VTint: Protecting Virtual Function Tables' Integrity

Chao Zhang (UC Berkeley)

Chengyu Song (Georgia Tech)

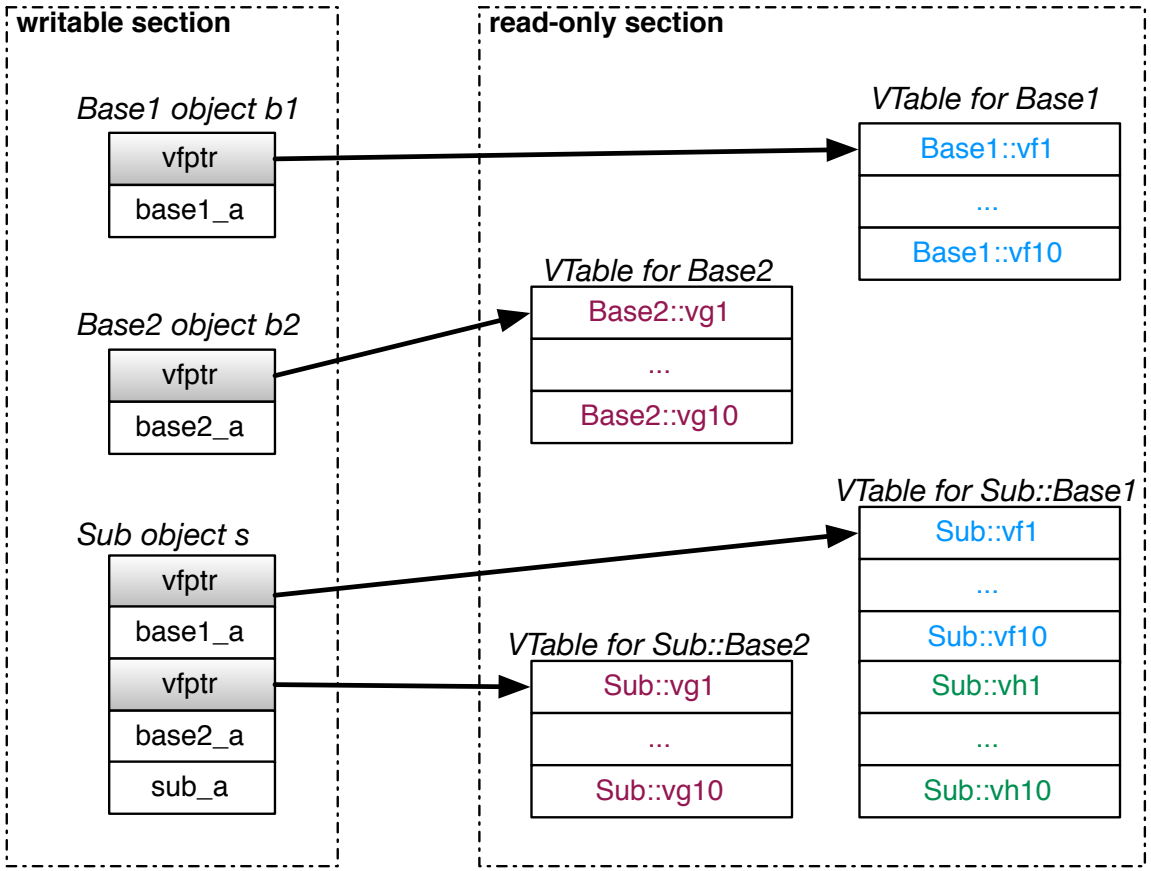
Kevin Zhijie Chen (UC Berkeley)

Zhaofeng Chen (Peking University)

Dawn Song (UC Berkeley)

VTable for Dynamic Dispatch (C++)

