FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Protecting Virtual Calls in Binary Programs:

From COTS Applications To CPS Applications

Chao Zhang (UC Berkeley)

Dawn Song (UC Berkeley)

Berkeley UNIVERSITY OF CALIFORNIA

Massachusetts Institute of Technology

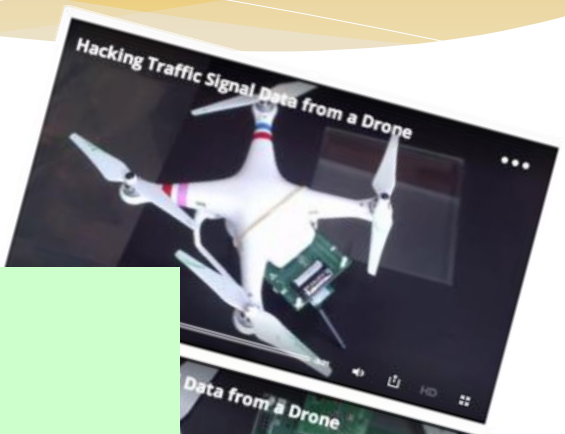
NSF

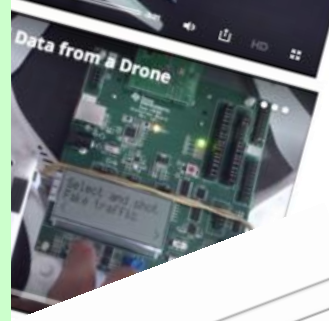
MICHIGAN

VANDERBILT UNIVERSITY

Lots of attack targets

(cars, traffic lights, navigation routes, signs, …)

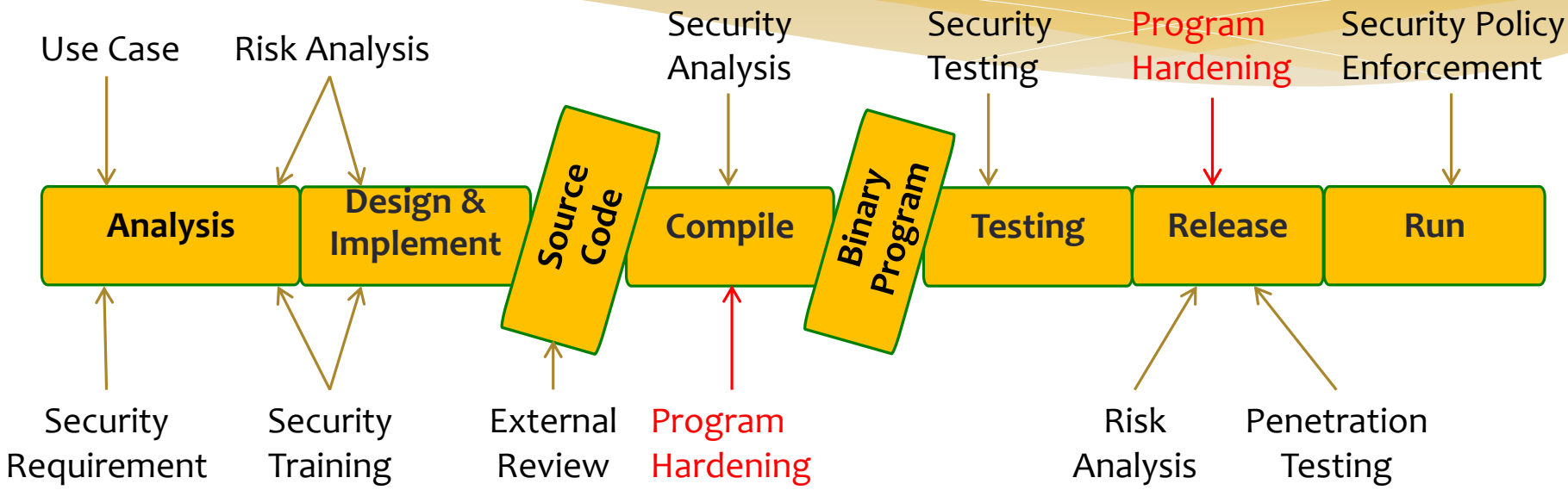




What do they have in common?

