33 – C++ Analysis

OOP
Sample (Simple) Class
SimpleClass Properties
Code for SimpleClass
Code for HelloWorld
Overloading and Name Mangling
Inheritance and Overloading
Virtual Functions
Socket Classes with Virtual Functions
Vtable Implementation
Object Creation and Deletion
OOP

• Malware written in C++ contains constructs that make analysis difficult if they are not understood.

• C++ uses classes to represent families of objects. Classes are implemented as structs that refer to function pointers.

• All instance objects belonging (directly) to the same class share the same methods (behavior) but have their own data structure data (attributes).
Sample (Simple) Class

class SimpleClass()
{
    public:
        int x;
    void HelloWorld()
    {
        if (x==10) printf("X is 10.\n");
    }
};
int _tmain(int argc, _TCHAR* argv[])
{
    SimpleClass myObject;
    myObject.x = 9;
    myObject.HelloWorld();
    SimpleClass myOtherObject;
    myOtherObject.x = 10;
    myOtherObject.HelloWorld();
}
SimpleClass Properties

• Instances of SimpleClass contain a single int data member, \( x \).

• The class contains a single (non-virtual) method HelloWorld. This means that every object referred to by a variable or reference of type SimpleClass implements the same behavior for that function.

• As a result, the address of HelloWorld can be allocated statically in programs using SimpleClass.
Code for SimpleClass

00401100  push ebp
00401101  mov  ebp,esp
00401103  sub  esp,1F0h
00401109  mov  [ebp+var_10], offset off_404768
00401110  mov  [ebp+var_C], 9
00401117  lea  ecx, [ebp+var_10]
0040111A  call sub_4115D0
0040111F  mov  [ebp+var_34], offset off_404768
00401126  mov  [ebp+var_30], 0Ah
0040112D  lea  ecx, [ebp+var_34]
00401130  call sub_4115D0
Code for HelloWorld

004115D0  push ebp
004115D1  mov  ebp,esp
004115D3  sub  esp,9Ch
004115D9  push ebx
004115DA  push esi
004115DB  push edi
004115DC  mov  [ebp+var_4], ecx
004115DF  mov  eax,[ebp+var_4]
004115E2  cmp  dword ptr [eax+4], 0Ah
004115E6  jnz  short loc_4115F6
004115E8  push offset aXIs10_ : “X is 10.\n”
00411ED  call ds::__imp__printf
Overloading and Name Mangling

- A class in C++ can contain multiple methods having the same name accepting different parameters.
- C++ uses *name mangling* to keep track of these differing methods.
- Example: method

  ```cpp
  void SimpleClass::TestFunction(int, int)
  ```

  has internal name

  ```
  ?TestFunction@SimpleClass@@QAEXZZ@Z
  ```

  to distinguish it from a methods with a different argument list. The string `@@QAEXZZ@Z` identifies the argument and return value types.
- Mangling choices are compiler-specific. IDA Pro can demangle names for most compilers.
Inheritance and Overloading

class Socket {
public:
    void setDestinationAddr (InetAddr *addr) {
        ...
    }
    ...
};

class UDPSocket : public Socket {
public:
    void sendData (char *buf, InetAddr *addr) {
        setDestinationAddr(addr);
    }
    ...
};
Inheritance and Overloading

• The class `Socket` does not contain a `sendData` method, but it does have a method for setting the socket destination address.

• Subclass `UDPSocket` of `Socket` implements method `sendData` on objects belonging to its class which uses the underlying `setDestinationAddress` method defined in its base class.

• Since these methods are still non-virtual, their addresses can be known at compile time.
Virtual Functions

• Virtual functions override the behavior of base class methods when applied to derived class object references or pointers.

• This usually supports cases where a single base class supports multiple derived classes, each of which implements the same method differently.

• Example: revised Socket Class
Socket Classes with Virtual Functions

class Socket {
public:
    virtual void sendData (char *buf, INetAddr *addr) {
        // raise error message
    }
    ...
};
class UDPSocket : public Socket {
public:
    virtual void sendData (char *buf, INetAddr *addr) {
        ...
    }
    ...
};
class TCPSocket : public Socket {
public:
    virtual void sendData (char *buf, INetAddr *addr) {
        ...
    }
    ...
};
Vtable Implementation

• The addresses of virtual functions associated with an class are contained in a virtual table (or vtable).

• Vtables can be recognized in code in a way similar to jump tables. Instead of an array of jump addresses, a vtable contains an array of function pointers.

• Virtual functions are not called directly by other code, therefore they will have no cross-references that call them.
Object Creation and Deletion

- C++ programs use new and delete to create and destroy objects.
- Focusing on calls to new shows where objects are dynamically created.
- IDA Pro generally will correctly identify calls to allocation/deallocation correctly.
Next Time

- Exercises, exercises, exercises.
- Quiz questions from previous quizzes.