Outils d'intrusion automatisée : risques et protections

Mathieu Blanc

Commissariat à l'Energie Atomique
DAM - Ile de France

SSTIC 2008
4 juin 2008

mathieu.blanc@cea.fr
moutane@rstack.org

Sommaire

Objectifs

- Présenter les outils d'intrusion automatisée ;
- Expliciter les risques liés à leur utilisation ;
- En comprendre le fonctionnement interne ;
- Proposer des moyens pour les détecter ou les bloquer.

De l'art du pentest

Dans la trousse à outils de l'auditeur, on trouve de nombreux outils :

- Scanner réseau ;
- Scanner de vulnérabilités ;
- Scanner de comptes et mots de passe ;
- ...

Bien évidemment les compétences de l'auditeur mais c'est une autre histoire...

Depuis quelques années, on trouve aussi des **outils d'exploitation automatisée de vulnérabilités**.

Outils d'intrusion automatisée

Les outils étudiés :

- Metasploit, CORE IMPACT, Immunity CANVAS

Caractéristiques spécifiques

- Intégration d'une base d'exploits ;
- Vulnérabilités distantes, locales, et *client-side* ;
- Exécution des exploits fiabilisée ;
- Possibilités d'obtenir des exploits *O-day* via des packs payants.

Pentest automatisé

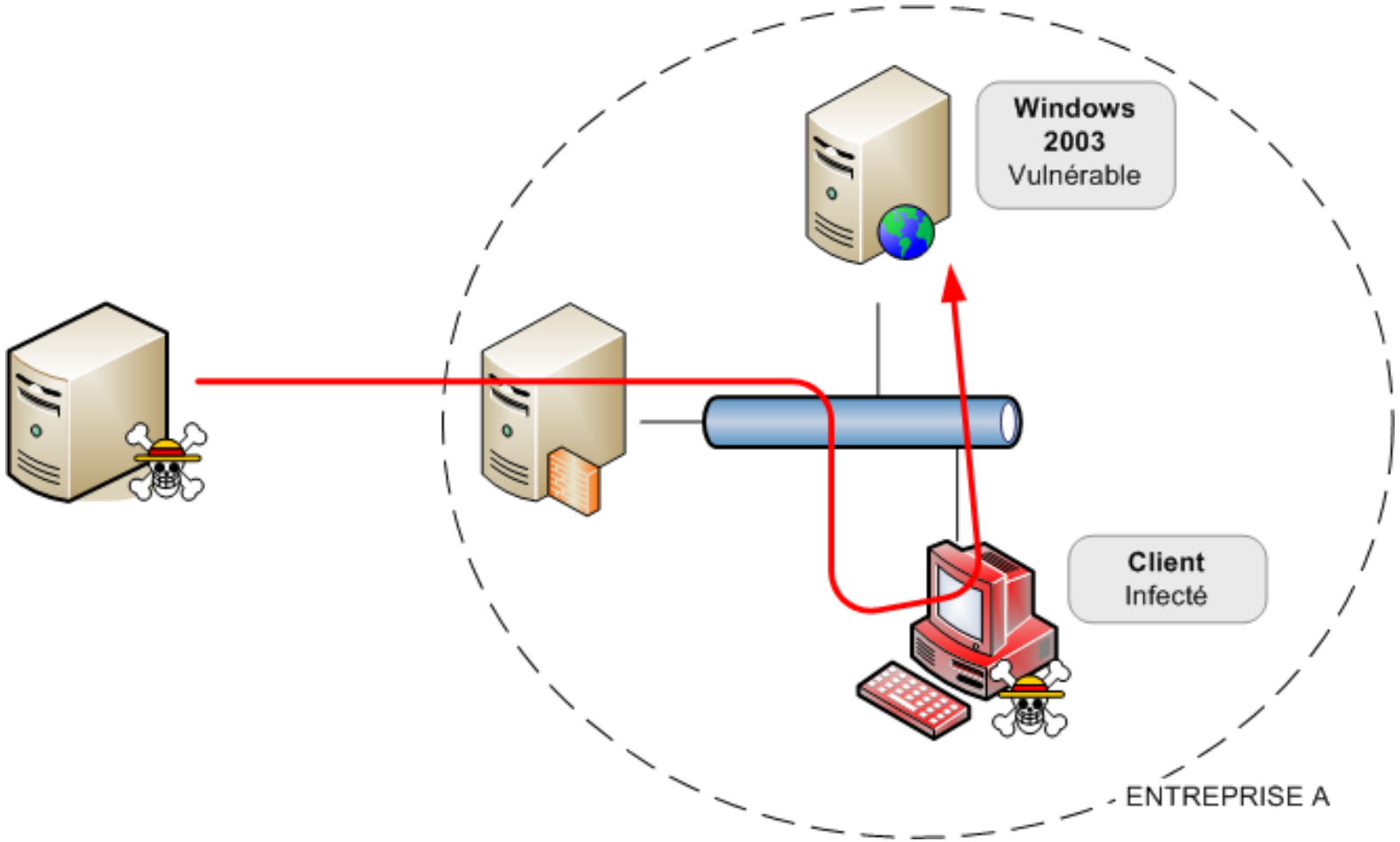
Ces outils réalisent de façon automatique :

