

# Audit d'applications .NET

## Le cas Microsoft OCS 2007 (R1 et R2)

SSTIC 2010

Nicolas RUFF  
EADS Innovation Works  
nicolas.ruff (à) eads.net

# Préambule



- Qui suis-je ?
  - Un « chercheur » en sécurité
    - Audit de systèmes, audit de produits, audit de réseaux, test d'intrusion, analyse de malwares, détection d'intrusion, conception d'architectures sécurisées, relations publiques, trolls ...
- Qu'est-ce que je fais ici ?
  - Un état de l'art de mes connaissances
    - Evidemment incomplètes
  - Un appel à la population
    - Si vous vous intéressez à « .NET », contactez moi !

# Introduction



- Auditer OCS 2007, pourquoi faire ?
  - L'application est très complexe
    - Donc coûteuse à auditer
  - De nombreux clients l'utilisent déjà
    - Est-ce à moi de payer pour la sécurité des autres ?
  - Microsoft fait attention à la sécurité de ses applications
  - Seul Microsoft aura le pouvoir de corriger les failles
- Alors, vraiment, pourquoi faire ?
  - (A part des conférences)

# Introduction

- Auditer pour ...
  - Pénétrer des systèmes tiers via OCS ?
    - Certainement pas 😊
  - Répondre à une exigence de sécurité ?
    - Toute nouvelle application doit être auditée
  - Identifier la surface d'attaque
    - Dépendances, technologies, options de compilation, ...
  - Définir des mesures de défense en profondeur
    - Activation de DEP, modifications de configurations, ...
  - Comprendre !
    - Développer des outils et des compétences

# Microsoft OCS 2007, c'est quoi ?

- Un produit de communications « unifiées »
- Deux versions: « R1 » (?) et « R2 »
- Plus de 200 Mo de binaires (une fois installé)
- Au moins 5 serveurs pour une infrastructure complète
  - Contrôleur de domaine + DNS, Pool, Edge, SQL, serveur CWA, ...
- Des clients
  - Live Meeting
  - Live Communicator « R1 » et « R2 »
  - Communicator Web Access (CWA)



# De l'ordre et de la méthode

- Il est impossible de tout tester !
- Il faut définir:
  - Les risques majeurs
    - Soit les bonnes questions à se poser
  - Les moyens (outils et méthodes) applicables
    - Pour répondre aux questions précédentes



# Risques

- Les services exposés à Internet sont les plus menacés
  - Par exemple
    - Hébergement de *meetings* anonymes
    - Fédération d'identités permettant le *chat* avec des tiers
    - CWA
  - A contrario
    - Le serveur SQL ne devrait être visible de personne
- Les failles peuvent être de 3 types
  - Conception
    - Ex. protocole sans authentification mutuelle
  - Implémentation
    - Ex. *buffer overflow*
  - Mise en œuvre chez le client
    - Ex. utilisation de clés privées trop « courtes »

# Méthodes

- Il y a plein de choses à faire !
  - Analyse documentaire
  - Outils de développement (SDKs)
  - Outils d'administration (ex. LcsCmd.exe)
  - Outils de mise au point (ex. OCSLogger.exe )
  - Analyse « boîte noire »
    - Ports TCP ouverts, permissions sur les ressources, schéma SQL, ...
  - Analyse réseau
    - Plus ou moins difficile: SIP, RDP, PSOM, C3P, TURN, ...
  - Analyse de code

# Analyse de code .NET

- La partie serveur du produit OCS repose essentiellement sur du *bytecode* .NET
  - Codes sources:
    - C# pour l'essentiel
    - J# pour la partie « historique » du produit
      - Rachetée à PlaceWare en 2003
    - VB.NET pour les composants de rapport d'erreur
- L'audit du serveur OCS va donc consister essentiellement dans l'analyse de *bytecode* .NET

# .NET – c'est quoi ?

- Un *bytecode* [ECMA-335, ISO 23271]
  - *Common Language Infrastructure* (CLI)
- Une machine virtuelle d'interprétation du *bytecode*
  - *Common Language Runtime* (CLR)
- Des librairies standard [ECMA-335, ISO 23271]
  - *Base Class Library* (BCL)
- Des librairies additionnelles
  - *Framework Class Library* (FCL)
- Des langages compilables en *bytecode* .NET
  - J#, F#, VB.NET, ASP.NET, Cobol.NET, IronPython, managed C++, ...
  - C# est le langage de référence [ECMA-334]

