Thomas Caplin
Sogeti / ESEC R&D
thomas@security-labs.org

Silverlight ou comment surfer à travers .NET

Mercredi 8 juin 2011

SOGETI

Introduction

De quoi allons-nous parler ?

- À partir d'une vieille (août 2010) faille du Framework .NET :
  - Comment exploiter une vulnérabilité de ce type
  - Avec à la clé un contournement des protections Windows
- Le plugin Silverlight offre une voie d'attaque bien plus fun !

Travaux précédents

- SSTIC 2010 : Audit d'applications .NET (Nicolas Ruff, EADS)
- Jeroen Frijters (IKVM) montra une technique d'exploitation .NET
- Il publia aussi un article sur la vulnérabilité présentée ici

Organisation

- Nous expliquerons en détails comment l'exécution de code est possible
- Et pourquoi les protections Windows sont outre-passées
- Nous verrons qu'il n'y a pas que Flash Player qui peut être dangereux pour vos navigateurs Web !

Plan 1/3

Introduction à Microsoft .NET

- Le Framework .NET
- Mécanismes internes

.NET : un forfait pour plein de pistes !

- Architecture du Framework .NET
- Les couches du Framework

Plan 2/3

Analyse d'une vulnérabilité .NET

- Présentation de la vulnérabilité
- Détails et dangers de cette vulnérabilité
- Preuve de concept : confusion de type

Exploitation d'une vulnérabilité .NET

- Technique d'exploitation
- Contournement des protections Windows : ASLR, DEP

Plan 3/3

Silverlight : une piste en or !

- Qu'est-ce que Silverlight ?
- Pourquoi c'est plus intéressant d'attaquer via Silverlight ?
- Résultats d'une attaque

Qu'est-ce que Microsoft .NET

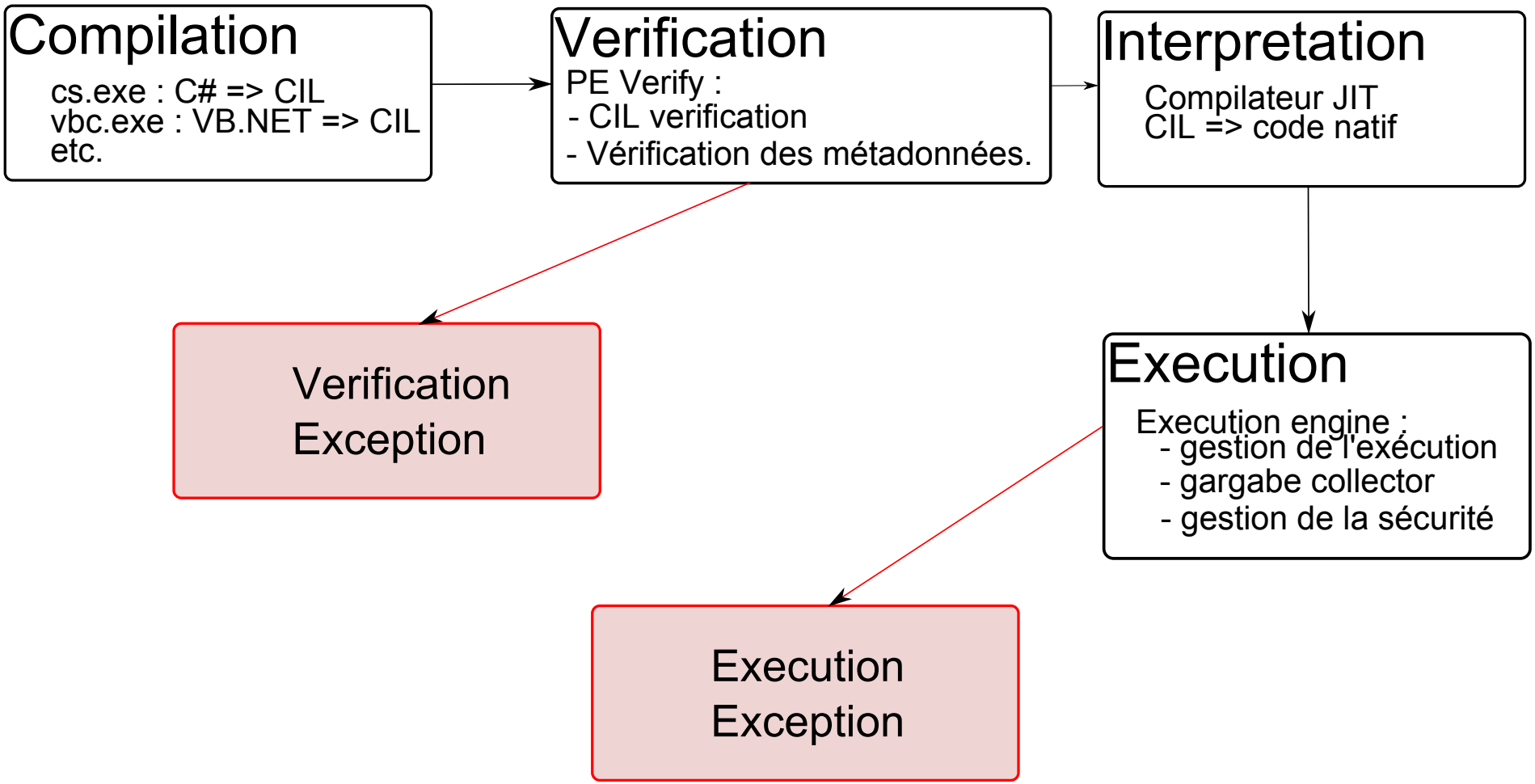
Le Framework .NET

- Créé en 2000 par Microsoft et proposant un nouveau langage : le C#
- Machine virtuelle basée sur la *Common Language Infrastructure* (CLI)
- Ensemble de bibliothèques de développement