1. Découverte des systèmes sur un réseau cible ;
2. Identification des versions des systèmes ;
3. Tentative d'exploitation des vulnérabilités connues correspondantes ;
4. Déploiement d'agents sur les machines compromises ;
5. Eventuellement élévation locale des privilèges ;
6. Répétition des étapes à partir des agents.

Automatisation des exploits client-side

Dévastateur dans les *"Crunchy networks"* !

Crunchy Networks



Risques liés à l'utilisation dans un audit

Les erreurs d'utilisation :

Par manque de maîtrise des outils

- Attaque d'ordinateurs en dehors du périmètre d'un audit ;
- Utilisation d'exploits pouvant provoquer des DoS sur des services sensibles.

Par manque d'information sur les exploits

- Toujours bien lire les informations sur l'exploit :
  - OS et versions cibles ;
  - Conditions de fonctionnement ;
  - Possibles conséquences de l'utilisation.
- DoS provoqué par un exploit considéré "sûr".

Risques d'utilisations malveillantes

Les attaques volontaires :

Par des pirates indépendants

- Certaines versions des outils commerciaux sur le p2p ;
- Outils comme Metasploit librement disponibles ;
- Environnements de développements d'exploits très performants ;
- Entre les mains d'un expert malveillant, ces outils peuvent être très dangereux.

Par des organisations avec des moyens financiers

- CORE IMPACT et Immunity CANVAS coûtent cher ;
- Emergence de la *cybercriminalité* et du commerce de code d'exploitation.

Information sur les clients de CORE IMPACT

Sur erreur du service commercial de CORE, nous avons pu voir une liste d'adresses recevant les *offline updates*.

Domaines	Nombre
Etatiques	58 %
Commerciaux	35 %
Autre	8 %

Prise en compte des risques liés à ces outils

Risques en cas de mauvaise manipulation

⇒ Connaître en détail le fonctionnement des outils.

