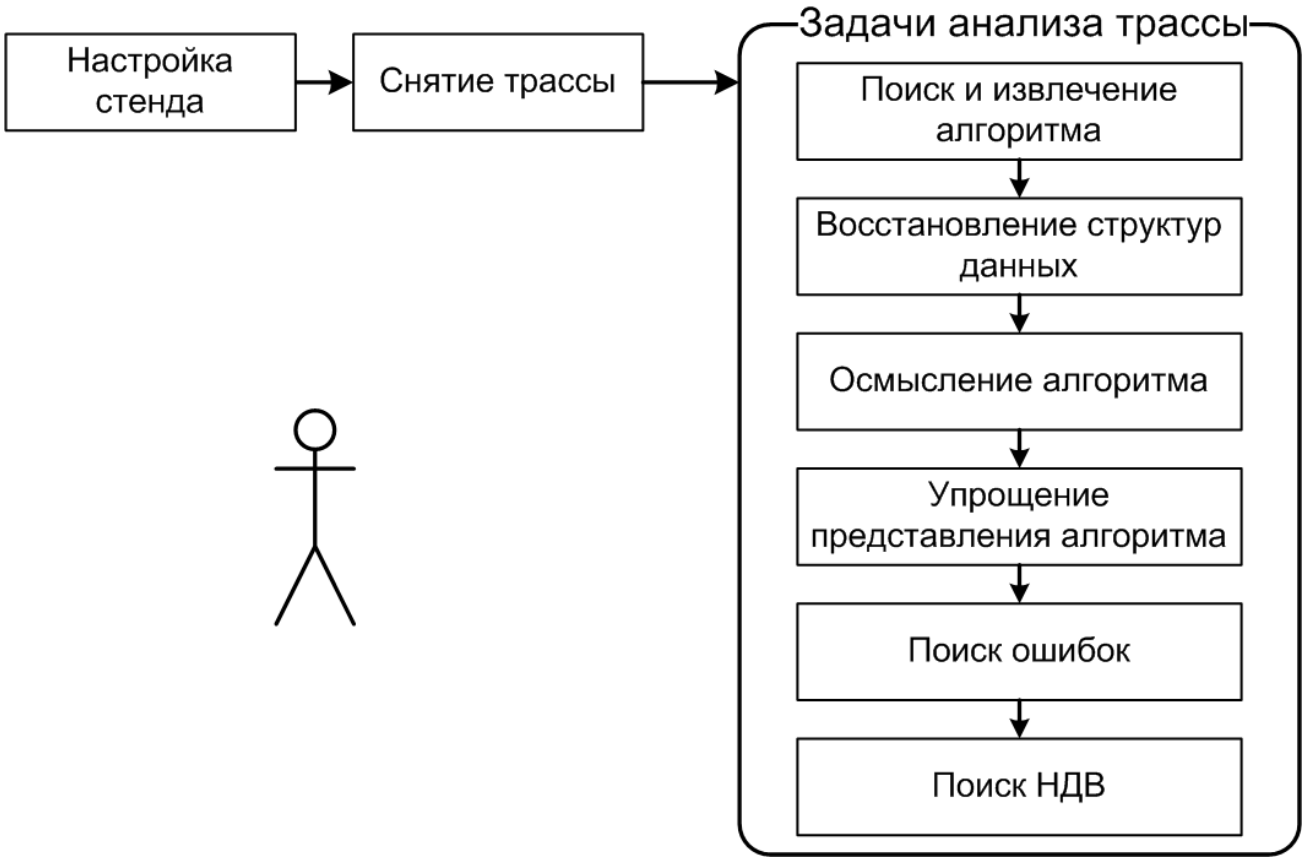

IDA - c:\temp\ar.idb (ar.exe)
File Edit Jump Search View Debugger Options Windows Help
Debugger
Structures
Enums
IDA View-EIP
.text:004010E0 public start
.text:004010E0 start proc near
.text:004010E0 jmp short loc_4010F2
.text:004010E2 dw 6266h
.text:004010E4 dd 2B2B433Ah, 4B4F4F48h
.text:004010EC db 90h
.text:004010ED db 0E9h
.text:004010EE dd offset ___CPPdebugHook
.text:004010F2 loc_4010F2: ; CODE XREF: start↑j
.text:004010F2 mov eax, TlsIndex
.text:004010F7 shl eax, 2
.text:004010FA mov dword_40E08F, eax
.text:004010FF push edx
.text:00401100 push 0 ; lpModuleName
.text:00401102 call GetModuleHandleA
.text:00401107 mov edx, eax
.text:00401109 call nullsub_2
.text:0040110E pop edx
.text:0040110F call nullsub_1
.text:00401114 call nullsub_3
.text:00401119 push 0 ; src
.text:0040111B call __ExceptInit
.text:00401120 pop ecx
Pseudocode-A
void __fastcall start(int a1, int a2, int a3, int a4)
{
int v4; // ST04_4@1
HMODULE v5; // eax@1
int v6; // eax@1
dword_40E08F = 4 * TlsIndex;
v4 = a2;
v5 = GetModuleHandleA(0);
v6 = nullsub_2(v5, v5);
nullsub_1(v6, v4);
nullsub_3();
_ExceptInit(0);
dword_40E093 = (int)GetModuleHandleA(0);
_startup(a4);
}
General registers
EAX 00000000
EBX 7FFD8000 debug005:7FFD8000
ECX 0012FFB0 Stack[00000FC0]:0012FFB0
EDX 7C91E514 ntdll.dll:ntdll_KiFastSystemCa
ESI 00000013
EDI 01C9865C
EBP 0012FFF0 Stack[00000FC0]:0012FFF0
ESP 0012FFC4 Stack[00000FC0]:0012FFC4
EIP 004010FA start+1A
EFL 00000246
CF 0 PF 1 AF 0 ZF 1 SF 0 TF 0 IF 1 DF 0 OF 0
FPU registers
ST0 0.0
ST1 0.0
ST2 0.0
ST3 0.0
ST4 0.0
ST5 0.0
ST6 0.0
ST7 0.0
CTRL 027F
STAT 0000
TAGS FFFF
IM 1 DM 1 ZM 1 OM 1 UM 1 PM 1 PC 2 RC 0 X 0
IE 0 DE 0 ZE 0 OE 0 UE 0 PE 0 SF 0 ES 0 C0 0 C1 0 C2 0 TOP 0 C3 0 B 0