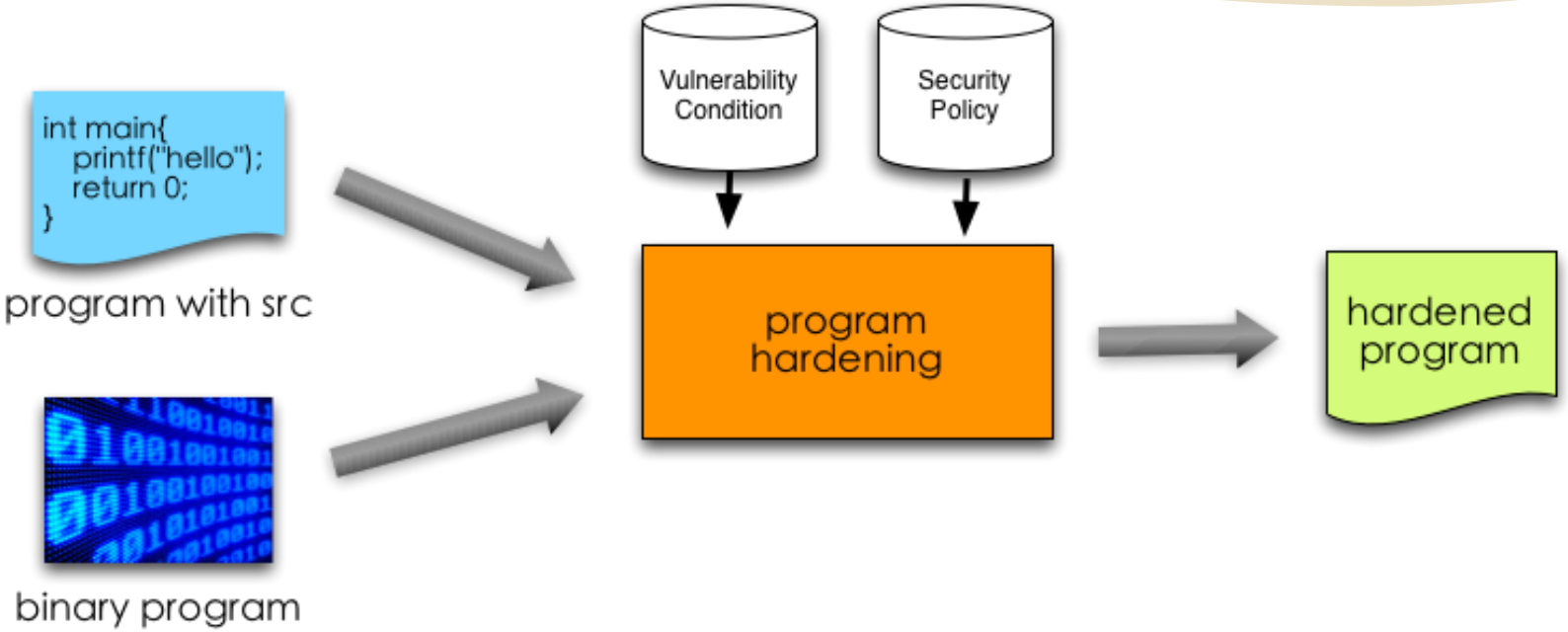
Software vulnerabilities and exploits.

Secure Software Development Life Cycle



* Existing solutions are not sufficient
  * Vulnerabilities are inevitable when designing and implementing.
  * Testing is not able to find out all potential vulnerabilities.
  * Runtime protection is not sufficient, and has compatibility issues.
* Proactive program hardening is a promising solution

