# Python в IDApro

докладчик Синяев А.А.

# Реверс-инжиниринг

**Обратная разработка** - исследование некоторого готового устройства или программы, а также документации на него с целью понять принцип его работы.
Например, чтобы обнаружить недокументированные возможности (в том числе програмные закладки), сделать изменение, или воспроизвести устройство, программу или иной объект с аналогичными функциями, но без копирования как такового.

# Краткое введение в дизассемблирование

Одним из способов изучения программ в отсутствии исходных текстов является дизассемблирование, – перевод двоичных кодов процессора в удобочитаемые мнемонические инструкции. С перового взгляда кажется: ничего сложного в такой операции нет, и один дизассемблер не будет сильно хуже любого другого. На самом же деле, ассемблирование – однонаправленный процесс с потерями, поэтому автоматическое восстановление исходного текста невозможно.

```
mov     ah,9
mov     dx, offset s0
int     21h
ret
s0      DB 'Hello,World!',0Dh,0Ah,'$'
```

(a) Исходная программа

```
mov     ah,9
mov     dx,0108h
int     21h
ret
s0 DB 'Hello,World!',0Dh,0Ah,'$'
```

(b) Дизассемблированная программа

```
mov     ah,09
mov     dx,0108h
int     21h
xor     ax,ax
int     16h
ret
s0 DB 'Hello,World!',0Dh,0Ah,'$'
```

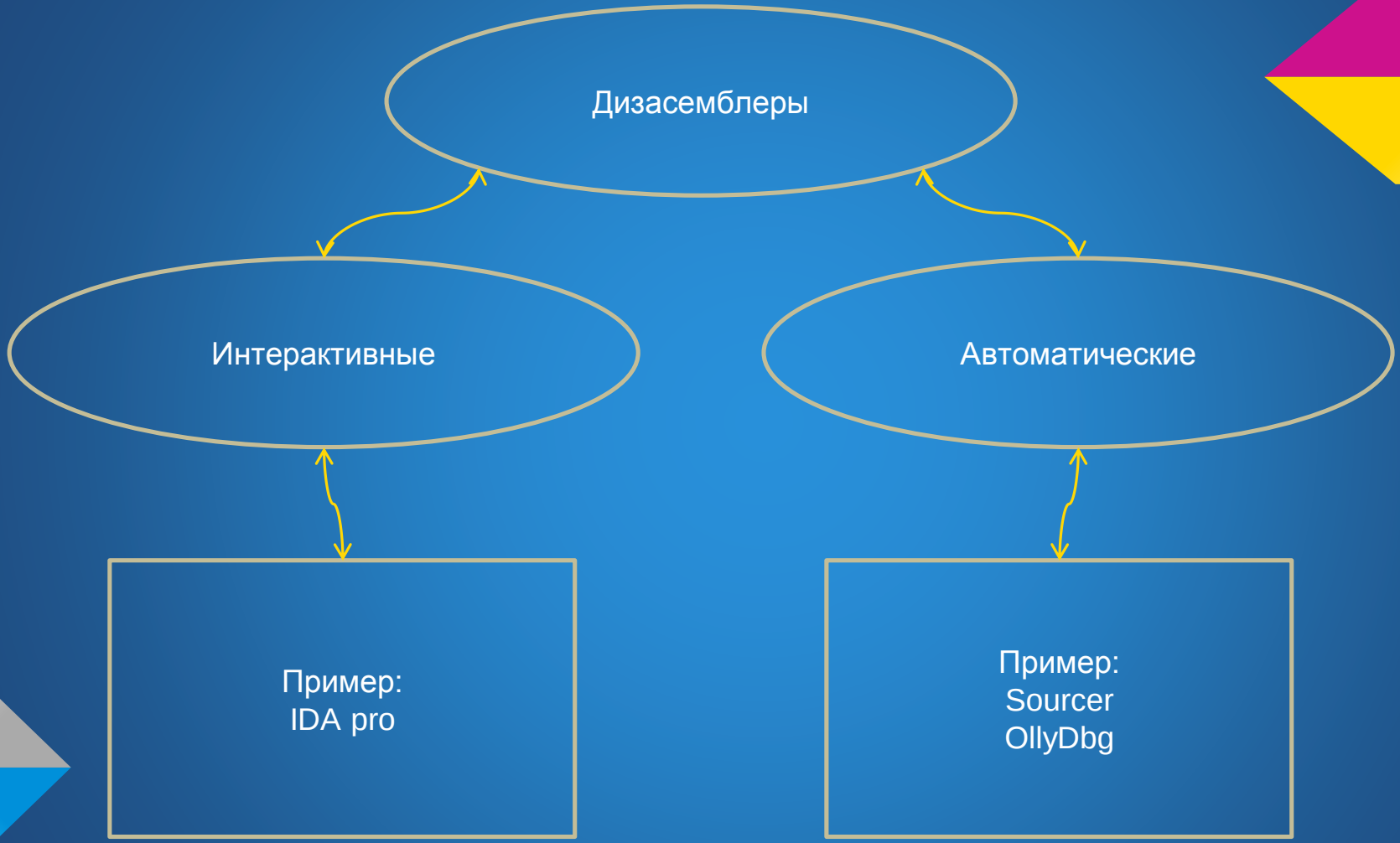
(c) Модифицированная программа

```
:0100 start             proc    near
:0100                   mov     ah, 9
:0102                   mov     dx, 108h
:0105                   int     21h
:0107                   xor     ax, ax
:0109                   int     16h
:010B                   retn
:010B aHelloWorld       db 'Hello,World!',0Dh,0Ah,'$'
:010C end               start
```

