



IBM Global Services

# Reversing C++

**Paul Vincent Sabanal**  
**X-Force R&D**  
**Mark Vincent Yason**  
**X-Force R&D**



IBM Internet Security Systems  
Ahead of the threat.™

© Copyright IBM Corporation 2007



Reversing C++

# Part I. Introduction



## Introduction > Purpose

- **Understand C++ concepts as they are represented in disassemblies**
- **Have a big picture idea on what are major pieces (classes) of the C++ target and how these pieces relate together (class relationships)**

## Introduction > Focus On...

- **(1) Identifying Classes**
- **(2) Identifying Class Relationships**
- **(3) Identifying Class Members**

## Introduction > Motivation

- **Increasing use of C++ code in malware**
  - Difficult to follow virtual function calls in static analysis
  - Examples: Agobot, Mytob, new malcodes from our honeypot
- **Most modern applications use C++**
  - For binary auditing, reversers can expect that the target can be a C++ compiled binary
- **General lack of publicly available information regarding the subject of C++ reversing**
  - Only good information is from Igor Skochinsky
  - [https://www.openrce.org/articles/full\\_view/23](https://www.openrce.org/articles/full_view/23)



Reversing C++

## Part II. Manual Approach





Reversing C++

## Part II. Manual Approach

### Identifying C++ Binaries & Constructs



## Manual Approach > Identifying C++ Binaries & Constructs

### ■ Heavy use of ecx (*this ptr*)

```
.text:004019E4      mov      ecx, esi  
.text:004019E6      push     0BBh  
.text:004019EB      call    sub_401120
```

### ■ ecx used without being initialized

```
.text:004010D0 sub_4010D0      proc near  
.text:004010D0           push    esi  
.text:004010D1           mov     esi, ecx  
.text:004010DD           mov     dword ptr [esi], offset off_40C0D0  
.text:00401101           mov     dword ptr [esi+4], 0BBh  
.text:00401108           call   sub_401EB0  
.text:0040110D           add    esp, 18h  
.text:00401110           pop    esi  
.text:00401111           retn  
.text:00401111 sub_4010D0      endp
```

## Manual Approach > Identifying C++ Binaries & Constructs

### ■ Parameters on the stack, `ecx = this ptr`

```
.text:00401994 push    0Ch
.text:00401996 call    ??2@YAPAXI@Z      ; operator new(uint)
.text:004019AB mov     ecx, eax
:::
.text:004019AD call    ClassA_ctor
```

### ■ Virtual function calls (indirect calls)

```
.text:00401996 call    ??2@YAPAXI@Z      ; operator new(uint)
:::
.text:004019B2 mov     esi, eax
:::
.text:004019FF mov     eax, [esi]       ;EAX = vftable
.text:00401A01 add    esp, 8
.text:00401A04 mov     ecx, esi
.text:00401A06 push   0CCh
.text:00401A0B call    dword ptr [eax]
```

## Manual Approach > Identifying C++ Binaries & Constructs

### ■ STL Code and Imported DLLs

Address	Ordinal	Name	Library
00402030		InterlockedExchange	KERNEL32
00402038		?endl@std@@YAAV?\$basic_ostream@DU?\$char_traits@D@std...	MSVCP80
0040203C		?setstate@?\$basic_ios@DU?\$char_traits@D@std@@@std@@QA...	MSVCP80
00402040		?cout@std@@@3V?\$basic_ostream@DU?\$char_traits@D@std@@@...	MSVCP80
00402044		?uncaught_exception@std@@YA_NXZ	MSVCP80
00402048		?sputn@?\$basic_streambuf@DU?\$char_traits@D@std@@@std@...	MSVCP80
0040204C		?_Osfx@?\$basic_ostream@DU?\$char_traits@D@std@@@std@@@...	MSVCP80

```
.text:00401201    mov      ecx, eax
.text:00401203    call     ds :?sputc@?$basic_streambuf@DU?$char_traits@D@std@@@std@@QAEHD@Z
; std::basic_streambuf<char, std::char_traits<char>>::sputc(char)
```

## Manual Approach > Class Instance Layout

### Class Instance Layout

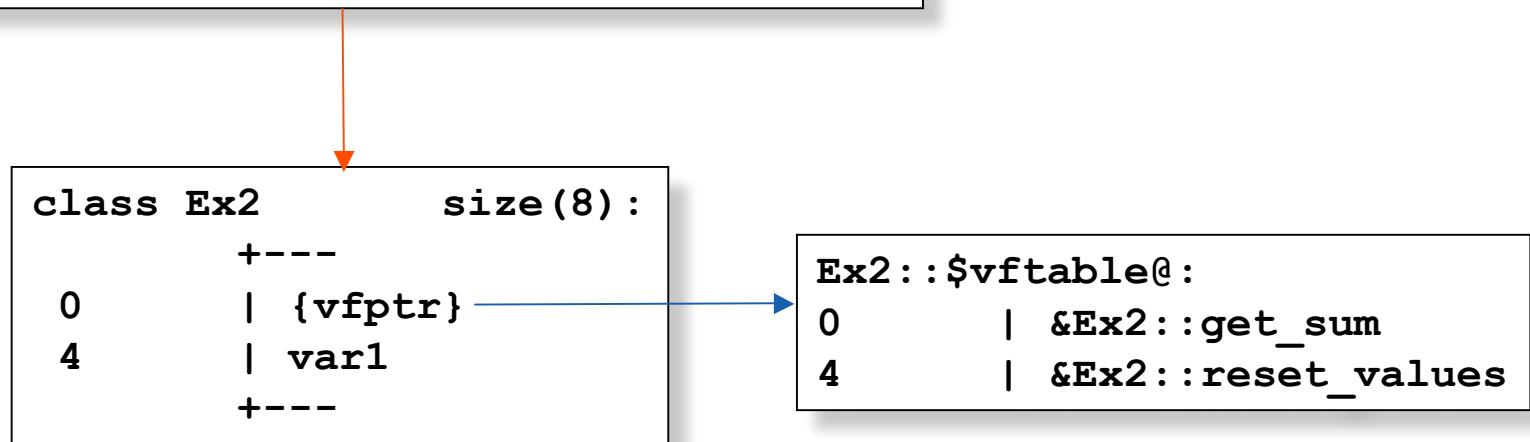
```
class Ex1
{
    int var1;
    int var2;
    char var3;
public:
    int get_var1();
};
```

```
class Ex1           size(12) :
                    +---+
0                 | var1
4                 | var2
8                 | var3
| <alignment member> (size=3)
+---+
```