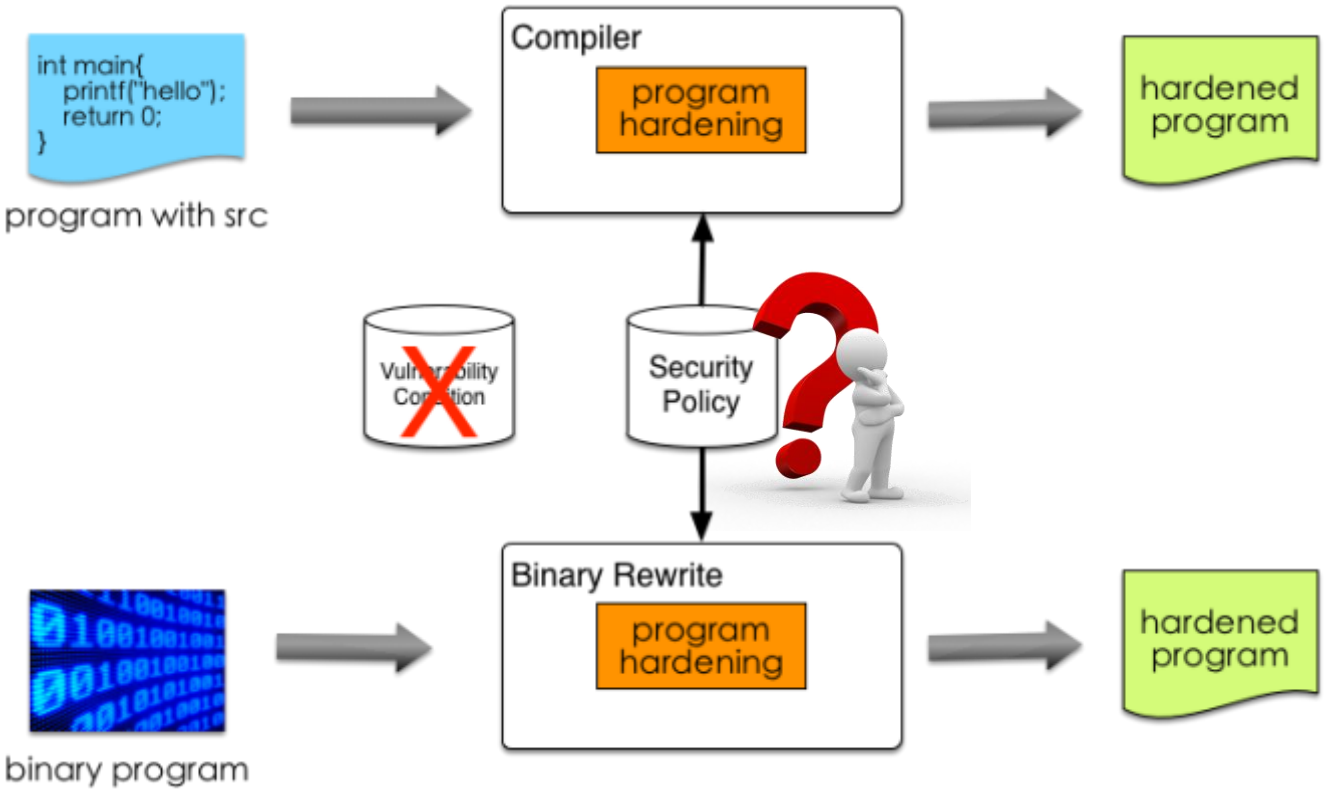
Program Hardening



* Fix vulnerabilities
* Deploy security checks

FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Our Solutions



FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

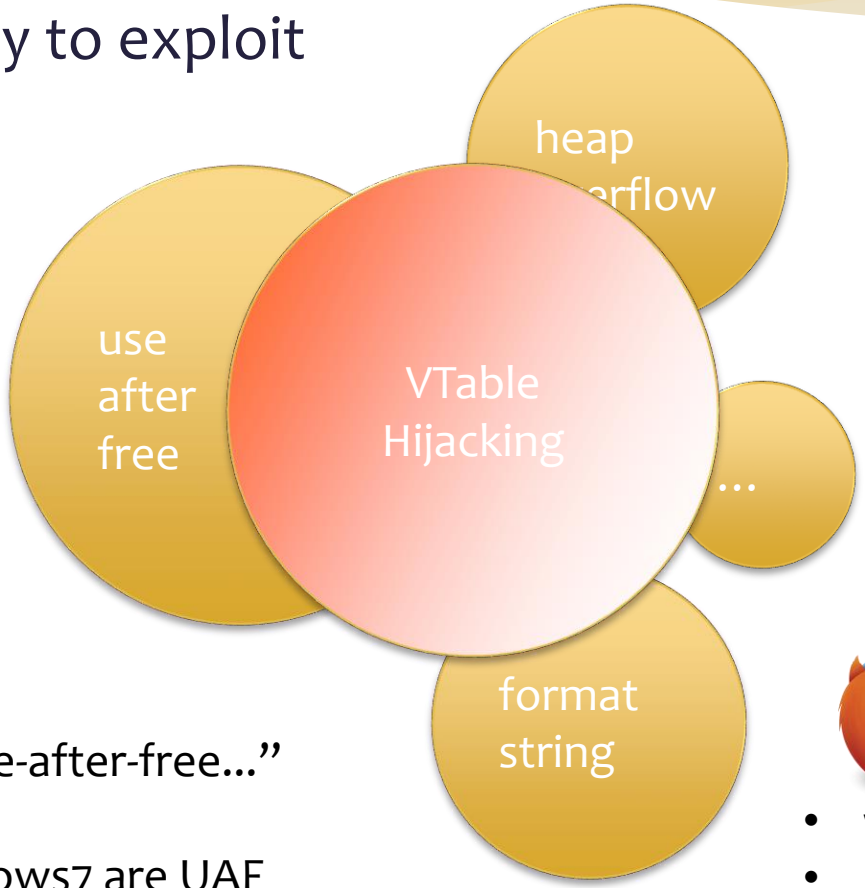
To select a security policy and enforce it,

Know your enemy first.

Sun Tzu

Real World Attacks: VTable Hijacking

* A common way to exploit



Google:
"80% attacks exploit use-after-free..."
Microsoft:
50% CVEs targeted Winows7 are UAF

- written in C++
- BIG Targets in the Cloud

FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

What is VTable?

* A data structure for supporting dynamic dispatch (C++)

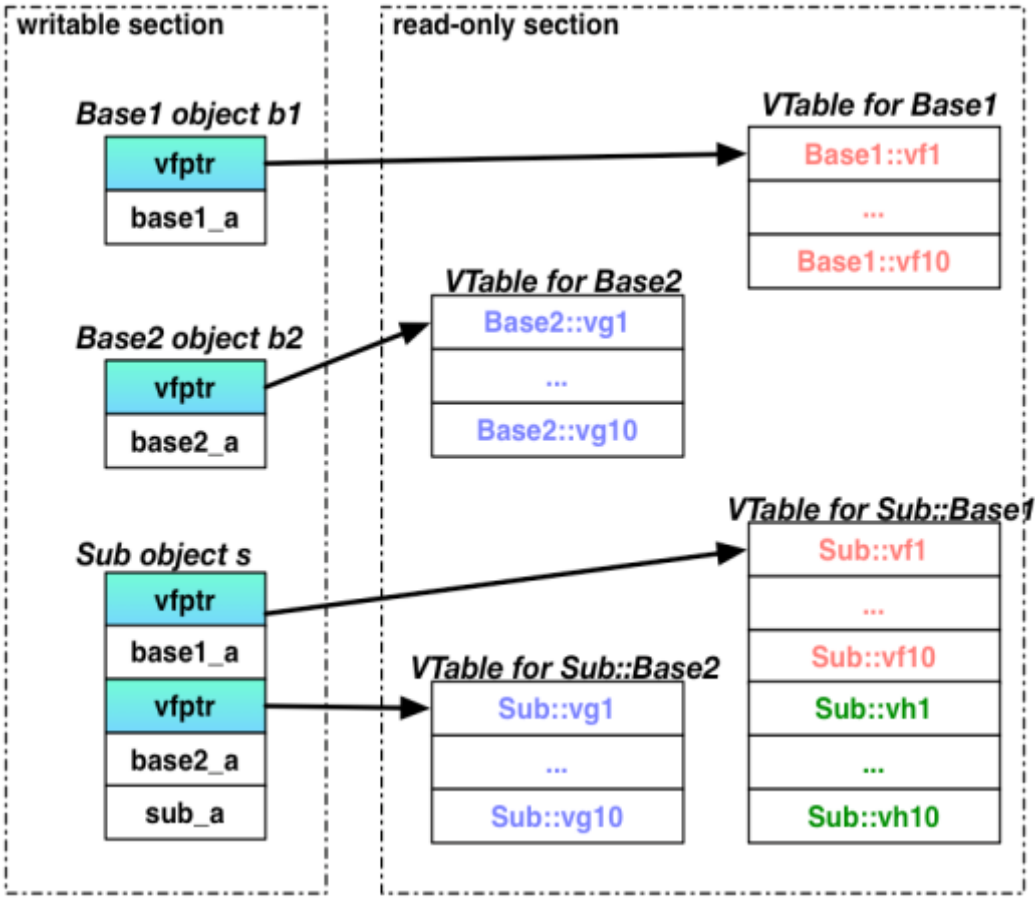
```
class Base1{… };  // virtual function vf1~vf10
class Base2{… };  // virtual function vg1~vg10
class Sub: Base1, Base2{… };

void foo(Base2* obj) {
   // What function will be invoked?
   // Depending on the runtime object.
   obj->vg4();
}

void main(){
   Base2* b2 = new Base2();
   foo(b2);
   Sub* s = new Sub();
   foo(s);
}
```