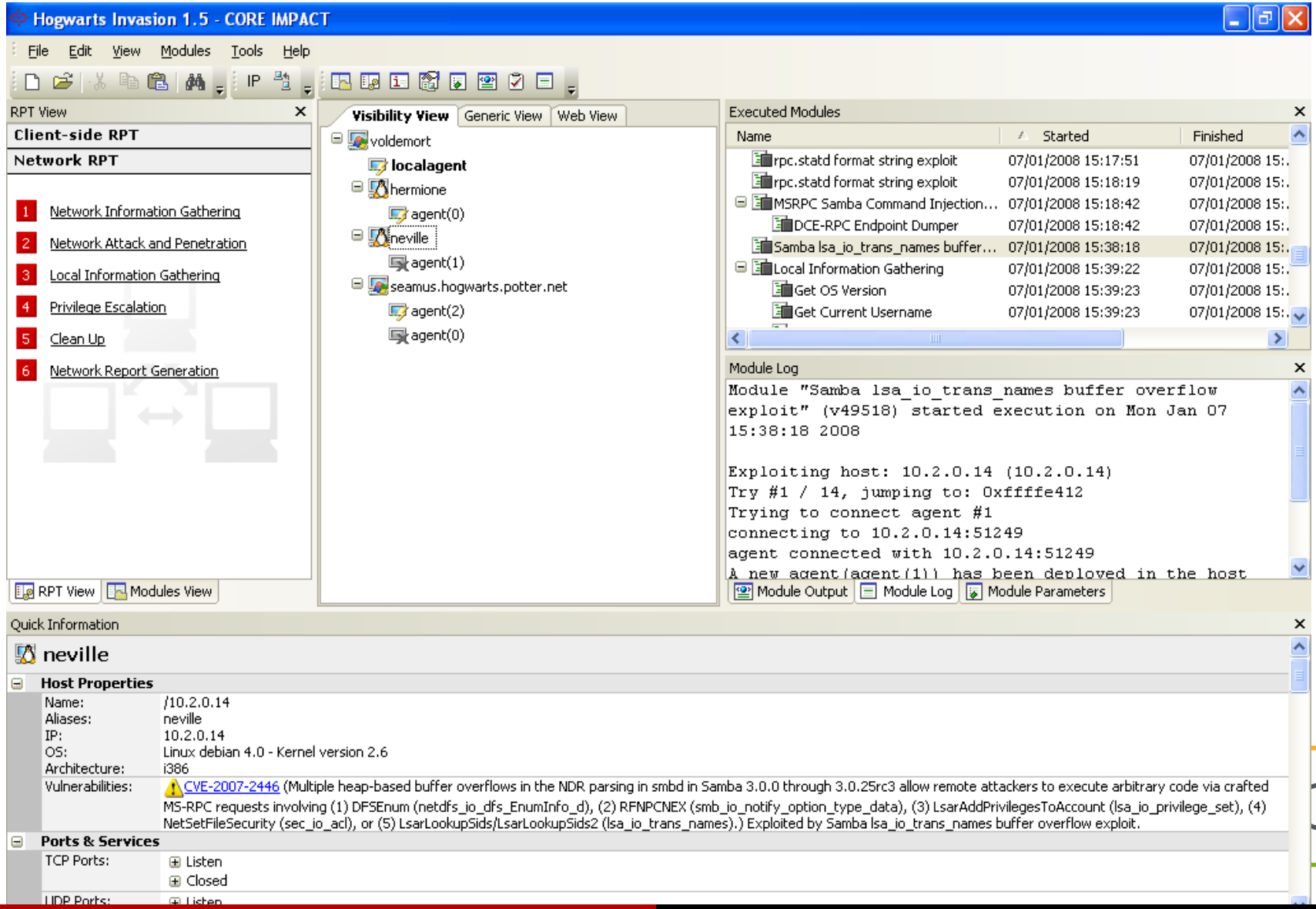
Attaques effectuées à l'aide de ces outils

⇒ Détecter et bloquer les attaques qui en émanent.

