Lesson 6: Introduction to Reversing C++ Binaries

Please Stop Compiling With O3

Leonardo Galli

flagbot (CTF@VIS)

November 2, 2021

Table of Contents

Readying IDA

Theory

Reconstructing Classes

C++ STL

- Strings
- Vectors
- Maps

Demo

Readying IDA

Useful Plugins

- HexRaysPyTools: Extremely useful for quickly creating structures without having to find every offset that might be a field.
- Classy: Makes working with vtables and child classes a lot easier.

Other Settings

- Make sure to regularly create a snapshot of your database! (File → Take Database Snapshot)
- Create / Open Classy Database (Classy → Create / Open)
- Make sure that compiler options are correct (Options → Compiler)
- (Optional) Always show demangled names (Options → Demangled names → Select Names)

Theory

C++ Class Layout in Memory

```
// Usually stored in the data section.
struct vtable {
        void (*func1)();
        void (*func2)();
};

struct class {
        vtable* vtbl;
        int member1;
        int member2;
};
```

C++ Patterns in a Decompiler

```
// Initialize new instance of class
__int64 v1 = operator new(sizeof(class));
*v1 = gvtable; // stored somewhere in data.
*(v1 + 4) = 0;
*(v1 + 8) = 0;
```

```
// Call a vtable function
(*(void (*)())(*(_QWORD*)v1 + 8))();
```

Reconstructing Classes

Finding vtables and Inheritence Hierarchy

- Search for vtable in IDA, you should find something like this:

```
40FFC0 ; `vtable for'Polygon
40FFC0 _ZTV7Polygon    dq 0                    ; offset to this
40FFC8                 dq offset _ZTI7Polygon  ; `typeinfo for'Polygon
40FFD0 vtable      dq offset sub_408D40    ; DATA XREF: sub_408D40+11↑o
```

- Go to `_ZTI7Polygon` and search cross references for “reference to parent’s type name”, giving you a location close to any child classes:

```
3D40 ; public Triangle :
3D40 ;   public /* offset 0x0 */ Polygon
3D40 ; `typeinfo for'Triangle
```

Using Classy

- With the information gathered from before, start creating the hierarchy in classy
- Then add the correct vtable to every class in classy
- If possible, rename the functions correctly and add arguments in classy
- Details in demo later!

Creating Structures for Members

- Start with the base class and search for cross references to the vtable
- These are locations where class is constructed
- Use Structure Builder (right click → Show Structure Builder) to scan variable that is assigned the vtable
- This will automatically try to figure out how the struct layout should look like
- Go into any functions that use the newly allocated struct and scan as well
- Once satisfied, click finalize, you will be prompted to save the struct
- We first need to make some changes!

Allowing Inheritence

- Surround everything except the first vtable field in another struct, named type_members, e.g.:

```
struct class {
    vtable* vtbl;
    struct class_members {
        int member1;
        int member2;
    } mbrs;
};
```

- Now you can hit save

Adding Subclasses

- Works very similar to the base class, but you search for cross references to the vtable of the subclass
- Also, you want to have the members struct inherit from the base member struct and delete any fields that are duplicate, e.g.:

```
struct subclass {
    subvtable* vtbl;
    struct subclass_members : class_members {
        // int member1; dup
        // int member2; dup
        int member3;
    } mbrs;
};
```

Renaming Vtable Functions

- Once you created a struct, renaming vtable functions is not as easy anymore.
- If you rename them, the created vtable struct will not be automatically renamed as well!
- However, you can just go to the location of the vtable in the data section and press V
- This will "recreate" the vtable fixing up the namings!

C++ STL

What is the C++ STL?