TAG0 3 TAG1 3 TAG2 3 TAG3 3 TAG4 3 TAG5 3 TAG6 3 TAG7 3
Hex View-1
004010E0 EB 10 66 62 3A 43 2B 2B 48 4F 4F 4B 90 E9 98 E0
004010F0 40 00 A1 8B E0 40 00 C1 E0 02 A3 8F E0 40 00 52
00401100 6A 00 E8 ED C1 00 00 8B D0 E8 4A 1B 00 00 5A E8
00401110 4C 0E 00 00 E8 43 1B 00 00 6A 00 E8 90 2F 00 00
00401120 59 68 34 E0 40 00 6A 00 E8 C7 C1 00 00 A3 93 E0
00401130 40 00 6A 00 E9 4F A0 00 00 E9 BE 2F 00 00 33 C0
00401140 A0 7D E0 40 00 C3 A1 93 E0 40 00 C3 60 BB 00 50
00401150 B0 BC 53 68 AD 0B 00 00 C3 B9 A4 00 00 00 0B C9
00401160 74 4D 83 3D 8B E0 40 00 00 73 0A B8 FE 00 00 00
00401170 E8 D7 FF FF FF B9 A4 00 00 00 51 6A 08 E8 84 C1
Breakpoints
Address Pass count Hardware Condition
.text:start
Stack view
0012FFC4 7C817077 kernel32.dll:7C817077
0012FFC8 01C9865C
0012FFCC 00000013
0012FFD0 7FFD8000 debug005:7FFD8000
0012FFD4 8054B6ED
0012FFD8 0012FFC8 Stack[00000FC0]:0012FFC8
0012FFDC 88BC9418
0012FFE0 FFFFFFFF
Output window
Debugger: loaded C:\WINDOWS\system32\kernel32.dll
Debugger: loaded C:\WINDOWS\system32\user32.dll
Debugger: loaded C:\WINDOWS\system32\gdi32.dll
Debugger: loaded C:\WINDOWS\system32\imm32.dll
Debugger: loaded C:\WINDOWS\system32\advapi32.dll
Debugger: loaded C:\WINDOWS\system32\rpcrt4.dll
Debugger: loaded C:\WINDOWS\system32\secur32.dll
Debugger: loaded C:\WINDOWS\system32\lpk.dll
Debugger: loaded C:\WINDOWS\system32\usp10.dll
4010E0: hit breakpoint
401F60: using guessed type int __fastcall nullsub_1(_DWORD, _DWORD);
402C58: using guessed type int __fastcall nullsub_2(_DWORD, _DWORD);
402C5C: using guessed type int nullsub_3(void);
40B188: using guessed type _DWORD __cdecl _startup(_DWORD);
40E08F: using guessed type int dword_40E08F;
40E093: using guessed type int dword_40E093;
IDC
AU: idle
Down
Disk: 377GB