class Sub: public Base1, Base2{…};




```
void foo(Base2* obj){
   obj→vg4();
}

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

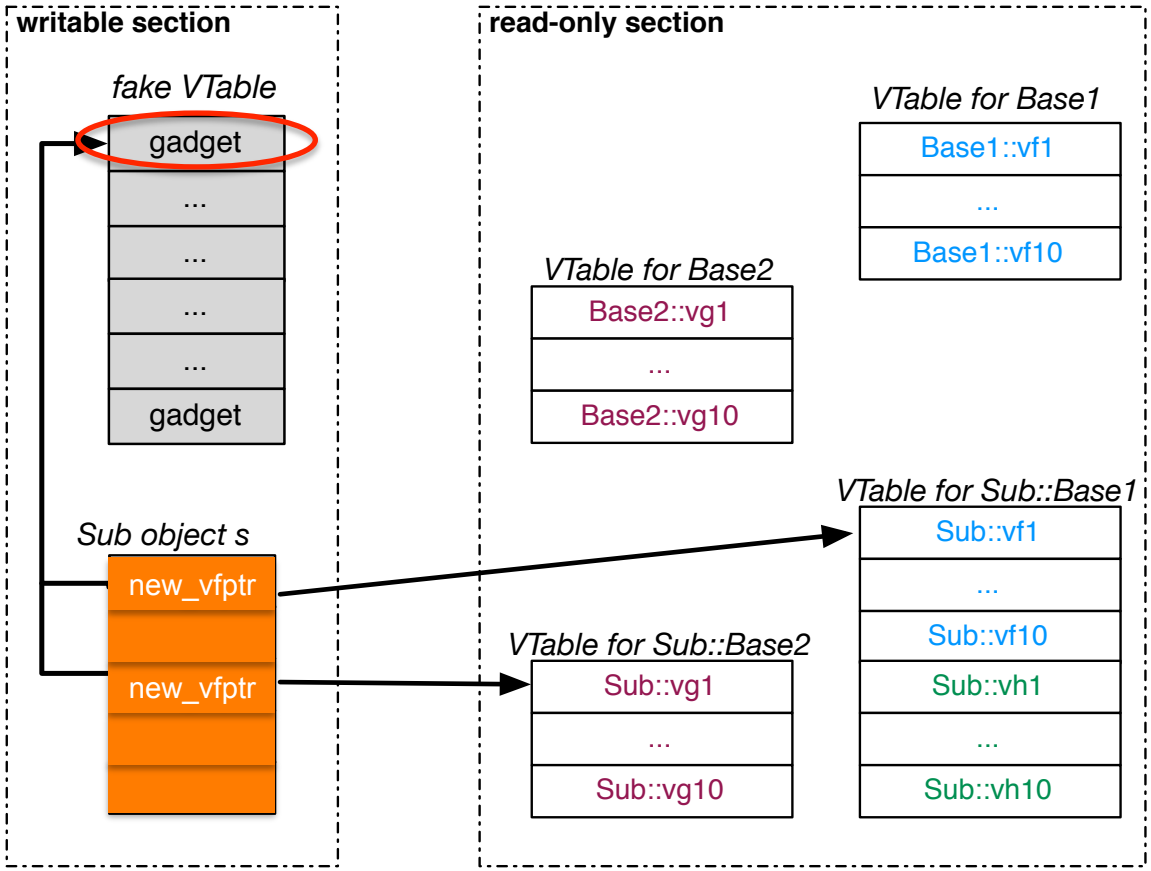
code section

```
; Function main()
push  SIZE
call    malloc()
mov   ecx, eax
call    Sub::Sub()
; now ECX points to the Sub object
add   ecx, 8
; now ECX points to the Sub::Base2 object
call    foo()
ret

