HexRays Decompiler API

Что такое декомпиляция? Зачем она нам нужна

- Процесс получения из машинного кода, текста программы на языке «высокого уровня»
- Значительно может ускорить и упростить анализ
- Или наоборот ☺️

Компиляция

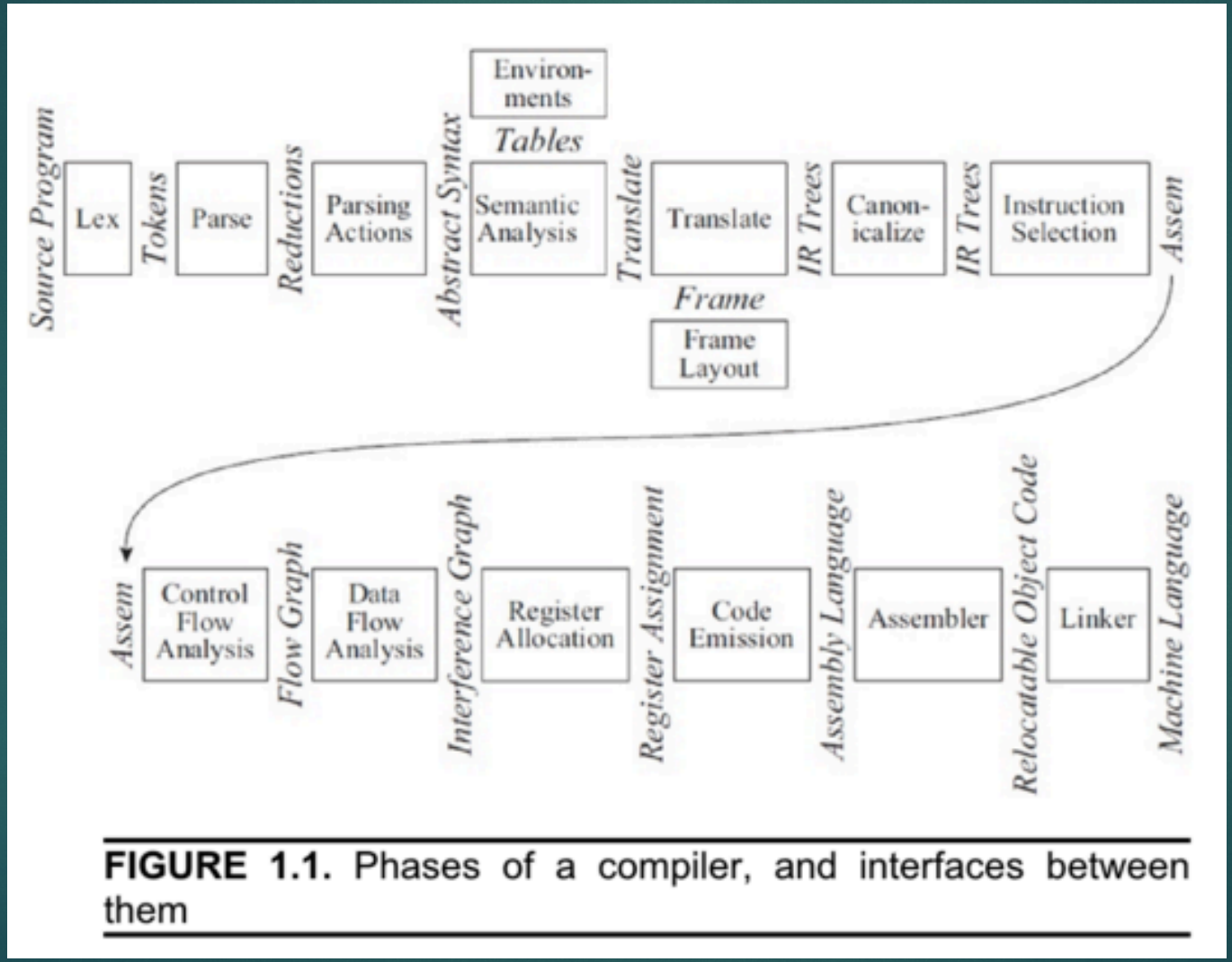


FIGURE 1.1. Phases of a compiler, and interfaces between them

Декомпиляция

- Синтаксический анализ
- Семантический анализ
- Генерация промежуточного представления
- Генерация графа выполнения программы
- Анализ потоков данных
- Анализ потоков выполнения
- Генерация кода