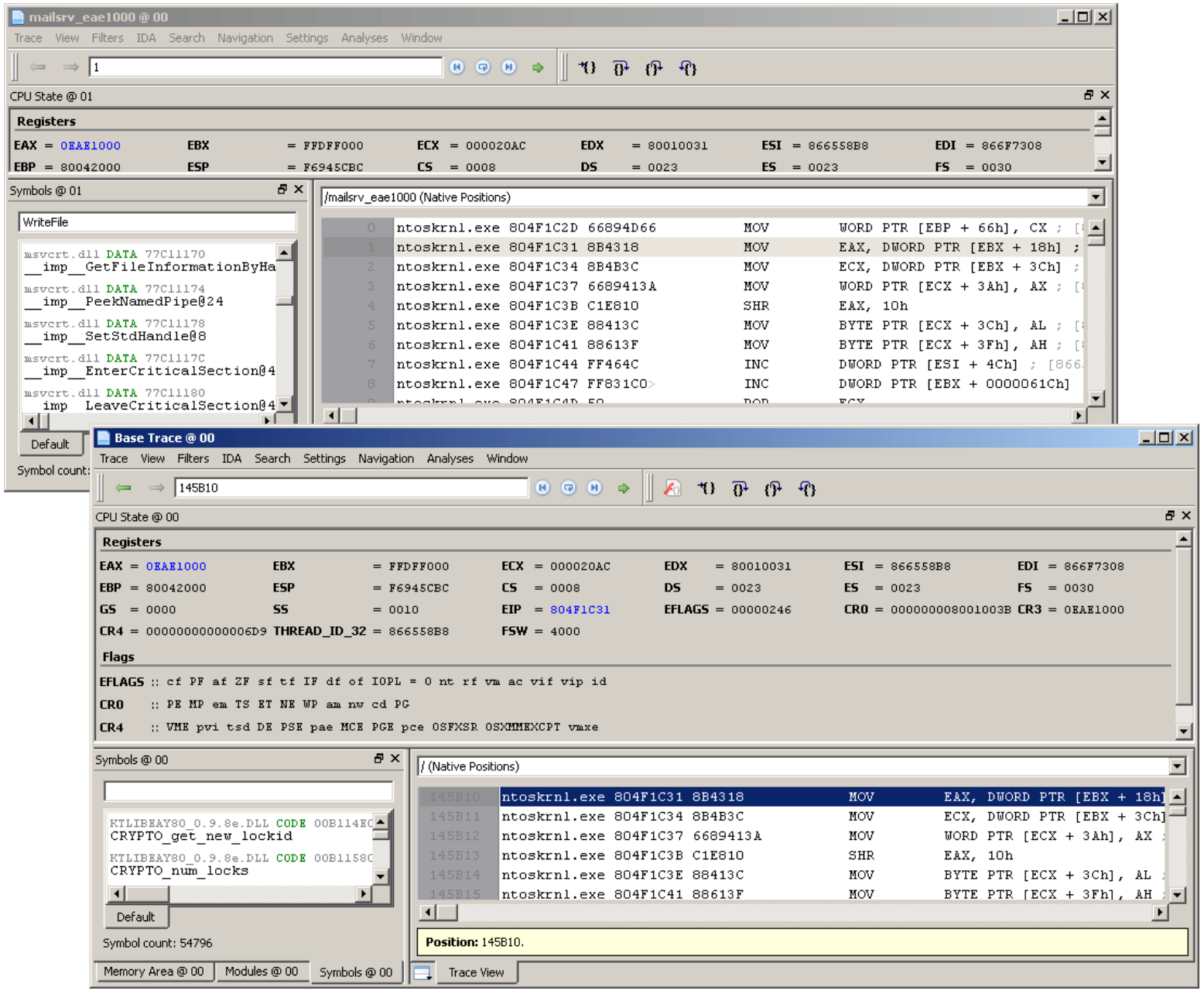
Методика анализа



- Аналитик запускает на симуляторе исследуемую программу.
- Трассировке подвергается выполнение соответствующего алгоритма, т.е. происходит проигрывание определенного сценария работы программы, который фиксируется в трассе.
- Трасса анализируется с целью осмысления алгоритма.
- Формальный результат осмысления
  - Отчет на естественном языке
  - Контрольный пример. Ассемблерная программа, выполнение которой должно повторить исходный сценарий
- Потенциально возможна декомпиляция контрольного примера в язык С.

Методика анализа. Порядок действий.




