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

David Dewey Jonathon Giffin
School of Computer Science
Georgia Institute of Technology
{ddewey, giffin}@gatech.edu

Common problem in C++

In C++ specifically, how does one convert and instance of an object into an instance of another object?

> "…use `static_cast` in all cases and see what the compiler says."
>
> B. Stroustrup. The Design and Evolution of C++. Pearson Education, 1994.

C++ Type confusion vulnerabilities

Adobe Flash Player SharedObject Type Confusion Vulnerability

CVE-2011-0611

Microsoft ATL/MFC ActiveX Type Confusion Vulnerability

CVE-2009-2494

Microsoft Office Excel Conditional Expression Ptg Type Confusion Vulnerability

CVE-2011-1989

The list goes on… and on… and on…

Reverse engineering C++ binaries is hard

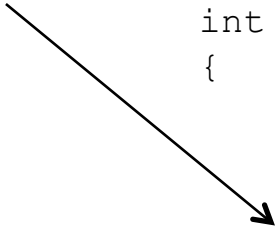
```
; END OF FUNCTION CHUNK FOR ?_SendRequestWithDrainComplete@HTTP_USER_REQUEST@@AAEKPAVPENDING_
; ---------------------------------------------------------------------------
; START OF FUNCTION CHUNK FOR ?SendRequest@HTTP_USER_REQUEST@@QAEKQAEK_KHP6GXPAXKK2K@ZKKH@Z

loc_EAFB93D:                            ; CODE XREF: HTTP_USER_REQUEST::SendRequest(uchar * (
                mov     eax, [ecx]
                call    dword ptr [eax+4]
                mov     [esi+14h], edi
                jmp     loc_EAE7DFD
; END OF FUNCTION CHUNK FOR ?SendRequest@HTTP_USER_REQUEST@@QAEKQAEK_KHP6GXPAXKK2K@ZKKH@Z
; ---------------------------------------------------------------------------
; START OF FUNCTION CHUNK FOR ?OnIoComplete@WEBIO_REQUEST@@UAEXKKPAVHTTP_ASYNC_OVERLAPPED@@@Z

loc_EAFB94A:                            ; CODE XREF: WEBIO_REQUEST::OnIoComplete(ulong,ulong
                mov     ecx, [ebp+arg_8]
                mov     eax, [ecx]
                inc     ebx
                call    dword ptr [eax]
                jmp     loc_EAEA333
; ---------------------------------------------------------------------------

loc_EAFB957:                            ; CODE XREF: WEBIO_REQUEST::OnIoComplete(ulong,ulong
                                        ; WEBIO_REQUEST::OnIoComplete(ulong,ulong,HTTP_ASYNC_
                mov     esi, [esi+20h]
                mov     eax, [esi]
                push    edi
                mov     ecx, esi
                call    dword ptr [eax+10h]
                jmp     loc_EAFB73C
; ---------------------------------------------------------------------------
```

As it turns out, these are all the same problem…

- Recently, many software-level vulnerabilities caused by C++ type confusion
- Compiled C++ code can be very difficult to analyze
  - IDS/IPS vendor wanted to provide signature coverage
  - Software consumer concerned with application security
  - Third-party interoperation
- Software developers regularly incorrectly use the `static_cast` operator
  - No compiler warning from most modern compilers
  - C++ standard only requires “cv-check”

Root of the problem

- This code compiles without warning with Visual Studio and g++ (< 4.6)
- Running this code causes a call to arbitrary memory



