When Virtual Hell Freezes Over- Reversing C++ Code

<3

Gal Zaban
@0xgalz




id;whoami

- Gal Zaban
- Reverse Engineer
- Security Researcher at Viral Security Group
- In my spare-time I like sewing

This is my own **private** research

Agenda

- REsearch
  - C++ Internals
    - Object Creation
    - Inheritance
    - Multiple Inheritance
    - Vtables
    - Virtual calls
- DEvelopment
  - IDAPython - Breakpoints
  - “Virtualor” - IDAPython framework that automates reverse engineering of C++

The Problem

Reversing C++ is Hard

Dynamic Object Creation

```
class Father1
{
public:
    Father1(int age) { _age = age; }
    virtual int GetAge() { return _age; }
    virtual void PrintAge() { cout << "Father1's age is: " << _age << endl; }
private:
    int _age;
};

int main()
{
    Father1* objA = new Father1(52);
```

Dynamic Object Creation

```
class Father1
{
public:
    Father1(int age) { _age = age; }
    virtual int GetAge() { return _age; }
    virtual void PrintAge() { cout << "Father1's age is: " << _age << endl; }
private:
    int _age;
};

int main()
{
    Father1* objA = new Father1(52);
```

```
push    8                   ; Size
call    operator new(uint)
```

Dynamic Object Creation

```
class Father1
{
public:
    Father1(int age) { _age = age; }
    virtual int GetAge() { return _age; }
    virtual void PrintAge() { cout << "Father1's age is: " << _age << endl; }
private:
    int _age;
};

int main()
{
    Father1* objA = new Father1(52);
```

```
call    j_gz_Father1_ctor
mov     [ebp+this_ptr_Father1], eax
```

Dynamic Object Creation

Object Creation	
Action	**Assembly**
Heap Allocation	call operator new(uint)
Constructor Call	call j_gz_Object_ctor

Basic Constructor

```
mov     eax, [ebp+p_this_object]
mov     dword ptr [eax], offset const Account::`vftable'
mov     eax, [ebp+p_this_object]
movsd   xmm0, [ebp+objects_member]
movsd   qword ptr [eax+8], xmm0
```

Object	Assembly
VTable	mov dword ptr [eax], VTable
Member1	movsd qword ptr [eax+8], xmm0
Member2	-
...	-
MemberX	-

How Does A Vtable Look Like?

```
class Father0
{
public:
    virtual void PrintHello() { cout << "Hello you!"; }
    virtual void PrintHelloMe() { cout << "Hello Father0!"; }
};

class FatherA : public Father0
{
public:
    virtual void PrintHelloMe() { cout << "Hello Father1!"; }
    virtual void PrintNum() { cout << x; }


private:
    int _x = 5;
};
```

Vtable In IDA

```
.rdata:1003AB04 off_1003AB04    dd offset sub_10001500
.rdata:1003AB04
.rdata:1003AB08                 dd offset sub_10002031
.rdata:1003AB0C                 dd offset sub_10001451
.rdata:1003AB10                 dd offset sub_1000141F
```

VTables and Virtual Calls

```
mov     [ebp+this_object], eax
mov     eax, [ebp+this_object]
mov     edx, [eax]
mov     esi, esp
mov     ecx, [ebp+this_object]
mov     eax, [edx+4]
call    eax
```

Assignment of the vtable to EDX

Move the virtual func to EAX

The Virtual Call

Multiple Inheritance

```
class FatherB
{
public:
    virtual void PrintHello() { cout << "Hello, I'm FatherB" ; }
    virtual void PrintFather() { cout << "FatherB"; }
};

class FatherA : public Father0
{
public:
    virtual void PrintHello() { cout << "Hello, I'm FatherA" ; }
    virtual void PrintFather() { cout << "FatherA"; }
private:
    int _x = 5;
};

class C : public FatherA, public FatherB
{
public:
    virtual void foo2() { cout << "C::foo2\n"; }
};
```

Multiple Inheritance

Multiple Inheritance Structure
FatherA
FatherB
C's Members

The Son's Full Object
C_A_VTable
FatherA_Member1
....
FatherA_MemberX
C_B_VTable
FatherB_Member1
...
FatherB_MemberX
C_Member1
...
C_MemberX