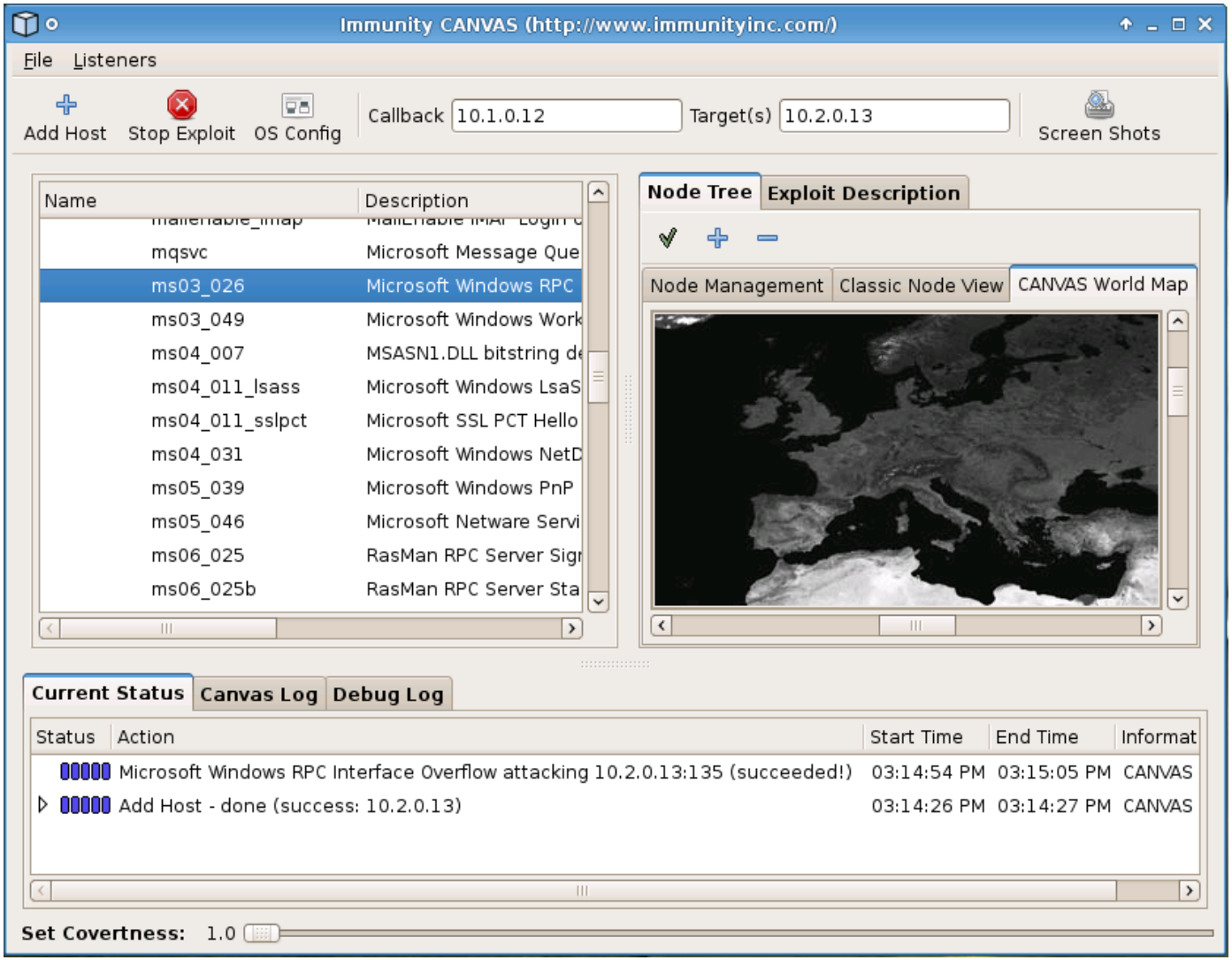
Les outils étudiés

- CORE IMPACT 7.X
  - Code source des exploits ;
- Immunity CANVAS 6.XX
  - Code source intégral ;
- Metasploit framework 3.X
  - Logiciel Open Source et gratuit ;

