Confusion de type en C++

Florent Saudel

10 juin 2016

AMOSSYS

Qui suis-je ?

Florent Saudel

Doctorant chez AMOSSYS

Recherche de vulnérabilités, *reverse*, analyse de binaire ...

Introduction

Plan

Pourquoi s'y intéresser ?

Description de la vulnérabilité

Comment l'exploiter ?

Conclusion

Pourquoi s'y intéresser ?

Les dernières CVE

Année	Logiciel	CVE
2016 (6)	Microsoft Edge	0003 (RCE), 0060 (MD)
	Microsoft Internet Explorer	0061 (RCE), 0112 (RCE)
	Adobe Flash AS2	1015 (RCE)
	Apple Safari	1778 (RCE)
2015 (10)	Microsoft JScript	6134 (RCE)
	Microsoft VBScript	6055 (RCE)
	Microsoft Internet Explorer	2443 (RCE), 2448 (RCE)...
	Adobe Flash AS2	7659 (RCE), 8439 (RCE)
	Google Chrome	1230 (RCE)

Remote Code Execution (RCE) : exécution de code arbitraire à distance
Memory Disclosure (MD) : fuite d'information sur la mémoire du processus

Pwn2Own 2013 - CVE-2013-0912 [3]

1 bug = 1 MD + 1 RCE

```
SVGElement* SVGViewSpec::viewTarget() const
{
    if (!m_contextElement)
        return 0;
    return static_cast< SVGElement* >(
        m_contextElement->treeScope()
                        -> getElementById (m_viewTargetString));
}
```

Element*

258 classes filles

Description de la vulnérabilité

L'origine

- « *bad cast* »
- 3 opérateurs : `static_cast`, `reinterpret_cast` et `dynamic_cast`

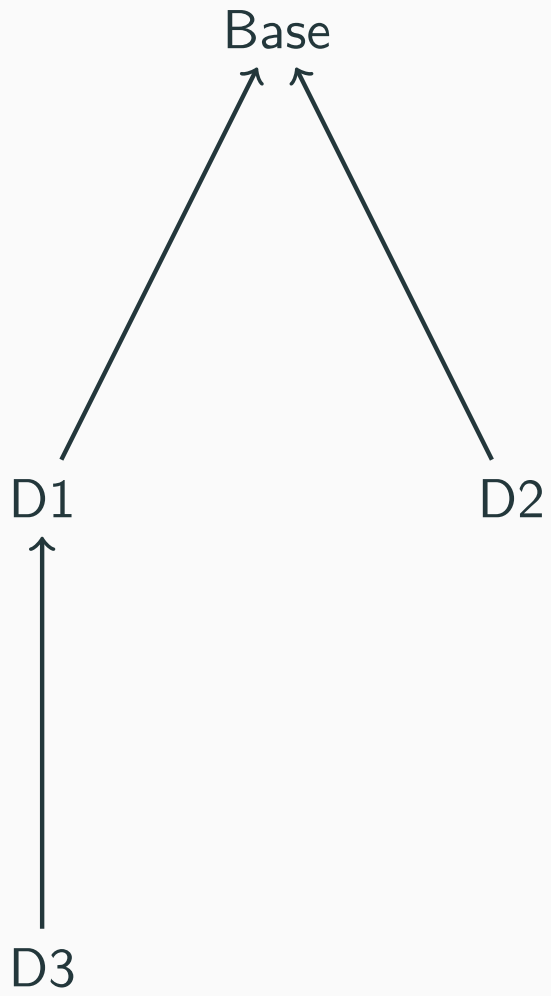
Exemple

```
int i = 12000;
void *p = reinterpret_cast<void *>(i);

struct B {};
struct D : B {};

B *b = new D(); // upcast implicite
D *another_d = static_cast<D *>(b); // downcast

D *dyn_another_d = dynamic_cast<D *>(b); // downcast
```

