vfGuard:

Strict Protection for Virtual Function Calls in COTS C++ Binaries

Aravind Prakash
Xunchao Hu
Heng Yin

Department of Electrical Engineering and Computer Science, Syracuse University

L.C.Smith College of Engineering and Computer Science

Motivation

Control-Flow Hijacking

– Subvert control-flow to execute malicious code.

– Deviate from the intended flow of control.

Motivation

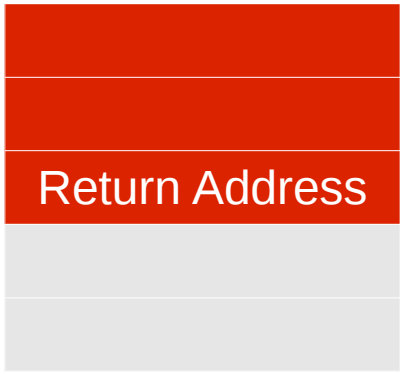
Control-Flow Hijacking
– Subvert control-flow to execute malicious code.
– Deviate from the intended flow of control.

```
void foo(char *s, char *d) {
    strcpy(d, s);
}
```

Motivation

Control-Flow Hijacking
– Subvert control-flow to execute malicious code.
– Deviate from the intended flow of control.

```
void foo(char *s, char *d) {
    strcpy(d, s);
}
```



Motivation: CFI

"The CFI security policy dictates that software execution must follow a path of a *Control-Flow Graph* determined ahead of time."

– Abadi et al., CCS'05.

Control-Flow Graph

– Binary level, Source code level, etc.

Allowable Targets(Branch) = min({Target 1, Target 2, … , Target n})

An example...

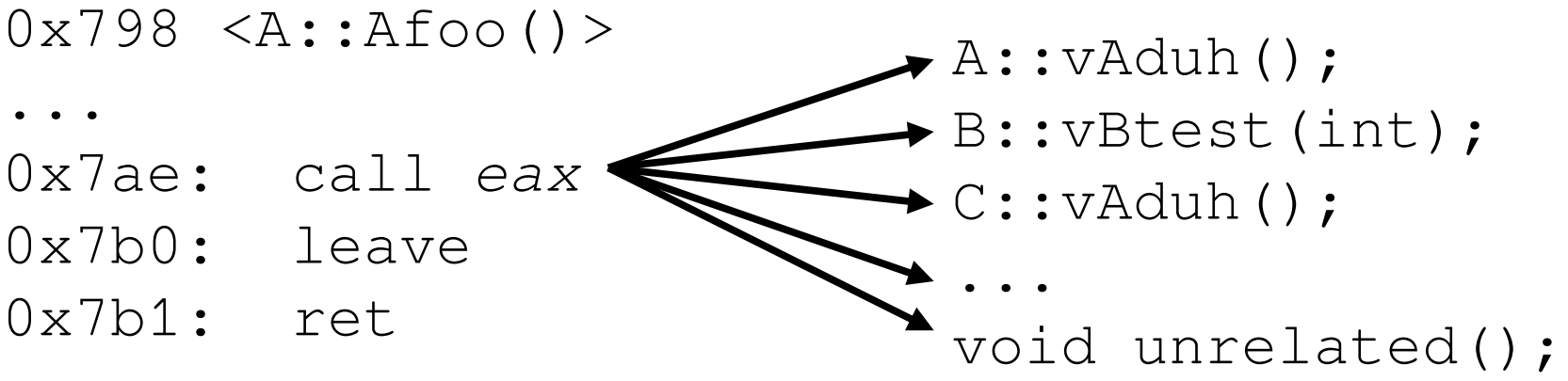
```
class A {
...
public:
virtual bool vAduh()
    {return true;}
virtual int vAtest(int
a)
    {return 0;}
virtual void Afoo()
    {this->vAduh();}
...
};
```

An example...

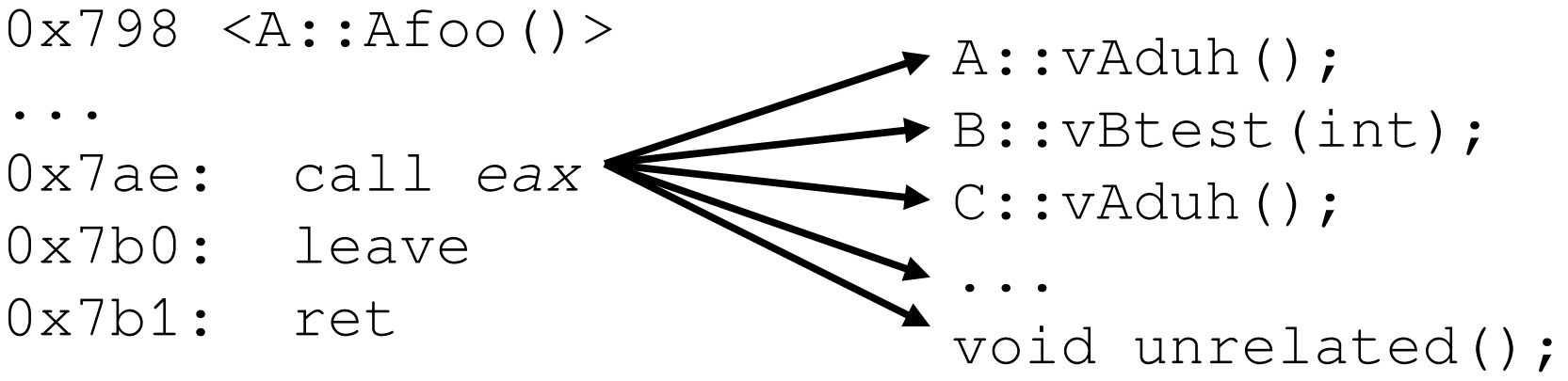
```
class A {
...
public:
virtual bool vAduh()
    {return true;}
virtual int vAtest(int
a)
    {return 0;}
virtual void Afoo()
    {this->vAduh();}
...
};
```

```
0x798 <A::Afoo()>
...
0x7ae:  call eax
0x7b0:  leave
0x7b1:  ret
```

Motivation: Attack Space



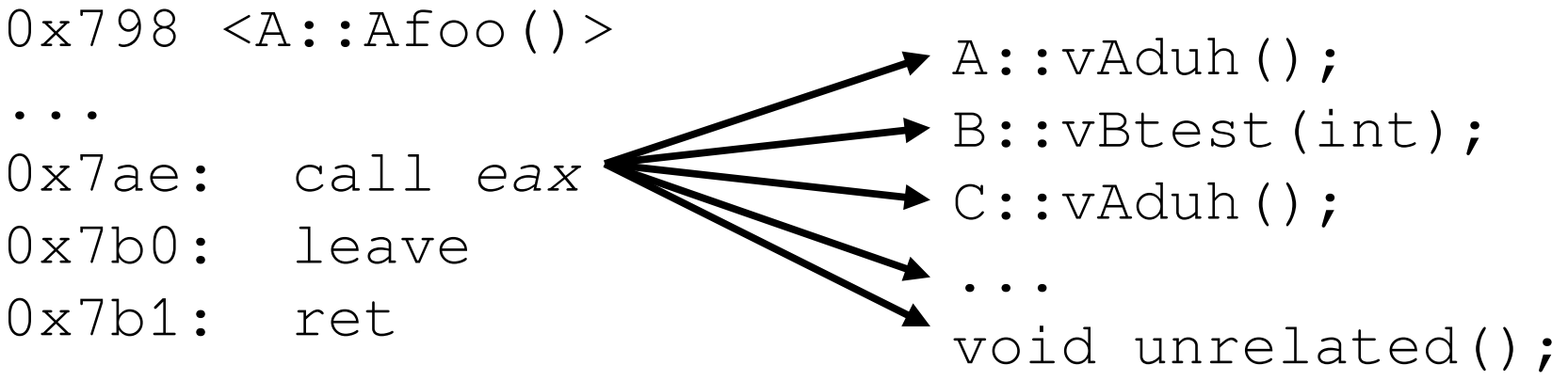
Motivation: Attack Space



CFI on the Binary...