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

VTable Hijacking in real world

+ Vulnerabilities like use-after-free
+ VTable Injection
+ ROP gadgets

- Pwn2Own 2014 Firefox
- Pwn2Own 2014 Chrome
- CVE-2014-1772 IE

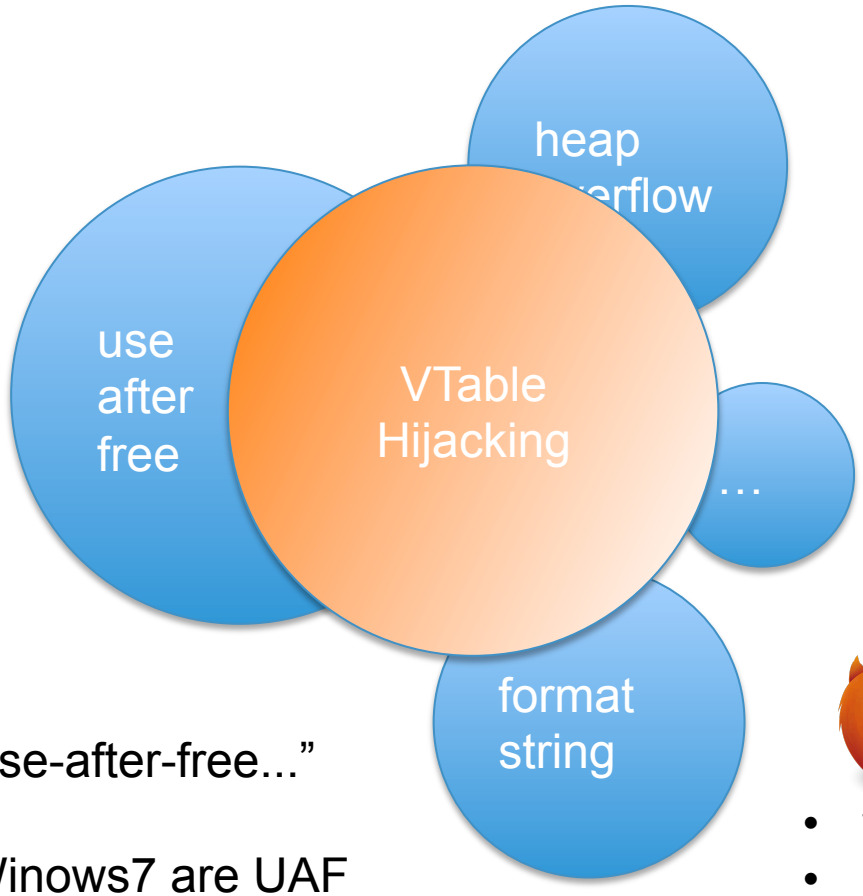


code section

```
; Function main()
push  SIZE
call    malloc()
mov   ecx, eax
call    Sub::Sub()
; now ECX points to the Sub object
add   ecx, 8
; now ECX points to the Sub::Base2 object
call    foo()
ret

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

VTable Hijacking in real world

- A common way to exploit

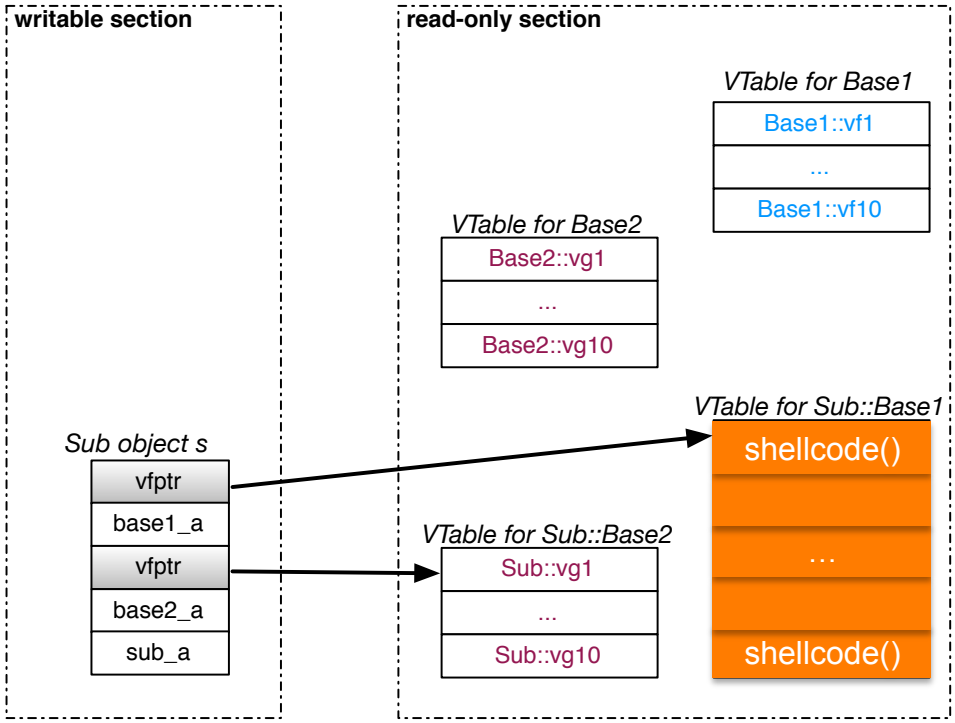


Google:
"80% attacks exploit use-after-free..."
Microsoft:
50% CVEs targeted Winows7 are UAF

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

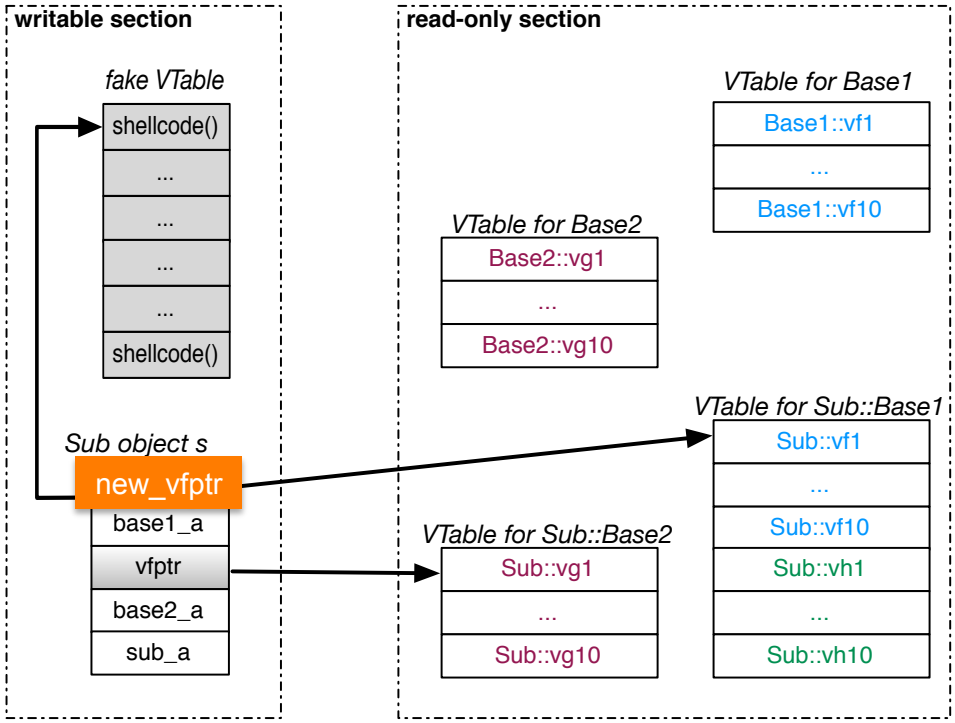
VTable Hijacking Classification

- VTable corruption
  - overwrite VTable
- VTable injection
- VTable reuse



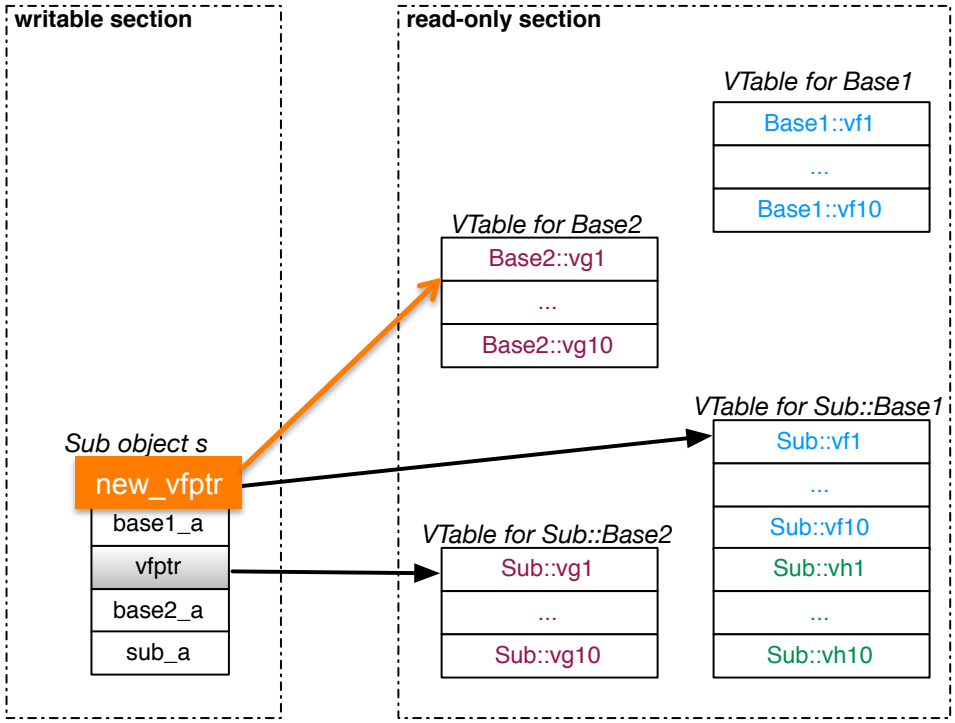
VTable Hijacking Classification

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



VTable Hijacking Classification

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



VTint

- Motivation
- VTint Design
- VTint Implementation
- Evaluation

Our solution: VTint

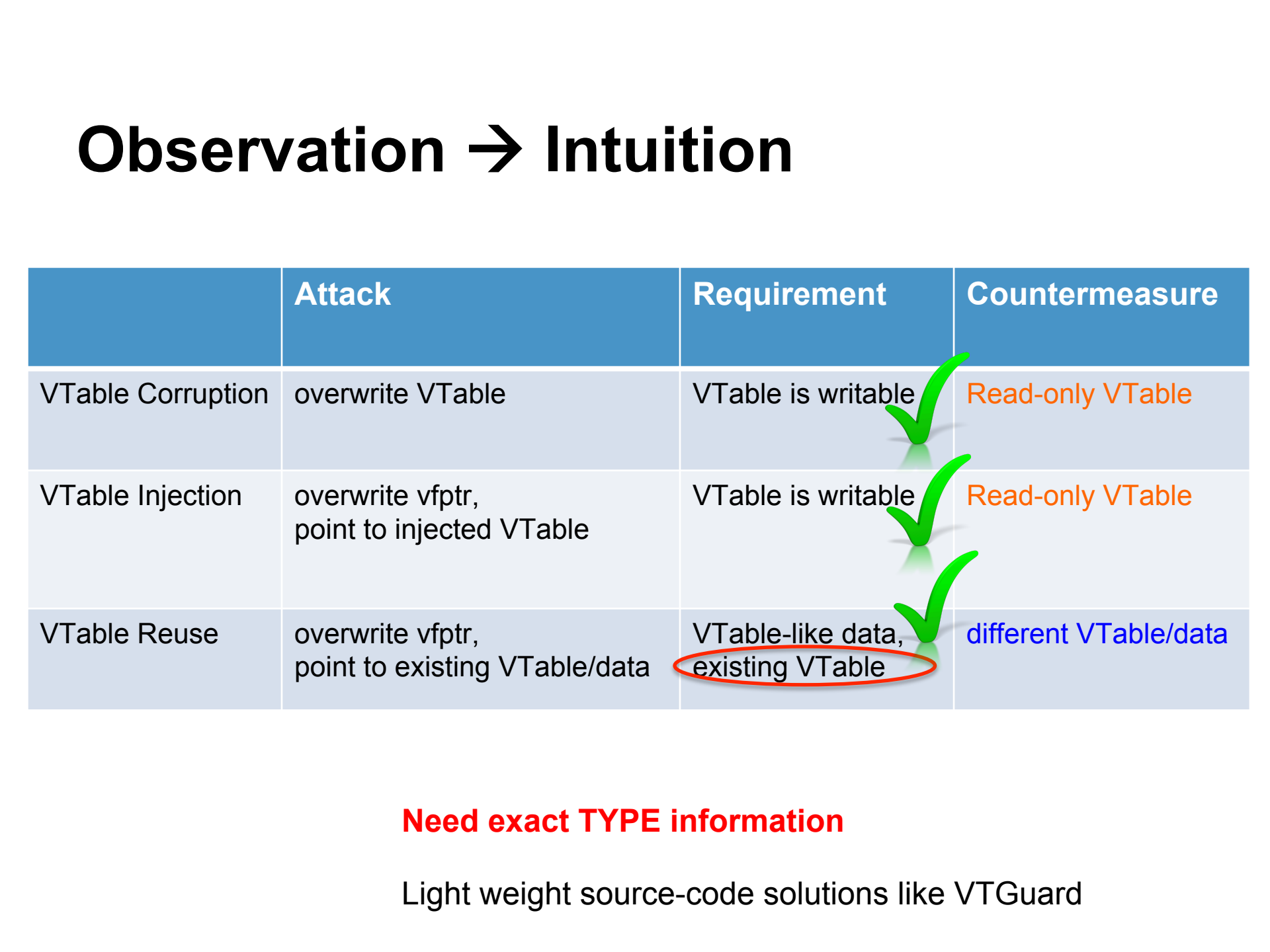
- Goal: VTable Hijacking
  - lightweight
  - binary
  - effective

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