Architecture

- Code source compilé dans le *Common Intermediate Language* (CIL)
- CIL exécuté par la VM .NET : *Common Language Runtime* (CLR)
- Une application .NET s'exécute dans un environnement d'exécution managé :
  - Gestion des permissions
  - Gestion du démarrage et du déroulement de l'application
  - *Garbage collector*

Mécanismes internes



Le vérifieur de code intermédiaire : PE Verifier

Caractéristiques

- Embarqué dans la VM .NET
- Empêche des attaques au niveau du bytecode
- Agit avant l'exécution de la classe

Vérifications

- Format du bytecode
- Paramètres des méthodes : type, nombre, placement
- Valeur de retour des méthodes
- Utilisation des pointeurs

Attaque sur le PE Verifier 1/3

Manipulation des paramètres d'une méthode

- Arguments et variables locales manipulés via la pile
- Identifiés par un index

Code C# valide

```
public void foo() {
int a, b;

a = 1712;
b = a;
}
```

Attaque sur le PE Verifier 2/3

Modification du code CIL généré

Code CIL généré :

```
.maxstack 2
.locals init (
[0] int32 a, [1] int32 b)
ldc.i4 0x6B0
stloc.0
ldloc.0
stloc.1
```

Code CIL corrompu :

```
.maxstack 2
.locals init (
[0] int32 a, [1] int32 b)
ldc.i4 0x6B0
stloc.0
ldloc.3
stloc.1
```

L'attaque

- Nous corrompons le CIL en modifiant un index : `ldloc.0` par `ldloc.3`
- Le but étant de lire à un endroit de la pile où nous ne sommes pas supposés pouvoir accéder

Attaque sur le PE Verifier 3/3

Pe Verifier failed

Nous utilisons l'outil `peVerify.exe` pour vérifier la validité de notre code.

```
Microsoft (R) .NET Framework PE Verifier. Version 3.5.21022.8
Copyright (c) Microsoft Corporation. All rights reserved.

[IL]: Error: [C:\...\attaque1.exe:
attaque1.Program::vuln][offset 0x00000008]
Unrecognized local variable number.
1 Error Verifying attaque1.exe
```

Le code CIL ne passe pas le vérifieur !

- Le vérifieur de bytecode détecte qu'un index n'est pas correct !
- L'application ne sera jamais exécutée

.NET : un forfait pour plein de pistes !