mailsrv_eae1000 @ 00
Trace View Filters IDA Search Navigation Settings Analyses Window
CPU State @ 01
Registers
EAX = 0EAE1000 EBX = FFDFF000 ECX = 000020AC EDX = 80010031 ESI = 866558B8 EDI = 866F7308
EBP = 80042000 ESP = F6945CBC CS = 0008 DS = 0023 ES = 0023 FS = 0030
Symbols @ 01
WriteFile
msvcrt.dll DATA 77C11170 __imp__GetFileInformationByHa
msvcrt.dll DATA 77C11174 __imp__PeekNamedPipe@24
msvcrt.dll DATA 77C11178 __imp__SetStdHandle@8
msvcrt.dll DATA 77C1117C __imp__EnterCriticalSection@4
msvcrt.dll DATA 77C11180 __imp__LeaveCriticalSection@4
Default
Symbol count:
/mailsrv_eae1000 (Native Positions)
0 ntoskrnl.exe 804F1C2D 66894D66 MOV WORD PTR [EBP + 66h], CX
1 ntoskrnl.exe 804F1C31 8B4318 MOV EAX, DWORD PTR [EBX + 18h]
2 ntoskrnl.exe 804F1C34 8B4B3C MOV ECX, DWORD PTR [EBX + 3Ch]
3 ntoskrnl.exe 804F1C37 6689413A MOV WORD PTR [ECX + 3Ah], AX
4 ntoskrnl.exe 804F1C3B C1E810 SHR EAX, 10h
5 ntoskrnl.exe 804F1C3E 88413C MOV BYTE PTR [ECX + 3Ch], AL
6 ntoskrnl.exe 804F1C41 88613F MOV BYTE PTR [ECX + 3Fh], AH
7 ntoskrnl.exe 804F1C44 FF464C INC DWORD PTR [ESI + 4Ch]
8 ntoskrnl.exe 804F1C47 FF831C0 INC DWORD PTR [EBX + 0000061Ch]
Base Trace @ 00
Trace View Filters IDA Search Settings Navigation Analyses Window
145B10
CPU State @ 00
Registers
EAX = 0EAE1000 EBX = FFDFF000 ECX = 000020AC EDX = 80010031 ESI = 866558B8 EDI = 866F7308
EBP = 80042000 ESP = F6945CBC CS = 0008 DS = 0023 ES = 0023 FS = 0030
GS = 0000 SS = 0010 EIP = 804F1C31 EFLAGS = 00000246 CR0 = 000000008001003B CR3 = 0EAE1000
CR4 = 00000000000006D9 THREAD_ID_32 = 866558B8 FSW = 4000
Flags
EFLAGS :: cf PF af ZF sf tf IF df of IOPL = 0 nt rf vm ac vif vip id
CR0 :: PE MP em TS ET NE WP am nw cd PG
CR4 :: VME pvi tsd DE PSE pae MCE PGE pce OSFXSR OSXMMEXCPT vmxe
Symbols @ 00
KTLIBEAY80_0.9.8e.DLL CODE 00B114EC CRYPTO_get_new_lockid
KTLIBEAY80_0.9.8e.DLL CODE 00B1158C CRYPTO_num_locks
Default
Symbol count: 54796
/ (Native Positions)
145B10 ntoskrnl.exe 804F1C31 8B4318 MOV EAX, DWORD PTR [EBX + 18h]
145B11 ntoskrnl.exe 804F1C34 8B4B3C MOV ECX, DWORD PTR [EBX + 3Ch]
145B12 ntoskrnl.exe 804F1C37 6689413A MOV WORD PTR [ECX + 3Ah], AX
145B13 ntoskrnl.exe 804F1C3B C1E810 SHR EAX, 10h
145B14 ntoskrnl.exe 804F1C3E 88413C MOV BYTE PTR [ECX + 3Ch], AL
145B15 ntoskrnl.exe 804F1C41 88613F MOV BYTE PTR [ECX + 3Fh], AH
Position: 145B10.
Memory Area @ 00
Modules @ 00
Symbols @ 00
Trace View

TrEx – актуальные работы

- Независимый от архитектуры анализ кода
  - Внутреннее представление
  - Распутывание кода
- Восстановление формата данных
- Подъем уровня представления, информационное окружение аналитика
  - стек вызовов, автоматические переменные
  - модули, символы
- Улучшение покрытия кода трассами