Существующие декомпиляторы(в произвольном порядке)

- Snowman
  - https://derevenets.com/
- HexRays
  - https://www.hex-rays.com/
- RetDec
  - https://retdec.com
- Radeco ?
- DREAM(DREAM++)
  - Я нашел только статьи с исследованиями ;[

HexRays: плюсы

- Интерактивный
- The ability to define types/structs in IDA (to reduce casting) is invaluable for killing ugly output ©
- Расширяемый

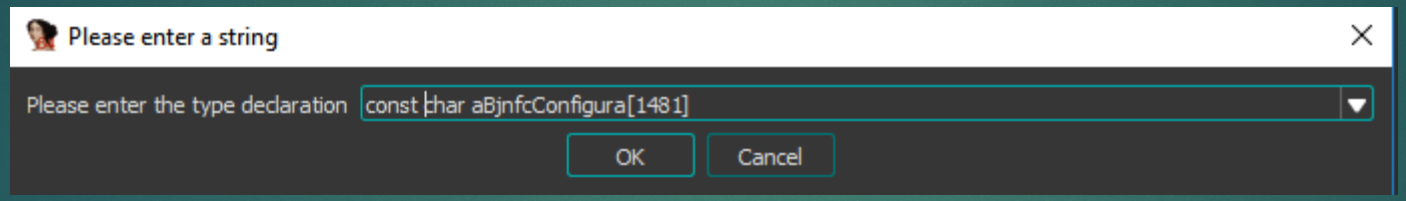
Зачем нам писать свой код поверх декомпилятора?

- Автоматическое распознавание структур
- Автоматическая смена типов
- «Макросы»
- Потому что могу

Пример 1: добавление const по щелчку

```
{
  v2 = write(v0, aBjnfcConfigura, unk_62A130);
  if ( v2 != -1 )
    break;
  v1 = *__errno_location();
```

buf: char[1481]




```
{
  v2 = write(
         v0,
         "# bjnfc configuration\n"
         "\n"
         "bind                 0.0.0.0 2443\n"
         "tls_dhparam         dh2048.pem\n"
         "\n"
         "validator v_file regex ^.+$\n"
         "validator v_device regex [[:alnum:]]{,20}\n"
         "\n"
         "validator v_signature regex ^[[:xdigit:]]+$\n"
         "validator v_chall regex [[:xdigit:]]{32}\n"
```

Пример 1: добавление const по щелчку

- Собственно, это не особо частый случай при работе с PE или ELF, когда лоадер правильно ставит разрешения на сегменты
- Но однажды у меня была прошивка, адреса сегментов в которой я не знал на момент начала анализа
- Все строки были без const
- И в тот момент когда я довольно резво научился вызывать комбинацию y<home>const<space><enter> я решил, что хочу делать это по shift+c

CONSTantine.py

- За основу был взят код из HexRaysPyTools
- Вызывается либо по нажатию Shift+C, либо из менюшки ПКМ в окне декомпилятора
- https://gitlab.com/zaytsevgu/spbctf171216/blob/master/CONSTantine.py

HexRays API: установка callback-а

- install_hexrays_callback
  - Принимает на вход callback
- int hexrays_cb_t(void *ud, hexrays_event_t event, va_list va)
- Если мы используем скрипт а не плагин, то надо отвязвать старый callback при перезагрузке скрипта
- remove_hexrays_callback
- https://www.hex-rays.com/products/decompiler/manual/sdk/hexrays_8hpp.shtml#a7c1ea443e86c64bdccdcad64042e4a36

HexRays API: Типы событий

- https://gitlab.com/zaytsevgu/spbctf171216/blob/master/demo_handler.py
- hxe_keyboard
- hxe_right_click
- hxe_double_click
- hxe_populating_popup
- hxe_maturity
- Из питона нам доступны не все события