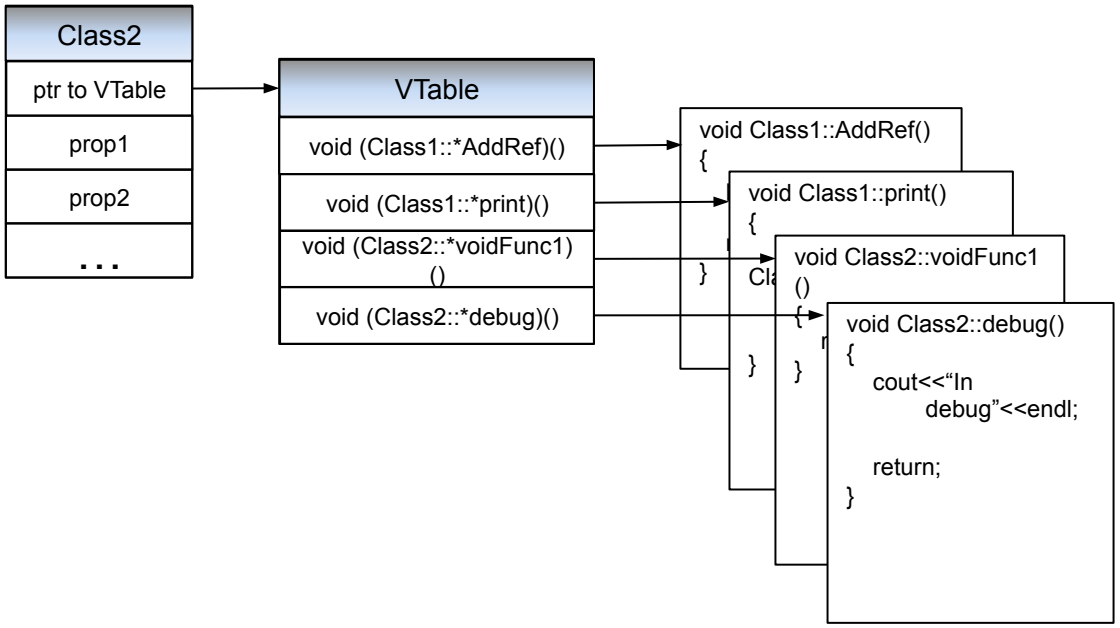
```
class class1 {
public:
     class1();
     ~class1();
     virtual void addRef();
     virtual void print();
};

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

int tmain(int argc, TCHAR* argv[])
{
     class1 C1;
     C1.addRef();
     C1.print();
     static cast<class2*>(&C1)->debug();
     return 0;
}
```

Same problem

- In the previous slide, the problem should be obvious to a developer
- Consider this code. `_tmain()` and `internalFunction()` may be “miles apart”
  - Separate libraries
    - Not caught by g++ 4.6
- Very common code construct in MS COM

```
int internalFunction(void *pv)
{
  static_cast<class1*>(pv)->addRef();
  static_cast<class1*>(pv)->print();
  static_cast<class1*>(pv)->debug();
  return 0;
}

int _tmain(int argc, _TCHAR* argv[])
{
  class1 *C1 = new class1;
  class2 *C2 = new class2;
  internalFunction((void *)C1);
  internalFunction((void *)C2);
  return 0;
}
```