-> Machine #1

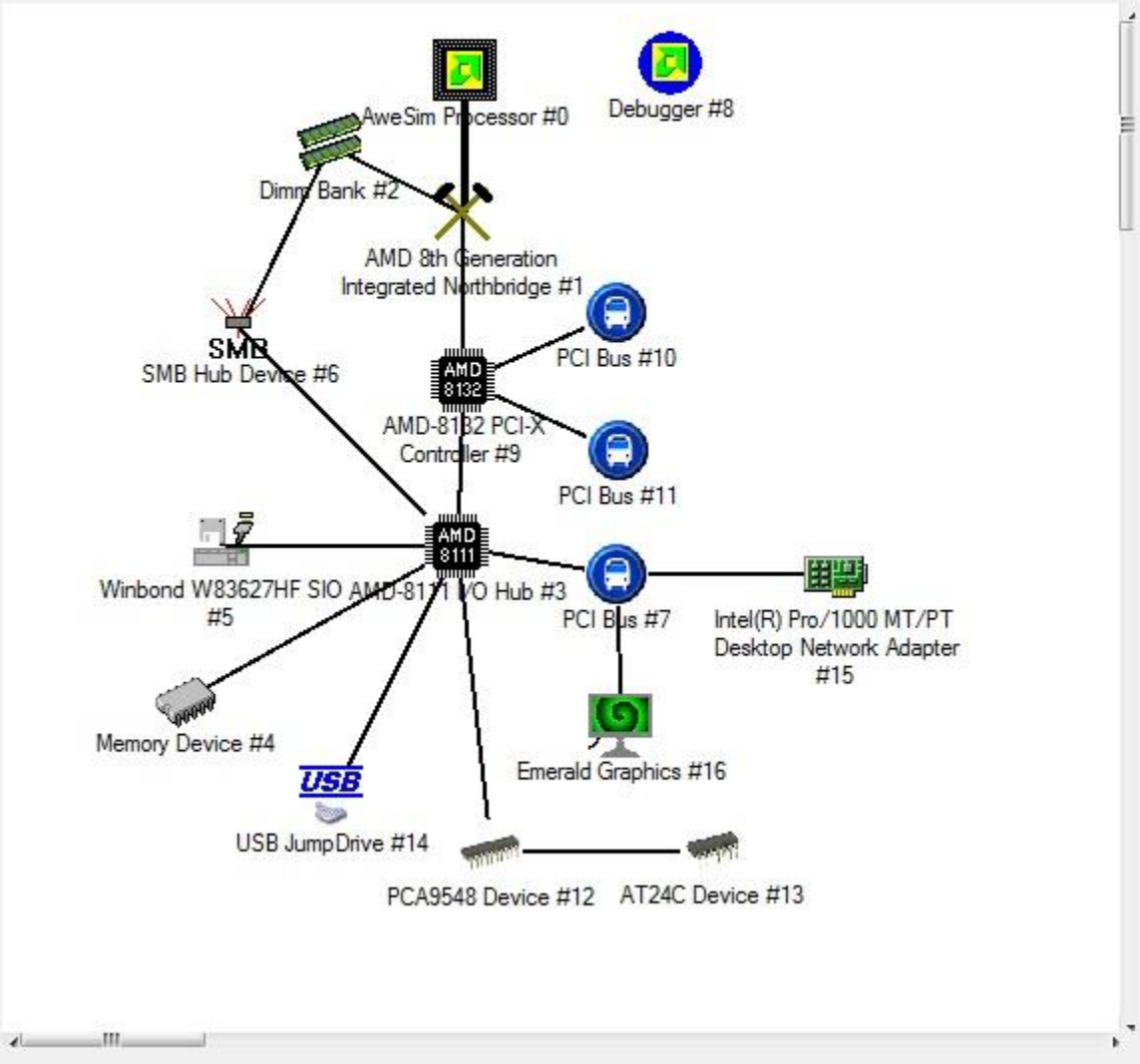
Drag Icons to insert new devices

Shift+drag to add connections

Help

Show Deprecated Devices

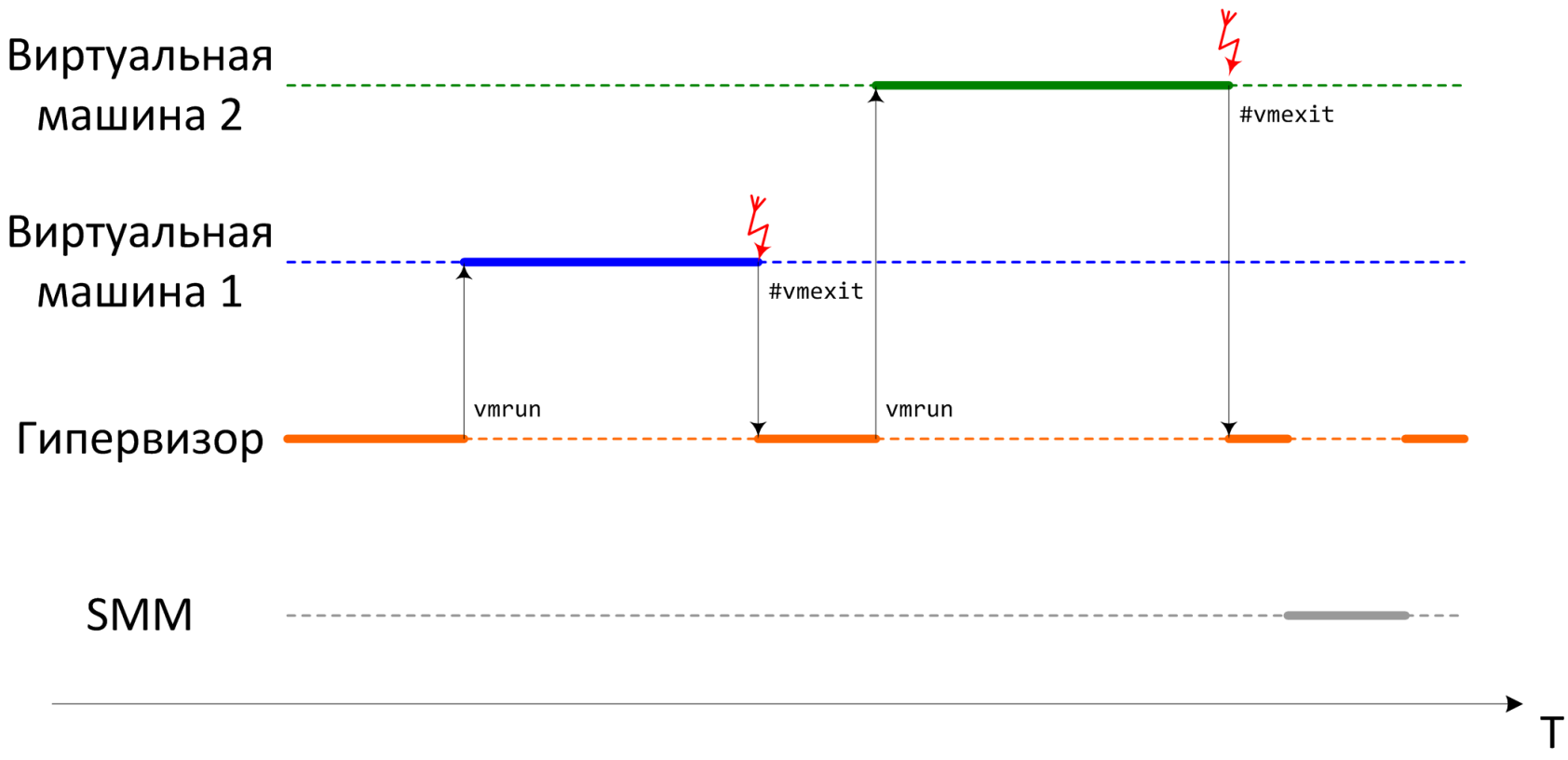
- AMD 8th Generation Integrated Northbridge
- AMD Family 12h Graphics NorthBridge
- AMD Istanbul 6-Core Socket L1
- AMD Magny-Cours 12-Core Socket G34
- AMD Sao Paulo 8-Core Socket G34
- AMD-8111 I/O Hub
- AMD-8131 PCI-X Controller
- AMD-8132 PCI-X Controller
- AMD-8151 AGP Tunnel
- AT24C Device
- ATI Radeon HD 3870
- ATI RD790/RD780/RX780 Host Bridge Tunnel
- ATI RD890S/RD890/RD780S/RX880 Host Bridge
- ATI-RS480 Host Bridge Tunnel
- ATI-RS780/RS880 Host Bridge Tunnel