SOGETI

Les différentes couches du Framework .NET

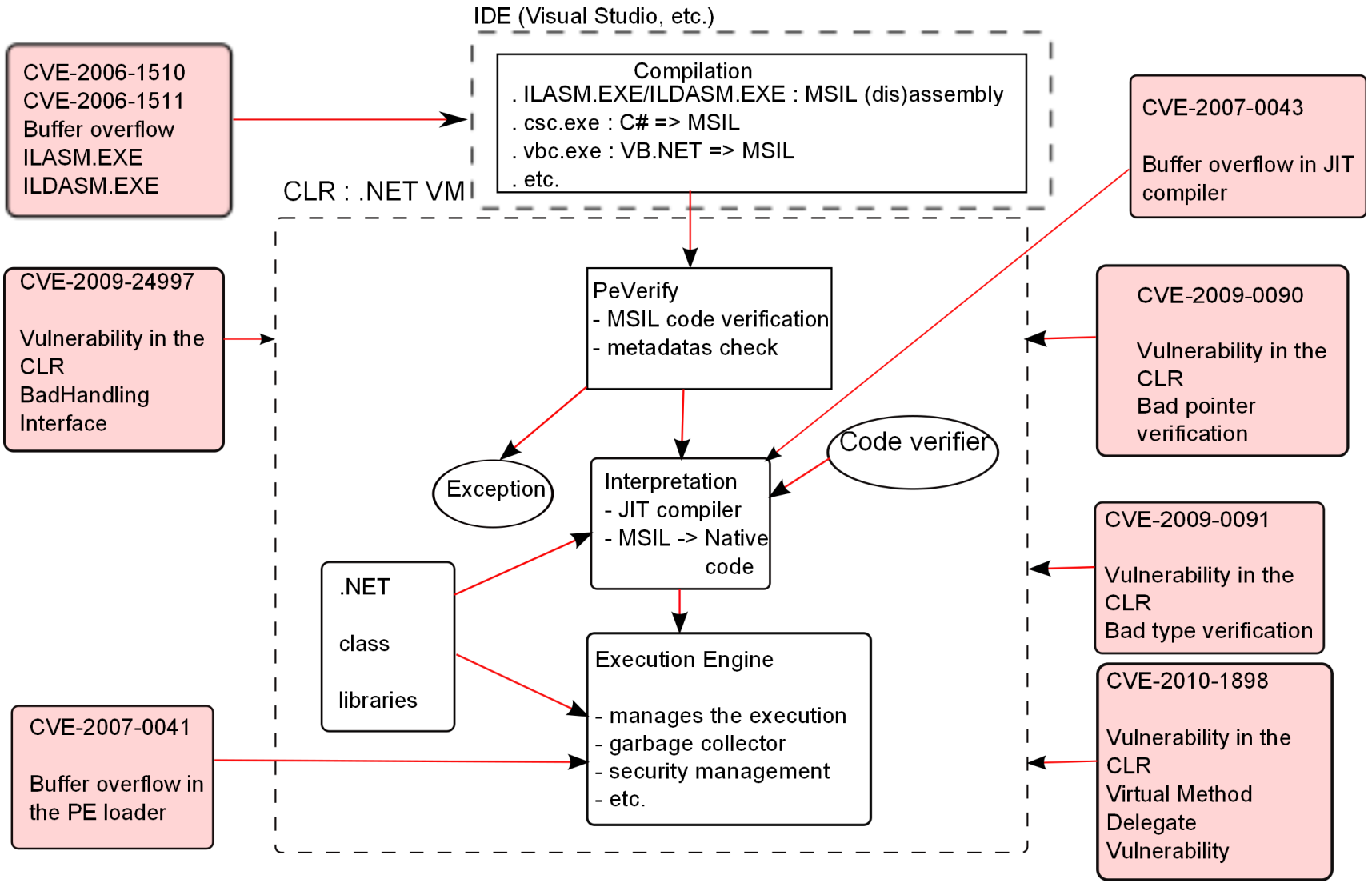
La couche native

- Le Framework repose sur des DLLs Windows natives
- `gdiplus.dll` et `gdi32.dll` pour la gestion graphique
- `winnm.dll` pour la gestion du son
- Une vulnérabilité dans ces bibliothèques pourrait compromettre le Framework

Le Framework

- Le coeur du Framework .NET
- CLR : la machine virtuelle
  - PE Verifier
  - Interpreter : compilateur JIT
  - Execution engine

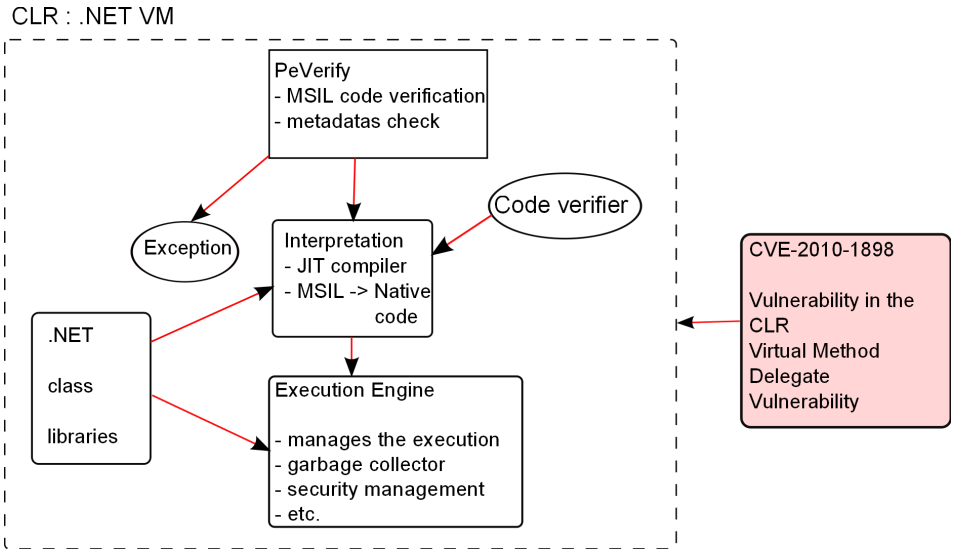
Le Framework



Attention, une crevasse !

Analyse d'une vulnérabilité .NET

La vulnérabilité



Détails & impact

- Trouvée par Eamon Nerbonne, corrigée par MS en août 2010
- Le CLR .NET compromis
- Toutes les applications .NET sont touchées
- Exécution de code arbitraire possible