## Manual Approach > Class Instance Layout

### Class Instance Layout

```
class Ex2
{
    int var1;
public:
    virtual int get_sum(int x, int y);
    virtual void reset_values();
};
```



## Manual Approach > Class Instance Layout

### Class Instance Layout

```
class Ex3: public Ex2
{
    int var1;
public:
    void get_values();
};
```

```
class Ex3           size(12):
+---+
| +--- (base class Ex2)
0   |   | {vptr}
4   |   | var1
| +---+
8   |   | var1
+---+
```

## Manual Approach > Class Instance Layout

### Class Instance Layout

```
class Ex4
{
    int var1;
    int var2;
public:
    virtual void func1();
    virtual void func2();
};

class Ex5: public Ex2, Ex4
{
    int var1;
public:
    void func1();
    virtual void v_ex5();
};
```



```
class Ex5           size(24):
+---+
| +--- (base class Ex2)
0   | | {vptr}
4   | | var1
| +---+
| +--- (base class Ex4)
8   | | {vptr}
12  | | var1
16  | | var2
| +---+
20  | var1
+---+
```



**Reversing C++**

## Part II. Manual Approach Identifying Classes



Manual Approach > Identifying Classes > Constructor/Destructor Identification

## ■ Global Objects

- Allocated in the data segment
- Constructor is called at program startup
- Destructor is called at program exit
- *this* pointer points to a global variable
- To locate constructor/destructor, examine cross-references

Manual Approach > Identifying Classes > Constructor/Destructor Identification

## ■ Local Objects

- Allocated in the stack
- Constructor is called at declaration
- *this* pointer points to an uninitialized local variable
- Destructor is called at block exit

## Manual Approach &gt; Identifying Classes &gt; Constructor/Destructor Identification

## ■ Local Objects

```
.text:00401060 sub_401060      proc near
.text:00401060
.text:00401060 var_C          = dword ptr -0Ch
.text:00401060 var_8           = dword ptr -8
.text:00401060 var_4           = dword ptr -4
.text:00401060
...(some code)...
.text:004010A4             add    esp, 8
.text:004010A7             cmp    [ebp+var_4], 5
.text:004010AB             jle    short loc_4010CE
.text:004010AB
.text:004010AB { ← block begin
.text:004010AD             lea    ecx, [ebp+var_8] ; var_8 is uninitialized
.text:004010B0             call   sub_401000 ; constructor
.text:004010B5             mov    edx, [ebp+var_8]
.text:004010B8             push   edx
.text:004010B9             push   offset str->WithinIfX
.text:004010BE             call   sub_4010E4
.text:004010C3             add    esp, 8
.text:004010C6             lea    ecx, [ebp+var_8]
.text:004010C9             call   sub_401020 ; destructor
.text:004010CE } ← block end
.text:004010CE
.text:004010CE loc_4010CE:          ; CODE XREF: sub_401060+4Bj
.text:004010CE             mov    [ebp+var_C], 0
.text:004010D5             lea    ecx, [ebp+var_4]
.text:004010D8             call   sub_401020
```

Manual Approach > Identifying Classes > Constructor/Destructor Identification

## ■ **Dynamically Allocated Objects**

- Allocated in the heap
- Created via operator *new*
  - Allocates memory in heap
  - Calls the constructor
- Destructor is called via operator *delete*
  - Calls destructor
  - De-allocates object instance

## Manual Approach &gt; Identifying Classes &gt; Constructor/Destructor Identification

## Dynamically Allocated Objects

```
.text:0040103D _main          proc near
.text:0040103D argc           = dword ptr  8
.text:0040103D argv           = dword ptr  0Ch
.text:0040103D envp           = dword ptr  10h

.push    esi
.push    4                  ; size_t
.text:00401040 call    ??2@YAPAXI@Z ; operator new(uint)
.test    eax, eax ; eax = address of allocated memory
.pop     ecx
.jz     short loc_401055
.mov    ecx, eax
.text:0040104C call    sub_401000 ; call to constructor
.mov    esi, eax
.jmp    short loc_401057
                ; CODE XREF: _main+Bj
.text:00401055 loc_401055: xor     esi, esi
                ; CODE XREF: _main+16j
.text:00401057 loc_401057: push    45h
.mov    ecx, esi
.call   sub_401027
.test    esi, esi
.jz     short loc_401072
.mov    ecx, esi
.text:00401066 call    sub_40101B ; call to destructor
.push    esi           ; void *
.call   j__free ; call to free thunk function
.pop     ecx
                ; CODE XREF: _main+25j
.text:00401072 loc_401072: xor     eax, eax
.pop     esi
.text:00401074 retn
.text:00401075 _main         endp
```

## Manual Approach > Identifying Classes > via RTTI

### ■ What is RTTI

- Run-time Type Information (RTTI)
- Used for identification of object type on run-time
- Generated for polymorphic classes (classes with virtual functions)
- Utilized by operators `typeid` and `dynamic_cast`
- Will give us important information on
  - Class Name
    - Rough idea what the class is all about
  - Class Hierarchy
- Consists of several data structures