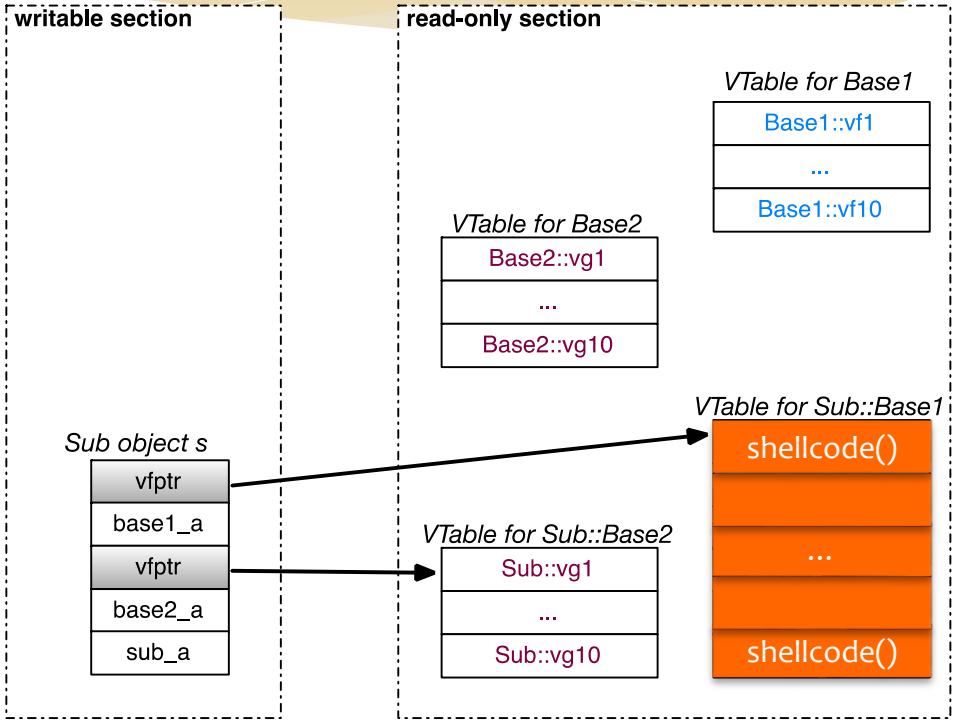
code section

```
; Function foo()
mov eax, [ecx]            ; read vtptr of Base2 object
mov edx, [eax+0x0C]       ; get vg4() from vtable
call  edx                 ; call Base2::vg4()
ret
```



VTable Hijacking

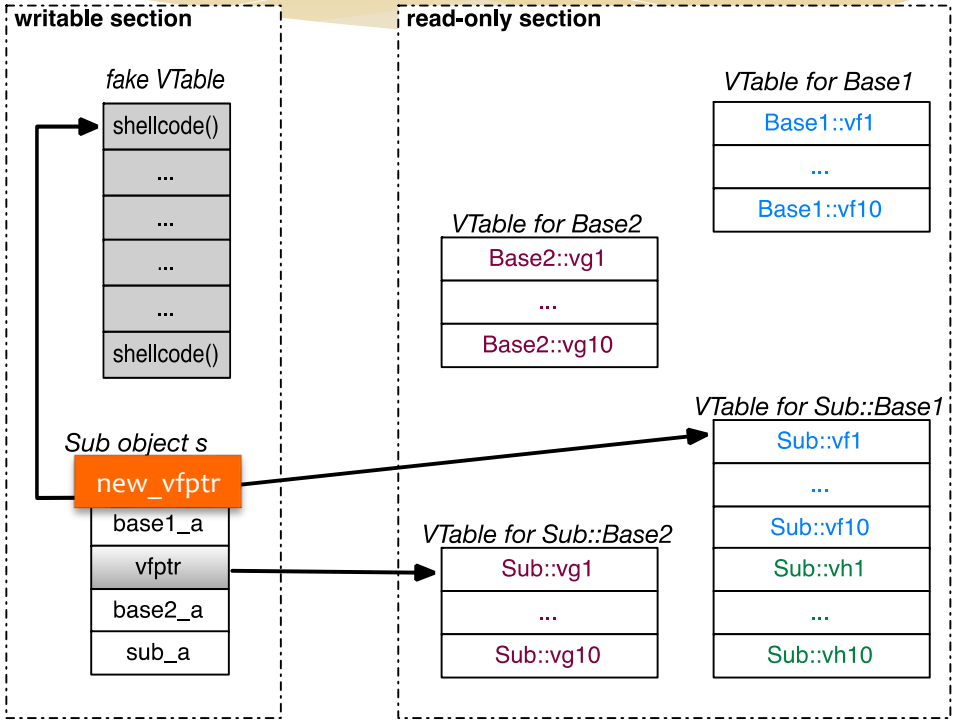
* VTable corruption
  * overwrite VTable
* VTable injection
* VTable reuse



FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

VTable Hijacking

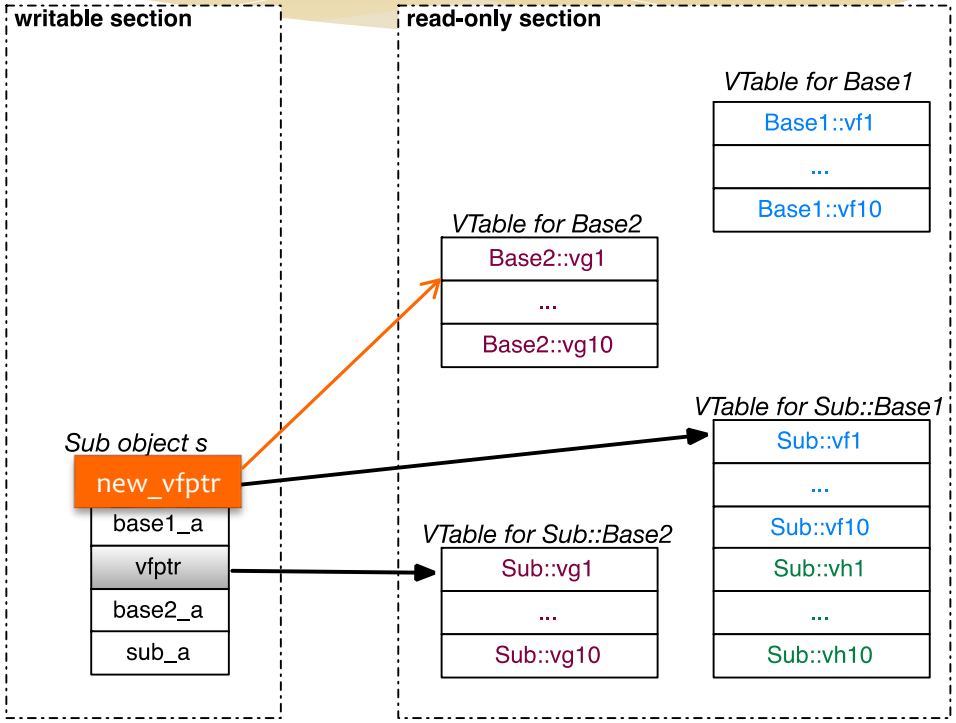
* VTable corruption
  * overwrite VTable
* VTable injection
  * overwrite vfptr
  * point to fake VTable
* VTable reuse



FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

VTable Hijacking

* VTable corruption
  * overwrite VTable
* VTable injection
  * overwrite vfptr
  * point to fake VTable
* VTable reuse
  * overwrite vfptr
  * point to existing VTable, data etc.



FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

The Question

* Goal: defense against VTable Hijacking
  * lightweight
  * binary program support
  * effective

* What security policies to deploy?
* How to deploy these policies to binary programs?

Our solution: VTint

Motivation

VTint Design

VTint Implementation

Evaluation on COTS Applications

Investigation on CPS Applications

Observation

	Attack	Requirement	
VTable Corruption	overwrite VTable	VTable is writable	
VTable Injection	overwrite vfptr, point to injected VTable	VTable is writable	
VTable Reuse	overwrite vfptr, point to existing VTable/data	VTable-like data, existing VTable	

Observation → Intuition

	Attack	Requirement	Countermeasure
VTable Corruption	overwrite VTable	VTable is writable	Read-only VTable
VTable Injection	overwrite vfptr, point to injected VTable	VTable is writable	Read-only VTable
VTable Reuse	overwrite vfptr, point to existing VTable/data	VTable-like data, existing VTable	different VTable/data

Need exact TYPE information

Light weight source-code solutions like VTGuard

The security policy

* Policy 1:
  * legitimate VTables should be placed in read-only memory
    * attackers cannot corrupt legitimate VTables
* Policy 2:
  * only read-only VTables can be used in runtime virtual calls
    * attackers cannot inject fake VTables
* Policy 3:
  * legitimate VTables are different from other data
    * attackers can hardly reuse other data as VTables