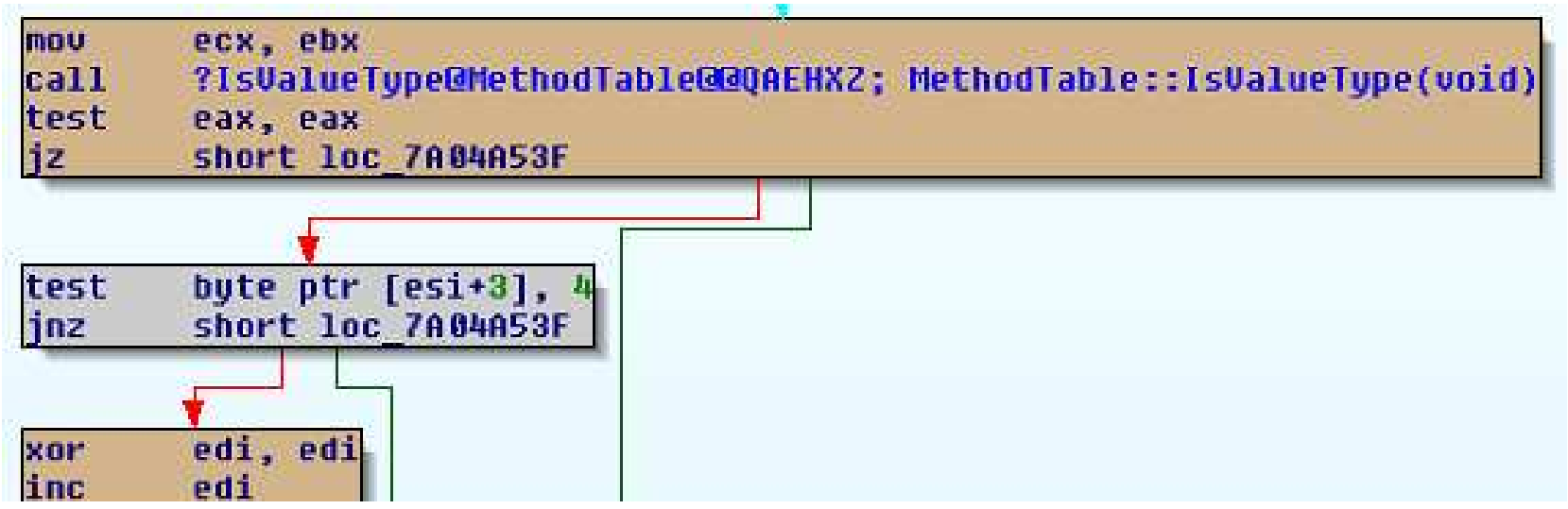
Du patch à la vulnérabilité 1/2

Analyse du correctif Microsoft

- Le bug est dans `mscorwks.dll`
- Cette DLL est le coeur de la VM
- Le bug résulte d'une mauvaise gestion des *delegates* sur les méthodes virtuelles
- Un *delegate* est une sorte de pointeur de fonction en .NET

ComDelegate : :BindToMethodInfo

Un seul bloc d'instructions supprimé par le correctif.



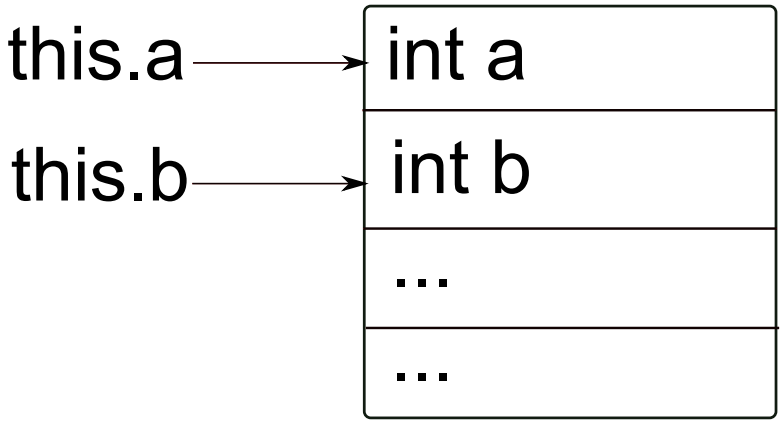
Du patch à la vulnérabilité 2/2

Nous avons analysé le bug aussi avec le code source du Framework disponible (ROTOR).

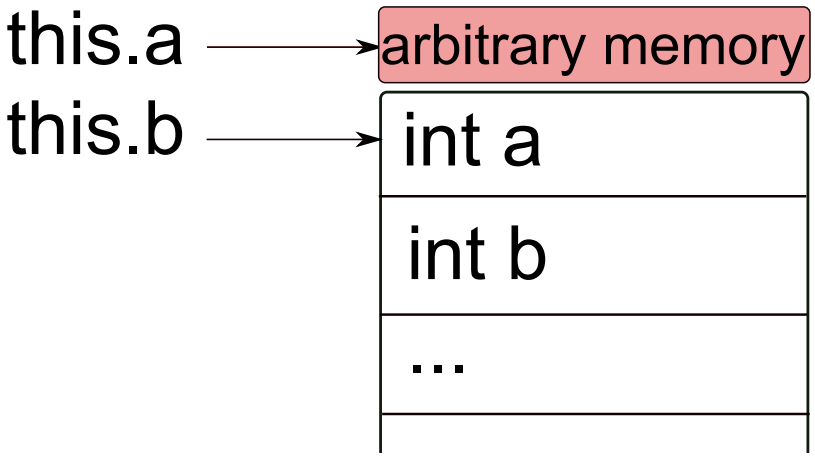
Conclusions

- En .NET, avec les `struct` méthodes, le pointeur **this** doit être ajusté de 4 octets
- Sauf dans le cas où la méthode est virtuelle
- C'est ici qu'est le bug : le compilateur JIT ajuste le pointeur **this** alors que ce n'est pas nécessaire