Function Calls w Multiple Inheritance

```
int main()
{
    Father0* p1 = new Father0();
    FatherA* p2 = new FatherA();
    FatherB* p3 = new FatherB();
    C* p4 = new C();

    p1->PrintHello();
    p2->PrintHello();
    p2->PrintFather();
    p3->PrintHello();
    p3->PrintFather();
    p4->FatherA::PrintFather();
    p4->FatherA::PrintHello();
    p4->FatherB::PrintFather();
    p4->FatherB::PrintHello();
    p4->foo2();
    return 0;
}
```

```
mov     ecx, [ebp+this_ptr_C]
add     ecx, 4
call    j_gz_FatherA_PrintFather
mov     ecx, [ebp+this_ptr_C]
add     ecx, 4
call    j_gz_FatherA_PrintHello
mov     ecx, [ebp+this_ptr_C]
call    j_gz_FatherB_PrintFather
mov     ecx, [ebp+this_ptr_C]
call    j_gz_FatherB_PrintHello
mov     eax, [ebp+this_ptr_C]
mov     edx, [eax]
mov     esi, esp
mov     ecx, [ebp+this_ptr_C]
mov     eax, [edx+8]
virtual call- C::foo2():
call    eax
```

It requires a lot of work

I wanted to make it fluffy

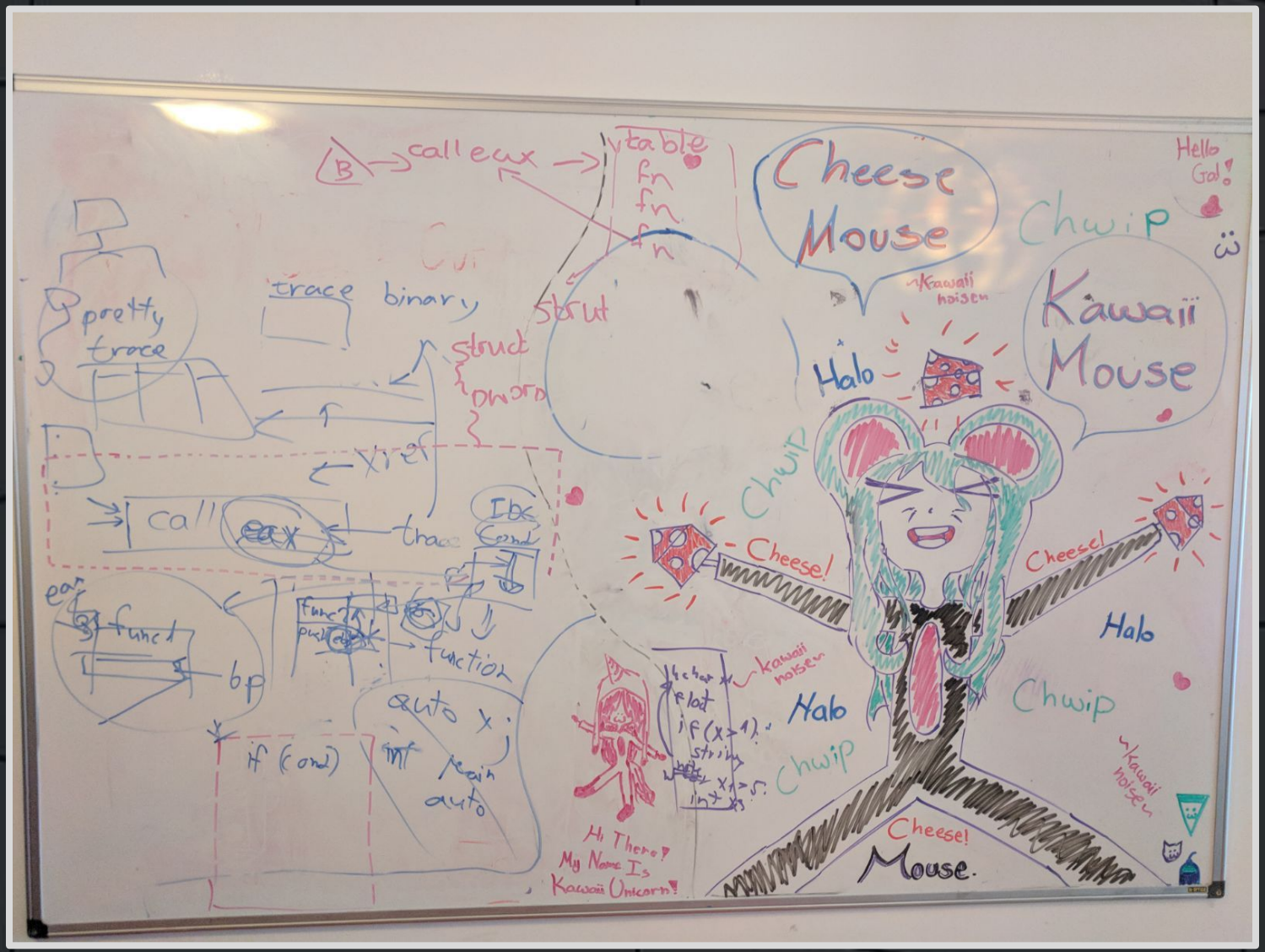
IDAPython + IDC =

IDAPython is ezpz to write

🤯

But IDC is more extensive

How it all began



Virtualor

Automated IDA tracing

- Create trace breakpoints on virtual calls
- Parse the trace file created by IDA

```
def define_function_trace(adr):
    return idc.SetBptAttr(adr, idc.BPTATTR_FLAGS, idc.BPT_ENABLED | idc.BPT_TRACE | idc.BPT_TRACEON | idc.BPT_TRACE_FUNC)
```

The Tracing problem

- It didn’t give a realtime solution for vtables
- This solution can only provide the specific function call and not all the vtable

How can we make it a dynamic solution?

- Taint backward to the instruction that assigns the relevant function to the register of the virtual call
- Create the structure of the vtable based on the vtable base pointer
- Correlate between the structure and the vtable pointer

IDAPython- How to create a Breakpoint

```
import idautils
import idaapi
import idc

class ConditionalBreakPoint:
    def __init__(self):
        pass

    def set_bp(self, adr, cnd):
        print "breakpoint on %08x" % adr
        idaapi.add_bpt(adr, 0, idc.BPT_SOFT)
        idaapi.enable_bpt(adr, True)
        idc.SetBptCnd(adr, cnd)

    def delete_bp(self, adr):
        idaapi.del_bpt(adr)
```

Hook VTables Pointers

- Find all the virtual calls
- Add breakpoints on the vtable's function assignment

```
while cur_addr >= start_addr:
    if idc.GetMnem(cur_addr)[:3] == "mov" and idc.GetOpnd(cur_addr, 0) == i_cnt:
        opnd2 = idc.GetOpnd(cur_addr, 1)
        place = opnd2.find('+') #
        register = ''
        offset = ''
        if place != -1: # if the function is not the first in the vtable
            register = opnd2[opnd2.find('[') + 1: place]
            offset = opnd2[place + 1: opnd2.find(']')]
            return register, offset, cur_addr
        else:
            offset = "0"
            register = opnd2[opnd2.find('[') + 1: opnd2.find(']')]
            return register, offset, cur_addr
    cur_addr = idc.PrevHead(cur_addr)
```

Conditional BP as a hook

- Write code inside the BP conditions
- Add false binary condition in order to disable the breakpoint prior to the BP execution

Conditionals BP and IDAPython

- By default IDAPython support only IDC Conditional Breakpoints
- In IDC conditions we cannot #include idc.idc

IDAPython internals

- Diving into the files of IDAPython modules
- We must find a way to change the condition to IDAPython

```python
def set_bpt_cond(ea, cnd, is_lowcnd=0):
    """
    Set breakpoint condition

    @param ea: any address in the breakpoint range
    @param cnd: breakpoint condition
    @param is_lowcnd: 0 - regular condition, 1 - low level condition

    @return: success
    """
    bpt = ida_dbg.bpt_t()

    if not ida_dbg.get_bpt(ea, bpt):
        return False

    bpt.condition = cnd
    if is_lowcnd:
        bpt.flags |= BPT_LOWCND
    else:
        bpt.flags &= ~BPT_LOWCND

    return ida_dbg.update_bpt(bpt)
```

```
// ************************************************
class Breakpoint
{
  // Breakpoint type. One of BPT_... constants
  attribute type;

  // Breakpoint size (for hardware breakpoint)
  attribute size;

  // Breakpoint condition (string)
  attribute condition;

  // Scripting language of the condition string
  // "IDC" for IDC, "Python" for Python etc. ('name' field of extlang_t)
  // if empty, default extlang is assumed
  attribute elang;

  // Breakpoint flags. Refer to BPTATTR_FLAGS
  attribute flags;

  // Breakpoint properties. Refer to BKPT_... constants
  attribute props;
```

```
class bpt_t(object):
    """
    Proxy of C++ bpt_t class
    """
  condition =  swig_property( ida_dbg.bpt_t_condition_get,  ida_dbg.bpt_t_condition_set)
  elang = _swig_property(_ida_dbg.bpt_t_elang_get, _ida_dbg.bpt_t_elang_set)
  __swig_destroy__ = _ida_dbg.delete_bpt_t
  __del__ = lambda self : None;
```