## Manual Approach > Identifying Classes > via RTTI

### ■ **RTTICCompleteObjectLocator**

- Contains pointers to two structures that identifies
  - Class information (**TypeDescriptor**)
  - Class Hierarchy (**RTTIClassHierarchyDescriptor**)
- Located just below the class' vftable

```
.rdata:00404128          dd offset ClassA_RTTICCompleteObjectLocator
.rdata:0040412C ClassA_vftable dd offset sub_401000 ; DATA XREF:...
.rdata:00404130          dd offset sub_401050
.rdata:00404134          dd offset sub_4010C0
.rdata:00404138          dd offset ClassB_RTTICCompleteObjectLocator
.rdata:0040413C ClassB_vftable dd offset sub_4012B0 ; DATA XREF:...
.rdata:00404140          dd offset sub_401300
.rdata:00404144          dd offset sub_4010C0
```

## Manual Approach > Identifying Classes > via RTTI

### ■ RTTICompleteObjectLocator

Offset	Type	Name	Description
0x00	DW	signature	Always 0?
0x04	DW	offset	Offset of vftable within the class
0x08	DW	cdOffset	?
0x0C	DW	pTypeDescriptor	Class Information
0x10	DW	pClassHierarchy Descriptor	Class Hierarchy Information

```
.rdata:004045A4 ClassB_RTTICompleteObjectLocator
    dd 0          ; COL.signature
.rdata:004045A8    dd 0          ; COL.offset
.rdata:004045AC    dd 0          ; COL.cdOffset
.rdata:004045B0    dd offset ClassB_TypeDescriptor
.rdata:004045B4    dd offset ClassB_RTTIClassHierarchyDescriptor
```

## Manual Approach > Identifying Classes > via RTTI

### ■ TypeDescriptor

- Contains the class name (which is an important information)
- Say **CPacketParser** and **CTCPPacketParser**

Offset	Type	Name	Description
0x00	DW	pVFTable	Always points to type_info's vftable
0x04	DW	spare	?
0x08	SZ	name	Class Name

```
.data:0041A098 ClassA_TypeDescriptor ; DATA XREF: ....
        dd offset type_info_vftable    ; TypeDescriptor.pVFTable
.data:0041A09C          dd 0           ; TypeDescriptor.spare
.data:0041A0A0          db '.?AVClassA@@',0 ; TypeDescriptor.name
```

## Manual Approach > Identifying Classes > via RTTI

### ■ **RTTIClassHierarchyDescriptor**

- Information about the class hierarchy
- Includes pointers to `BaseClassDescriptors` for each base class

Offset	Type	Name	Description
0x00	DW	signature	Always 0?
0x04	DW	attributes	Bit 0 – multiple inheritance Bit 1 – virtual inheritance
0x08	DW	numBaseClasses	Number of base classes. Count includes the class itself
0x0C	DW	pBaseClassArray	Array of <code>RTTIBaseClassDescriptor</code>

## Manual Approach > Identifying Classes > via RTTI

### ■ **RTTIClassHierarchyDescriptor**

- Example class declaration

```
class ClassA {...}
class ClassE {...}
class ClassG: public virtual ClassA, public virtual ClassE {...}
```

- Corresponding RTTIClassHierarchyDescriptor

```
.rdata:004178C8 ClassG_RTTIClassHierarchyDescriptor ; DATA XREF: ...
.rdata:004178C8          dd 0                      ; signature
.rdata:004178CC          dd 3                      ; attributes
.rdata:004178D0          dd 3                      ; numBaseClasses
.rdata:004178D4          dd offset ClassG_pBaseClassArray ; pBaseClassArray
.rdata:004178D8 ClassG_pBaseClassArray
                      dd offset oop_re$RTTIBaseClassDescriptor@4178e8
.rdata:004178DC          dd offset oop_re$RTTIBaseClassDescriptor@417904
.rdata:004178E0          dd offset oop_re$RTTIBaseClassDescriptor@417920
```

## Manual Approach > Identifying Classes > via RTTI

### ■ **RTTIBaseClassDescriptor**

- Information about the base class
- Contains the TypeDescriptor for the base class

Offset	Type	Name	Description
0x00	DW	pTypeDescriptor	TypeDescriptor of this base class
0x04	DW	numContainedBases	Number of direct bases of this base class
0x08	DW	PMD.mdisp	vftable offset
0x0C	DW	PMD.pdisp	vtable offset (-1: vtable is at displacement PMD.mdisp inside the class)
0x10	DW	PMD.vdisp	Displacement of the base class vftable pointer inside the vtable
0x14	DW	attributes	?
0x18	DW	pClassDescriptor	RTTIClassHierarchyDescriptor of this base class

## Manual Approach > Identifying Classes > via RTTI

### ■ **vtable (virtual base class table)**

- Contains information necessary to locate the actual base class within class
- Generated for multiple virtual inheritance and used for *upcassing* (casting to base classes)

```
class ClassG           size(28):  
    +---  
0     | {vfptr}  
4     | {vbptr}  
    +---  
    +--- (virtual base ClassA)  
8     | {vfptr}  
12    | class_a_var01  
16    | class_a_var02  
    | <alignment member> (size=3)  
    +---  
    +--- (virtual base ClassE)  
20    | {vfptr}  
24    | class_e_var01  
    +---
```

```
ClassG::$vtable@:  
0      | -4  
1      | 4 (ClassGd(ClassG+4)ClassA)  
2      | 16 (ClassGd(ClassG+4)ClassE)
```