Le bug



Au niveau de l'appel à une méthode virtuelle depuis un delegate

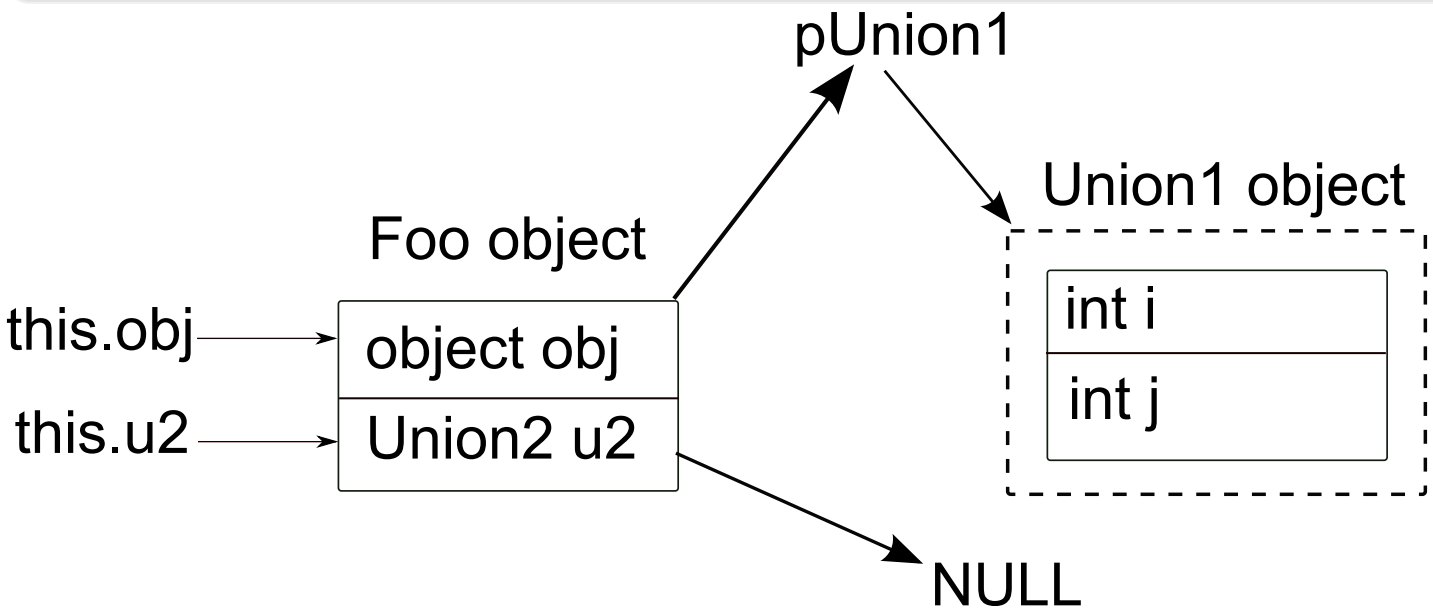


Confusion de type 1/3

Création du delegate : avant l'exploitation de la vulnérabilité

La création du *delegate* génère un objet *Foo* avec un pointeur *Union1*.

```
public struct Foo { object obj; Union2 u2;
  public Foo(object obj){
    this.obj = obj;
    this.u2 = null; }
...
MethodInfo method = typeof(Foo).GetMethod("ToString");
MyDelegate d = (MyDelegate)Delegate.CreateDelegate(typeof(MyDelegate),
               new Foo(u1),method);
```

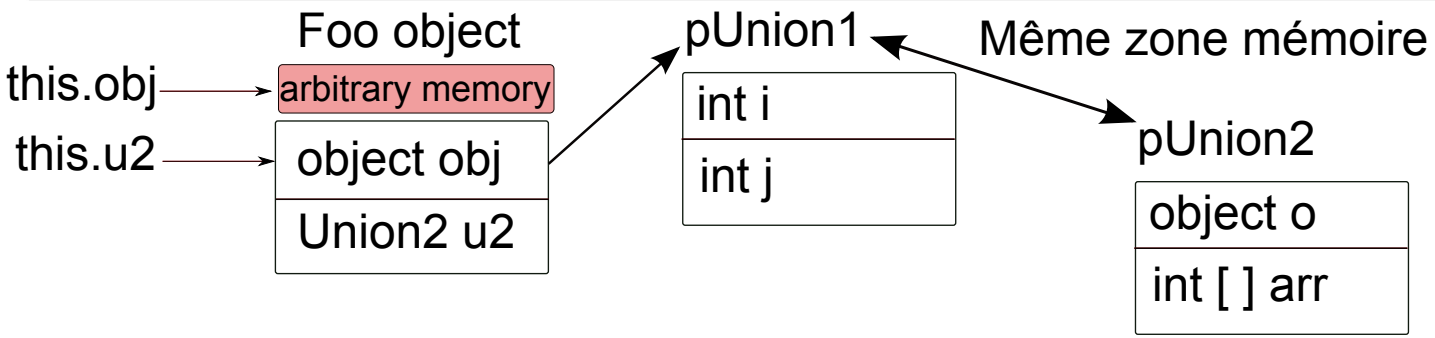


Confusion de type 2/3

Appel du delegate : exploitation

Quand le *delegate d* est appelé, la méthode virtuelle *toString* qui lui est associée est appelée.
Ainsi le bug est déclenché.

```
public override string ToString() {
  Program.pUnion2 = this.u2;
  return null; } }
...
d(); // appel du delegate : entraine l'appel a toString
```



.