Our solution: VTint

Motivation

VTint Design

VTint Implementation

Evaluation on COTS Applications

Investigation on CPS Applications

Challenges

* Source code → Native code

```
class Base1{… };  // virtual function vf1~vf10
class Base2{… };  // virtual function vg1~vg10
class Sub: Base1, Base2{… };

void foo(Base2* obj) {
   obj->vg4();
}

void main(){
   Base1* b1 = new Base1();
   Base2* b2 = new Base2();
   foo(b2);
   Sub* s = new Sub();
   foo(s);
}
```

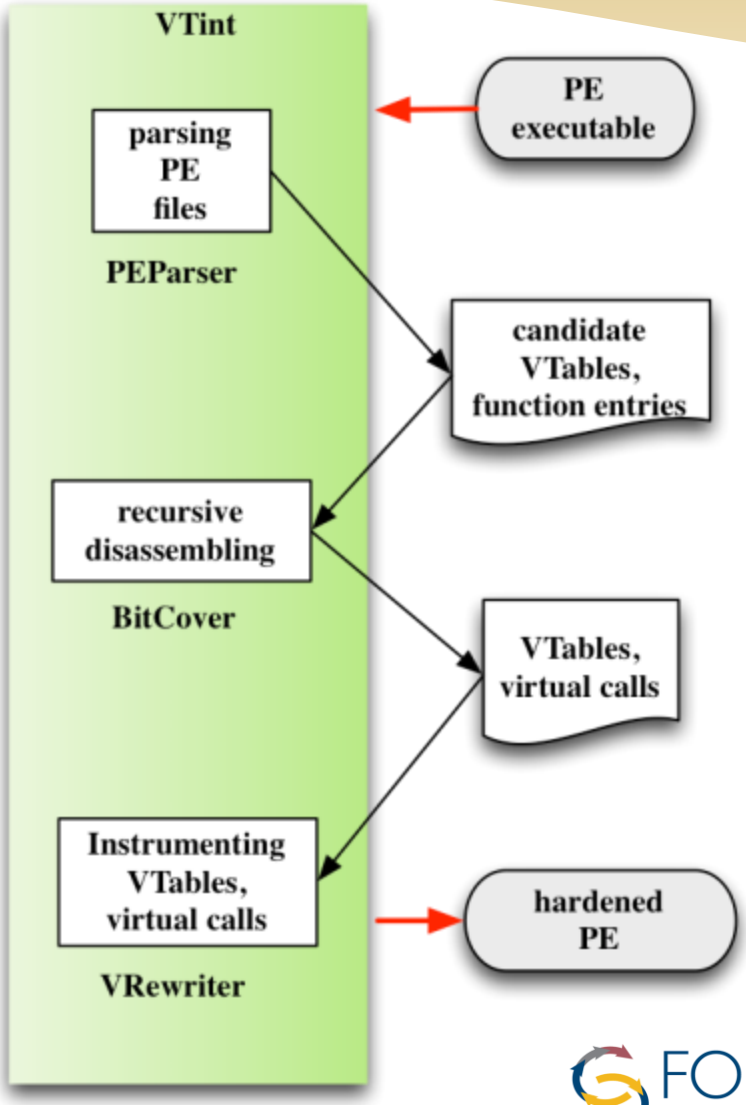
code section

```
; Function foo()
mov eax, [ecx]          ; read vtptr of Base2 object
mov edx, [eax+0x0C]     ; get vg4() from vtable
call  edx               ; call Base2::vg4()
ret
```

- A lot of information are missing
  - types, virtual call, VTables…

- How to recover high-level information from binary programs?
  - Which are virtual calls
  - Which are VTables?

Architecture



- Binary parsing
  - candidate VTables
  - candidate functions
- Disassembling
  - code/data
  - constructor functions
  - VTables
  - virtual calls
- Binary rewriting

FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Binary Rewriting

* Security Policy
  * Place legitimate VTables in read-only sections
  * Enforce runtime VTables to be read-only
  * Differentiate VTables from other data
* Rewriting


```
; get vtable ptr from object
mov  eax, [ecx+8]
```

check vtable page has VTID

check vtable page is read-only

```
; get virtual func ptr from vtable
mov  edx, [eax+24]
; call virtual function
call  edx
```

Our solution: VTint

Motivation

VTint Design

VTint Implementation

Evaluation on COTS Applications

Investigation on CPS Applications

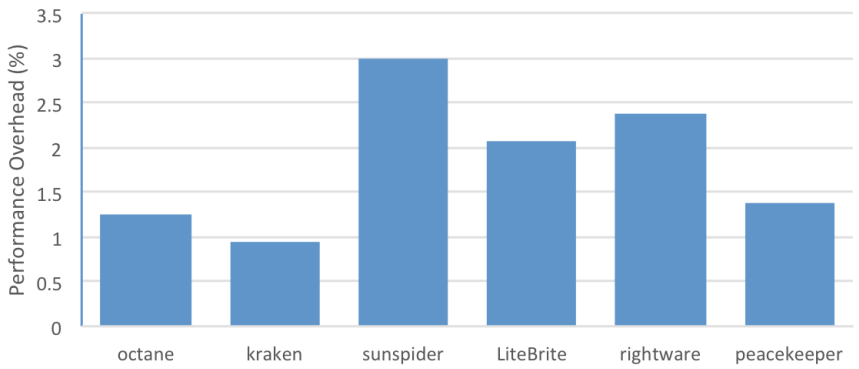
Static Analysis Results

* Firefox analysis
  * fast analysis for each module
  * small file size overhead