Способы виртуализации

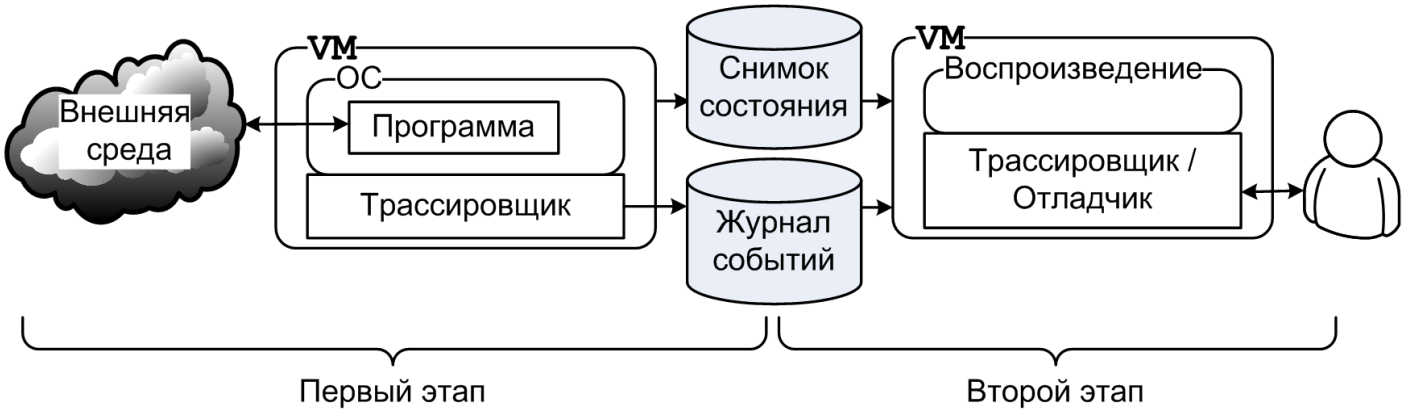
- Программный симулятор
  - Пример: AMD SimNOW
- Бинарная трансляция
  - Пример: VMWare Workstation
- Паравиртуализация
  - Пример: Xen
- Аппаратная виртуализация:
  - AMD SVM (Secure Virtual Machine)
  - Intel VT-x (Virtualization Extensions)
  - Пример: VMWare Workstation, Microsoft Hyper-V, Xen, KVM

Аппаратная виртуализация



Двухпроходная трассировка

- Цель – получение детальной трассы, исключая возможность обнаружения процесса трассировки извне
- Все взаимодействие с внешним окружением фиксируется на первом этапе в журнале событий
  - Крайне низкие накладные расходы на трассировку
- Второй этап – детерминированное воспроизведение
  - Подключение интерактивного отладчика
  - Снятие детальной трассы для последующего анализа