## Manual Approach > Identifying Classes > via RTTI

### ■ RTTIBaseClassDescriptor (example)

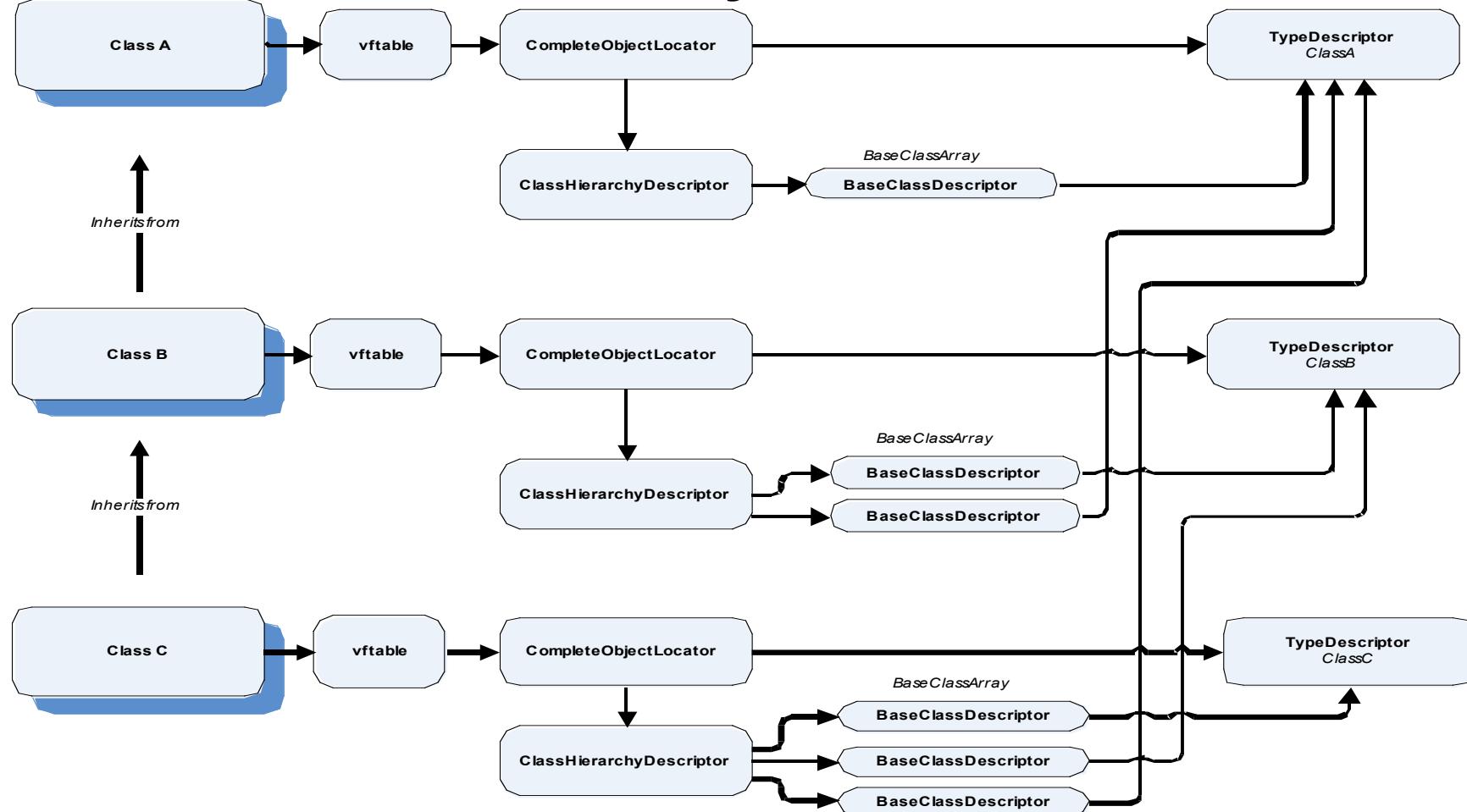
```
class ClassG           size(28):
    +---+
0     | {vfptr}
4     | {vbptr}
    +---+
    +--- (virtual base ClassA)
8     | {vfptr}
12    | class_a_var01
16    | class_a_var02
    | <alignment member> (size=3)
    +---+
    +--- (virtual base ClassE)
20    | {vfptr}
24    | class_e_var01
    +---+
```

```
ClassG::$vtable@:
0          | -4
1          | 4 (ClassGd(ClassG+4)ClassA)
2          | 16 (ClassGd(ClassG+4)ClassE)
```

```
.rdata:00418AFC RTTIBaseClassDescriptor@418afc      ; DATA XREF: ...
              dd offset oop_re$ClassE$TypeDescriptor
.rdata:00418B00  dd 0                      ; numContainedBases
.rdata:00418B04  dd 0                      ; PMD.mdisp
.rdata:00418B08  dd 4                      ; PMD.pdisp
.rdata:00418B0C  dd 8                      ; PMD.vdisp
.rdata:00418B10  dd 50h                     ; attributes
.rdata:00418B14  dd offset oop_re$ClassE$RTTIClassHierarchyDescriptor ;
pClassDescriptor
```

## Manual Approach > Identifying Classes > via RTTI

### RTTI Data Structures Layout





**Reversing C++**

## Part II. Manual Approach Identifying Class Relationship



## Manual Approach > Identifying Relationship > Constructor Analysis

### ■ Single Inheritance

```
.text:00401010 sub_401010    proc near
.text:00401010
.text:00401010 var_4          = dword ptr -4
.text:00401010
.text:00401010
.text:00401010 push    ebp
.text:00401011 mov     ebp, esp
.text:00401013 push    ecx
.text:00401014 mov     [ebp+var_4], ecx ; get this ptr to current object
.text:00401017 mov     ecx, [ebp+var_4] ;
.text:0040101A call    sub_401000 ; call class A constructor
.text:0040101F mov     eax, [ebp+var_4]
.text:00401022 mov     esp, ebp
.text:00401024 pop    ebp
.text:00401025 retn
.text:00401025 sub_401010 endp
```

## Manual Approach > Identifying Relationship > Constructor Analysis

### ■ Multiple Inheritance

```
.text:00401020 sub_401020      proc near
.text:00401020
.text:00401020 var_4          = dword ptr -4
.text:00401020
.text:00401020                 push    ebp
.text:00401021                 mov     ebp, esp
.text:00401023                 push    ecx
.text:00401024                 mov     [ebp+var_4], ecx
.text:00401027                 mov     ecx, [ebp+var_4] ; ptr to base class A
.text:0040102A                 call    sub_401000 ; call class A constructor
.text:0040102A
.text:0040102F                 mov     ecx, [ebp+var_4]
.text:00401032                 add     ecx, 4 ; ptr to base class C
.text:00401035                 call    sub_401010 ; call class C constructor
.text:00401035
.text:0040103A                 mov     eax, [ebp+var_4]
.text:0040103D                 mov     esp, ebp
.text:0040103F                 pop    ebp
.text:00401040                 retn
.text:00401040
.text:00401040 sub_401020      endp
```