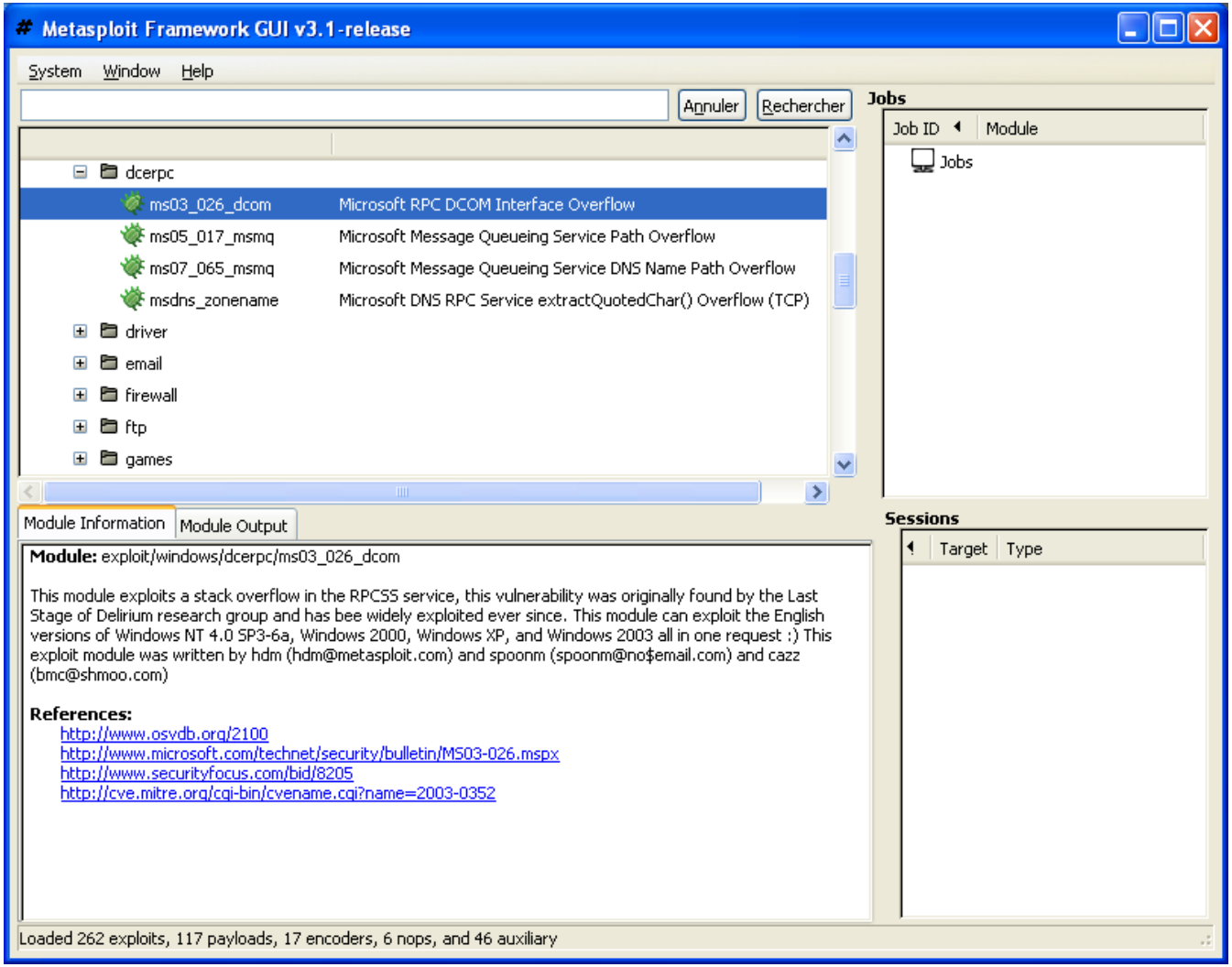
CORE IMPACT 7.X



Immunity CANVAS 6.XX



Metasploit framework 3.X



Fonctionnalités des outils

Systèmes d'exploitation cibles :

- Microsoft Windows (2000, XP, 2003) ;
- MacOS X (peu d'exploits) ;
- *BSD (essentiellement OpenBSD) ;
- Linux (rarement multi-distribution) ;
- Solaris, AIX, IRIX, HPUX (plus ou moins).

Chiffrement des connexions :

- Exploitation et envoi de l'agent en clair ;
- Après, négociation d'une clé de chiffrement entre agent et console.