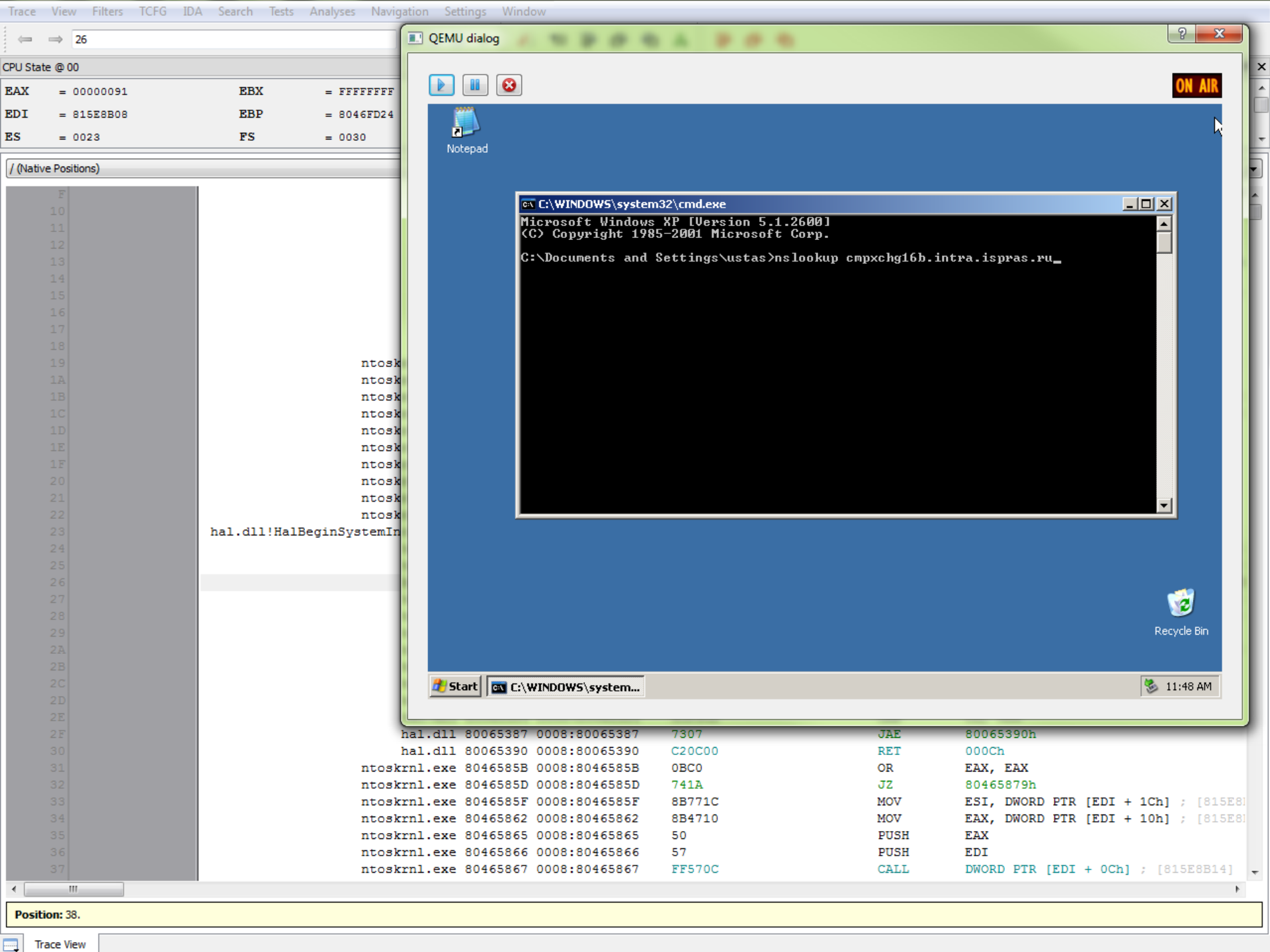


Необходимые для восстановления трассы данные и события

- Начальное состояние гостевой системы:
  - Содержимое занятой памяти
  - Значения регистров
- События во время выполнения:
  - Прерывания
  - Доступ к портам I/O, MSR’ам
  - Данные I/O, в т.ч. данные DMA-транзакций