# Analyse de code .NET - natif

- Tout application exécutée sur un processeur x86/x64 est transformée en code natif à un moment ou un autre
  - Sauf en cas d'interprétation complète du *bytecode*
    - Très peu performante !
  - Compilation « JIT » (*Just-In-Time*)
    - Réalisée en mémoire par le Framework .NET à l'exécution
  - Précompilation dans le « GAC » (*Global Assembly Cache*)
    - Réalisée par l'outil NGEN.EXE
    - « %windir%\assembly\NativeImages\* »
    - « \*.ni.dll »

# Analyse de code .NET - natif

- Les outils d'analyse x86/x64 classiques fonctionnent
  - Analyse statique des fichiers « \*.ni.dll »
    - IDA Pro
  - Analyse dynamique des processus en cours d'exécution
    - WinDbg
    - OllyDbg, SoftIce, ...
- Mais ces techniques perdent le bénéfice du typage fort du *bytecode* .NET !
  - A l'exception du débogueur « PEBrowse »

# Analyse de code .NET - statique

- Le *bytecode* .NET conserve toute la sémantique du code source
- Désassemblage
  - Il est possible de récupérer un listing au format « IL »
  - Outils ILDASM.EXE et ILASM.EXE fournis avec Visual Studio
  - Désassemblage et réassemblage sans perte d'informations
    - Modulo les ressources, la signature (*Strong Name*), ...
- Décompilation
  - Il est possible de récupérer un listing au format C#
  - Outil « Reflector.NET » (gratuit)
    - Ou autres outils commerciaux comme « 9Rays Decompile »
  - Décompilation et recompilation dans Visual Studio !

# Analyse de code .NET - dynamique

- WinDbg sait déboguer du *bytecode* .NET
  - Grâce à l'extension « SOS »
    - `.loadby sos.dll mscorwks`
    - `!help`
  - Voir aussi les extensions « SOSEX », etc.
- Tout le monde peut écrire son débogueur
  - Interfaces de débogage offertes par le CLR
    - Objets COM « ICorDebug\* »
    - Cf. outil MDbg

# Analyse de code .NET - *profiling*

- Tout le monde peut écrire son profileur
  - Interfaces de *profiling* offertes par le CLR
    - Objets COM « ICorProfiler\* »
    - Cf. outils Logger, dotTrace, .NETMon, ...
- Le profileur peut instrumenter le code au moment de la compilation JIT
  - SetILFunctionBody()
- Le profileur est un objet COM en code natif
  - Développement long et fastidieux ...

# Analyse de code .NET - réflexion

- Namespace « System.Reflection.\* »
  - Permet une introspection (statique ou dynamique) du code
  - Toutes les primitives sont fournies
    - Chargement de classes, désassemblage, manipulation de code source, ...
  - Cf. outil Reflector
- Namespace « System.Reflection.Emit.\* »
  - Permet une génération (statique ou dynamique) de code
  - Toutes les primitives sont fournies
    - Création d'*assemblies* sur disque, assemblage symbolique, ...
  - Seul limite: il n'est pas possible de modifier du code existant
    - MethodRental ne fonctionne que sur du code généré dynamiquement

# Analyse de code .NET - Phoenix

- Phoenix Framework: le *metasm* du « .NET »
  - Phoenix en 3 mots
    - Un projet de recherche Microsoft
    - Un environnement permettant de travailler du code source jusqu'au *bytecode* ... et inversement
      - Compilation, décompilation, manipulation des assemblies, ...
    - Une fondation pour les futurs compilateurs Microsoft
  - Cf. analyseur statique « Cthulhu »
    - Matt Miller / ToorCon 2007

# ShowTime !

1. Générer et exécuter dynamiquement du *bytecode* en utilisant la réflexion
2. Ré-implémenter (partiellement) Reflector.NET en 10 lignes de C#
3. Surprise 😊

# ShowTime !

☆ <http://www.pcinpact.com/actu/news/48857-microsoft-songsmith-fan-linus-trovalds.htm>

Connectez-vous Design

# PC INpact

Navigation

Cherchez... Trouvez

- Toute l'Actualité
- Tests et dossiers
- Astuces
- Comparateur de prix
- Communauté
- Divers et à propos

Derniers articles

 Computex 2010 : le bilan (2/2)

L'actualité informatique et multimédia

## Linus Torvalds l'avoue : il est devenu grand fan de Microsoft

ce n'est pas sale, Linus ! 

Pointé par nos confrères d'[Ecrans.fr](#), cette petite phrase de Linus Torvalds sur son blog : « *Fantastique !* » s'écrit le personnage qui tout de go raconte son histoire d'amour naissante avec Microsoft .