## Manual Approach > Identifying Relationship

### ■ Multiple Inheritance

```
class A size(4) :  
    +---  
0     | a1  
    +---  
class C size(4) :  
    +---  
0     | c1  
    +---  
class D size(12) :  
    +---  
        | +--- (base class A)  
0     | | a1  
        | +---  
        | +--- (base class C)  
4     | | c1  
        | +---  
8     | d1  
    +---
```

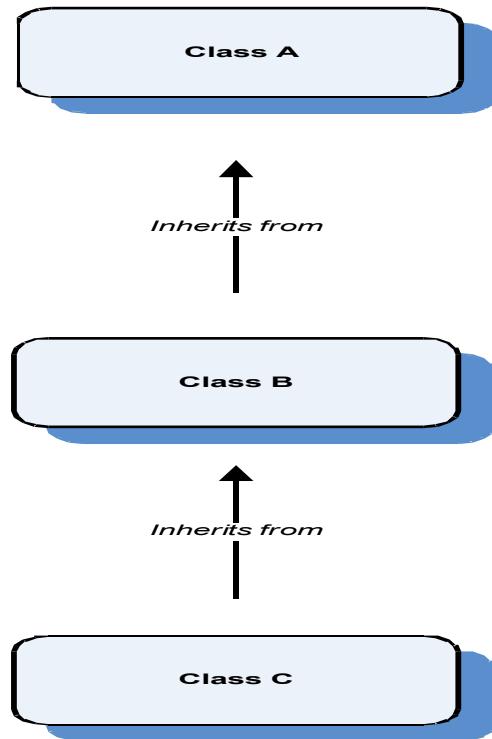
## Manual Approach > Identifying Relationship > via RTTI

- **Using RTTIClassHierarchyDescriptor**
- **Contain pointers to RTTIBaseClassDescriptors (BCDs)**

Offset	Type	Name	Description
0x00	DW	signature	Always 0?
0x04	DW	attributes	Bit 0 - multiple inheritance Bit 1 - virtual inheritance
0x08	DW	numBaseClasses	Number of base classes. Count includes the class itself
0x0C	DW	pBaseClassArray	Array of RTTIBaseClassDescriptor

## Manual Approach > Identifying Relationship > via RTTI

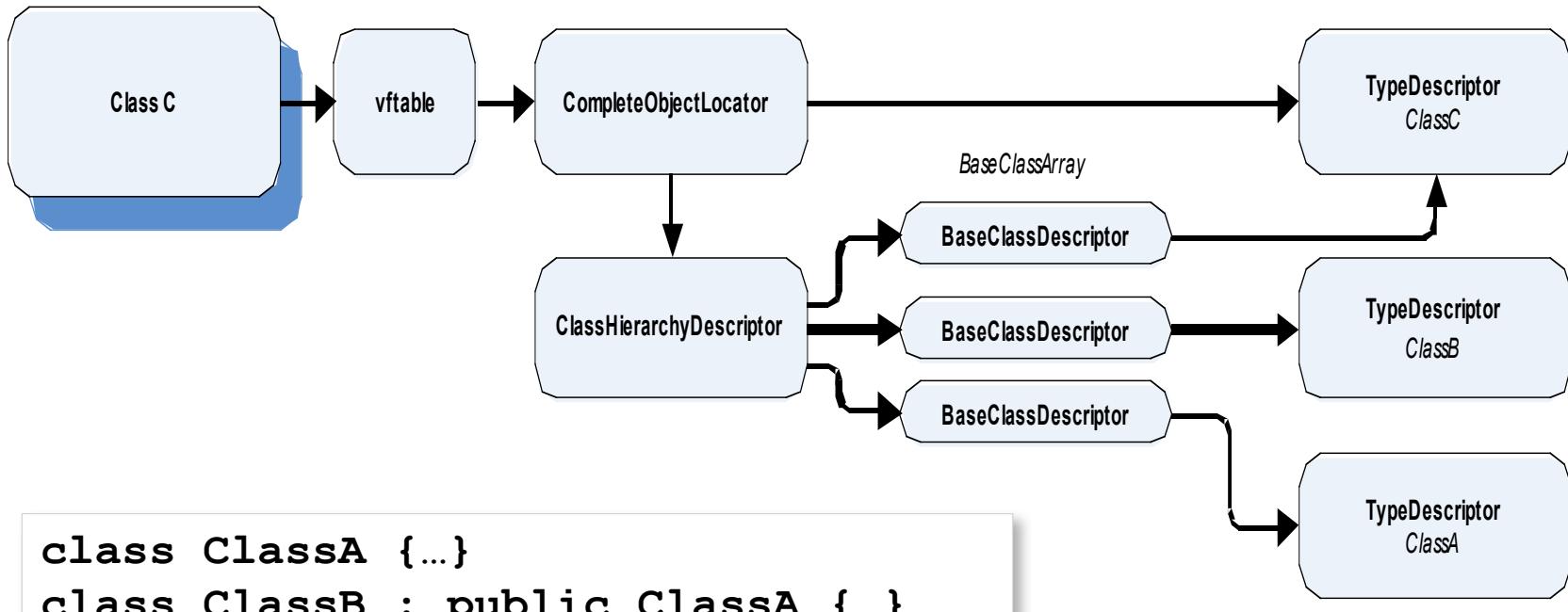
### ■ Example: C inherits B inherits A



```
class ClassA {...}  
class ClassB : public ClassA {...}  
class ClassC : public ClassB {...}
```

## Manual Approach > Identifying Relationship > via RTTI

### Example: C inherits B inherits A





**Reversing C++**

## Part II. Manual Approach Identifying Class Members



## Manual Approach > Identifying Class Members

### ■ Class Member Variable

```
.text:00401003          push    ecx  
.text:00401004          mov     [ebp+var_4], ecx  
.text:00401007          mov     eax, [ebp+var_4]  
.text:0040100A          mov     dword ptr [eax + 8], 12345h
```