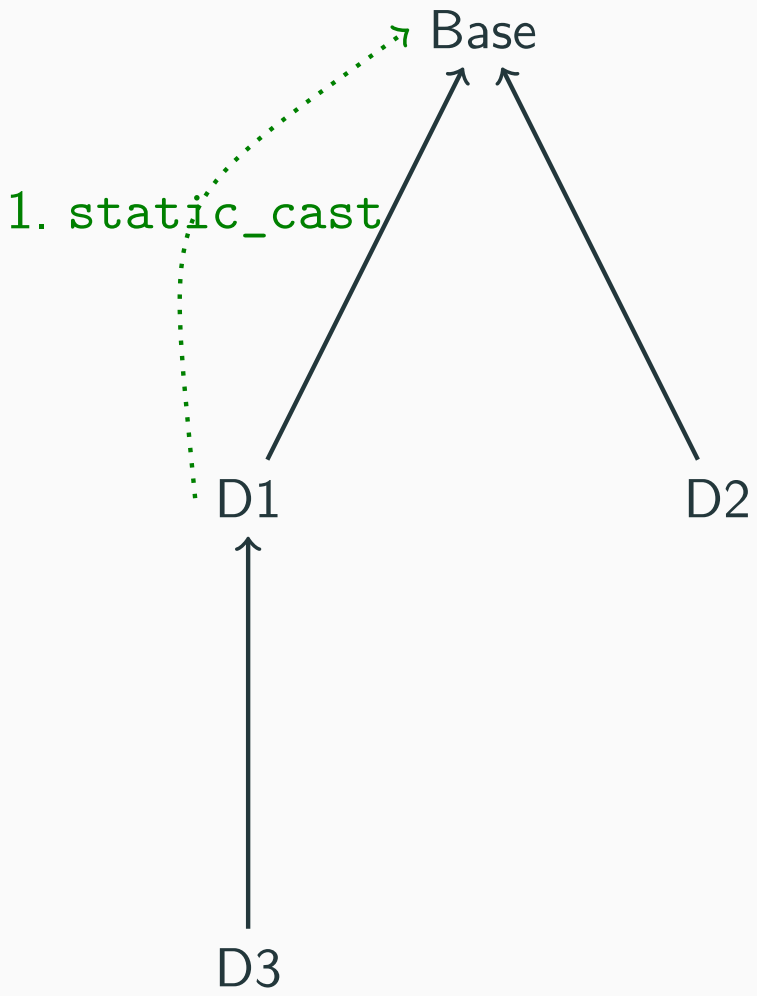
Les opérateurs

Opérateurs	Vérification(s)	Ralentissement
`reinterpret_cast`	aucune	0
`static_cast`	compilation	0
`dynamic_cast`	compilation et exécution	++

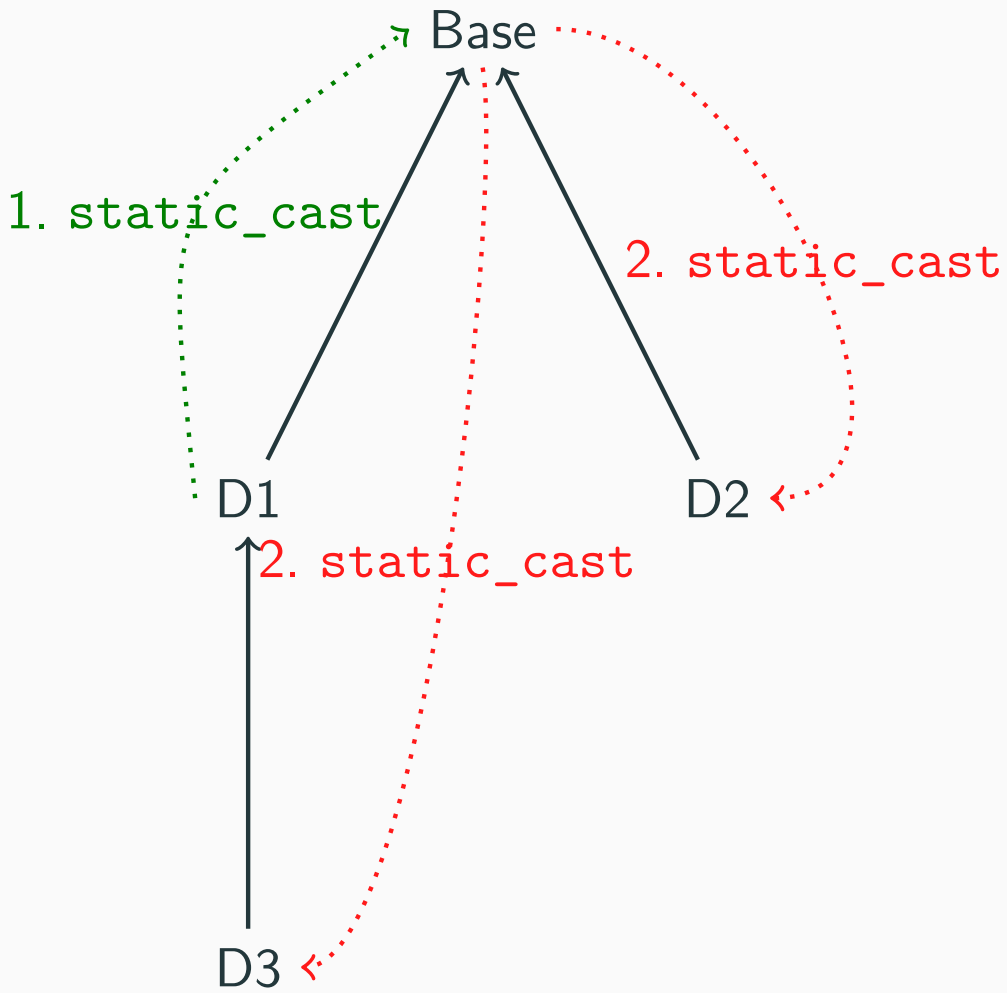
Le « *bad downcast* »