VTint vs. DEP

	VTint
VTable Corruption	Read-only VTable
VTable Injection	Read-only VTable
VTable Reuse	different VTable/data

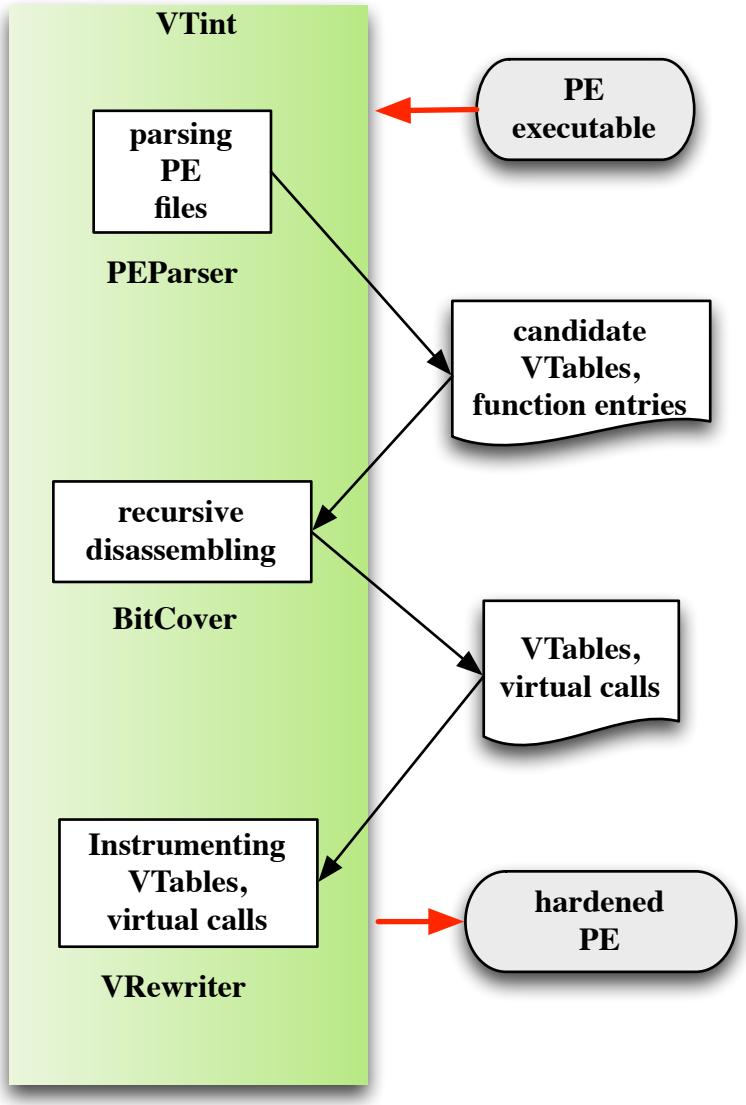
	DEP
Code Corruption	Read-only Code Sec
Code Injection	Read-only Code Sec *(writable sections will not be executed)*
Code Reuse	NO

- Similar to DEP
  - lightweight, and can be binary-compatible
- Different from DEP
  - after hardening, the attack surface is smaller

VTint

- Motivation
- VTint Design
- VTint Implementation
- Evaluation

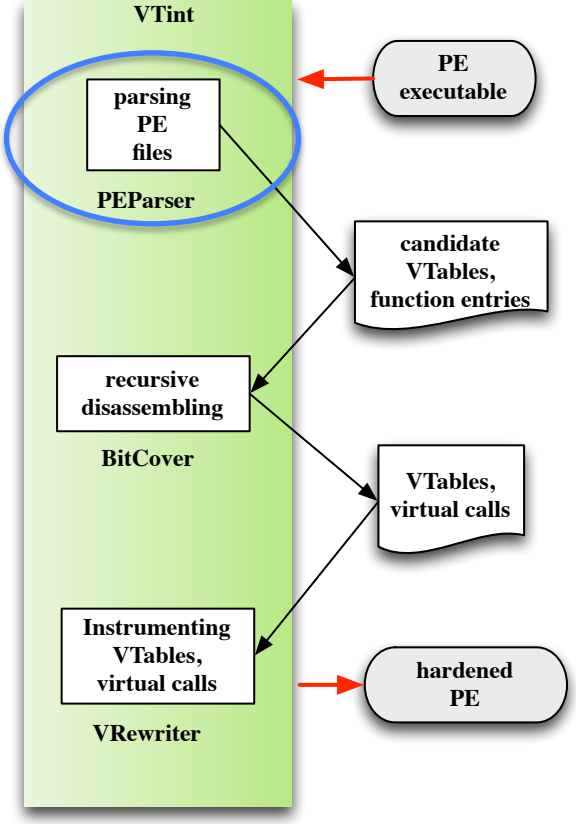
Architecture



- Binary parsing
- Disassembling
- Binary rewriting

Binary Parsing

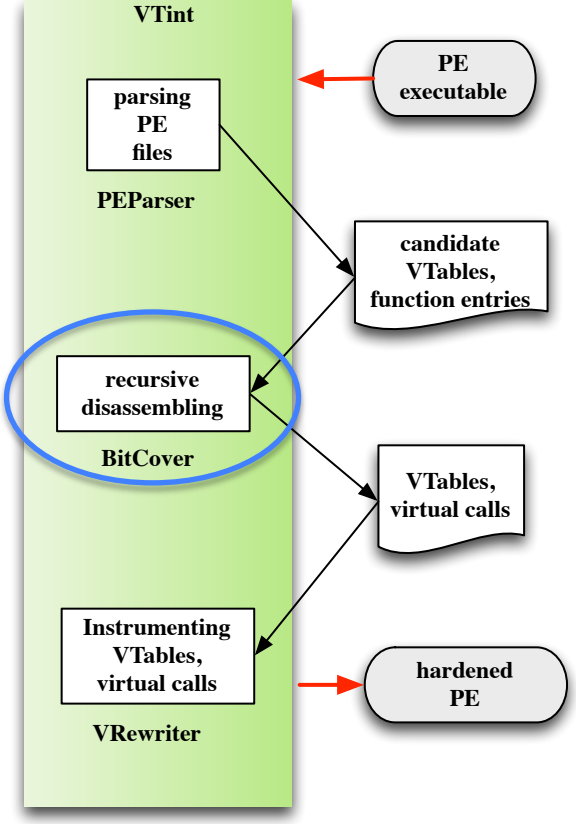
- PE format