## Manual Approach > Identifying Class Members

### ■ Virtual Functions

```
.text:00401C21          mov     ecx, [ebp+var_1C] ; ecx = this pointer
.text:00401C24          mov     edx, [ecx] ; edx = ptr to vftable
.text:00401C26          mov     ecx, [ebp+var_1C]
.text:00401C29          mov     eax, [edx+4]
.text:00401C2C          call    eax ; call virtual function
```

## Manual Approach > Identifying Class Members

### ■ Non-virtual member functions

```
.text:00401AFC
.text:00401B01
.text:00401B04
```

push	0CCh
lea	ecx, [ebp+var_C] ; ecx = this pointer
call	sub_401110



push	ebp
mov	ebp, esp
push	ecx
mov	[ebp+var_4], ecx ; ecx

```
.text:00401110
.text:00401111
.text:00401113
.text:00401114
```

used



Reversing C++

# Part III. Automation



## Automation > OOP\_RE

- Developed in Python
- Uses the IDAPython platform
- Identifies Classes, Relationships and Members
- Using Static Analysis

The screenshot shows a terminal window with two main sections. The top section displays assembly code from a debugger, specifically the .text section, with several pushes and a call instruction to sub\_4022C0. The bottom section shows the output of the OOP\_RE tool, which is version 0.8.0 beta. It logs the analysis process, starting with loading and reconstructing RTTI data, searching for vftables, retrieving virtual functions, and finally completing the analysis.

```
.text:004010C9          push    offset _mainClassname, [main] Class *d2
.text:004010D4          push    edx
.text:004010D9          call    sub_4022C0

-----
IDAPython version 0.8.0 beta (serial 0) initialized
Python interpreter version 2.4.2 final (serial 0)

OOP_RE v.1.0
[*] oop_re: analysis started.
[*] oop_re: loading and reconstructing rtti data...
[*] oop_re: warning: no rtti data found :(
[*] oop_re: searching for possible vftables (via heuristics)...
[*] oop_re: retrieving corresponding virtual functions of vftables...
[*] oop_re: searching constructors via new()...
[*] oop_re: updating ctors list and creating new classes using data found in ctor search...
[*] oop_re: analyzing remaining ctors/dtors (via heuristics)...
[*] oop_re: analyzing constructors to search for base classes...
[*] oop_re: analysis completed.
```

## Automation > Why a Static Approach?

- **Difficult to perform runtime analysis on some platforms (Symbian)**
- **Of course, a hybrid approach may produce more exact results**



Reversing C++

## Part III. Automation Automated Analysis Strategies



## Automation &gt; Strategies &gt; 1. Polymorphic Class Identification via RTTI

**■ Leverage RTTI data to accurately extract:**

- Polymorphic Classes
- Polymorphic class Name
- Polymorphic class Hierarchy
- Polymorphic class Virtual Function Table and Virtual Functions
- Polymorphic class Destructors/Constructors

## Automation &gt; Strategies &gt; 1. Polymorphic Class Identification via RTTI

## ■ Searching RTTI-related structures

- Via virtual function table (vftable) searching:
  - ─ If item is DWORD
  - ─ If item is a pointer to a Code
  - ─ If item is being referenced by a Code and the instruction in this referencing code is a `mov` instruction (vftable assignment)
- `RTTIClassBCompleteObjectLocator` is just below a vftable

```
.rdata:004165B0          dd offset ClassB_RTTIClassBCompleteObjectLocator@00
.rdata:004165B4 ClassB_vftable
.rdata:004165B4          dd offset sub_401410 ; DATA XREF:...
.rdata:004165B8          dd offset sub_401460
.rdata:004165BC          dd offset sub_401230
```

## Automation &gt; Strategies &gt; 1. Polymorphic Class Identification via RTTI

## ■ Verifying RTTICCompleteObjectLocator

- Verify if RTTICCompleteObjectLocator points to a valid TypeDescriptor
- TypeDescriptor is valid if TypeDescriptor.name starts with “.?AV”

```
.rdata:00418A28 ClassB_RTTICCompleteObjectLocator@00
.rdata:00418A28          dd 0                  ; signature
.rdata:00418A2C          dd 0                  ; offset
.rdata:00418A30          dd 0                  ; cdOffset
.rdata:00418A34          dd offset ClassB_TypeDescriptor
.rdata:00418A38          dd offset ClassB_RTTIClassHierarchyDescriptor
```

```
.data:0041B01C ClassB_TypeDescriptor
                  dd offset type_info_vftable
.data:0041B020          dd 0                  ; spare
.data:0041B024 a_avclassb@@      db '.?AVClassB@@',0    ; name
```

## Automation &gt; Strategies &gt; 1. Polymorphic Class Identification via RTTI

## ■ Class Information from RTTI (Summary)

```
new_class()
    - Identified from TypeDescriptors
new_class.class_name
    - Identified from TypeDescriptor.name
new_class.vftable/vfuncs
    - Identified from vftable-RTTICompleteObjectLocator
      relationship
new_class.ctors_dtors
    - Identified from functions referencing the vftable
new_class.base_classes
    - Identified from
RTTICompleteObjectLocator.pClassHierarchyDescriptor
```

## Automation &gt; Strategies &gt; 2. Polymorphic Class Identification (w/o RTTI)

## ■ Polymorphic Classes Identification (w/o RTTI)

- Via vtable searching (*previously discussed*)
- Base classes are not yet identified
- Class name will be automatically generated

```
new_class()
    - Identified from vtable
new_class.class_name
    - Auto-generated (based from vtable address, etc.)
new_class.vftable/vfuncs
    - Identified from vtable
new_class.ctors_dtors
    - Identified from functions referencing the vtable
```

## ■ Simple Data Flow Analyzer Algo

1. If the variable/register is overwritten, stop tracking
2. If EAX is being tracked and a call is encountered, stop tracking. (We assume that all calls return values in EAX).
3. If a call is encountered, treat the next instruction as a new block
4. If a conditional jump is encountered, follow the register/variable in both branches, starting a new block on each branch.
5. If the register/variable was copied into another variable, start a new block and track both the old variable and the new one starting on this block.
6. Otherwise, track next instruction.