Deux pointeurs différents (pUnion1 et pUnion2) pointent sur le même objet Union1.

Confusion de type 3/3

Conclusions

- Grâce au décalage du pointeur **this**, nous obtenons deux pointeurs de types différents (Union1 and Union2) qui pointent sur la même zone mémoire
- Modifier les champs de pUnion1 modifiera ceux de pUnion2
- **En manipulant un tableau, nous pouvons lire et écrire n'importe où en mémoire**

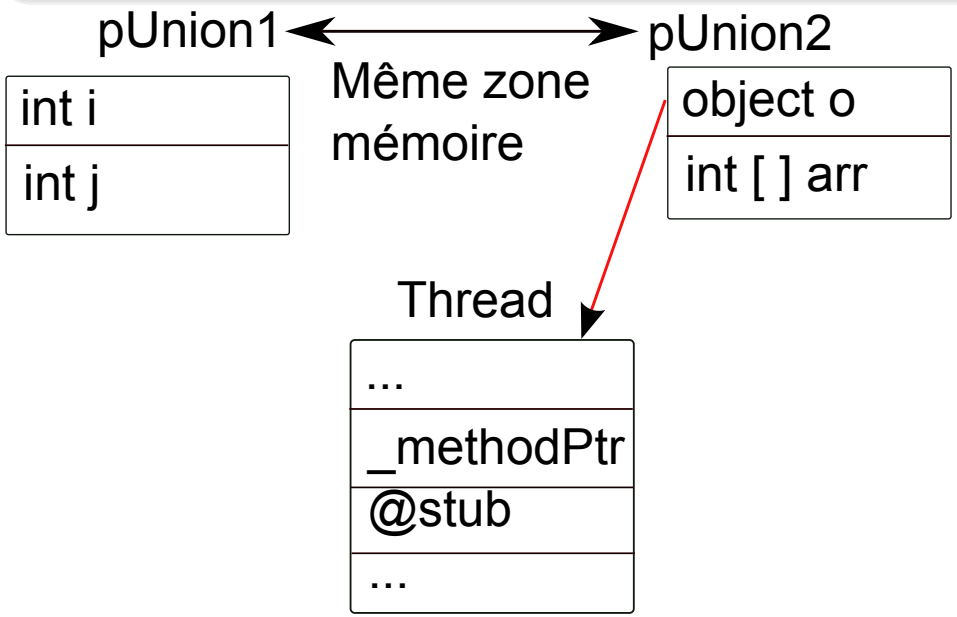
Exploitation d'une vulnérabilité .NET

Exploitation d'une vulnérabilité .NET

Technique d'exploitation 1/3

Exploitation de la confusion de type

- Avec le champ tableau, nous pouvons lire et écrire en mémoire
- Grâce à cet accès arbitraire à la mémoire, nous pouvons accéder directement aux champs de notre objet Thread
- En lisant le champ i, nous lisons l'adresse de l'objet