The new BP Creation

```
def set(self, break_p=False):
    print "breakpoint on %08x" % self.address
    print "=Breakpoint Added=\n"
    idaapi.add_bpt(self.address, 0, idc.BPT_SOFT)
    idaapi.enable_bpt(self.address, True)
    bpt = idaapi.bpt_t()
    idaapi.get_bpt(self.address, bpt)
    bpt.elang = self.elang
    bpt.condition = self.condition.get_text()
    idaapi.update_bpt(bpt)
    print bpt.condition
```

The Hook Purpose

- Create IDA structures of the vtables
- Connect the structures with the virtual calls
- Add comments and references to the code
- Correlate the vtable base pointer to its struct

The Hook location

- The breakpoint located on the assignment of the relevant function to the register.

```
mov     esi, esp
mov     ecx, [ebp+this_obj]
mov     eax, [edx+4]
call    eax
```

Get The Vtable Pointer

What Created the Hook

```
p_vtable = idc.GetRegValue(\"""" + reg_vtable + """\")

pv_func_addr = idc.GetRegValue(\"""" + reg_vtable + """\") + """ + offset + """
```

```
mov     esi, esp
mov     ecx, [ebp+this_obj]
mov     eax, [edx+4]
call    eax
```

Get The Vtable Pointer

- And this is how it looks in the hook's condition:

Edit script

Please enter script body

```
p_vtable = idc.GetRegValue("edx")
pv_func_addr = idc.GetRegValue("edx") + 4
```

```
mov     esi, esp
mov     ecx, [ebp+this_obj]
mov     eax, [edx+4]
call    eax
```

Get Functions From Vtable

What Created the Hook

```
all_functions = []

if """ + offset + """ > 0:

    cnt = 0

    while cnt <= """ + offset + """:

        pv_func_addr = idc.GetRegValue( \""""" + reg_vtable + """\") + cnt

        v_func_addr = get_wide_dword(pv_func_addr)

        v_func_name = GetFunctionName(v_func_addr)

        all_functions.append(v_func_name)

        cnt += 4
```

Now we have we have the vtable!

Create The Structure

What Created the Hook	The Vtable Name
`struct_id = add_struc(-1, "vtable_" + hex(p_vtable), 0)`	`vtable_0x1379ba8L`

Add Vtable Functions as Members

What Created the Hook	Functions Members Examples

`cnt = 0`
`for func_name in all_functions:`
 	
`  idc.add_struc_member(struct_id, "v_" + func_name,`
`  cnt*4 , FF_DWRD, -1, 4)`
`  cnt += 1`
	`v_sub_1359e84`
	
	OR
	
	`v_gz_calc_size`

This is how the structure looks like now...

```
vtable_0x1379ba8L struc
v_gz_j_get_balance dd ?
v_sub_1371497    dd ?
vtable_0x1379ba8L ends
```

Unfortunately It's not Fluffy Enough..

Because we also want comments!

Add Comments To The Structure

- Add where the function were assigned
- Add function’s names to existing comments
  - using the same function from different parts of the code.

Add Comments To The Structure

What Created the Hook

```
cmt_curr = idc.GetMemberComment(struct_id, cnt*4, 1)

# New Comment
if cmt_curr== None:
   if """ + offset + """ == cnt*4:
       idc.SetMemberComment(struct_id, cnt*4 , "Was used in address:"
                                + " """+ hex(start_addr) + """" , 1)

# Adding function's names to existing comment
else:

   cmt_new = cmt_curr
   cmt_new += ", " + " """+ hex(start_addr) + """   "
   idc.SetMemberComment(struct_id, cnt*4 , cmt_new , 1)
```

Add Comments To The Assembly

What Created the Hook

```
virtual_call_addr = """ + hex(start_addr) + """

last_text = idc.get_cmt(virtual_call_addr, 1)
if last_text == None:
   last_text = ""

idc.set_cmt(virtual_call_addr, last_text + "vtable structure is:
" + "vtable_" + hex(p_vtable) + ", function: " + curr_func, 1)
```

And One Last Thing To Add ...

Structure Offset and False Condition

What Created the Hook

```
idc.op_stroff(virtual_call_addr, 1, struct_id, 0)

"Gal" == "IDA"
```

Now The Hook Is Finished!