Structure of a C++ object after compilation

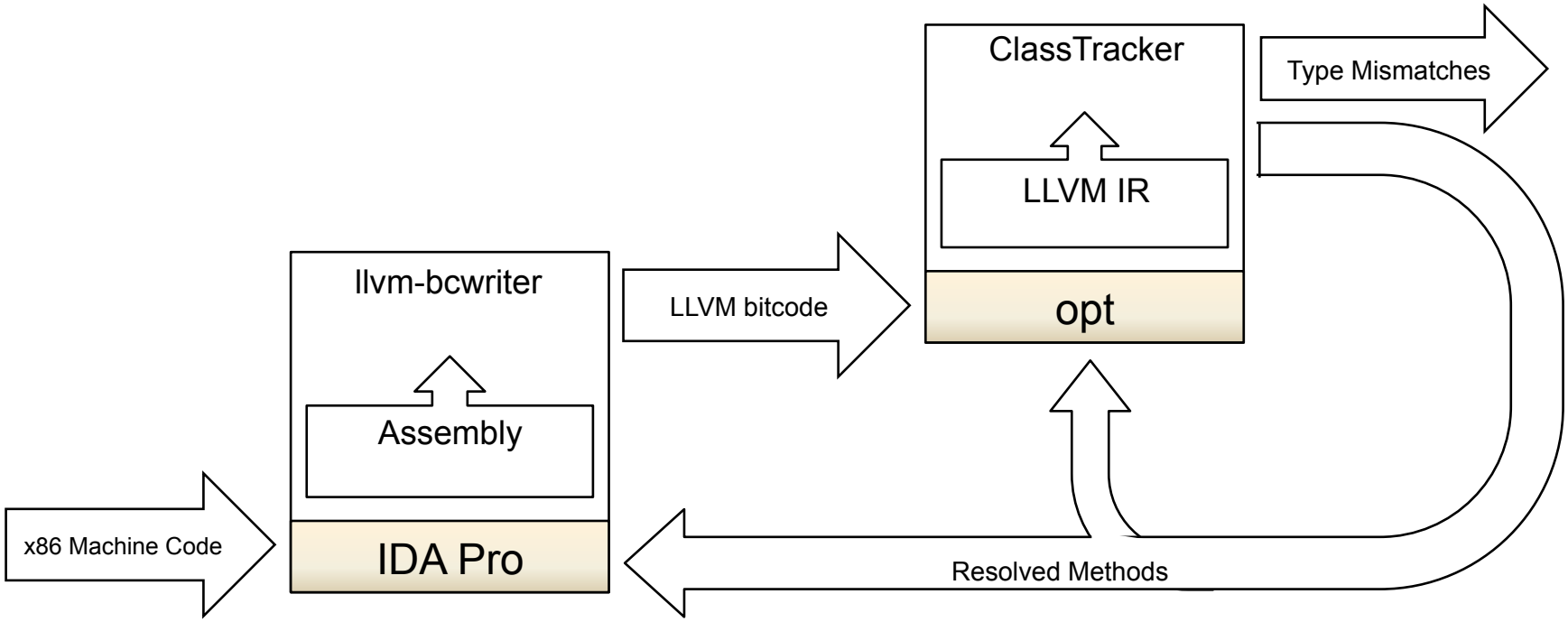



```
00402138 off 402138 dd offset sub 4010D0
0040213C dd offset sub 4010A0
00402140 dd offset nullsub 1
00402144 dd offset sub 4010B0
00402148 dd offset dword 402274
0040214C off 40214C dd offset sub 4010D0
00402150 dd offset sub 4010A0
00402154 align 8
00402158 db 48h ; H
00402159 db 0
0040215A db 0
0040215B db 0
0040215C db 0
```

RECALL

- Reconstruct C++ objects from binary code
- Perform reaching definition analysis on object definitions to determine which object is being referenced at a given use point (make reverse engineering easier)
- Perform a “congruence check” to determine the safety of the use of a given object (detects vtable escape vulnerabilities)

High-level architecture of RECALL



x86 to SSA

- First, we translate x86 machine code into an SSA-based IR
- We chose an SSA-based IR to make translation simpler
  - x86 assembly is mostly triple-based
  - Use-def chains are implicit (core requirement for reaching definitions)
  - Problems with going to higher-level IR
- Chose the LLVM IR due to the robustness of the LLVM analysis framework
- LLVM is attractive from a licensing perspective

Object reaching definition analysis

$$REACH_{IN}[S] = \bigcup_{p \in pred[S]} REACH_{OUT}[p]$$

$$REACH_{OUT}[S] = GEN[S] \cup (REACH_{IN}[S] - KILL[S])$$

Where:

GEN is the set of objects that are instantiated in a given basic block

KILL is the set of objects that are deleted in a given basic block

For interprocedural analysis, $REACH_{IN}$ at the entry of a function F is equal to REACH[c] at the call to F from a call site c