Technique d'exploitation 2/3

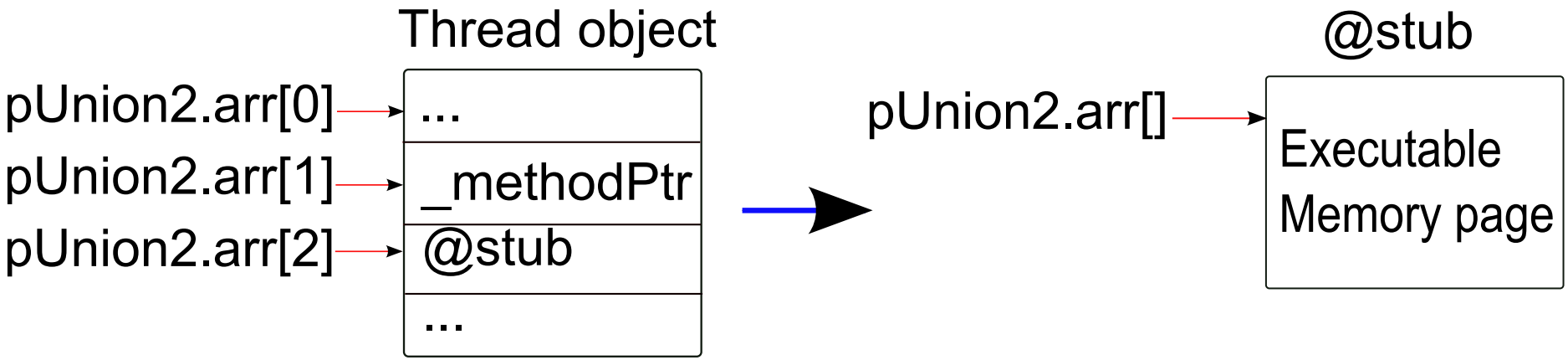
Lecture en mémoire

First step, memory read : @stub

pUnion2.o = thread

pUnion1.j = pUnion1.i

pUnion1.j = pUnion2.arr[2] - 12



Lecture en mémoire

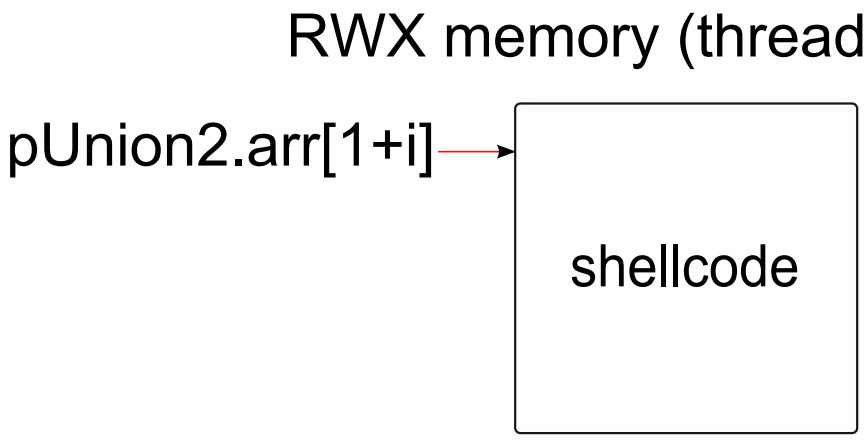
- Le stub du Thread est dans une zone mémoire exécutable
- Notre but : écraser le stub par notre code encoquillé
- **Maintenant, le champ tableau de notre pointeur pointe sur le stub**

Technique d'exploitation 3/3

Écriture en mémoire

Dernière étape : écriture du shellcode en mémoire

```
for (int i = 0; i < shellcode.Length / 4; i++)
   u2.arr[1 + i] = BitConverter.ToInt32(shellcode, i * 4);
```



Écriture en mémoire

- Notre shellcode est copié dword par dword dans une zone mémoire exécutable
- Grâce au compilateur JIT, cette zone est accessible en écriture

Et les protections Windows ?

Address space layout randomization (ASLR)

- Pas d'adresse en dur dans notre exploit
- Uniquement des appels à l'API du Framework
- ASLR contournée naturellement

Data Execution Prevention (DEP)

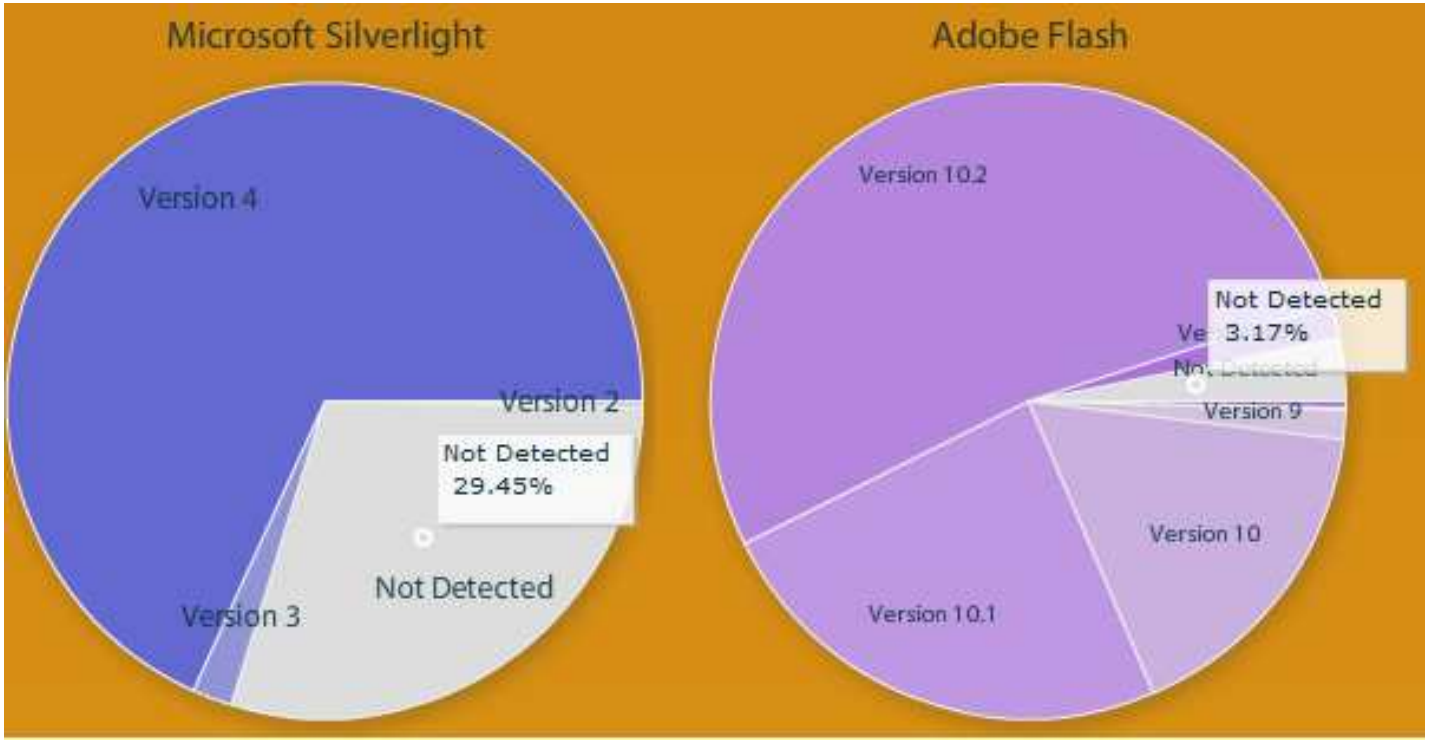
- La compilation JIT laisse le stub dans une zone en écriture et exécution
- Shellcode est copié dans le stub d'un objet Thread
- DEP contourné en utilisant une zone mémoire exécutable

Silverlight : une piste en or !