BinCFI : [Zhang, Usenix'13], CCFIR : [Zhang, S&P'13]

Motivation: Attack Space

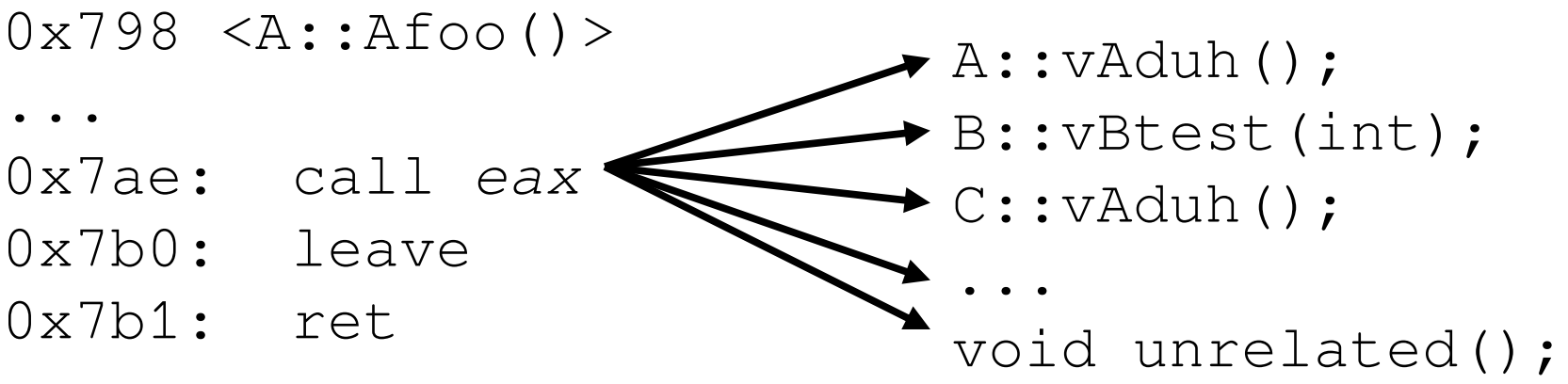


CFI on the Binary...

BinCFI : [Zhang, Usenix'13], CCFIR : [Zhang, S&P'13]

Coarse grained, Low precision

Motivation: Attack Space

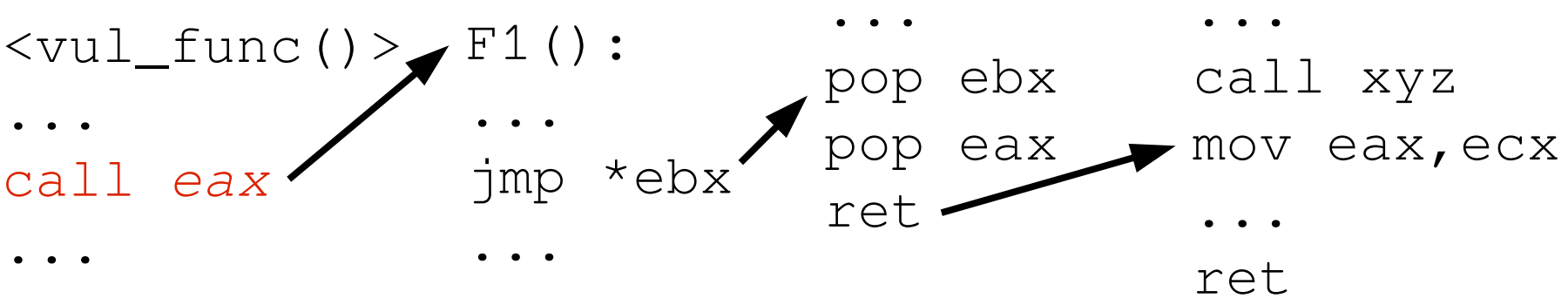


CFI on the Binary...

BinCFI : [Zhang, Usenix'13], CCFIR : [Zhang, S&P'13]

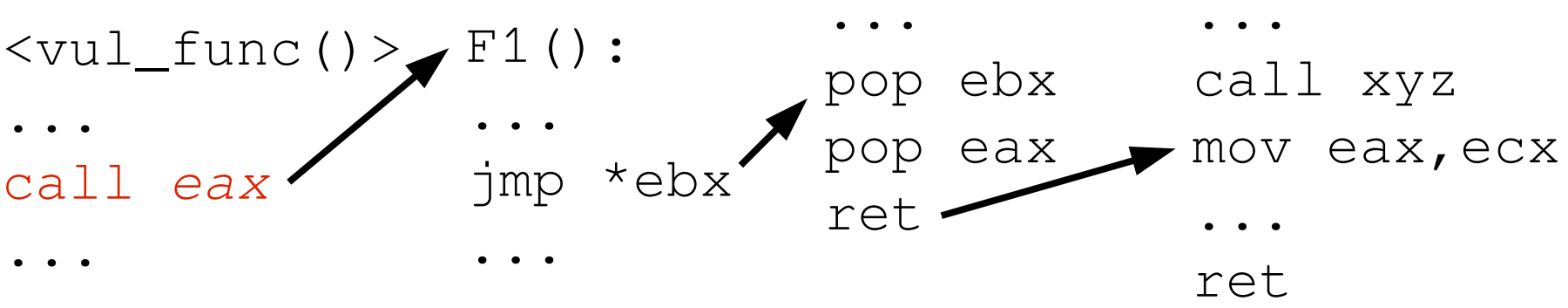
Low Precision ⟶ High Overhead

Motivation: Attack against Coarse-Grained CFI



Goktas et al., S&P'14
Carlini and Wagner, Usenix'14

Motivation: Attack against Coarse-Grained CFI



Goktas et al., S&P'14
Carlini and Wagner, Usenix'14

Low Precision → Attack Space

C++ Virtual Function Dispatch

C++ language
– Widely Used
– Object-Oriented: Polymorphism

Characteristics of C++ binary
– Large fraction indirect call instructions are virtual function dispatch.

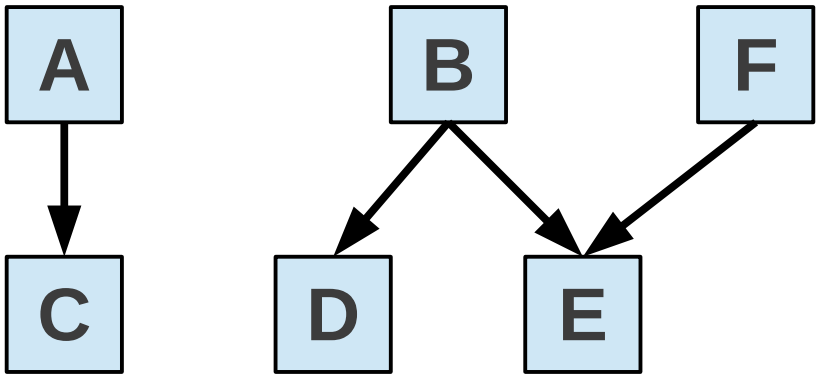
C++ Virtual Function Dispatch

C++ language
– Widely Used
– Object-Oriented: Polymorphism

Characteristics of C++ binary
– Large fraction indirect call instructions are virtual function dispatch.

Allowable Targets(ICI) = min({Target 1, Target 2, … , Target n})

An example...

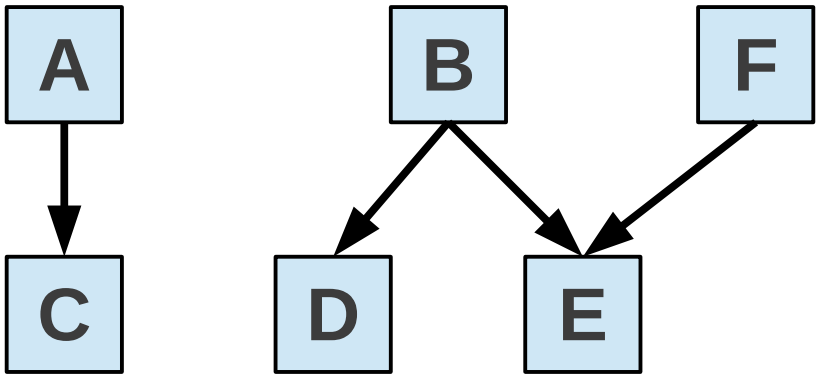


...

```
0x798 <A::Afoo()>
...
0x7ae:  call eax
0x7b0:  leave
0x7b1:  ret
```

```
A::vAduh();
B::vAtest(int);
C::vAduh();
void unrelated();
D::vAduh();
E::foo();
```

An example...