The Hook

```
p_vtable = idc.GetRegValue(\""" + reg_vtable + """\")
pv_func_addr = idc.GetRegValue(\""" + reg_vtable + """\") + """ + offset + """
curr_func = ""
all_functions = []
if """ + offset + """ > 0:
    cnt = 0
    while cnt <= """ + offset + """:
        pv_func_addr = idc.GetRegValue(\""" + reg_vtable + """\") + cnt
        v_func_addr = get_wide_dword(pv_func_addr)
        v_func_name = GetFunctionName(v_func_addr)
        all_functions.append(v_func_name)
        cnt += 4

        #virtual_call_addr = idc.NextHead(idc.here())
struct_id = add_struc(-1, "vtable_" + hex(p_vtable), 0)
cnt = 0
for func_name in all_functions:
    print "hello"
    idc.add_struc_member(struct_id, "v_" + func_name, cnt*4 , FF_DWRD, -1, 4)
    cmt_curr = idc.GetMemberComment(struct_id, cnt*4, 1)
    if cmt_curr== None:
        if """ + offset + """ == cnt*4:
            idc.SetMemberComment(struct_id, cnt*4 , "Was used in address:" + " """ + hex(start_addr) + """ , 1)
            curr_func = func_name
    else:
        cmt_new = cmt_curr
        cmt_new += ", " + " """ + hex(start_addr) + """ "
        idc.SetMemberComment(struct_id, cnt*4 , cmt_new , 1)
        print cmt_new
    cnt += 1
virtual_call_addr = """ + hex(start_addr) + """
last_text = idc.get_cmt(virtual_call_addr, 1)
if last_text == None:
    last_text = ""
idc.set_cmt(virtual_call_addr, last_text + "vtable structure is: " + "vtable_" + hex(p_vtable) + ", function: " + curr_func, 1)
idc.op_stroff(virtual_call_addr, 1, struct_id, 0)
"gal" == "IDA"
```

Before

```
mov     eax, [ebp+var_14]
mov     edx, [eax]
mov     esi, esp
mov     ecx, [ebp+var_14]
mov     eax, [edx+4]
call    eax
cmp     esi, esp
call    j___RTC_CheckEsp
mov     eax, [ebp+var_20]
mov     edx, [eax]
mov     esi, esp
mov     ecx, [ebp+var_20]
mov     eax, [edx]
call    eax
cmp     esi, esp
```

After- vtable structures

```
vtable_0x1279b34L struc ; (sizeof=0x8, mappedto_47)
j_gz_FatherA_PrintHello dd ?            ; XREF: _main_0+164/r ; Was called from address: 0x01272A26L
j_gz_FatherA_PrintFather dd ?           ; XREF: _main_0+179/r ; Was called from address: 0x01272A3CL
vtable_0x1279b34L ends

; ---------------------------------------------------------------------------

vtable_0x138a34b1L struc ; (sizeof=0x8, mappedto_48)
j_gz_FatherB_PrintHello dd ?            ; XREF: _main_0+18F/r ; Was called from address: 0x01272A51L
j_gz_FatherB_PrintFather dd ?           ; XREF: _main_0+1A4/r ; Was called from address: 0x01272A67L
vtable_0x138a34b1L ends
```

After- The Disassembly

```
mov     eax, [ebp+this_ptr2]
mov     edx, [eax]
mov     esi, esp
mov     ecx, [ebp+this_ptr2]
mov     eax, [edx+vtable_0x1279b34L.j_gz_FatherA_PrintFather] ; Was called from address: 0x01272A3CL
call    eax             ; vtable struct is: vtable_0x1279b34L, function j_gz_FatherA_PrintFather
cmp     esi, esp
call    j___RTC_CheckEsp
mov     eax, [ebp+this_ptr3]
mov     edx, [eax]
mov     esi, esp
mov     ecx, [ebp+this_ptr3]
mov     eax, [edx+vtable_0x138a34b1L.j_gz_FatherB_PrintHello] ; Was called from address: 0x01272A51L
call    eax             ; vtable struct is: vtable_0x138a34b1L, function j_gz_FatherB_PrintHello
```

```
.text:01272A3C                 call    eax             ; vtable struct is: vtable_0x1279b34L,
.text:01272A3E                 cmp     esi, esp
.text:01272A40                 call    j___RTC_CheckEsp
.text:01272A45                 mov     eax, [ebp+this_ptr3]
.text:01272A48                 mov     edx, [eax]
.text:01272A4A                 mov     esi, esp
.text:01272A4C                 mov     ecx, [ebp+this_ptr3]
.text:01272A4F                 mov     eax, [edx+vtable_0x138a34b1L.j_gz_FatherB_PrintHello] ;
.text:01272A51                 call    eax             ; vtable struct is: vtable_0x138a34b1L,
```

What’s next?

- Add structures for all the objects (local, static, dynamic) and the inheritance.

- Add logic to the names of the functions in the vtables based on their code: strings, function calls, loops and more.

thank you!

@0xgalz