« *Cela va peut-être choquer tout le monde, mais je dois admettre que je ne suis généralement pas un grand fan de la plupart des logiciels de Microsoft (« Non, vraiment, Linus ? Dis-nous en plus ! »). Mais je vais devoir reconnaître que j'ai eu tort.*

*Non, je ne parle pas de Windows 7. Je parle de Songsmith, qui est clairement, un vrai produit de génie. Oui, oui, les pubs sont douloureusement affligeantes, mais, utilisé correctement, ce qu'il peut produire, c'est indubitablement de l'art. »*

Une histoire d'amour qui est né en regardant ce clip « White Wedding » du très blond Billy Idol qui a été songsmithé dans les règles de l'art. Avec un résultat tellement craquant que Linus évoque un risque de syndrome du canal carpien à force de cliquer sur « replay ». (voir une sélection de vidéo dont la bande son a été triturée avec Microsoft Songsmith)

# Analyse de code .NET - CLR

- Un CLR compatible 1.0 et 2.0 a été publié par Microsoft
  - Shared Source CLI (SSCLI alias projet Rotor)
  - Non supporté depuis plusieurs années
    - Mais le bogue MS09-061 a été trouvé par la lecture du code source
- Pour les versions ultérieures
  - Une partie du code source est disponible
    - <http://referencesource.microsoft.com/netframework.aspx>
  - Une partie des symboles de débogage privés est disponible
    - <http://referencesource.microsoft.com/symbols>

# .NET – où sont les failles ?

- Les avantages de .NET
  - Typage fort, vérifications à l'exécution = pas de *cast* improbable
  - Pas de manipulation de pointeurs = pas de *buffer overflow*
  - *Garbage Collector* = pas de *double free*
  - Bibliothèques éprouvées (ex. crypto)
  - Politique de sécurité configurable au niveau du CLR
  - Signature de code
- Les failles potentielles
  - Failles logiques (ex. injections SQL, *backdoors*, ...)
  - P/Invoke (appel à du code natif)
  - InteropServices (appel à des objets COM)
  - « unsafe », « stackalloc », « StructLayout », ...
  - Failles d'implémentation dans le CLR (ex. MS09-061)

# .NET – où sont les failles ?

- Historique de sécurité du produit
  - MS06-033 & MS06-056
    - Fuite d'information et XSS dans ASP.NET
  - MS07-040 & MS09-061
    - « Vraie » évasion de la machine virtuelle
  - MS08-052 & MS09-062
    - Failles GDI+ affectant .NET par rebond
  - MS09-036
    - Déni de service sur ASP.NET
- C'est peu comparé à la JVM Sun ...
  - Mais c'est sans compter sur les failles corrigées silencieusement !

# ShowTime !

- Exploitation d'une faille .NET (cas général)
  - Le mot clé « StructLayout » permet de contrôler le placement des objets en mémoire
  - A l'aide d'une « union », on accède au contenu d'un objet via un autre
  - Ces opérations ne peuvent être mises en œuvre que par du code « de confiance »
- La faille MS09-061
  - Une vérification commentée dans le code source permet de simuler la logique de l'union
  - ... quel que soit le niveau de confiance du code
- Impact
  - Nul, sauf si vous êtes capables de faire exécuter du bytecode « .NET » à la victime
  - Exemple de scénarios:
    - SilverLight, déploiement « ClickOnce », Windows Azure, ...
  - Ou le scénario Dowd + Sotirov (BlackHat 2008)
    - `<OBJECT classid="ControlName.dll#namespace.ClassName">`

# OCS 2007 - méthodes

- Traces internes
  - Faire appel aux outils fournis avec l'application avant toute chose !
  - Ex. OCSLogger / OCSTracer
- Test statique de classes avec IronPython
- Décompilation
  - Fonctionne uniquement sur le code C# de la version « R1 »
  - Les outils actuels ne sont pas efficaces sur du code « J# » ou du *bytecode* 3.5
- Recompilation
  - Il existe quelques *assemblies* qui vérifient les *Strong Names* « en dur »
- Analyse dynamique avec WinDbg
  - Ex. Identification des appels aux générateurs d'aléa
  - Ex. Identification du code de support pour le protocole PSOM (port TCP/8057)

# OCS 2007 – résultats (extrait)

The screenshot shows the 'Office Communications Server 2007 Logging Tool' window. It is divided into several sections:

- Logging Options:**
  - Components:** A list of components with checkboxes. 'DataMCU' is checked and highlighted in blue.
  - Level:** Radio buttons for Fatal Errors, Errors, Warnings, Information, Verbose, and All. 'All' is selected.
  - Flags:** A list of flags with checkboxes: TF\_COMPONENT, TF\_PROTOCOL, TF\_CONNECTION, TF\_SECURITY, TF\_DIAG, and All Flags. All are checked.
- Global Options:**
  - Log File Options:** Radio buttons for Circular, Sequential, and New File. 'Sequential' is selected. A 'Maximum Size' field is set to '20 MB'. There is an 'Append to log file' checkbox which is unchecked.
  - Real Time Options:** A 'Real Time Monitoring' checkbox is checked. Below it is an unchecked 'Display only, no log file' checkbox.
  - Filter Options:** An 'Enable Filters' checkbox is unchecked. There is a 'Clear Filters' button and four input fields for URI and FQDN.

At the bottom, there is a 'Log File Folder' field containing 'C:\WINDOWS\Tracing' and a 'Browse ...' button. Below that are five buttons: 'Start Logging', 'View Log Files', 'Analyze Log Files', 'Advanced Options', 'Exit', and 'Help'.

No active log session. Check the components you wish to log in the list on the left. For each checked component, configure the log level and flags for that component. Click Start Logging button to start logging the checked components with the configured level and flags.

# OCS 2007 – résultats (extrait)

- >>> import clr
- >>> clr.AddReference("Microsoft.RTC.Server.DataMCU.Application.Shared.dll")
- >>> import placeware.io.PWPath
- >>> from System import Array
- >>> a = Array[str]("")
- >>> placeware.io.PWPath.main(a)
- Testing ...
- checkPWPathSyntax() - true
- convertToUnixSyntax() - /awm/vol/vol-01/eng/work/foo/bar.html
- convertToWindowsSyntax() - \\awm\vol\vol-01\eng\work\foo\bar.html
- getPWPath() - awm:/vol/vol-01/eng/work/foo/bar.html
- getUnixPath() - /awm/vol/vol-01/eng/work/foo/bar.html
- getWindowsPath() - \\awm\vol\vol-01\eng\work\foo\bar.html
- >>> b = Array[str](["c:\\windows\\notepad.exe"])
- >>> placeware.io.PWPath.main(b)
- Testing ...
- checkPWPathSyntax() - false
- convertToUnixSyntax() - null
- convertToWindowsSyntax() - null
- getPWPath() - c:\windows\notepad.exe
- getUnixPath() - null
- getWindowsPath() - null

# OCS 2007 – résultats (extrait)

- Génération d'aléa
  - Les générateurs « `System.Random` » et « `java.util.random` » ne sont pas « sûrs »
    - Algorithme soustractif de Donald Knuth
    - Initialisé avec le `TickCount` de la machine
  - Ils sont pourtant utilisés à certains endroits
    - Ex. `placeware.apps.aud.SlideFiles` (génération du nom de fichier)

# OCS 2007 – résultats (extrait)

- Reconstruction du protocole PSOM (TCP/8057)
  - Opérations supportées
    - private enum FrameCode : byte
    - {
    - Authentication = 0x55,
    - BreakChannel = 6,
    - CloseChannel = 0,
    - DataRecord = 0x16,
    - NoCode = 0xff,
    - OpenChannel = 0x37,
    - SetChannel = 4,
    - Signature = 0x56
    - }
    - Le code 5 correspond à la fonction « Abort »

# OCS 2007 – résultats (extrait)

- Reconstruction du protocole PSOM (TCP/8057)
  - Schéma d'authentification
    - Le client s'authentifie via le protocole SIP
    - Le serveur SIP génère un ticket d'authentification, contenant un élément aléatoire de 8 octets et une durée de vie de 2 minutes
    - Le serveur SIP transmet le ticket au client dans le champ sAuthId
    - Le client a 2 minutes pour se reconnecter sur le port TCP/8057 et présenter son ticket
    - Le ticket est détruit à la première tentative d'authentification (réussie ou non)
- Remarque: cette liste de résultats obtenus est non exhaustive 😊

# Conclusion

- L'audit applicatif est une activité de plus en plus complexe
  - Car la taille et la complexité des applications augmente
- Pour cette même raison, l'audit applicatif intéresse de plus en plus les clients
  - Comment faire confiance à une énorme application monolithique, accessible « en direct » sur Internet ?
- L'audit d'applications « .NET » est relativement aisée
  - Sous réserve de disposer des bons outils et des bonnes méthodes !
- Et ... Microsoft OCS 2007 est plutôt « sûr » 😊
  - Ceci ne veut pas dire qu'il n'y aura pas de failles !

Questions ?