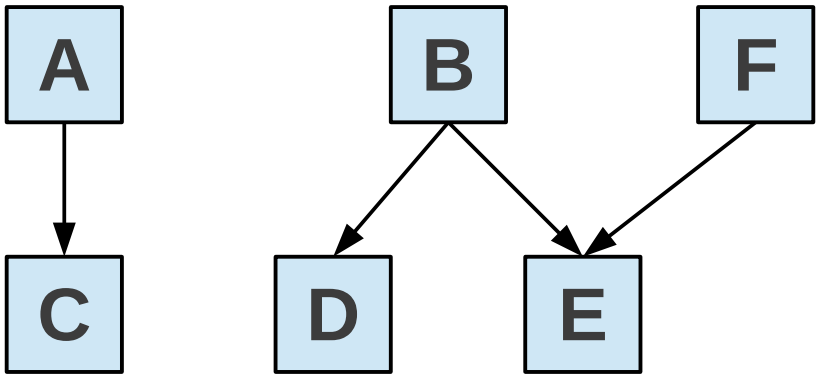


...

```
0x798 <A::Afoo()>
...
0x7ae:  call eax
0x7b0:  leave
0x7b1:  ret
```

```
A::vAduh();
B::vAtest(int);
C::vAduh();
void unrelated();
D::vAduh();
E::foo();
```

An example...

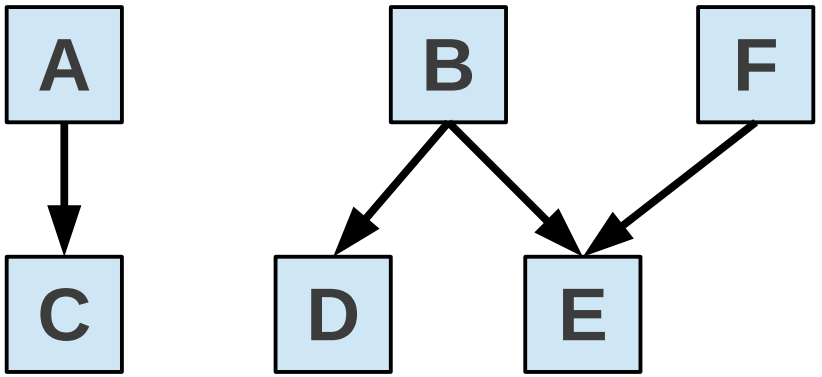


...

```
0x798 <A::Afoo()>
...
assert(eax == A::vAduh || eax == C::vAduh)
0x7ae:   call eax
0x7b0:   leave
0x7b1:   ret
```

```
A::vAduh();
B::vAtest(int);
C::vAduh();
void unrelated();
D::vAduh();
E::foo();
```

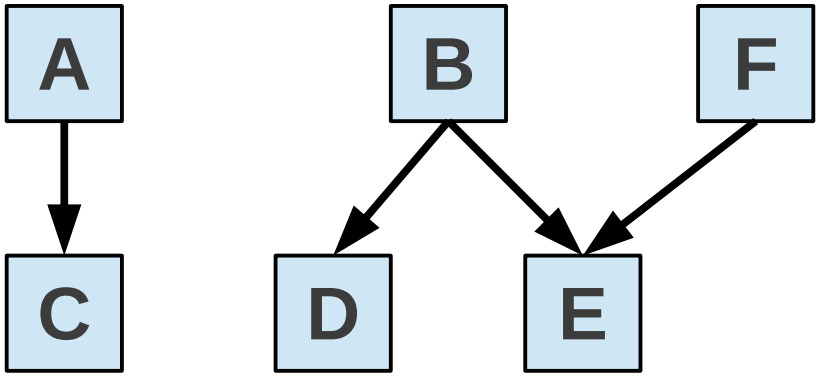
An example...



...

High-level Semantics → Higher Precision Low Attack Space

An example...

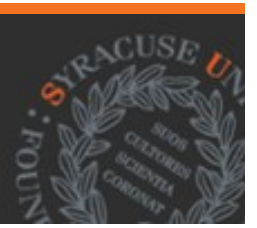


...

High-level Semantics → Higher Precision Low Attack Space

What semantics to recover?
How to recover them from the binary?

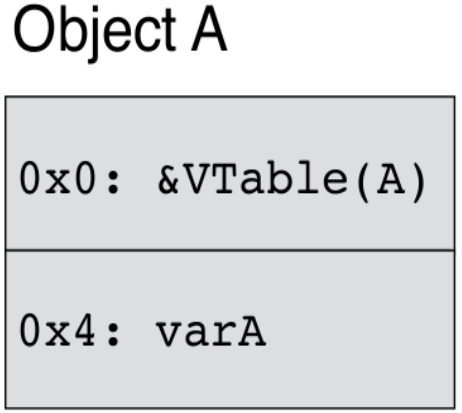
Virtual Tables in C++



```
class A {
int varA;
public:
virtual bool vAduh()
    {return true;}
virtual int vAtest(int a)
    {return 0;}
void Afoo()
    {this->vAduh();}
...
};
```

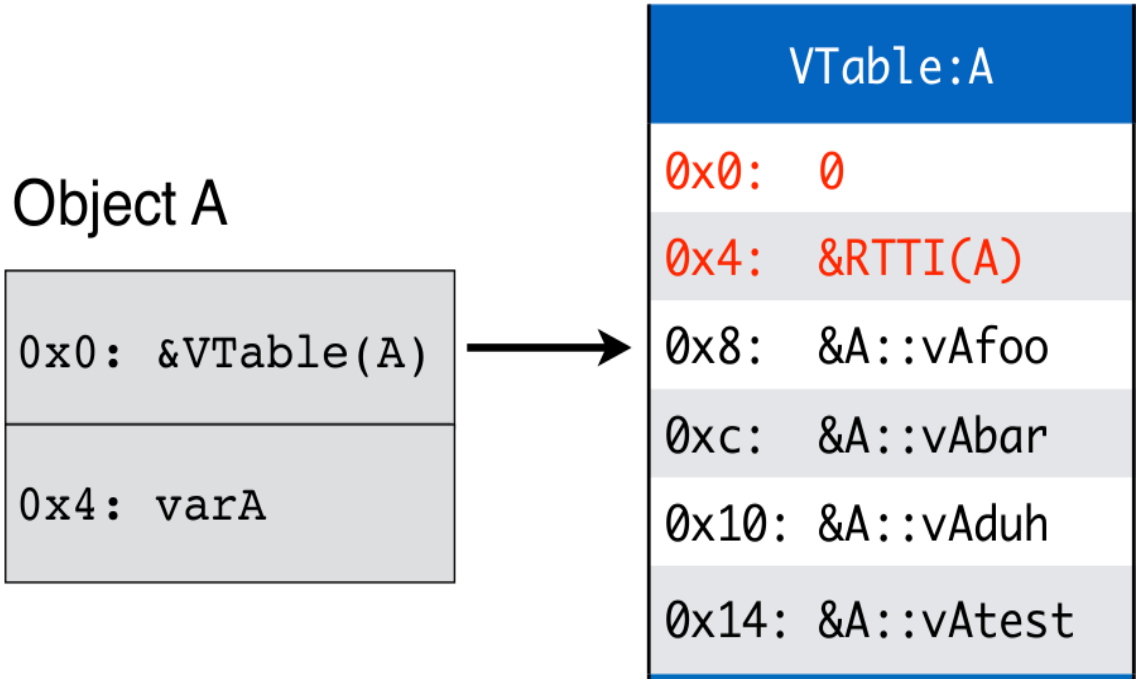
Virtual Tables in C++

```
class A {
int varA;
public:
virtual bool vAduh()
    {return true;}
virtual int vAtest(int
    {return 0;}
void Afoo()
    {this->vAduh();}
...
};
```