Automation > Strategies > 3. Class Identification via Constructor / Destructor Search

## ■ Constructor Identification

- For dynamically allocated objects
  1. Look for calls to new() .
  2. Track the value returned in EAX
  3. When tracking is done, look for the earliest call where the tracked register/variable is ECX. Mark this function as constructor.
- For local objects
  - For local objects, we do the same thing. Instead of initially tracking returned values of new(), we first locate instructions where an address of a stack variable is written to ECX, then start tracking ECX

## ■ Inheritance Identification

1. Track this pointer (ECX)
2. Check blocks with ECX as tracked variable
3. See if there is call to a constructor
4. To handle multiple inheritance, track pointers to offsets relative to object address

Automation > Strategies > 5. Class Member Identification

## ■ **Member Variables**

- track the `this` pointer from the point the object is initialized.
- note accesses to offsets relative to the *this* pointer.

Automation > Strategies > 5. Class Member Identification

## ■ **Non-virtual Functions**

- track the this pointer from the point the object is initialized.
- note all blocks where ECX is the tracked variable, then mark the call in that block, if there is any, as a member of the current class.

Automation > Strategies > 5. Class Member Identification

## ■ Virtual Functions

- To identify virtual functions, we simply have to locate vftables first through constructor analysis.

**After all of this is done, we then reconstruct the class using the results of these analysis.**



Reversing C++

# Part III. Automation Enhancing Disassembly



## Automation &gt; Disassembly Enhancement

- **RTTI structures reconstruction, naming, commenting**

```
<Original>
.rdata:004165A0          dd offset unk_4189E0
.rdata:004165A4 off_4165A4
                        dd offset sub_401170      ; DATA XREF:...
.rdata:004165A8          dd offset sub_4011C0
.rdata:004165AC          dd offset sub_401230
.rdata:004165B0          dd offset unk_418A28
```

```
<Processed>
.rdata:004165A0  dd offset oop_re$ClassA$RTTICompleteObjectLocator@00
.rdata:004165A4 oop_re$ClassA$vftable@00
                  dd offset sub_401170 ; DATA XREF: ...
.rdata:004165A8  dd offset sub_4011C0
.rdata:004165AC  dd offset sub_401230
```

## Automation &gt; Disassembly Enhancement

- RTTI structures (another example)

```
<Original>
.rdata:004189E0 dword_4189E0      dd 0          ; DATA XREF:...
.rdata:004189E4                  dd 0
.rdata:004189E8                  dd 0
.rdata:004189EC                  dd offset off_41B004
.rdata:004189F0                  dd offset unk_4189F4
```

```
<Processed>
.rdata:004189E0 oop_re$ClassA$RTTICCompleteObjectLocator@00
                 dd 0          ; RTTICCompleteObjectLocator.signature
.rdata:004189E4 dd 0          ; RTTICCompleteObjectLocator.offset
.rdata:004189E8 dd 0          ; RTTICCompleteObjectLocator.cdOffset
.rdata:004189EC dd offset oop_re$ClassA$TypeDescriptor
.rdata:004189F0 dd offset oop_re$ClassA$RTTIClassHierarchyDescriptor
```

## Automation > Disassembly Enhancement

### ■ Improving the call graph

- Add cross references on virtual function calls
- Result in more accurate call graph
- Will yield improvements on binary diffing results



