Static detection of C++ vtable escape vulnerabilities in binary code

By Huseyin Ulusoy, 4/25/2012
Outline

- Introduction
- Problem Definition
- Object Reaching Definition Analysis
- Implementation and Results
- Conclusion
- References
Introduction

- Static binary code analysis is widely used to find security defects.
- Vulnerabilities in binaries compiled from object-oriented languages remain undetected.
- Dynamic dispatch makes the static analysis harder.
- C++ permits type confusion errors via its static_cast operator.
- Many widely deployed softwares have vulnerabilities introduced by object-oriented programming languages.
Problem Definition

```cpp
class class1 {
    public:
        class1();
        ~class1();
        virtual void addRef();
        virtual void print();
    };

class class2 : public class1 {
    public:
        class2();
        ~class2();
        virtual void voidFunc1() {};
        virtual void debug() {};
    };

int _tmain(int argc, _TCHAR* argv[])
{
    class1 Cl1;
    Cl1.addRef();
    Cl1.print();
    static_cast<class2*>(&Cl1)->debug();
    return 0;
}
```

Source code

Compiled binary
Problem Definition

class class1 {
public:
  class1();
  ~class1();
  virtual void addRef();
  virtual void print();
};

class class2 : public class1 {
public:
  class2();
  ~class2();
  virtual void voidFunc1(); {}
  virtual void debug();
};

int _tmain(int argc, _TCHAR* argv[])
{
  class1 Cl;
  Cl.addRef();
  Cl.print();
  static_cast<class2*>(&Cl)->debug();
  return 0;
}

Source code

Vtable
Problem Definition

The binary code compiled with C++ may contain:
- Lots of indirect calls due to virtual functions
- Vtable to resolve virtual methods in run time
- Type confusion errors because of static_cast operator in C++
Vtable Escape Vulnerability

- Vtable Escape Vulnerability has impacted many widely used softwares

Patch dates:
- Microsoft Excel in March 2010
- Adobe Flash Player in April 2011
- Microsoft ATL in July 2009
Object Reaching Definition Analysis

“Object Reaching Definition Analysis” is created for data-flow analysis with three goals:

1. Resolve the indirect virtual function calls present in binary code due to virtual function dispatch
2. Statistically construct the program’s call graph to improve existing automated analyses
3. Identify object vtable congruence failures
Object Reaching Definition Analysis

Object Reaching Definition Analysis is composed of five parts:

- Object identification
- Constructor analysis
- Reaching definition analysis
- Virtual function resolution
- Object congruence
Object identification

Since binary file does not contain obvious identifiers of object instantiation, four heuristic functions are proposed:

1. Stack declaration + inline constructor
2. Heap declaration + inline constructor
3. Stack declaration + not inline constructor
4. Heap declaration + not inline constructor
Constructor analysis

Three tests are used to determine the length of vtable

1. If a data element is pointed to by another program point, it is likely the start of some other data structure
2. If a pointer in the table does not point to a function, it is likely the next element past the end of table
3. Zero padding (compilers often pad the end of a data structure with zeros.)
Reaching definition analysis

A fixed-point interprocedural reaching definition analysis is performed each object definition encountered during previous step to analyze object flow to indirect function call

\[
REACH_{IN}[S] = \bigcup_{p \in \text{pred}[S]} REACH_{OUT}[p] \quad (1)
\]

\[
REACH_{OUT}[S] =
GEN[S] \cup (REACH_{IN}[S] - KILL[S]) \quad (2)
\]
Virtual function resolution

- Single Object definition
- Single Object definition with null value
- Decidable number of object definitions
- Undecidable number of object definitions
Object congruence

- Congruence objects are made up of the same number of methods and properties.
- If more than one object definition reaches a given use, object congruence becomes important.
- Objects are marked as safe or unsafe for congruence with vtable and object definition information.
- If one of the possible reaching objects is unsafe, the whole program is marked as unsafe.
Authors created a framework called RECALL to implement and test Object Definition Reaching Analysis.

RECALL uses LLVM and IDA pro.
Comparison with HexRays

- HexRays is unable to resolve virtual function calls.
- HexRays identifies the first call to first element in the vtable, but all subsequent calls are calls to base pointer plus an offset.
- RECALL is able to fully resolve all of the virtual functions.

```c
int __cdecl wmain()
{
    int v0; // ed182
    int v1; // es185

    if ( operator new(8u) )
        v0 = sub_401070();
    else
        v0 = 0;
    if ( operator new(8u) )
        v1 = sub_4010A0();
    else
        v1 = 0;
    ++(void __thiscall **)(__DWORD)v0(v0);
    *(void __thiscall **)((__DWORD *)v0 + 4)(v0);
    ++(void __thiscall **)(__DWORD)v1(v1);
    *(void __thiscall **)((__DWORD *)v1 + 4)(v1);
    ++(void __thiscall **)(__DWORD)v1 + 12)(v1);
    return 0;
}
```

HexRays output
Conclusion

- Development of C++ caused the introduction of new vulnerability types
- These vulnerabilities are beyond of existing data flow analysis
- The presented data flow analysis can help remediate these new vulnerabilities
References