Etapes de la propagation automatique

1. Exploitation de machines distantes :
   - Par découverte automatique de machines vulnérables ;
   - Par exploitation de vulnérabilités sur des clients ;
2. Déploiement d'agents par un **code d'exploitation multi-étages** ;
3. Rebond automatique sur l'agent par **syscall proxying**.

Furtivité

La furtivité n'est pas l'objectif des outils étudiés :

- Code d'exploitation et de l'agent non chiffrés ;
- Phase d'exploitation automatique très bruyante sur le réseau.

Multi-stage shellcode

Déploiement d'agents par un système de shellcode à étages :

- Etage 2 :
  - Téléchargement du code de l'agent sur l'hôte compromis ;
  - Exécution du code de l'agent ;
  - Eventuellement migration de l'agent dans un autre processus.
- Etage 1 :
  - Détournement de l'exécution ;
  - Exécution d'un code binaire de type "Recv and Execute" ;
  - Attente d'une connexion ou *connect back*.

Rebondir sur les agents

Deux techniques pour exécuter du code sur les agents :

Upload de code source ou script vers l'agent

- Fonctionne avec tous les outils sans modification ;
- Problème : nécessité de compilateur ou interpréteur sur l'hôte distant ;

Syscall Proxy

- **Exécution mandatée d'appels système** ;
- Principe : faire exécuter les appels système d'un programme local sur une machine distante ;
- Minimise la dépendance à la configuration distante ;
- Adapté aux outils écrits en langage de script.

Syscall Proxy

Principe implanté dans

- CORE IMPACT ;
- Immunity CANVAS (MOSDEF).

Mise en oeuvre complexe

- Capture des fonctions système ;
- Sérialisation des arguments ;
- Exécution par l'agent ;
- Réception du résultat.

Voir les publications des développeurs CORE et CANVAS.

Principes de détection

Nous avons observé plusieurs phases dans l'exploitation :

1. Etage 1 du shellcode ;
2. Etage 2 du shellcode : agent ;
3. Communications liées au syscall proxying.

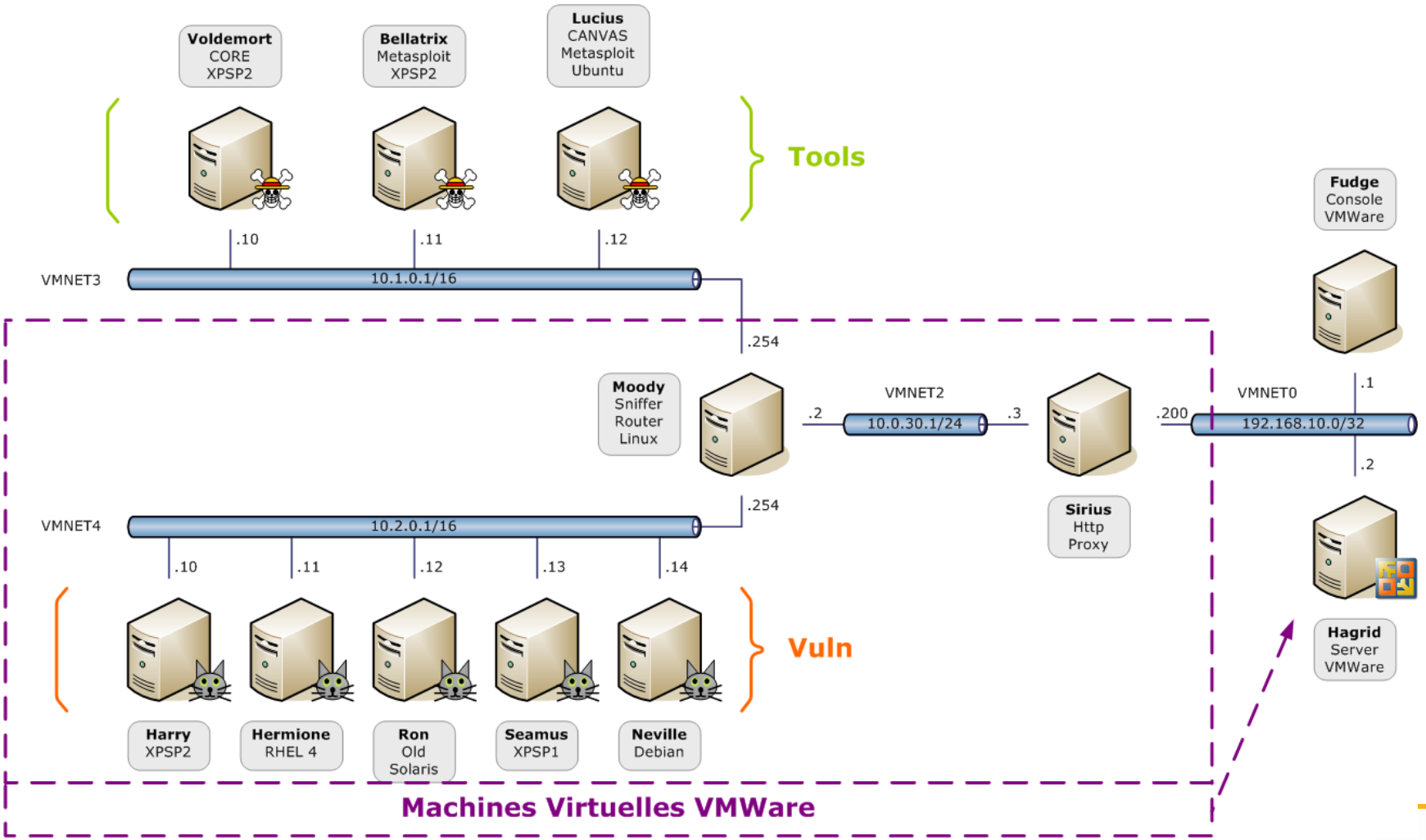
Objectifs de la détection :