Le « *bad downcast* »

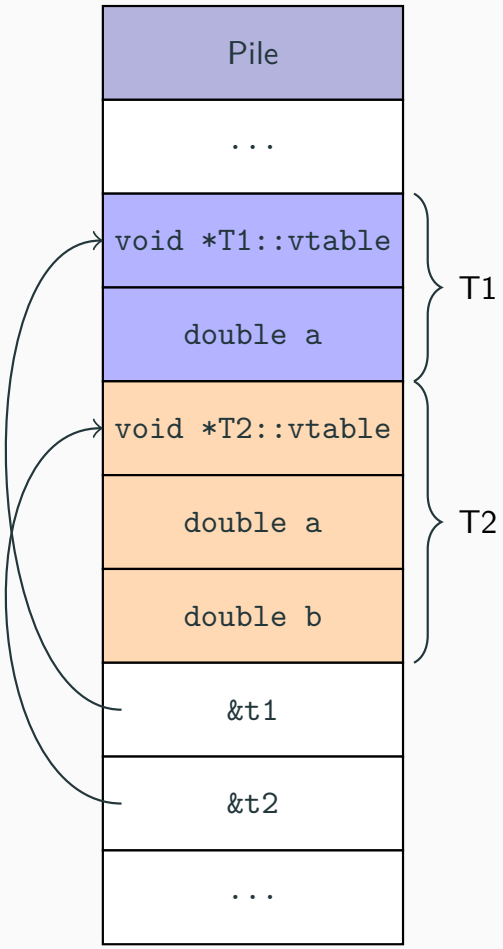


Le « *bad downcast* »