Vous avez dit Silverlight ?

Plugin Web .NET

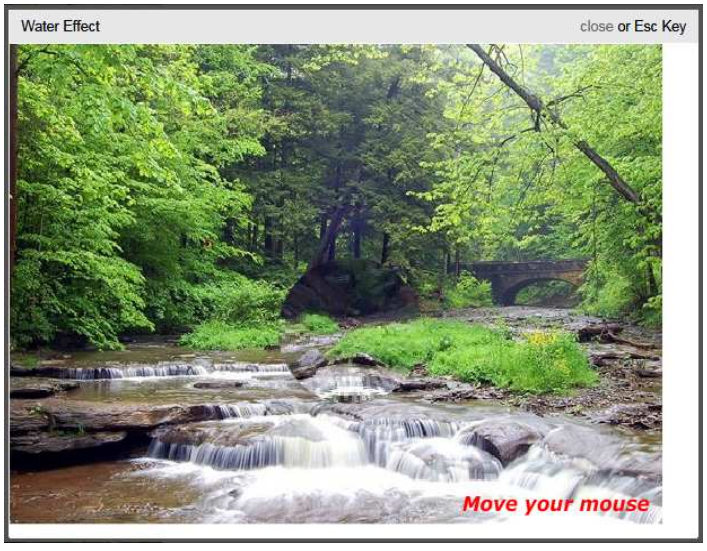
- Plugin pour navigateur Web écrit en .NET
- Concurrent direct de Flash player
- Exécuté côté client par sa machine virtuelle .NET



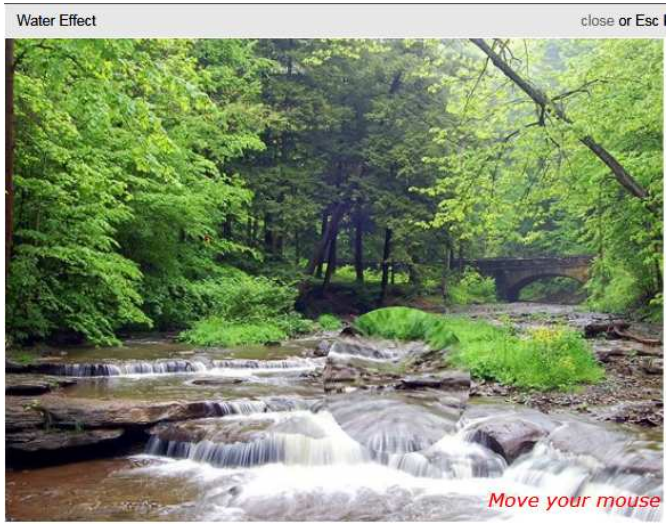
Source : riastats.com

Silverlight : un Flash player Microsoft !

A Flash animation



A Silverlight animation



Utilisation de Silverlight

De nombreux sites embarquent des applications Silverlight :

- Microsoft, Silverlight.net
- France télévision
- Liste complète : `http://www.realsoftwaredevelopment.com/top-10-silverlight-real-world-implementations/`

Pourquoi attaquer via Silverlight ?

Attaque classique

- Attaque classique peu intéressante : même idée que pour un virus classique
- Nous devons envoyer l'application .NET malicieuse à la victime et espérer qu'elle la lance

Attaque via Silverlight

- Silverlight permet une attaque à travers le navigateur Web
- C'est une attaque classique côté client
- Nous avons seulement besoin de contrôler un site Internet qui contiendra notre application Silverlight malicieuse
- Chaque client qui chargera la page, et donc l'application Silverlight embarquée, sera attaqué

Vers l'infini et au delà !

Démonstration !

.

Silverlight : une piste en or !

Résultats de l'exploit

- L'exploit contourne l'ASLR et le DEP avec tous les navigateurs
- Pour la majorité des navigateurs, le plugin Silverlight n'est pas sandboxé
- L'exploit n'utilise pas de *heap spraying*, il est **immédiat** et **très stable**

Web browser	ASLR	DEP	Sandbox
Windows Internet Explorer 9	BYPASSED	BYPASSED	SANDBOXED
Firefox 4	BYPASSED	BYPASSED	NOT SANDBOXED
Google Chrome	BYPASSED	BYPASSED	NOT SANDBOXED

À retenir

Conclusion

- .NET : énorme composant, surface d'attaque gigantesque
- Type de vulnérabilités difficile à trouver et exploiter
- ASLR et DEP trivialement contournées
- Silverlight offre une solution pour attaquer .NET via les navigateurs Web
- Flash Player n'est pas le seul plugin à risque !
- Absence de *heap spraying* : exploit très stable, pas de crash du navigateur
- Plugin Silverlight non sandboxé dans Firefox 4 et Google Chrome !

¿ Questions ?

SOGETI