**Reversing C++**

# Part III. Automation Visualization



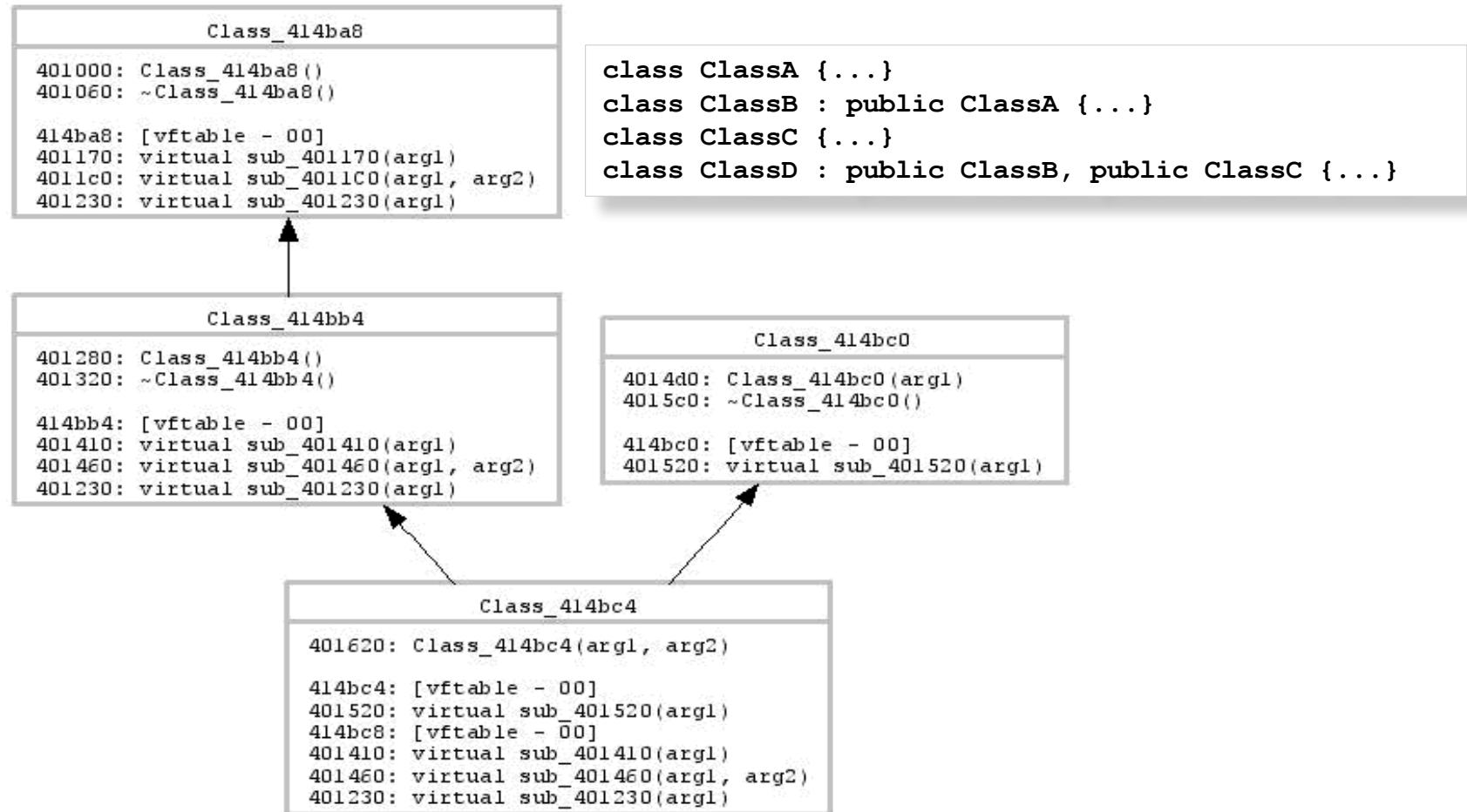
## Automation > Visualization

### ■ UML Diagram Generation

- Using `pydot`
- Create a node for each class
- Create an edge from each base classes
- Pretty simple (once you have the data :) and Cool too... ☺
- Very effective if RTTI exists (class names)
- EXE2UML ?

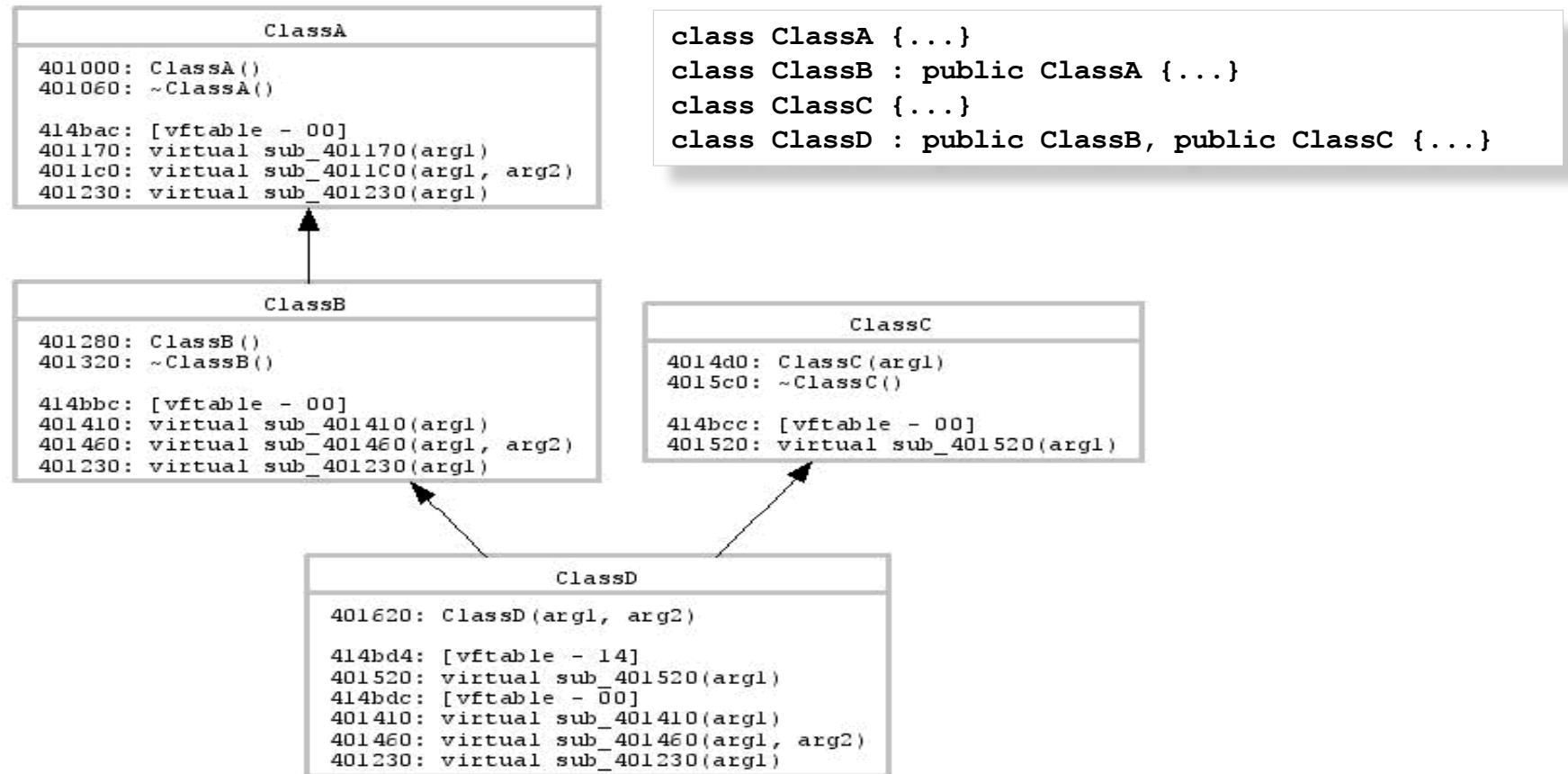
## Automation &gt; Visualization

## ■ UML Diagram Example (w/o RTTI)



## Automation &gt; Visualization

## ■ UML Diagram Example (w/ RTTI)





Reversing C++

Demo...





IBM Global Services

# Thank you!

Questions?

**Paul Vincent Sabanal**  
**X-Force R&D**  
**Mark Vincent Yason**  
**X-Force R&D**



IBM Internet Security Systems  
Ahead of the threat.™

© Copyright IBM Corporation 2007