- Détection générique des outils ;
- Pas de faux positif ;

⇒ Signature sur le code de l'agent

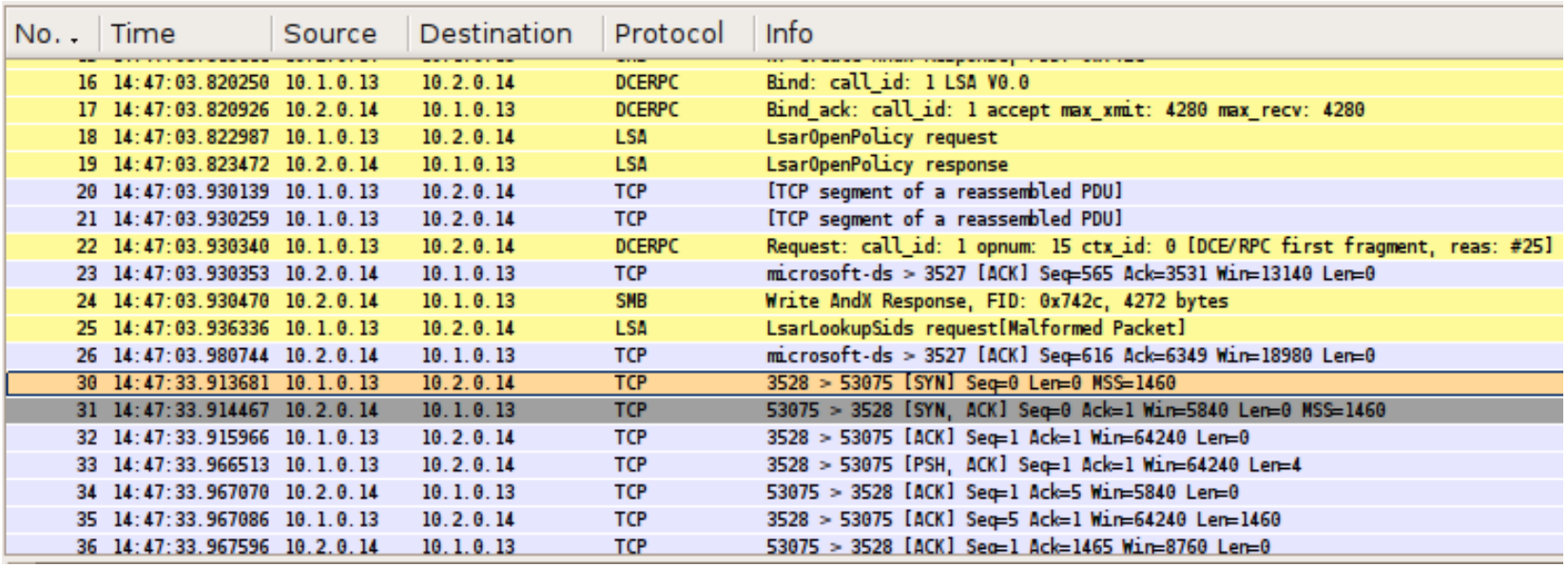
- Un outil = une signature ;
- Etape 2 ⇔ attaque réussie.