- Standard Template Library, is the library containing all the C++ types you know and love: `std::string, std::vector, std::map`
- Two major issues present itself when reversing C++ binaries with STL types:
  - When compiled with `03` , about 90% of STL code will be inlined
  - Memory layouts of STL types are different for every major OS and often not very intuitive

C++ STL

Strings

Memory Layout

- struct of size 32, if string is less than 16 bytes, everything is stored in the struct
- otherwise, allocated on heap, in steps of powers of 2

```cpp
struct basic_string
{
    char *begin_; // actual string data
    size_t size_; // actual size
    union
    {
        size_t capacity_; // used if larger than 15 bytes
        char sso_buffer[16]; // used if smaller than 16 bytes
    };
};
```

Inlined Initializers

```
// v47 is of type basic_string!
v48 = 0LL;
v47 = (__int64)&v49;
LOBYTE(v49) = 0;
// once retyped:
v47.size_ = 0LL;
v47.begin_ = v47.sso_buffer;
v47.sso_buffer[0] = 0;
```

Inlined Constructors

```
// somewhere in the function, you might have this:
std::__throw_logic_error("basic_string::_M_construct null not valid");
// The whole function is probably just a string constructor,
// possible signatures:
string_construct(basic_string*, char* begin, char* end);
string_construct(basic_string*, basic_string*); // Copy
string_construct(basic_string*, char* begin, size_t size);
```

C++ STL

Vectors

Memory Layout

- struct of size 24, stores start, end and max pointer
- array is allocated on the heap, pointer type is dependent on vector elements

```
// Stores Point objects
struct vector_point
{
    Point* start;
    Point* end;
    Point* max;
};
// Stores Point pointers, more common
struct vector_point_p
{
    Point** start;
    Point** end;
    Point** max;
};
```

Inlined Size

- In case of storing pointers, size calculation is straight forward:

```
size_t size = (vec.end - vec.start) >> 3; // div by 8
```

- In other cases, the division might look more painful:

```
// assume vec is a vector<char[5]>;
// div by 5
size_t size = ((vec.end - vec.start) * 0xCCCCCCCCCCCCCCCD) >> 2;
```

Inlined Methods

```
// Will often contain something like this:
std::__throw_out_of_range_fmt
// However, might not indicate that the whole function is from STL!
// Common append inlined method:
Point** end = vec->end;
if ( end == vec->max ) {
    // allocate more memory
    result = sub_4090C0(&vec->start, end, (Point *)&newp);
} else {
    if ( end ) {
        result = newp;
        *end = newp;
    }
    vec->end = end + 1;
}
```

C++ STL

Maps

Memory Layout

- Implemented using a red-black tree, so complex memory layout

```
enum std::_Rb_tree_color : __int32
{
    _S_red = 0x0,
    _S_black = 0x1,
};
struct std::_Rb_tree_node_base
{
    std::_Rb_tree_color _M_color;
    struct std::_Rb_tree_node* _M_parent;
    struct std::_Rb_tree_node* _M_left;
    struct std::_Rb_tree_node* _M_right;
};
```

Memory Layout

```
struct std::map
{
    void* allocator; // uninteresting
    std::_Rb_tree_node_base _M_header;
    size_t _M_node_count;
};
// For a map of type map<string, Point*>
struct std::_Rb_tree_node : std::_Rb_tree_node_base
{
    struct string_point_pair
    {
        basic_string string;
        Point* point;
    } pair;
};
```

Inlined Initalization

```
v42 = operator new(0x30LL);
*(_DWORD *)(v42 + 8) = 0;
*(_QWORD *)(v42 + 16) = 0LL;
*(_QWORD *)(v42 + 40) = 0LL;
*(_QWORD *)(v42 + 24) = v42 + 8;
*(_QWORD *)(v42 + 32) = v42 + 8;
// After applying type:
map = (std::map*)operator new(0x30LL);
map->_M_t._M_impl._M_header._M_color = 0;
map->_M_t._M_impl._M_header._M_parent = 0LL;
map->_M_t._M_impl._M_node_count = 0LL;
map->_M_t._M_impl._M_header._M_left = &map->_M_t._M_impl._M_header;
map->_M_t._M_impl._M_header._M_right = &map->_M_t._M_impl._M_header;
```

Demo

Demo Time