Indentifying object instantiation

- Stack-allocated

  Implement object structure heuristics

  - Inline constructor
  - Explicit constructor

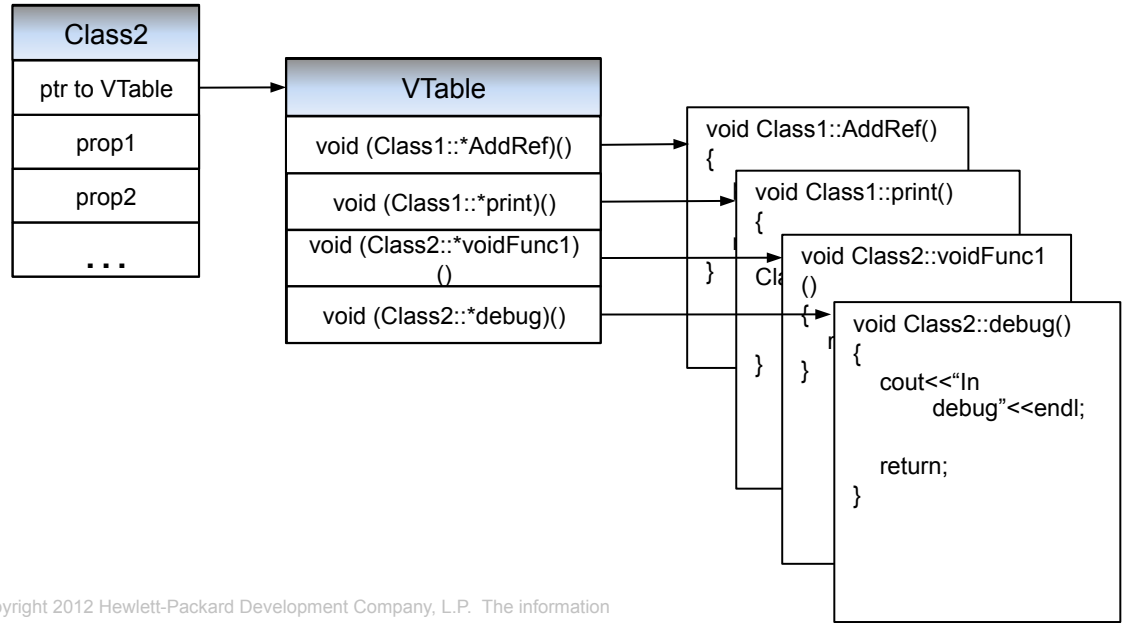
- Heap-allocated – new() operator

  Call to `YAPAXI(uint size)`

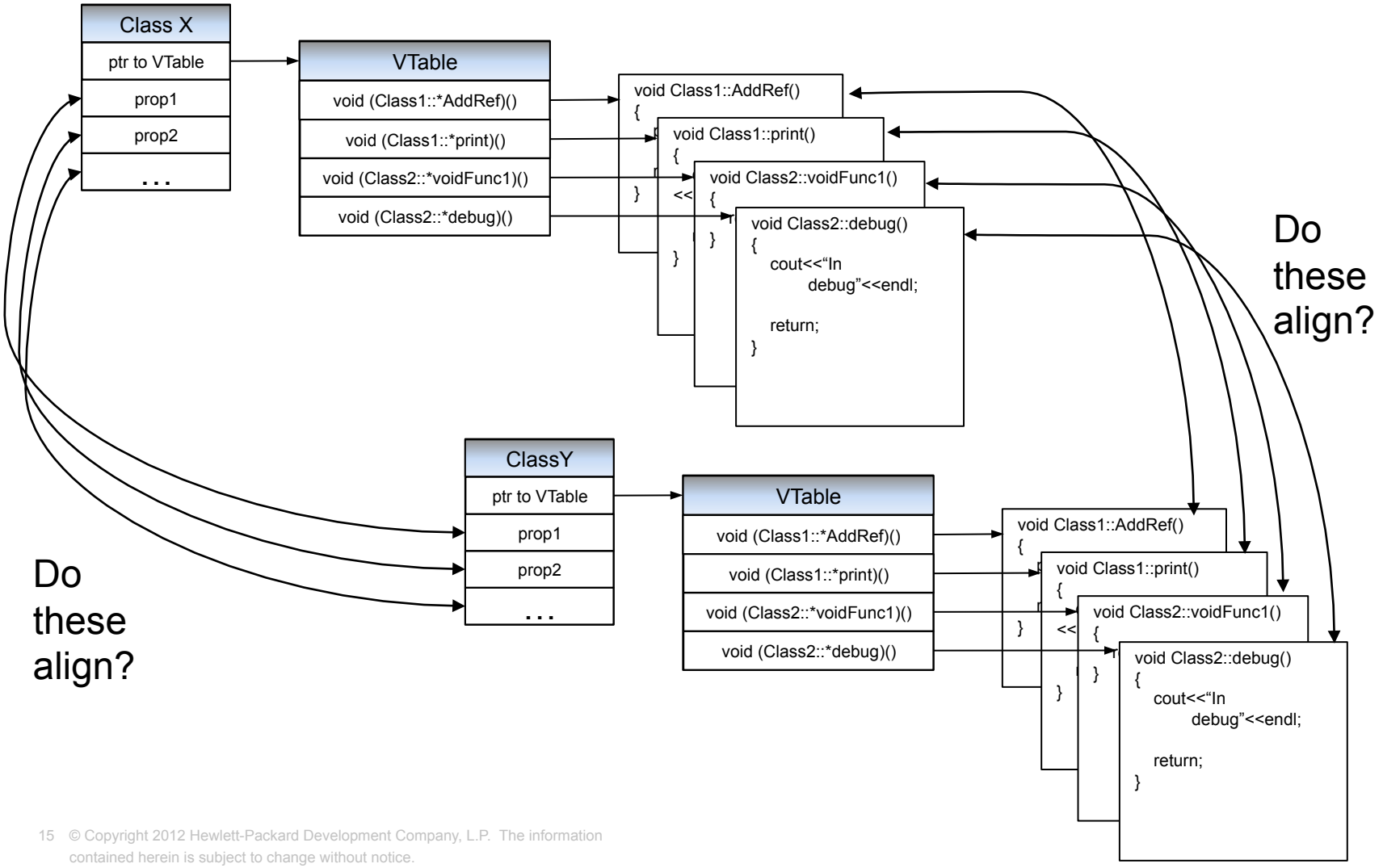
  - Inline constructor
  - Explicit constructor