  - relocation table
  - import/export table



- Output:
  - candidate function entries
    - relocation entries, export entries, EntryPoint
  - candidate VTables
    - addresses of VTables are in the relocation table
    - entries in VTables are also in the relocation table

Disassembling

- Goal
  - recover CFG
    - find out all functions, instructions
  - recover high-level information
    - constructor functions
    - real VTables
    - virtual function calls



- recursive disassembly
  - starting from candidate function entries
  - targeting normal PE binaries, with relocation table

Disassembling (1)
Identify Constructor Function

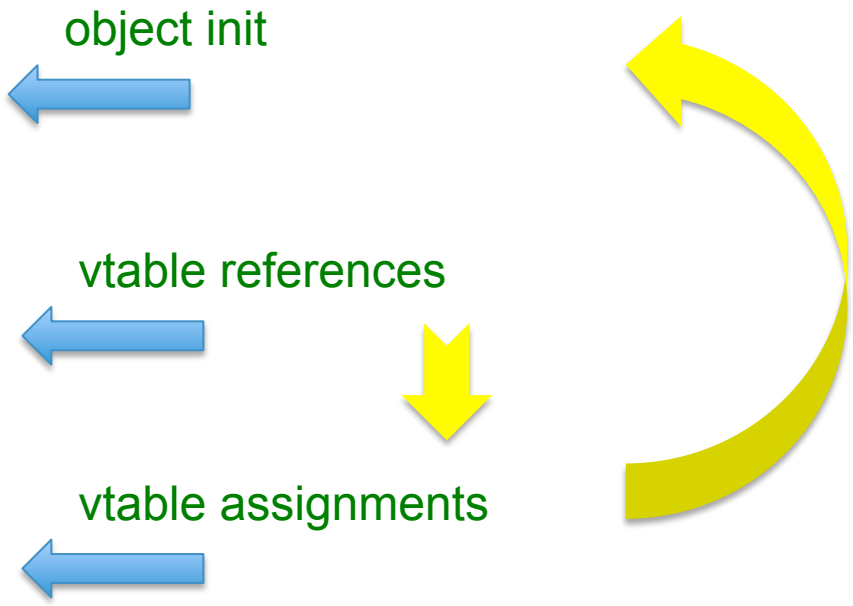
- Basic Pattern