IDAPython ;[

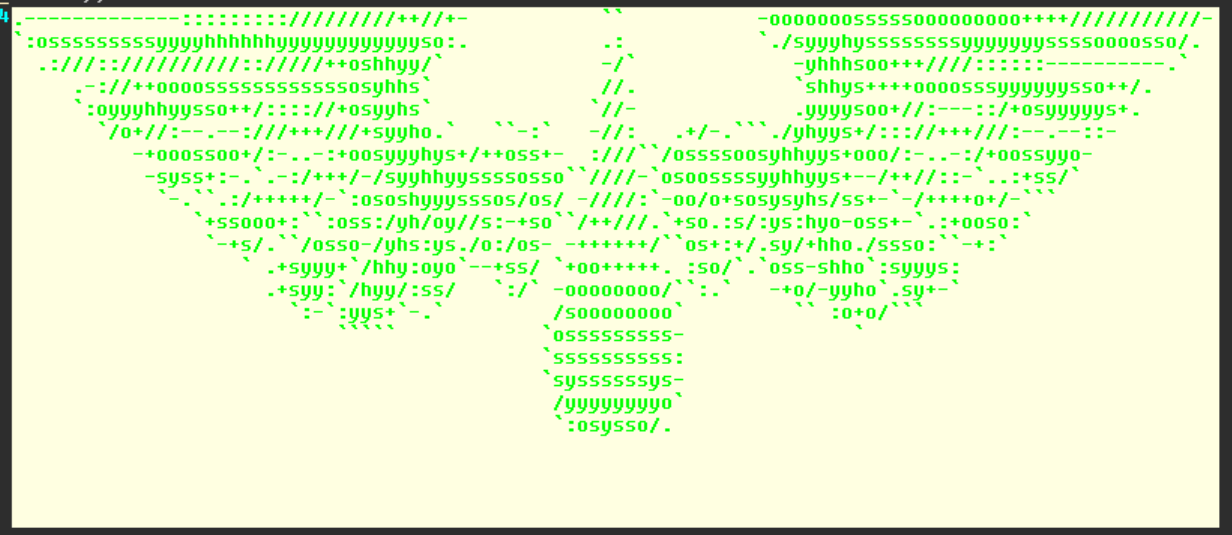
- https://github.com/idapython/src/blob/master/pywraps/py_hexrays.hpp
- __hexrays_python_callback обрабатывает и отдает в питоновский callback не все события, а для ряда событий отдает не все параметры
- Как вариант можно попробовать собрать свой IDAPython с событиями и параметрами
- Или если мы хотим работать со всеми событиями, то придется писать на C++

Скелет кода на C++

- https://gitlab.com/zaytsevgu/spbctf171216/tree/master/HexRPlugin

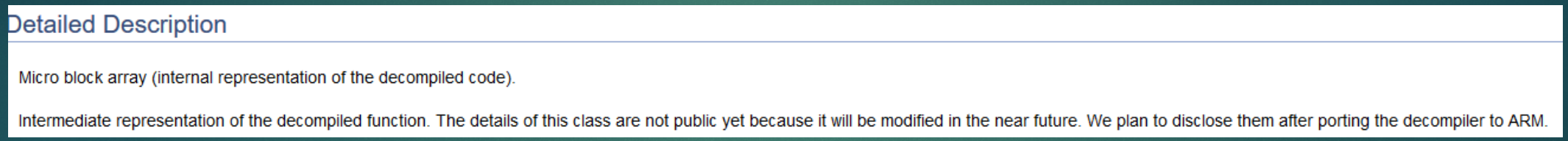
```
_DWORD *__thiscall sub_402660(_DWORD *this, LPCSTR lpsz, LPCSTR a3)
{
  _DWORD *v3; // esi@1
  CHAR *v4; // eax@1

  v3 = this;
  *this = &off_405254;
  v4 = (CHAR *)operator new(0x12A8u);
  JUMPOUT(v4, 0, &algn_4026AF);
  v3[1] = sub_402FA0(v4
  return v3;
}
```



C++: Можем ли мы теперь все?