Virtual Tables in C++

```
class A {
int varA;
public:
virtual bool vAduh()
    {return true;}
virtual int vAtest(int
    {return 0;}
void Afoo()
    {this->vAduh();}
...
};
```



Virtual function dispatch must target a function within a VTable

Virtual Function Dispatch in C++

```
0x798 <A::Afoo()>:
   798: push ebp
   799: mov  ebp, esp
   79b: sub  esp, 0x18
   79e: mov  eax, DWORD PTR [ebp+0x8]
   7a1: mov  eax, DWORD PTR [eax]
   7a3: add  eax, 8
   7a6: mov  eax, DWORD PTR [eax]
   7a8: mov  edx, DWORD PTR [ebp+0x8]
   7ab: mov  DWORD PTR [esp], edx
   7ae: call eax
   7b0: leave
   7b1: ret
```

*gcc 4.6

Virtual Function Dispatch in C++

```
0x798 <A::Afoo()>:
   798: push ebp
   799: mov  ebp, esp
   79b: sub  esp, 0x18
   79e: mov  eax, DWORD PTR [ebp+0x8]
GetVT 7a1: mov  eax, DWORD PTR [eax]
   7a3: add  eax, 8
   7a6: mov  eax, DWORD PTR [eax]
   7a8: mov  edx, DWORD PTR [ebp+0x8]
   7ab: mov  DWORD PTR [esp], edx
   7ae: call eax
   7b0: leave
   7b1: ret
```

*gcc 4.6

Virtual Function Dispatch in C++

```
0x798 <A::Afoo()>:
   798: push ebp
   799: mov  ebp, esp
   79b: sub  esp, 0x18
   79e: mov  eax, DWORD PTR [ebp+0x8]
GetVT 7a1: mov  eax, DWORD PTR [eax]
   7a3: add  eax, 8  ; Offset in VTable
GetVF 7a6: mov  eax, DWORD PTR [eax]
   7a8: mov  edx, DWORD PTR [ebp+0x8]
   7ab: mov  DWORD PTR [esp], edx
   7ae: call eax
   7b0: leave
   7b1: ret
```

*gcc 4.6

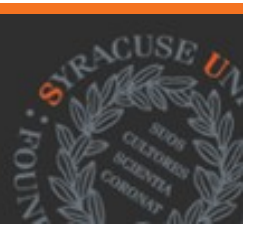
Virtual Function Dispatch in C++

```
0x798 <A::Afoo()>:
   798: push ebp
   799: mov  ebp, esp
   79b: sub  esp, 0x18
   79e: mov  eax, DWORD PTR [ebp+0x8]
GetVT 7a1: mov  eax, DWORD PTR [eax]
   7a3: add  eax, 8  ; Offset in VTable
GetVF 7a6: mov  eax, DWORD PTR [eax]
   7a8: mov  edx, DWORD PTR [ebp+0x8]
SetThis 7ab: mov  DWORD PTR [esp], edx
   7ae: call eax
   7b0: leave
   7b1: ret
```

this ptr on stack

*gcc 4.6

Virtual Function Dispatch in C++

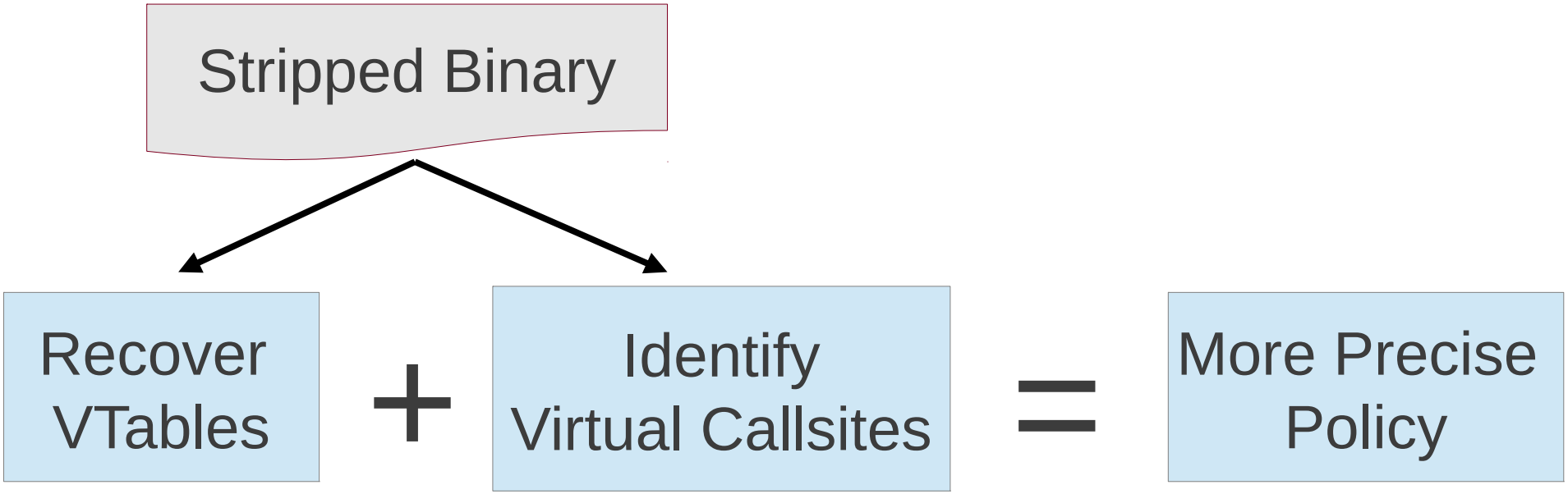


```
0x798 <A::Afoo()>:
   798: push ebp
   799: mov  ebp, esp
   79b: sub  esp, 0x18
   79e: mov  eax, DWORD PTR [ebp+0x8]
GetVT 7a1: mov  eax, DWORD PTR [eax]
   7a3: add  eax, 8  ; Offset in VTable
GetVF 7a6: mov  eax, DWORD PTR [eax]
   7a8: mov  edx, DWORD PTR [ebp+0x8]
SetThis 7ab: mov  DWORD PTR [esp], edx
CallVF 7ae: call eax
   7b0: leave
   7b1: ret
```