```
; allocate object memory
push SIZE
call   malloc()
mov  ecx, eax
```

```
; get VTable ptr
mov  eax, vfptr
```

```
; assign VTable to object
mov  [ecx], eax
```

- Identification
  - we know candidate vtables



Disassembling (2)
Identify VTables

- Basic Pattern

```
; assign to objects in constructors
mov  [ecx], vfptr
```

find vtable assignments

- Identification
  - we know candidate vtables

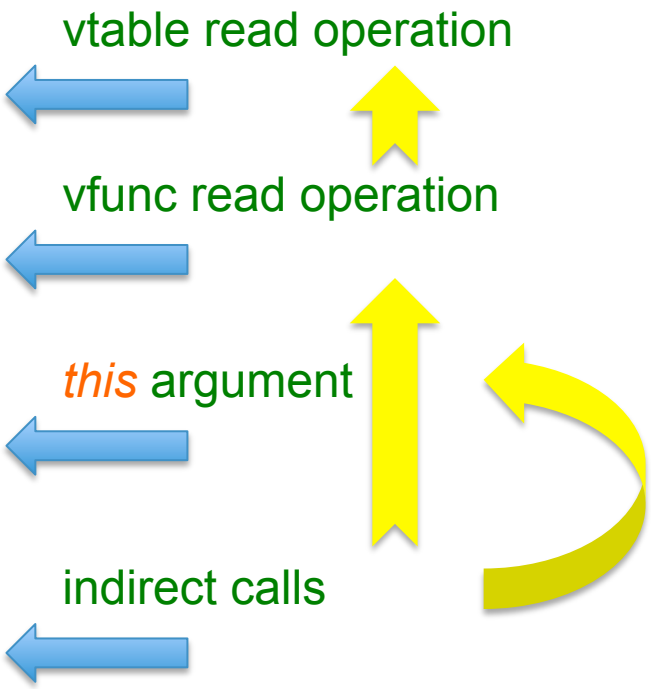
- VTable size
  - unable to get exact size in binaries
  - we can estimate the maximum size
    - continuous relocation entries
    - adjacent RTTI, this adjustors, base offsets

Disassembling (3)
Identify Virtual Function Calls

- Basic Pattern

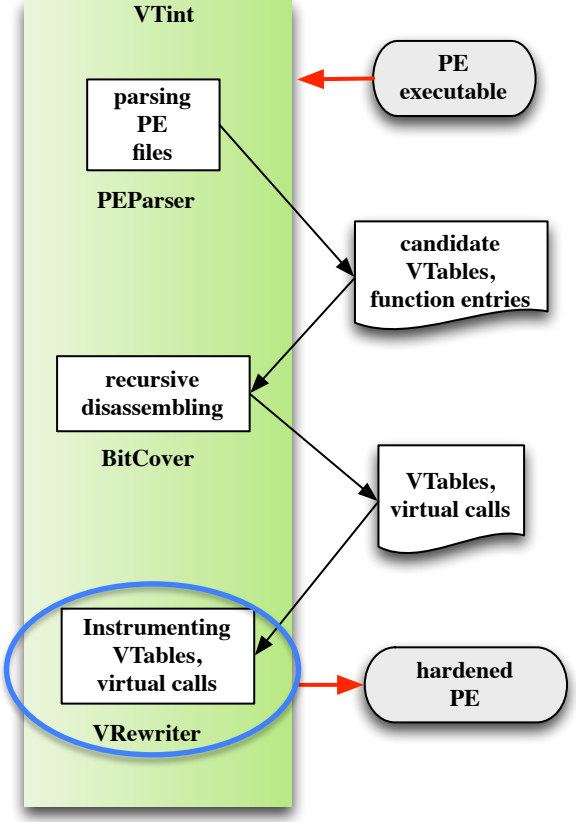
```
; get vtable ptr from object
mov  eax, [ecx+8]