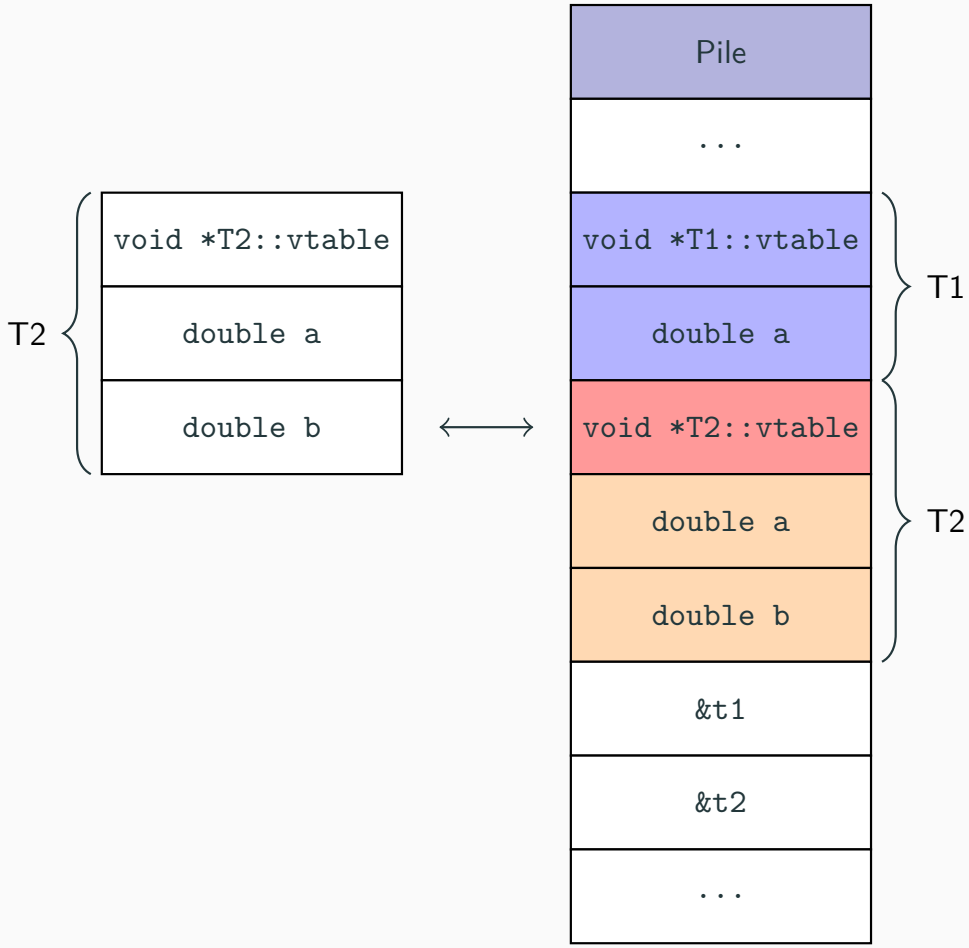
Comment l'exploiter ?

Première et seconde primitives : lecture et écriture en mémoire



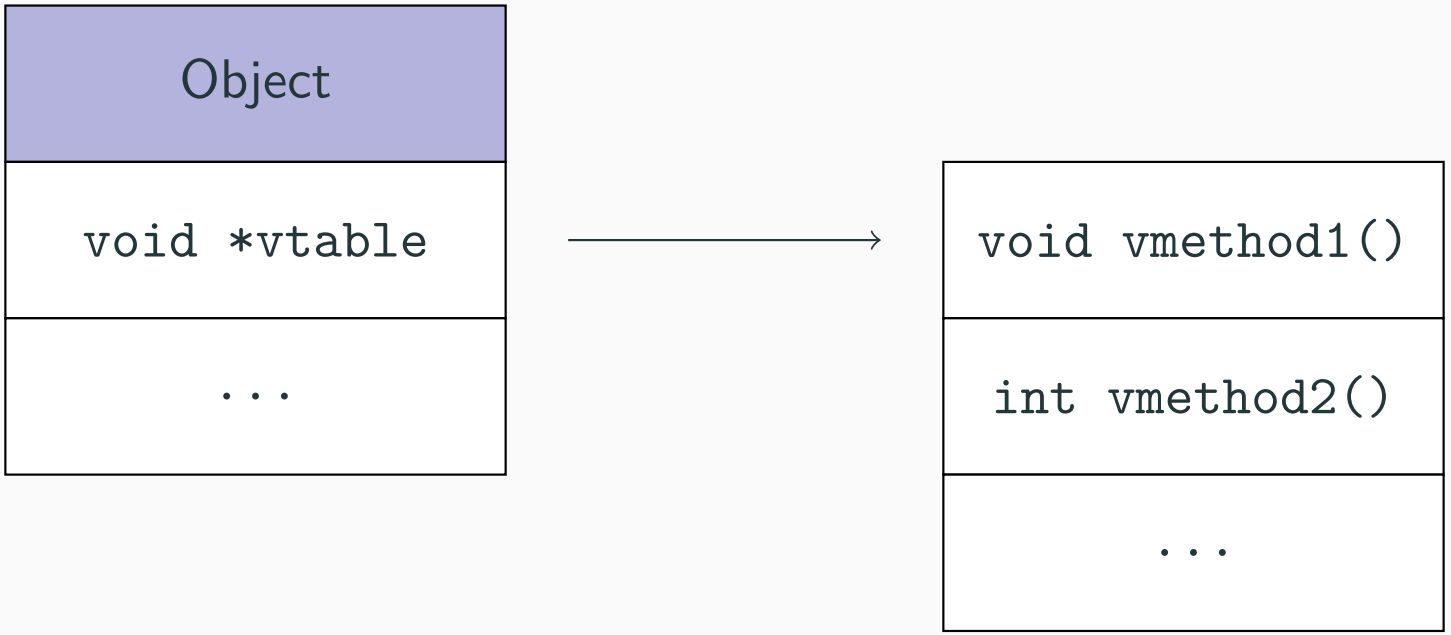

```
struct I {
    virtual void foo() = 0;
};

struct T1 : public I {
    void foo() { cout << "T1\n"; }
    double a = 1;
};

struct T2 : public I {
    void foo() { cout << "T2\n"; }
    double a = 2;
    double b = 3;
};

T1 t1; T2 t2;
I *a[2] = {&t1, &t2};

T2 *pt2 = static_cast<T2 *>(a);
pt2[0]->b;
```