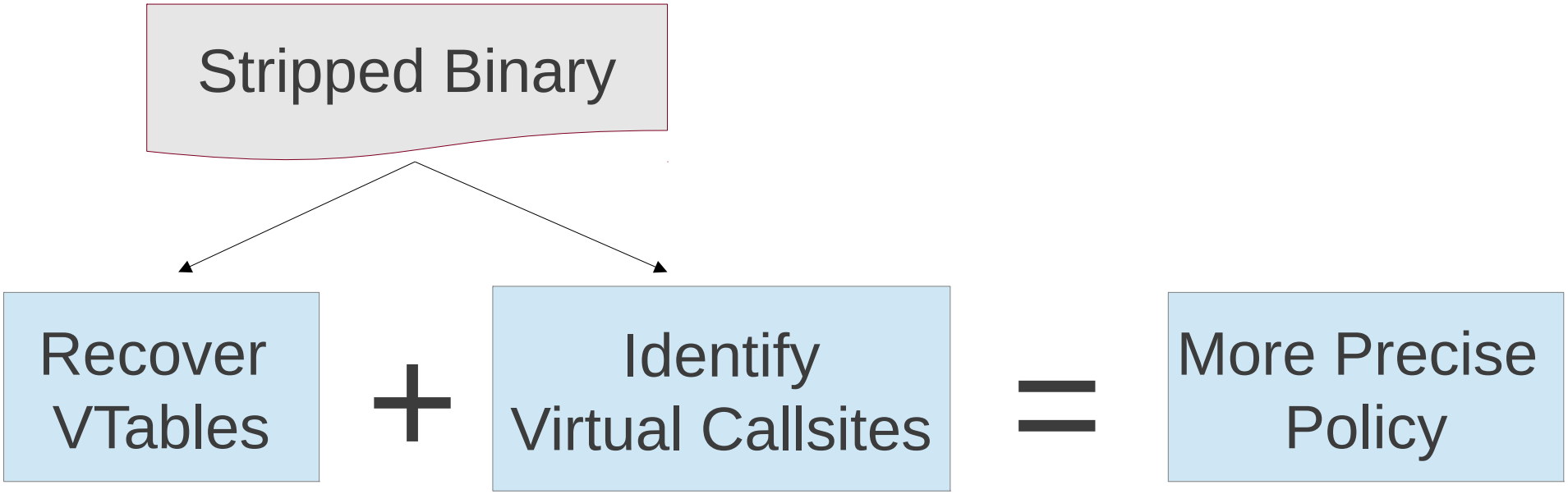
this ptr on stack

*gcc 4.6

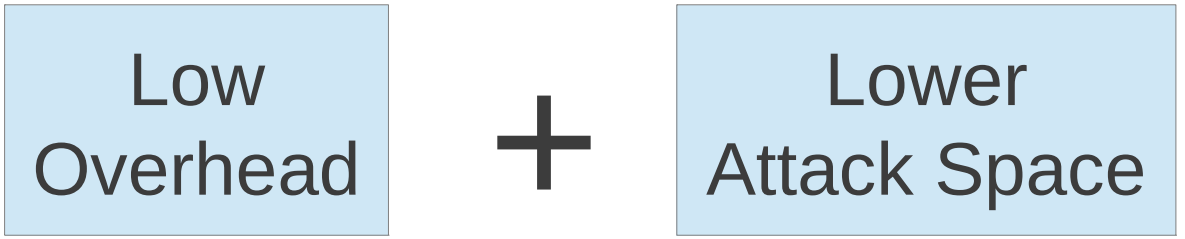
Our Solution



Our Solution



That is...



vfGuard

Soundness: In order to be sound, vfGuard must:

– Identify *all* Vtables. 0 false negatives.

– Do *not* identify a non-callsite as a callsite.
0 false positives.

Callsite Identification

Address	Instruction	IR-SSA form	After Propagation and Constant Folding
0x798	`push` ebp	$deref(esp_0) = ebp_0$ $esp_1 = esp_0 - 4$	$deref(esp_0) = ebp_0$ $esp_1 = esp_0 - 4$
0x799	`mov` ebp, esp	$ebp_1 = esp_1$	$ebp_1 = esp_0 - 4$
0x79b	`sub` $esp, 0x18h$	$esp_2 = esp_1 - 0x18$	$esp_2 = esp_0 - 0x1C$
0x79e	`mov` $eax, [ebp + 8]$	$eax_0 = deref(ebp_1 + 8)$	$eax_0 = deref(esp_0 + 4)$
0x7a1	`mov` $eax, [eax]$	$eax_1 = deref(eax_0)$	$eax_1 = deref(deref(esp_0 + 4))$
0x7a3	`add` $eax, 8$	$eax_2 = eax_1 + 8$	$eax_2 = deref(deref(esp_0 + 4)) + 8$
0x7a6	`mov` $eax, [eax]$	$eax_3 = deref(eax_2)$	$eax_3 = deref(deref(deref(esp_0 + 4)) + 8)$
0x7a8	`mov` $edx, [ebp + 8]$	$edx_0 = deref(ebp_1 + 8)$	$edx_0 = deref(esp_0 + 4)$
0x7ab	`mov` $[esp], edx$	$deref(esp_2) = edx_0$	$deref(esp_2) = deref(esp_0 + 4)$
0x7ae	`call` eax	`call` eax_3	`call` $deref(deref(deref(esp_0 + 4)) + 8)$

Callsite Identification

Address	Instruction	IR-SSA form	After Propagation and Constant Folding
0x798	`push` ebp	$deref(esp_0) = ebp_0$ $esp_1 = esp_0 - 4$	$deref(esp_0) = ebp_0$ $esp_1 = esp_0 - 4$
0x799	`mov` ebp, esp	$ebp_1 = esp_1$	$ebp_1 = esp_0 - 4$
0x79b	`sub` $esp, 0x18h$	$esp_2 = esp_1 - 0x18$	$esp_2 = esp_0 - 0x1C$
0x79e	`mov` $eax, [ebp + 8]$	$eax_0 = deref(ebp_1 + 8)$	$eax_0 = deref(esp_0 + 4)$
0x7a1	`mov` $eax, [eax]$	$eax_1 = deref(eax_0)$	$eax_1 = deref(deref(esp_0 + 4))$
0x7a3	`add` $eax, 8$	$eax_2 = eax_1 + 8$	$eax_2 = deref(deref(esp_0 + 4)) + 8$
0x7a6	`mov` $eax, [eax]$	$eax_3 = deref(eax_2)$	$eax_3 = deref(deref(deref(esp_0 + 4)) + 8)$
0x7a8	`mov` $edx, [ebp + 8]$	$edx_0 = deref(ebp_1 + 8)$	$edx_0 = deref(esp_0 + 4)$
0x7ab	`mov` $[esp], edx$	$deref(esp_2) = edx_0$	$deref(esp_2) = deref(esp_0 + 4)$
0x7ae	`call` eax	`call` eax_3	`call` $deref(deref(deref(esp_0 + 4)) + 8)$

`call` *deref (deref (exp) + offset),*

`call` *deref (deref (exp))*

VTable Recovery

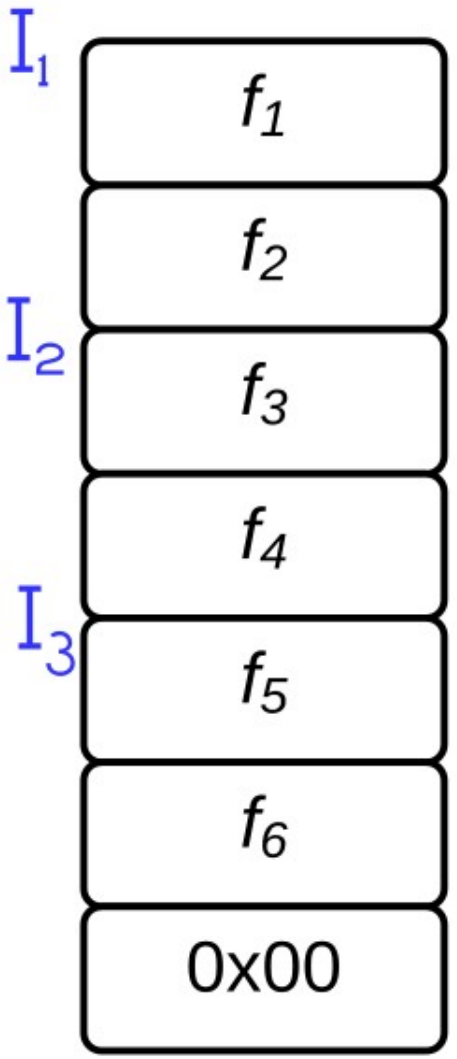
ABI-Specific VTable Signature
– Contains array of function pointers
– May contain optional fields

Characteristics of Vtables
– Present in read only sections
– VPTR initialized in constructors
(Vtable address occurs as immediate value)