QEMU

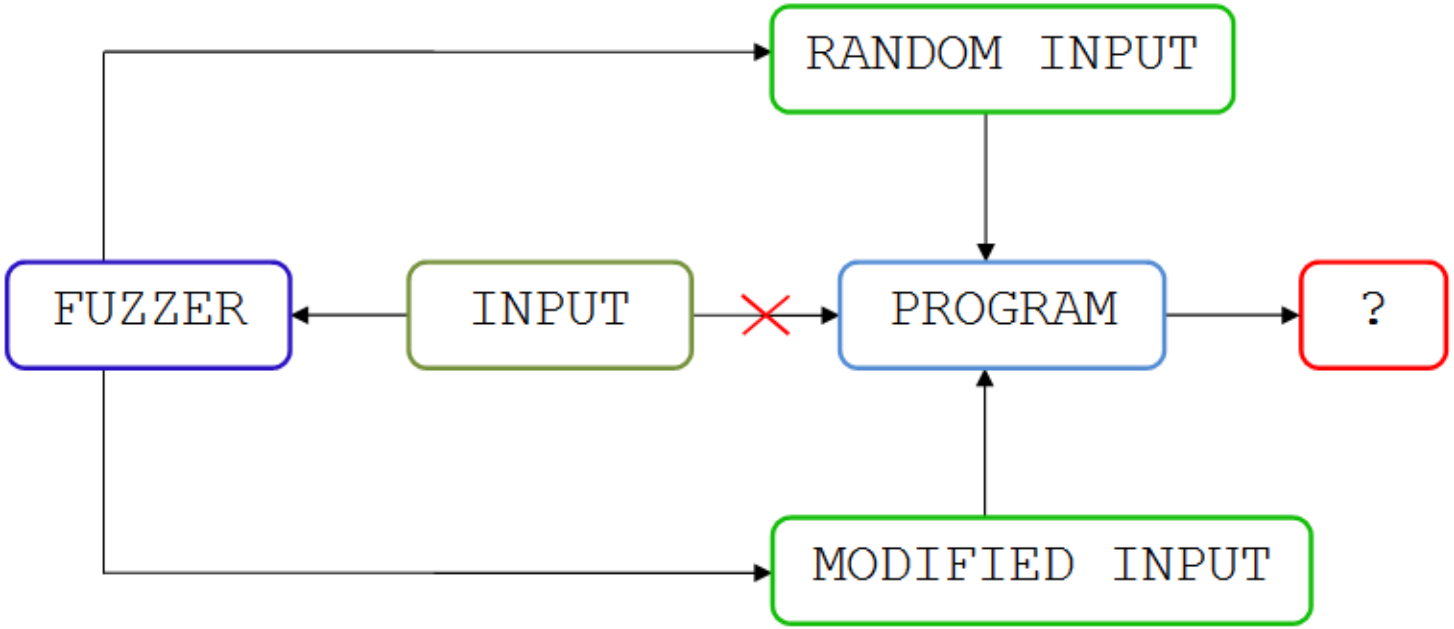
- Программа с открытым кодом
- Возможность полносистемной симуляции
  - Бинарная трансляция
- Легкое расширение списка поддерживаемой периферии
- Поддержка распространенных платформ
  - x86, PowerPC, ARM, Sparc, …
- Активно используется разработчиками мобильных платформ
  - Android, Symbian, Tizen, …
- Недостаток: известны случаи противодействия со стороны программ


QEMU dialog
ON AIR
Notepad
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\ustas>nslookup cmpxchg16b.intra.ispras.ru
Recycle Bin
Start
C:\WINDOWS\system...
11:48 AM

Trace View

Фаззинг

- Фаззинг – технология тестирования программного обеспечения, когда вместо ожидаемых входных данных программе передаются случайные или специально сформированные данные



Метрики сложности бинарного кода

количественные оценки сложности приложений и их частей

Задачи, решаемые с помощью метрик:

- Классификация приложений по сложности анализа
  (построение эталонных шкал сложности и ранжирование по ним анализируемых приложений)
- Оценка трудоемкости анализа модулей и функций
  (распределение подзадач между аналитиками и оптимизация рабочего процесса)
- Профилирование приложения для поиска потенциально интересного аналитику кода
  (наиболее сложные для анализа участки, обычно запутанные)
- Локализация запутанных участков для ускорения автоматического анализа
  (непрозрачные предикаты, диспетчеры, виртуальные машины)

Метрики сложности бинарного кода

Классификация метрик, применимых к бинарному коду

1. Количественные
   - LOC; среднее число инструкций в функции.
   - метрика Джилба (учитывает количество управляющих конструкций)
   - ABC = sqrt(A^2+B^2+C^2) (Assignment, Branch, Condition)
2. Метрики сложности потока управления
   - Цикломатическая сложность CFG
   - Метрики Харрисона & Мейджела, Пивоварского
   - Метрика граничных значений
3. Метрики сложности данных
   - Метрики Чепина, Кафура, Овиедо, спена
4. Гибридные
   - Взвешенные суммы мер из (1,2,3)
5. Метрики запутанности кода
   - BCD = sqrt(B^2+C^2)/LOC

Итого: направления работ

- Модели
  - поиск ошибок
  - описание алгоритмов, обработки данных
  - анализ реализации политик безопасности
- Бинарная трансляция
- QEMU: фаззинг, поддержка специфической аппаратуры
- Распутывание кода, метрики кода
- Анализ сетевого трафика и его сопоставление с бинарным кодом

СПАСИБО ЗА ВНИМАНИЕ!