(d) Неработоспособный результат

```
JMP Label                          00: E90100
Align 4                            03: 00
Label: XOR  AX,AX                  04: 33 C0
RET                                06: C3
(a) Исходная программа             (b) Машинный код

      00: E9 01 00      jmp   04
      03: 00 33         add   [bp][di],dh
      05: C0 C3         rol   bl,-070;
      (c) Дизассемблированный текст
```


Дизасемблеры
Интерактивные
Автоматические
Пример:
IDA pro
Пример:
Sourcer
OllyDbg

# Основы IDC

С помощью IDC можно быстро разрабатывать сценарии, позволяющие автоматизировать множество рутинных действий. Язык вобрал в себя основные синтаксические элементы традиционного C, поэтому прост для освоения. Полная документация идет вместе с IDA.

# Пример кода IDC

```
#include <idc.idc>
static main()
{
        auto ea, func, ref; // получаем текущий адрес курсора
        ea = ScreenEA(); // в цикле от начала (SegStart) до
                         // конца (SegEND) текущего сегмента
        for (func=SegStart(ea); func != BADADDR && func < SegEnd(ea); func=NextFunction(func))
        {
                // если текущий адрес является
                // адресом функции
                if (GetFunctionFlags(func) != -1)
                {
                        Message("Function %s at 0x%xn",
                        GetFunctionName(func), func);
                        // находим все ссылки на данную
                        // функцию и выводим
                        for (ref=RfirstB(func); ref != BADADDR; ref=RnextB(func, ref))
                        {
                                Message(" called from %s(0x%x)n",
                                GetFunctionName(ref), ref);
                        }
                }
        }
}
```

# Результат

Function start at 0x401000
Function sub_401060 at 0x401060
called from start(0x401006)
Function sub_401090 at 0x401090
called from sub_4010E0(0x401185)
Function sub_4010E0 at 0x4010e0

# Пример кода IDAPython

```
from idautils import *
ea = ScreenEA()
for funcea in Functions(SegStart(ea), SegEnd(ea)):
        print "Function %s at 0x%x" % (GetFunctionName(funcea),
        funcea)
for ref in CodeRefsTo(funcea, 1):
        print " called from %s(0x%x)" % (GetFunctionName(ref), ref)
```

## Alphabetical list of IDC functions

The following conventions are used in the function descriptions:

```
  'ea' is a linear address
  'success' is 0 if a function fails, 1 otherwise
  'void' means that function returns no meaningful value (always 0)
  'anyvalue' means that function may return value of any type
```

AddAutoStkPnt2
AddUserStkPnt
AddBptEx
AddBpt
AddCodeXref
AddConstEx
AddEntryPoint
AddEnum
AddHotkey
AddSegEx
AddSourceFile
AddStrucEx
AddStrucMember
AltOp
Analysis
AnalyzeArea
Appcall
AppendFchunk
ApplySig
ApplyType
AskAddr
AskFile
AskIdent
AskSeg
AskSelector
AskStr
AskYN
AttachProcess
AutoMark2
AutoMark
AutoShow
Batch
BeginEA
BeginTypeUpdating
Byte
CanExceptionContinue
DelStkpnt
ChangeConfig
CheckBpt
CheckTraceFile
DiffTraceFile
ChooseFunction
CleanupAppcall
ClearTraceFile
CmtIndent
CommentEx
Comments
Compile
CompileEx
CreateArray