Première et seconde primitives : lecture et écriture en mémoire




```
struct I {
    virtual void foo() = 0;
};

struct T1 : public I {
    void foo() { cout << "T1\n"; }
    double a = 1;
};

struct T2 : public I {
    void foo() { cout << "T2\n"; }
    double a = 2;
    double b = 3;
};

T1 t1; T2 t2;
I *a[2] = {&t1, &t2};

T2 *pt2 = static_cast<T2 *>(a);
pt2[0]->b;
```

Méthode virtuelle et vtable

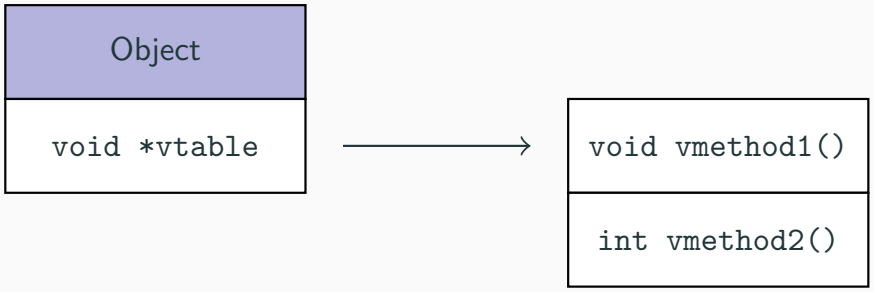


Appel de méthode virtuelle

```
mov     rax,QWORD PTR [object]
mov     rax,QWORD PTR [rax]
mov     rax,QWORD PTR [rax+offset]
mov     rdx,QWORD PTR [object]
mov     rdi,rdx
call    rax
```

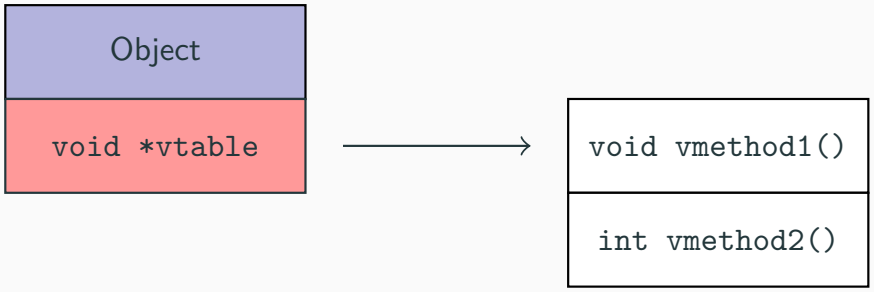
```
rax = this;
rax = *(this) = vtable;
rax = vtable[offset] = method;
rdx = this;
rdi = this;
this->method();
```

Appel de méthode virtuelle (II)