Tracking object types

- For each object, create a structure mapping the structure of the object
- Tag each object type with the virtual address of the constructor



Congruence Check



Caveats

- Not designed for the analysis of malware or obfuscated code
- Does not require RTTI or debug symbols
- Focus is on code compiled with Visual Studio, but techniques can be generalized to other compilers
- If an object is allocated and the class pointer is stored in a collection, when the pointer is retrieved, we cannot track the type (future work)

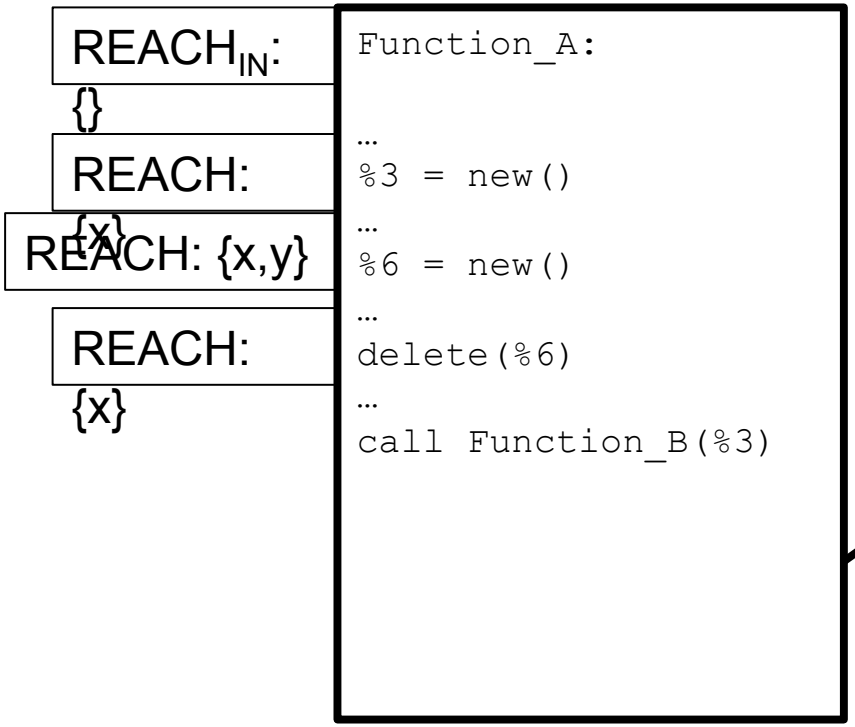
Results

- Able to reconstruct and analyze objects from sample code that models:

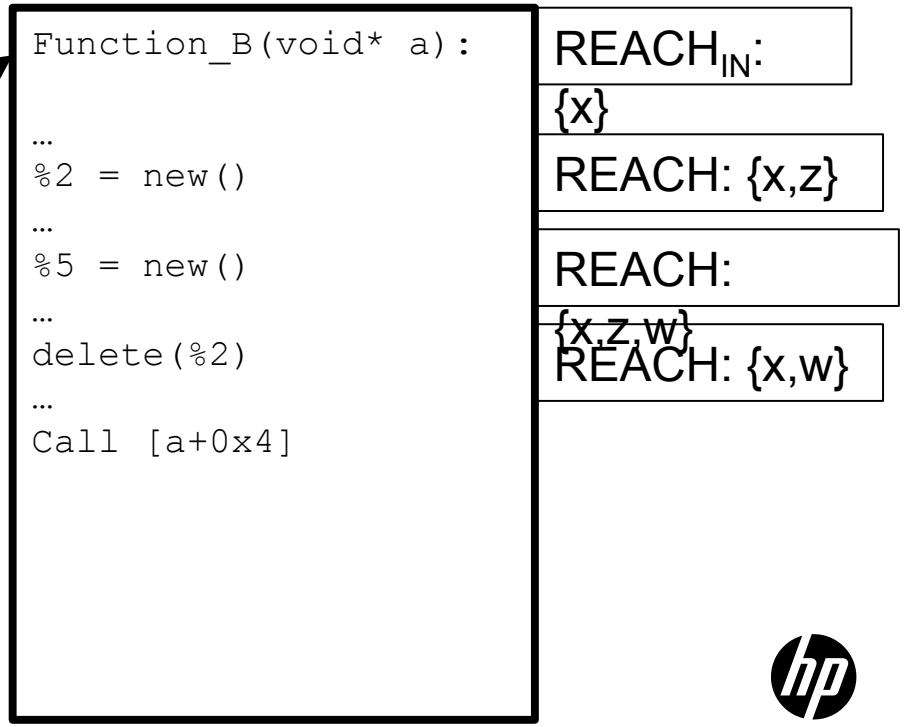
  [stack-allocated, heap-allocated] x [inlined ctor, explicit ctor]

- Able to identify vulnerabilities in microbenchmarks designed to simulate real vulnerabilities:
  - Simulated CVE-2011-0611(Adobe Reader)
  - Simulated CVE-2010-0258 (Microsoft Excel)

Why microbenchmarks?



- Analysis is performed interprocedurally
- Procedures can be analyzed independent of their location in the binary
- "Moving" procedures does not impact the correctness of the analysis



Select Related Work

D. F. Bacon and P. F. Sweeney. Fast static analysis of C++ virtual function calls. In Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), 1996.

B. Calder and D. Grunwald. Reducing indirect function call overhead in C++ programs. In Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL), Portland, Oregon, 1994.

C. Meadows. A procedure for verifying security against type confusion attacks. In IEEE Computer Security Foundations Workshop (CSFW), Pacific Grove, California, June 2003.

H. Pande and B. Ryder. Data-flow-based virtual function resolution. In Proceedings of the Third International Symposium on Static Analysis (SAS), 1996.

H. D. Pande and B. G. Ryder. Static type determination for C++. In Proceedings of the 6th USENIX C++ Technical Conference, 1994.

A. Slowinska, T. Stancescu, and H. Bos. Howard: A dynamic excavator for reverse engineering data structures. In Proceedings of the Network and Distributed Systems Security Symposium (NDSS), 2011.

D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena. BitBlaze: A new approach to computer security via binary analysis. In International Conference on Information Systems Security, 2008.

J. Viega, J. T. Bloch, Y. Kohno, and G. McGraw. ITS4: A static vulnerability scanner for C and C++ code. In Proceedings of the 16th Annual Computer Security Applications Conference (ACSAC), 2000.

Conclusion

- In our paper, we make the following contributions:
  - Resolve vtable dispatch calls in compiled binaries
  - Programmatically identify vtable escape vulnerabilities introduced by C++ developers
  - Construct a general C++ decompilation framework for use in other analyses

Questions?

ddewey@gatech.edu
giffin@gatech.edu