- Хотя теперь мы получаем до 1000 событий при декомпиляции одной функции вместо 30, для части событий мы не знаем ничего о параметрах ;[



Detailed Description

Micro block array (internal representation of the decompiled code).

Intermediate representation of the decompiled function. The details of this class are not public yet because it will be modified in the near future. We plan to disclose them after porting the decompiler to ARM.

HexRays API: hxe_create_hint

- Вызывается при наведении мыши на переменную/функцию etc
  - vdui_t *vu
  - qstring *result_hint
  - int *implines

- https://github.com/alexhude/FRIEND

HexRays API: hxe_maturity

- demo_handler.py – если раскомментировать пару строк, то создаст в папке с .idb пачку лог файлов с текущим телом функции
  - cfunc_t *cfunc
  - ctree_maturity_t new_maturity
- Часто используется и можно гуглить hxe_maturity и находить всякие интересные плагины для декомпилятора
- http://devirtualize.readthedocs.io/en/latest/index.html

HexRays API: hxe_keyboard

- Позволяет реагировать на нажатия клавиш
  - vdui_t *vu
  - int key_code (VK_...)
  - int shift_state
- Ряд клавиш (N,Y,...-?) перехватить не удастся
- В демках есть vim-like режим навигации ☺
- https://www.hex-rays.com/contests/2016/hexlight/hexrays_hlight.py

HexRays API: Доступ к AST

- Чтобы получать полезные скрипты нам необходимо как-то получать доступ к AST, который хранит в себе декомпилятор
- Так например в CONSTantine.py функция check принимает вторым параметром узел AST, который соответствует позиции курсора
- Помимо этого мы можем получить объект cfunc_t с помощью функции decompile
- cfunc_t - Decompiled function. Decompilation result is kept here

Lisper.py

- https://gitlab.com/zaytsevgu/spbctf171216/blob/master/lisper.py
- Простенький скрипт, который частично переводит AST в некое подобие кода на псевдо-lisp