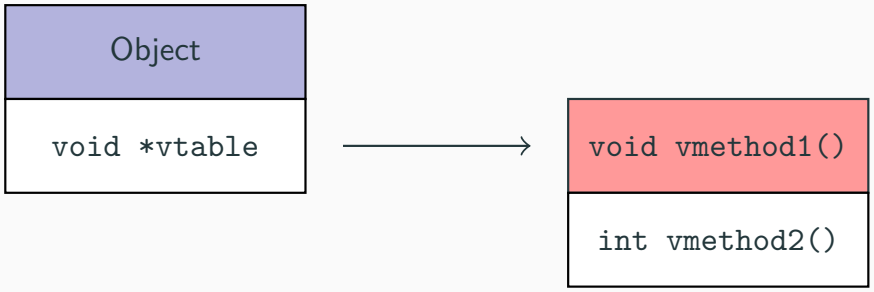

```
rax = this;
rax = *(this) = vtable;
rax = vtable[offset] = method;
rdx = this;
rdi = this;
this->method();
```

Appel de méthode virtuelle (II)



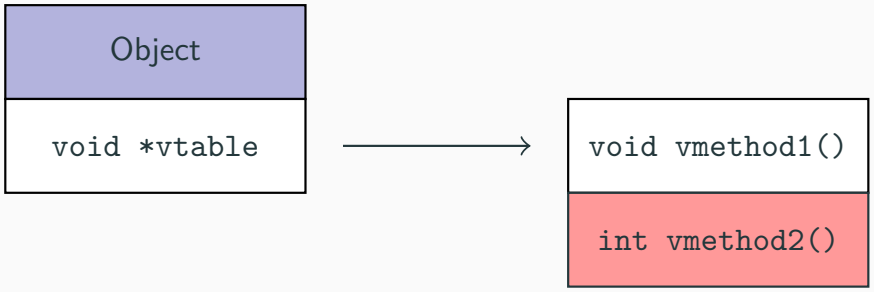

```
rax = this;
rax = *(this) = vtable;
rax = vtable[offset] = method;
rdx = this;
rdi = this;
this->method();
```

Appel de méthode virtuelle (II)



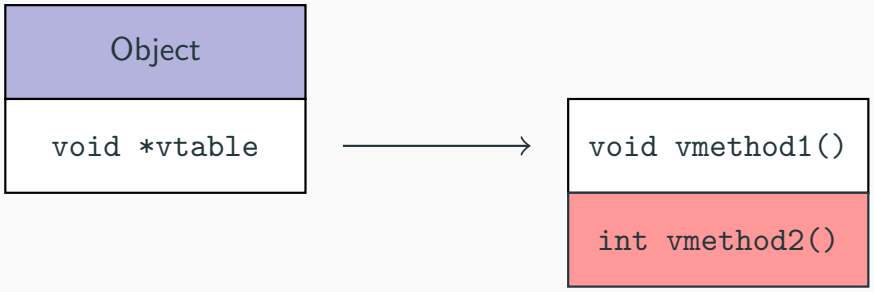

```
rax = this;
rax = *(this) = vtable;
rax = vtable[offset] = method;
rdx = this;
rdi = this;
this->method();
```

Appel de méthode virtuelle (II)




```
rax = this;
rax = *(this) = vtable;
rax = vtable[offset] = method;
rdx = this;
rdi = this;
this->method();
```

Appel de méthode virtuelle (II)