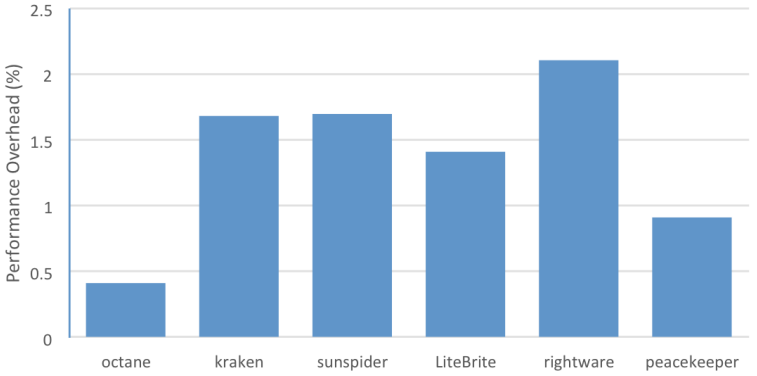
App	analysis time (sec)	file size (KB)			VTable info		
		orig	new	size overhead	#inst	#vtables	#vcalls
crashreporter.exe	1.8	116	117	0.52%	18,461	3	15
updater.exe	3.7	271	276	1.77%	112,693	9	17
webapprt-stub.exe	1.6	96	97	0.61%	38,589	2	17
D3DCompiler_43.dll	74.3	2,106	2,202	4.53%	2,135,041	48	1338
d3dx9_43.dll	36.9	1,998	2,184	9.33%	627,400	124	4152
gkmedias.dll	84.9	4,221	4,493	6.45%	2,130,418	483	5542
libEGL.dll	0.99	59	64	7.99%	17,772	3	156
libGLESv2.dll	23.7	473	519	9.91%	913,890	87	983
mozjs.dll	123.6	2,397	2,444	1.95%	4,553,743	35	174
msvcp100.dll	5.0	421	450	6.79%	78,586	116	438
msvcr100.dll	13.2	770	778	0.92%	291,484	91	270
xul.dll	328.9	15,112	17,768	17.57%	5,801,649	6548	54743

Performance Evaluation

* Firefox



* Chrome



- Average performance overhead is less than 2%

FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Attack Surface of Firefox

library/ executable	VTable			Call Instruction		
	#	assign	read	call	iCall	vCall
makeconv	173	470	1831	62625	6.66%	42.85%
genrb	173	473	1831	68429	6.35%	41.10%
icuinfo	173	470	1844	66600	6.35%	42.40%
genccode	173	470	1831	61037	6.81%	42.95%
gencmn	173	470	1831	61051	6.81%	42.95%
icupkg	175	476	1845	63197	6.89%	41.36%
pkgdata	175	476	1845	64363	6.75%	41.43%
gentest	174	471	1846	66640	6.35%	42.42%
gennorm2	179	478	1837	61831	6.78%	42.73%
gendict	174	472	1831	60896	6.83%	42.94%
js	1420	1991	3626	262502	23.26%	5.87%
libxul.so	15801	26212	72874	1720021	15.21%	27.48%

Protection Effect

* Real World Exploits

CVE-ID	App	Vul Type	POC Exploit	Protected
CVE-2010-0249	IE6	use-after-free	*vtable injection* [5]	YES
CVE-2012-1876	IE8	heap overflow	*vtable injection* [37]	YES
CVE-2013-3205	IE8	use-after-free	*vtable injection* [7]	YES
CVE-2011-0065	FF3	use-after-free	*vtable injection* [39]	YES
CVE-2012-0469	FF6	use-after-free	*vtable injection* [15]	YES
CVE-2013-0753	FF17	use-after-free	*vtable injection* [22]	YES

Limitations

* Binary disassembling
* High-level information recovery
  * Constructor functions
  * VTables
  * Virtual function calls
* Reusing existing VTables
  * call existing virtual functions