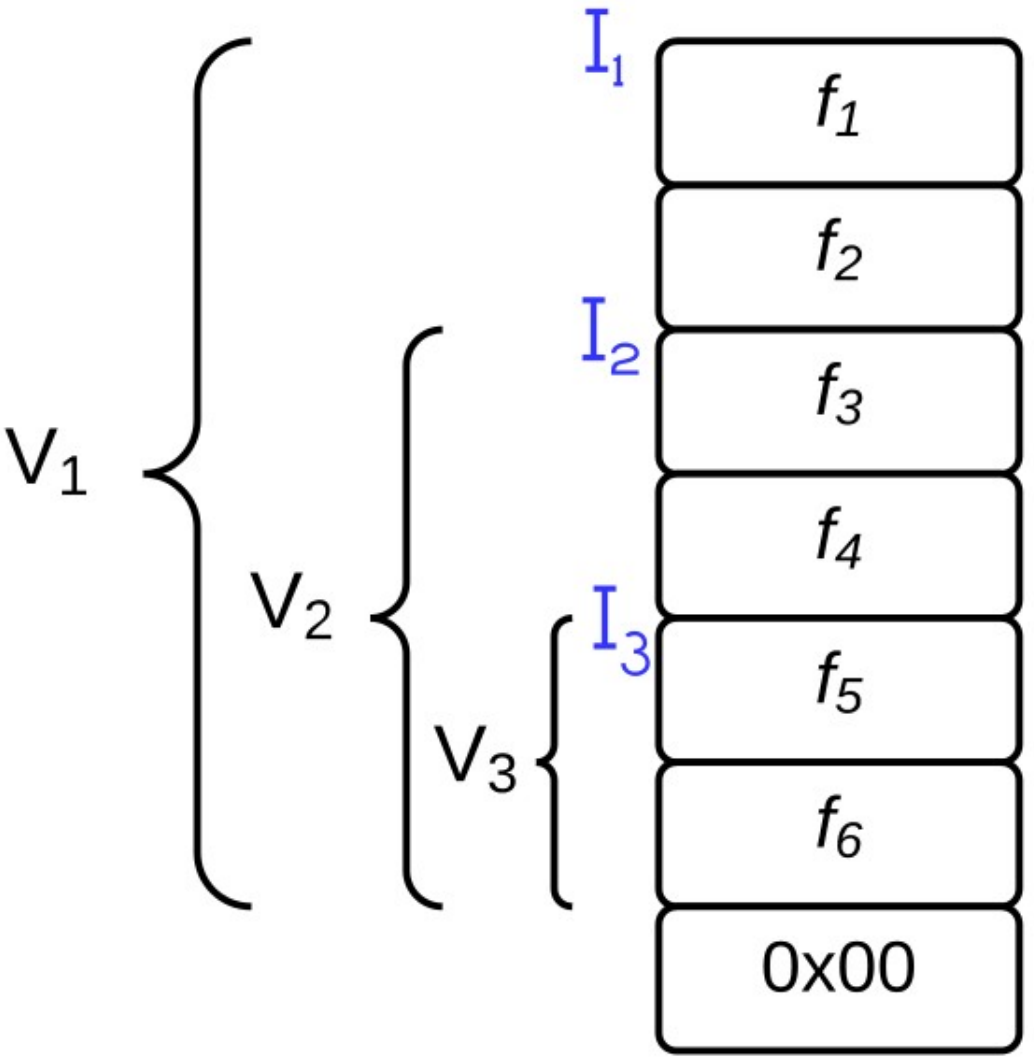
+ Identify *all* valid addresses in readonly regions that occur as immediate values in the code sections.
+ Check each such address for potential Vtable

VTable Recovery

```
.text
...
mov $I1, eax
...
mov $I2, edx
...
lea $I3, eax
...
```



VTable Recovery



Policy Generation – Basic Policy

Targets (*offset*) = { vfptrs in all Vtables at *offset* }

VTable:A	VTable:C
0x40: &A::vAfoo	0x58: &C::vAfoo
0x44: &A::vAbar	0x5c: &A::vAbar
0x48: &A::vAduh	0x60: &C::vAduh
0x4c: &A::vAtest	0x64: &C::vAtest

Polymorphic functions are present at the same offset

Policy Generation – Filters

Nested Virtual Call Filter:

– *this* pointer reuse

– Vfn belongs to Vtables that vAfoo belongs to.

```
Class A {
virtual void vAfoo() { this->vAduh(); }
};
```

– Filtered targets for nested virtual callsites.

`Caller.`*this* `==` `Callee.`*this*

`call` *deref (deref (deref(this)) + offset),*

Policy Generation – Filters

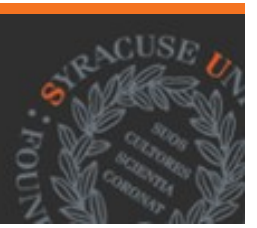
Calling Convention Filter:

– Calling convention at callsite must match calling convention at callee.

– Eliminate targets that don't match callsite calling convention.

$$\texttt{Callsite}.\mathit{conv} == \texttt{Callee}.\mathit{conv}$$

Inheritances



Can vfGuard deal with multiple and virtual inheritances?

Inheritances

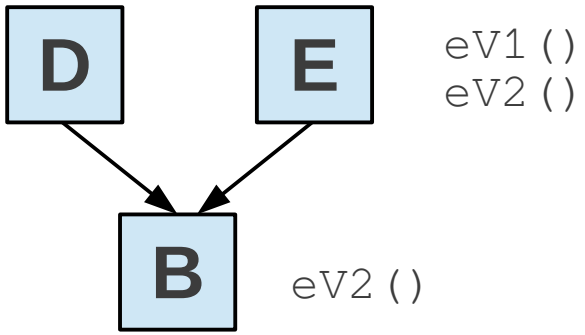
Can vfGuard deal with multiple and virtual inheritances?

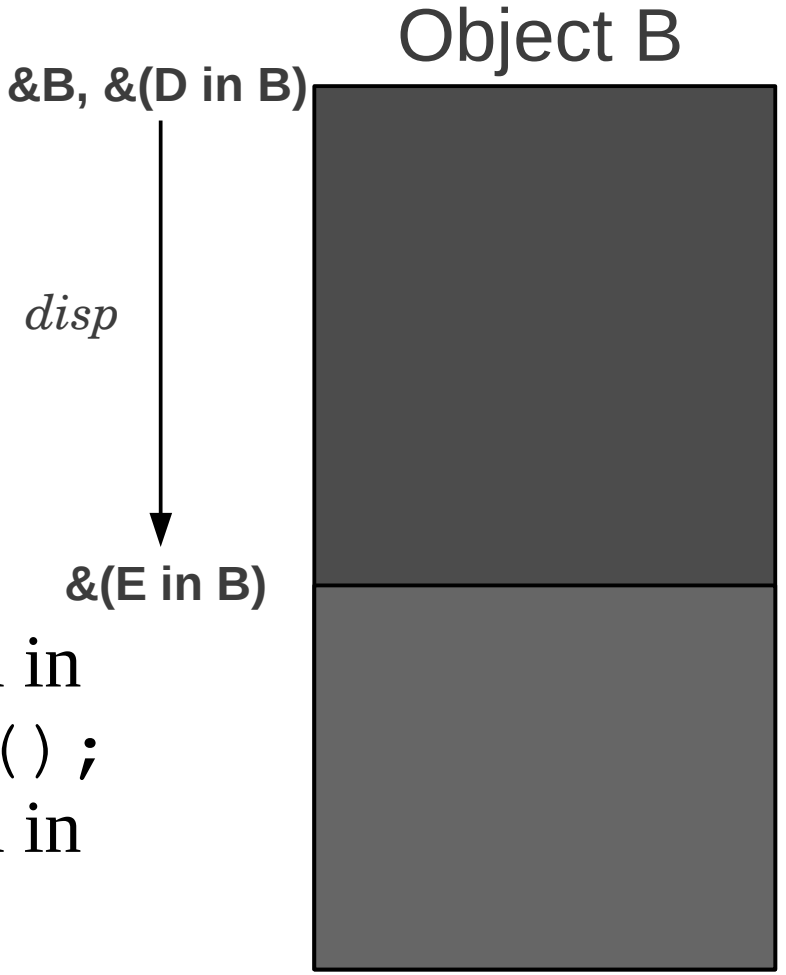
- **Yes**

Inheritances

Can vfGuard deal with multiple and virtual inheritances?