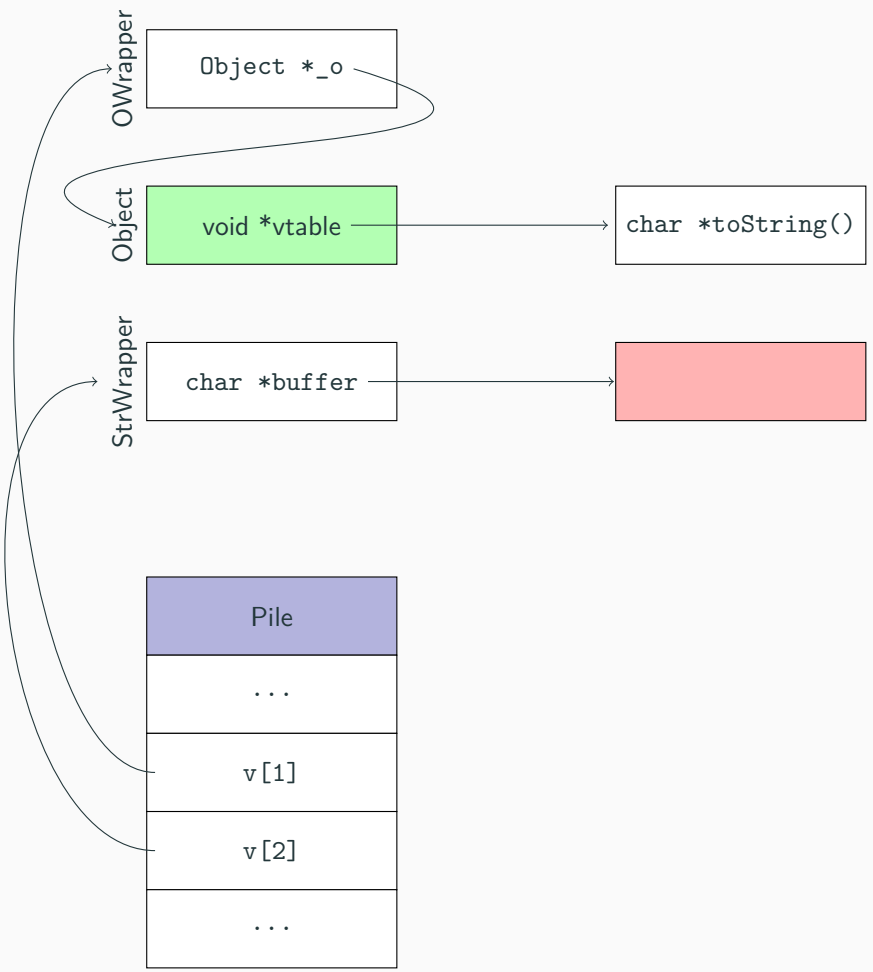



```
rax = this;
rax = *(this) = vtable;
rax = vtable[offset] = method;
rdx = this;
rdi = this;
this->method();
```

Troisième primitive : détournement de flot de contrôle

```
struct Object {
    virtual char *toString() { /*...*/ }
};

class Wrapper { };
struct OWrapper : public Wrapper {
        char *toString() {
            return _o.toString();
        }
        Object _o = new Object();
};
struct StrWrapper : public Wrapper {
    char *str = "AAAAAAAA";
};

Wrapper *v[2] = {new OWrapper,
    new StrWrapper};

for (int i = 0; i < 2; i++) {
    static_cast<OWrapper *>(v[i])
        ->toString();
}
```

Troisième primitive : détournement de flot de contrôle




```
struct Object {
    virtual char *toString() { /*...*/ }
};

class Wrapper { };
struct OWrapper : public Wrapper {
        char *toString() {
            return _o.toString();
        }
        Object _o = new Object();
};
struct StrWrapper : public Wrapper {
    char *str = "AAAAAAAA";
};

Wrapper *v[2] = {new OWrapper,
    new StrWrapper};

for (int i = 0; i < 2; i++) {
    static_cast<OWrapper *>(v[i])
        ->toString();
}
```

Conclusion

Une classe de programmes plus « vulnérable » ?

Classe

Navigateurs et interpréteurs

Arguments

- Hiérarchie de classes importante.
  WebKit `WebCore::Node` possède près de 300 classes filles.
- Langage de script

Comment détecter cette vulnérabilité ?

« ***Sanitizers*** »

- UBsan [4] (Clang puis GCC)
- CaVeR [6] (basé sur un « *fork* » de Clang)

Comment détecter cette vulnérabilité ?

« *Sanitizers* »

- UBsan [4] (Clang puis GCC)
- CaVeR [6] (basé sur un « *fork* » de Clang)

Fonctionnement

Transforme à l'aide d'une passe au sein du compilateur les `static_cast` en `dynamic_cast`.

Limitations

- dynamique $\Rightarrow$ couverture
- compilation $\Rightarrow$ code source

Références

- Issue 180763 - chromium - PWN2own : Bad cast in SVGViewSpec : :viewTarget - Monorail.
- Itanium C++ ABI.
- MWR Labs Pwn2own 2013 Write-up - Webkit Exploit - mwrlabs.
- UndefinedBehaviorSanitizer (UBSan) - The Chromium Projects.
- Chris Rohlf : **C++ - AppSec 2014, 2014.**
- B. Lee, C. Song, T. Kim et W. Lee : **Type Casting Verification : Stopping an Emerging Attack Vector.** p. 81–96, 2015.
- Natalie Silvanovich : **Project Zero : One Perfect Bug : Exploiting Type Confusion in Flash.**