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

; prepare this ptr for callee
add   ecx, 8

; call virtual function
call   edx
```

- Idenfication
  - we know indirect calls



Binary Rewriting

- Security Policy
  - Enforce 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
```

Info Leakage?

No problem!

VTint

- Motivation
- VTint Solution
- VTint Implementation
- Evaluation

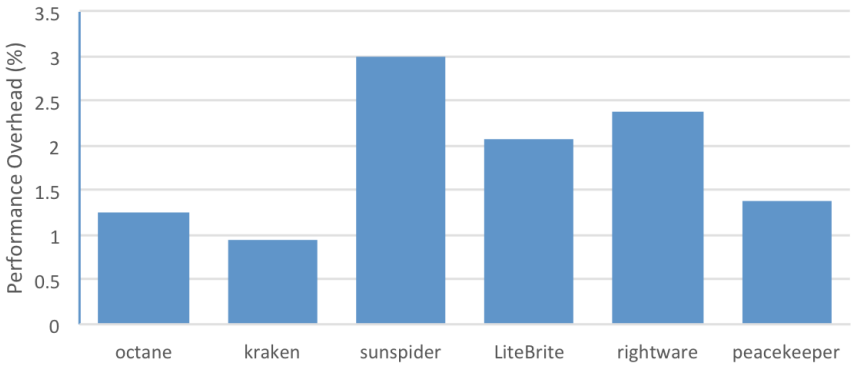
Static Analysis Results

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

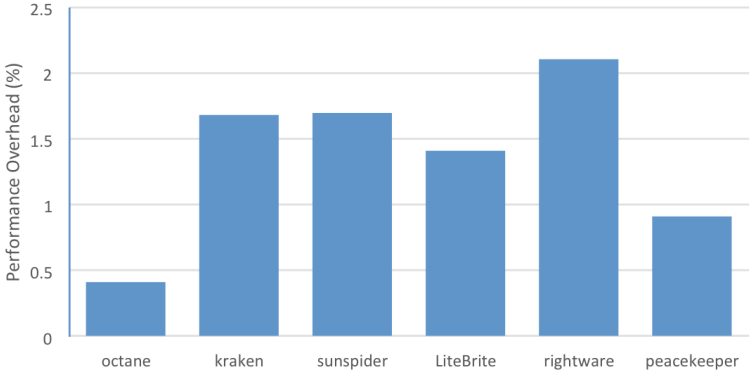
App	analysis time (sec)	file size (KB)			VTable info		
		orig	new	size overhead	#inst	#vtables	#vcalls
crashreporter.exe	1.8	116	117	0.52%	18,461	3	15
updater.exe	3.7	271	276	1.77%	112,693	9	17
webapprt-stub.exe	1.6	96	97	0.61%	38,589	2	17
D3DCompiler_43.dll	74.3	2,106	2,202	4.53%	2,135,041	48	1338
d3dx9_43.dll	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%

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

Conclusion

- VTable hijacking is popular and critical
- 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%

Thanks!