Our solution: VTint

Motivation

VTint Design

VTint Implementation

Evaluation on COTS Applications

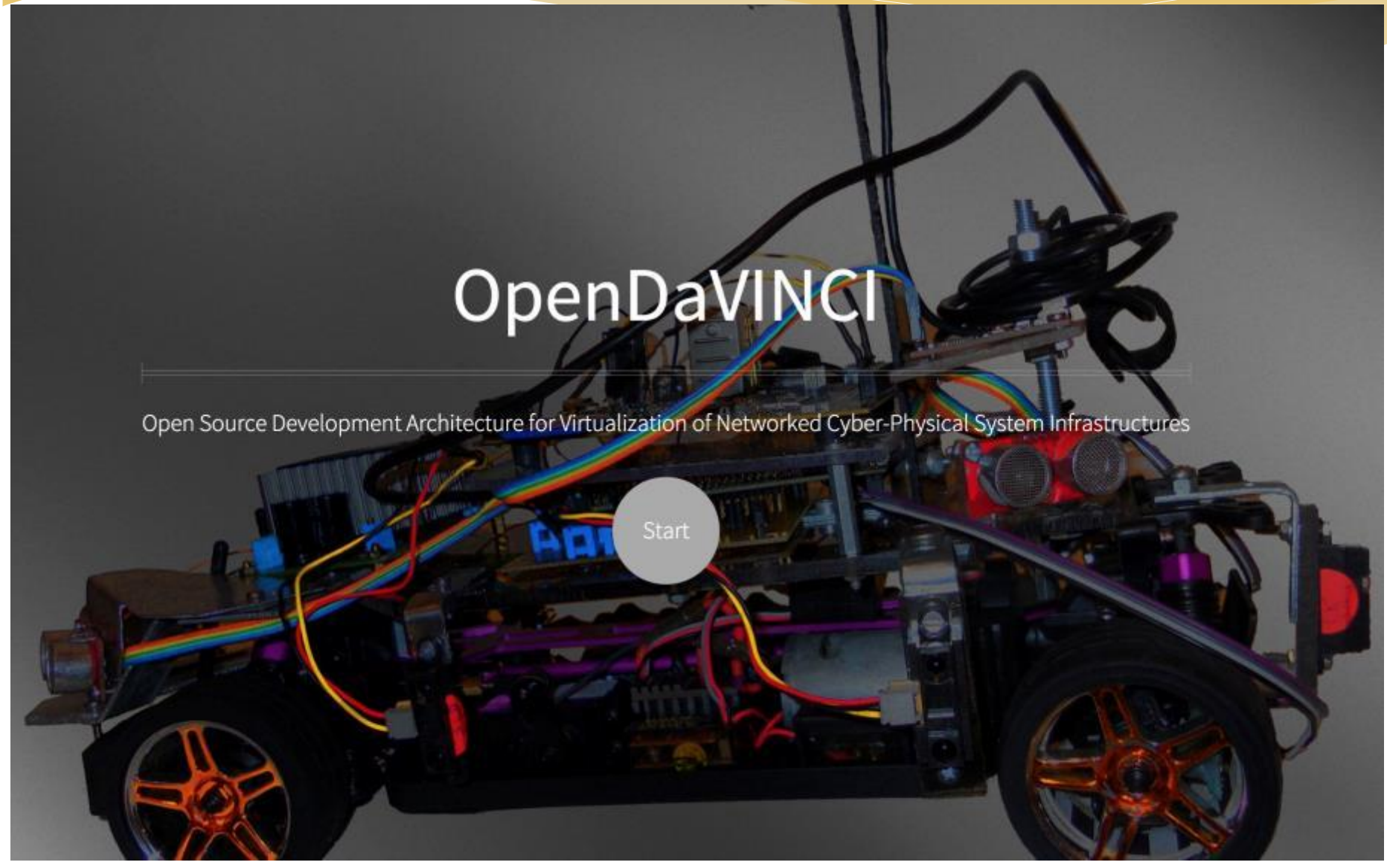
Investigation on CPS Applications

CPS Applications Written in C++

* C++ is mature and efficient programming language, widely used in COTS application development

* CPS applications also utilize C++
  * Simulators based on SystemC
    * Modeling Time-Triggered Ethernet in SystemC/TLM for Virtual Prototyping of Cyber-Physical Systems
  * Middleware developed in C++
    * The Design and Performance of Cyber-Physical Middleware for Real-Time Hybrid Structural Testing, wucse-2009-27
  * Hardware control system, Network Communication etc.
    * HVDC industrial controller

Sample


OpenDaVINCI
Open Source Development Architecture for Virtualization of Networked Cyber-Physical System Infrastructures
Start

Attack Surface of OpenDaVINCI

	#vtable	#vcall
RuntimeControl~1	175	1325
RuntimeControl~2	161	826
RuntimeControl~3	155	780
QueueTestSuite	63	650
ControlFlowTes~	138	643
ConferenceClie~	124	607
TCPTestSuite	55	465
ConferenceFact~	74	447
DMCPConnection~	76	421
ConnectionTest~	62	417
DMCPDiscoverer~	60	411
DataStoreTestS~	61	401
AbstractCIDMod~	58	365
UDPTestSuite	43	346
CommandLinePar~	34	343
KeyValueConfig~	35	332

	#vtable	#vcall
TimeFactoryTes~	40	329
SharedPointerT~	31	325
DisposalTestSu~	30	309
TimeStampTestS~	34	306
ConditionTestS~	39	302
ClockTestSuite	27	297
NetstringsProt~	35	290
RunnerTestSuit~	27	290
FalseSerializa~	37	288
ContainerTestS~	36	287
ServiceTestSui~	40	286
SerializationT~	32	279
StringProtocol~	33	275
SharedMemoryTe~	25	273
MutexTestSuite	28	259
TreeNodeTestSu~	25	250

* Most modules have virtual calls and VTables
* The attack surface is large enough for real world attacks.

Conclusion

* VTable hijacking is popular and critical
  * Real-world exploits against COTS applications exist.
  * CPS applications also have a large attack surface.
* Existing solutions are not perfect
* VTint is a lightweight, binary-compatible and effective defense against VTable hijacking, similar to DEP

defense solution	*vtable hijacking*			info leakage	binary support	perf. overhead
	corrupt	inject	reuse			
VTGuard	N	N	Y	N	N	0.5%
SD-vtable	N	Y	Y	N/A	N	30%
SD-method	Y	Y	Y	N/A	N	7%
DieHard	partial	partial	partial	N/A	N	8%
VTint	Y	Y	partial	Y	Y	2%

FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Thanks!

Q&A

Berkeley
UNIVERSITY OF CALIFORNIA

Massachusetts Institute of Technology

NSF

MICHIGAN

VANDERBILT UNIVERSITY