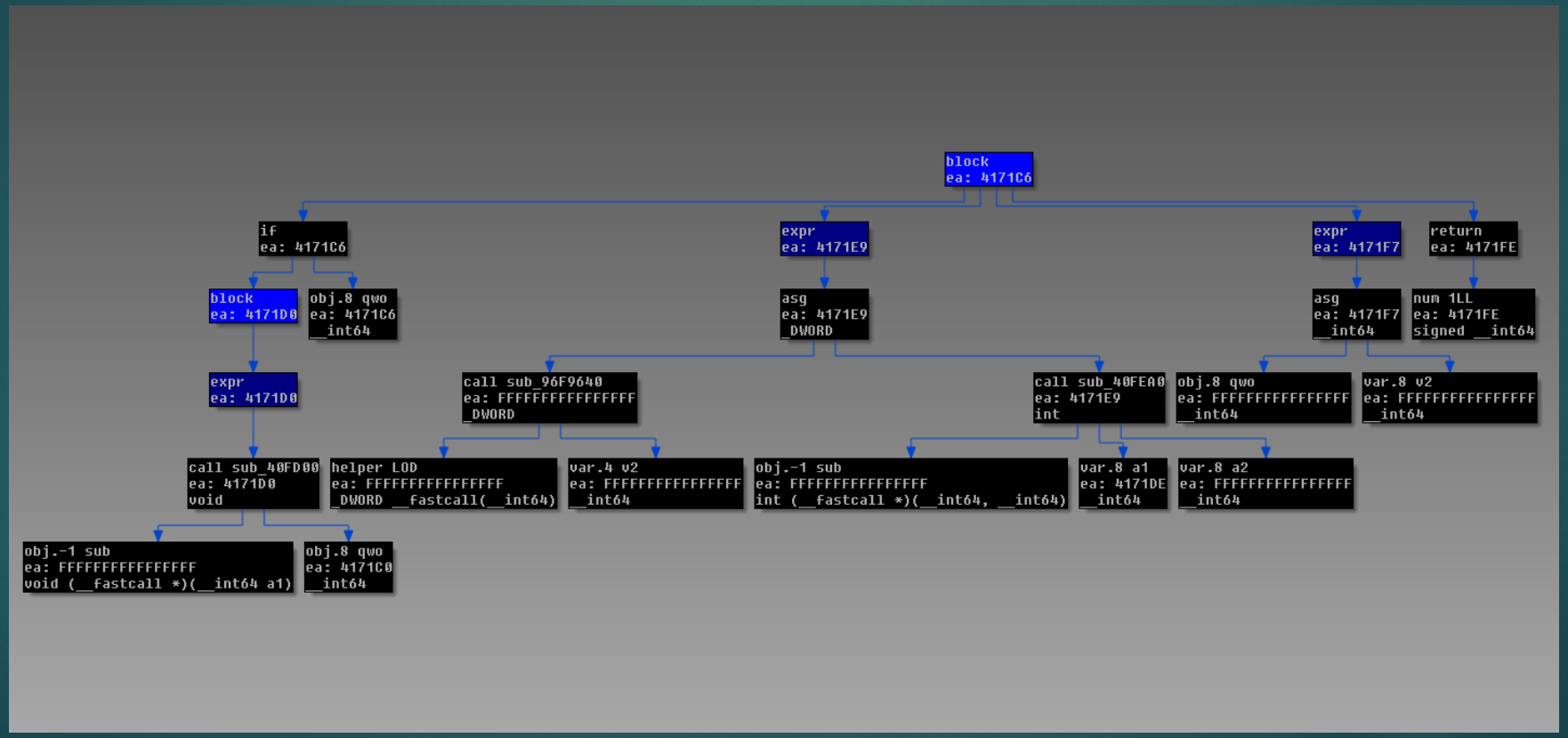
```
signed __int64 __fastcall sub_4171B0(__int64 a1, __int64 a2)
{
  __int64 v2; // rax@3

  if ( qword_62ABD0 )
    sub_40FD00(qword_62ABD0);
  LODWORD(v2) = sub_40FEA0(a1, a2);
  qword_62ABD0 = v2;
  return 1LL;
}
```

```
(cblock
    (if (obj qword_62ABD0)
        (cblock (call (obj sub_40FD00)))
    )
    (asg
        (call (helper))
        (call (obj sub_40FEA0))
    )
    (asg
        (obj qword_62ABD0)
        (var v2)
    )
    (return (num 1))
)
```

HexRaysCodeXplorer

- Помимо всего прочего умеет отображать cfunc_t в виде графа



HexRays API: внутри cfunc_t

- cfucn_t->body
- Объект типа cinst_t
- Который в свою очередь содержит ----------→
- cinst_t->op
- cinst_t->opname(python)

```
union {
    cblock_t *    cblock
    cexpr_t *    cexpr
    cif_t *    cif
    cfor_t *    cfor
    cwhile_t *    cwhile
    cdo_t *    cdo
    cswitch_t *    cswitch
    creturn_t *    creturn
    cgoto_t *    cgoto
    casm_t *    casm
};
```

HexRays API: cexpr_t

- Содержимое этого union-а зависит от поля type
- В питоне также есть opname

```
union {
   cnumber_t *   n
   fnumber_t *   fpc
   struct {
      union {
         var_ref_t   v
         ea_t        obj_ea
      }
      int            refwidth
   }
   struct {
      cexpr_t       *   x
      union {
         cexpr_t    *   y
         carglist_t *   a
         uint32   m
      }
      union {
         cexpr_t *   z
         int   ptrsize
      }
   }
   cinsn_t *   insn
   char *   helper
   char *   string
};
```

AST Matcher

- Возникла идея написать себе плагинчик, который будет позволять искать в AST определенные шаблоны, чтобы потом с ними чего-нибудь делать
- ATOM WINAPI AddAtom(_In_ LPCTSTR lpString);
- Например если у нас есть в коде 30+ подряд вызовов данной функции с сохранением результата в глобальных переменных, то было бы неплохо автоматически уметь переименовывать эти переменные
- Похожим способом можно искать например STL шаблоны
- Возможно автоматически сворачивать куски кода

DEMO