– **Yes**





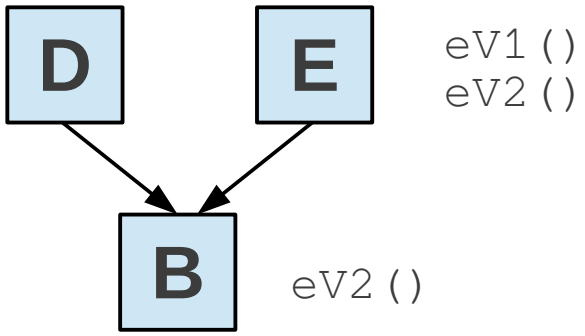
Two Cases:

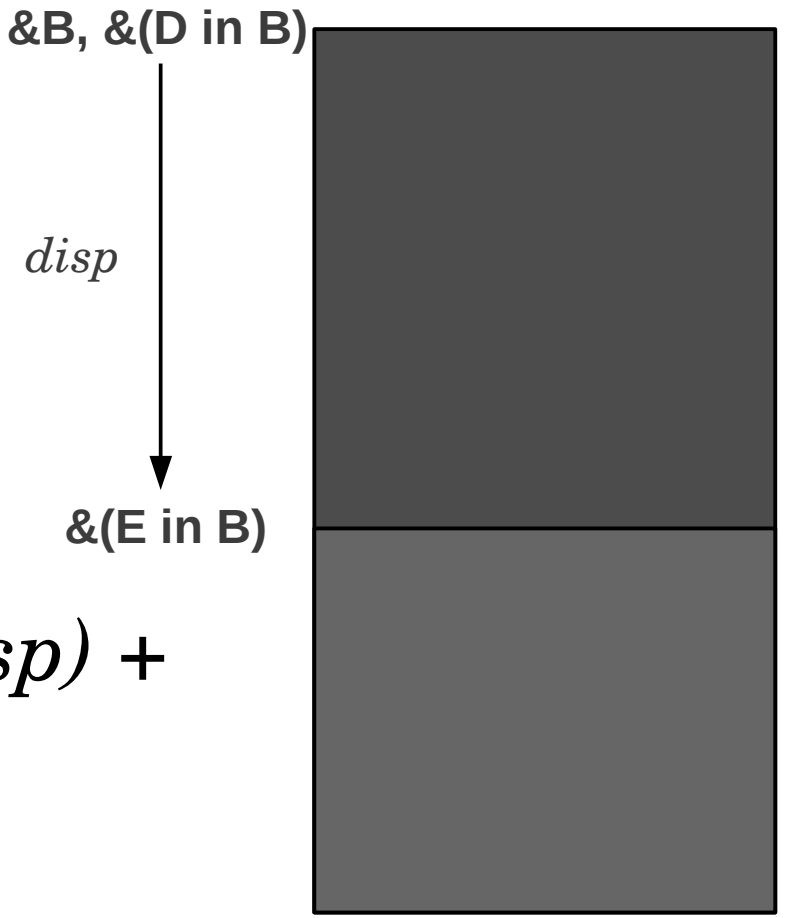
1. Derived class object invokes vfn in secondary base class. e.g., `b.eV1();`
2. Derived class object invokes vfn in secondary base class that it has overridden. e.g., `b.eV2();`

Inheritances

Can vfGuard deal with multiple and virtual inheritances?

– **Yes**





Case 1:

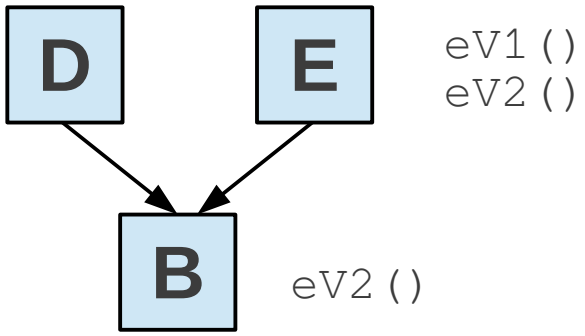
`call` *deref (deref (exp + disp) + offset)*

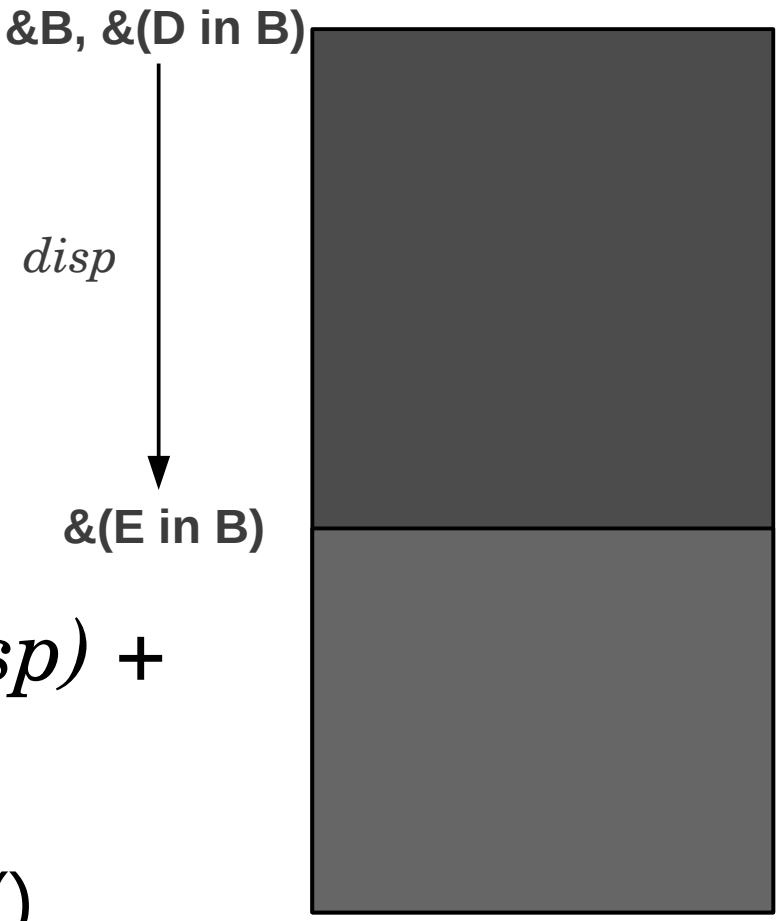
Target → &E::eV1()

Inheritances

Can vfGuard deal with multiple and virtual inheritances?

– **Yes**





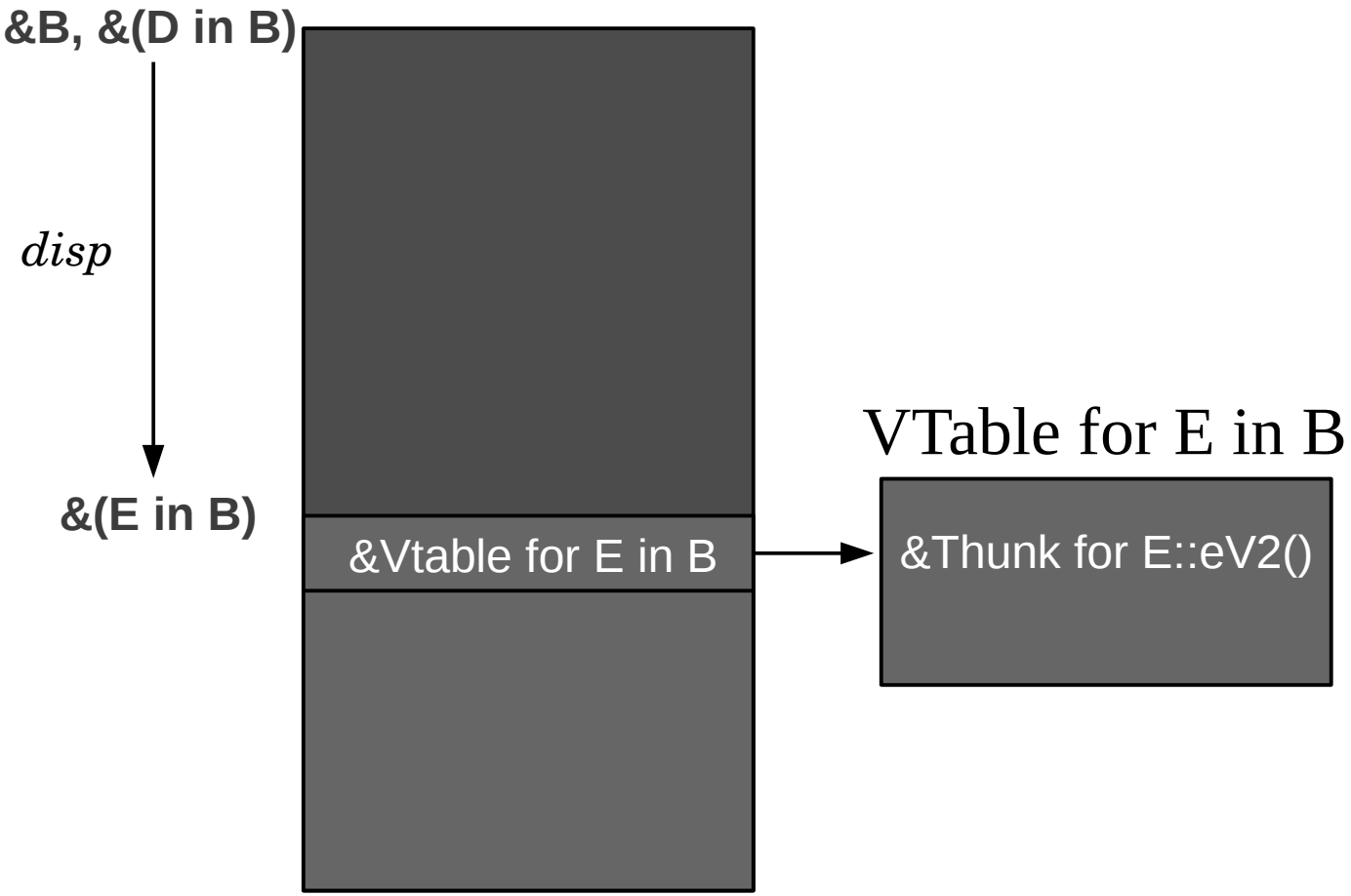
Case 2:

`call` *deref (deref (exp + disp) + offset)*

Target → &Thunk to B::eV2()

Inheritances

Case 2



Thunk for E::eV2():

```
sub $disp,this
jmp E::eV2()
```

Experimental Results – Identification Accuracy

Vtable Identification

Program	Ground Truth	vfGuard	FP	FN
SpiderMonkey	811	942	13.9%	0
dplus-browser_0.5b	270	334	19.1%	0
TortoiseProc.exe	568	595	4.7%	0

Callsite Identification

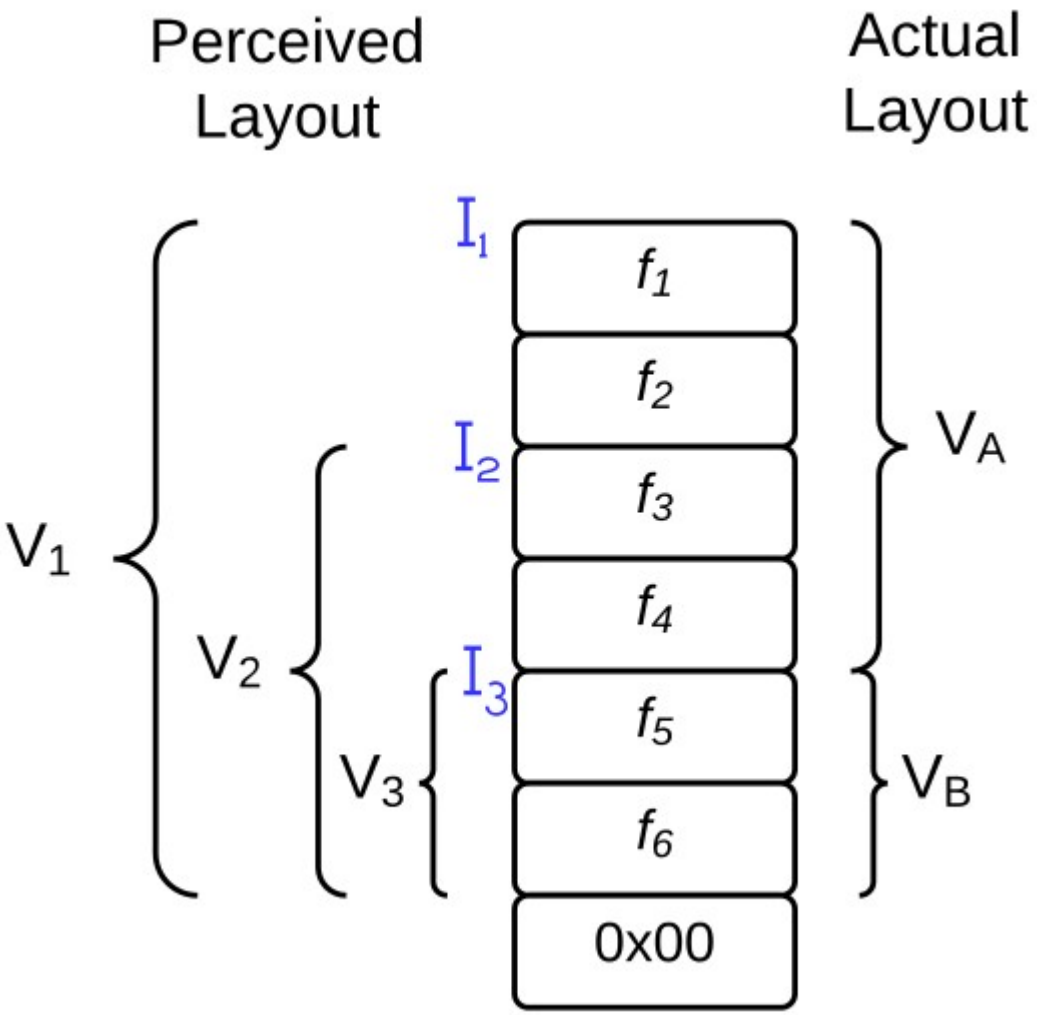
Program	Ground Truth	vfGuard	FP	FN
SpiderMonkey	1780	1754	0	1.4%
dplus-browser_0.5b	309	287	0	7.1%

Experimental Results

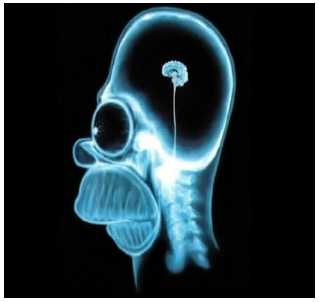
Policy Precision

Program	Total VTables Identified	Total Callsites Identified (CS)	Avg. Targets per CS (Basic Policy)	# Nested CS	Avg. Targets per CS (NCF)	Avg. Targets per CS (NCF+CCF)	Estimated `call` Targets – BinCFI	Call Target Reduction w.r.t BinCFI
ExplorerFrame.dll	736	6314	231	257	227	223	8964	97.5%
msxml3.dll	587	3321	96	219	88	84	6822	98.8%
jscript.dll	129	1170	39	55	38	38	2314	98.4%
mshtml.dll	1174	3583	292	211	258	257	16287	98.3%
WMVCore.dll	736	7516	268	562	256	244	8845	97.3%

Vtables Identification – False Negatives



Future Work

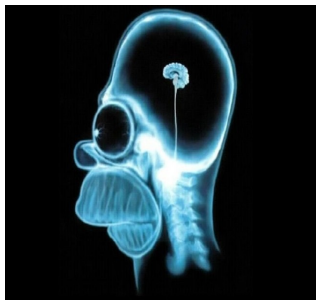


```
#include <stdio.h>
int main() {
….
}
….
```

```
0011001001010111
1101011101010111
1111110101010001
0001001011100110
0101011001110100
```

Steady dilution of intended control-flow

Future Work



```
#include <stdio.h>
int main() {
….
}
….
```

```
0011001001010111
1101011101010111
1111110101010001
0001001011100110
0101011001110100
```

Steady dilution of intended control-flow

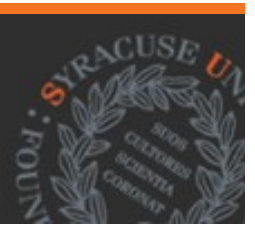
– Recovery of more high-level semantics to obtain better CFG.

Questions?

Thank you!

Aravind Prakash
arprakas@syr.edu

Policy Coverage



Policy Coverage

Program	Total # Indirect `call` instructions	Total # Indirect `jmp` instructions	Total # `ret` instructions	Total # Indirect `call`s analyzed (instructions successfully transformed to IR)	% of analyzed `call`s protected	% of Total indirect `call`s protected
ExplorerFrame.dll	7797	87	7266	7042	89.7%	81.0%
msxml3.dll	5439	78	6157	4045	82.1%	61.1%
jscript.dll	2235	5	4430	1678	69.7%	52.3%
mshtml.dll	9843	352	15479	4598	77.9%	36.4%
WMVCore.dll	9748	50	8497	8223	91.4%	77.1%