Architecture d'observation



Mise en évidence des 2 étapes

Samba 3.0.24 (LsaLookupSids) sur Debian 4.1



No.	Time	Source	Destination	Protocol	Info
16	14:47:03.820250	10.1.0.13	10.2.0.14	DCERPC	Bind: call_id: 1 LSA V0.0
17	14:47:03.820926	10.2.0.14	10.1.0.13	DCERPC	Bind_ack: call_id: 1 accept max_xmit: 4280 max_recv: 4280
18	14:47:03.822987	10.1.0.13	10.2.0.14	LSA	LsarOpenPolicy request
19	14:47:03.823472	10.2.0.14	10.1.0.13	LSA	LsarOpenPolicy response
20	14:47:03.930139	10.1.0.13	10.2.0.14	TCP	[TCP segment of a reassembled PDU]
21	14:47:03.930259	10.1.0.13	10.2.0.14	TCP	[TCP segment of a reassembled PDU]
22	14:47:03.930340	10.1.0.13	10.2.0.14	DCERPC	Request: call_id: 1 opnum: 15 ctx_id: 0 [DCE/RPC first fragment, reas: #25]
23	14:47:03.930353	10.2.0.14	10.1.0.13	TCP	microsoft-ds > 3527 [ACK] Seq=565 Ack=3531 Win=13140 Len=0
24	14:47:03.930470	10.2.0.14	10.1.0.13	SMB	Write AndX Response, FID: 0x742c, 4272 bytes
25	14:47:03.936336	10.1.0.13	10.2.0.14	LSA	LsarLookupSids request[Malformed Packet]
26	14:47:03.980744	10.2.0.14	10.1.0.13	TCP	microsoft-ds > 3527 [ACK] Seq=616 Ack=6349 Win=18980 Len=0
30	14:47:33.913681	10.1.0.13	10.2.0.14	TCP	3528 > 53075 [SYN] Seq=0 Len=0 MSS=1460
31	14:47:33.914467	10.2.0.14	10.1.0.13	TCP	53075 > 3528 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
32	14:47:33.915966	10.1.0.13	10.2.0.14	TCP	3528 > 53075 [ACK] Seq=1 Ack=1 Win=64240 Len=0
33	14:47:33.966513	10.1.0.13	10.2.0.14	TCP	3528 > 53075 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=4
34	14:47:33.967070	10.2.0.14	10.1.0.13	TCP	53075 > 3528 [ACK] Seq=1 Ack=5 Win=5840 Len=0
35	14:47:33.967086	10.1.0.13	10.2.0.14	TCP	3528 > 53075 [ACK] Seq=5 Ack=1 Win=64240 Len=1460
36	14:47:33.967596	10.2.0.14	10.1.0.13	TCP	53075 > 3528 [ACK] Seq=1 Ack=1465 Win=8760 Len=0

Attaques contre Windows (cf. article)

CORE IMPACT

```
content :"|51525033D25268|send|8BC45268|recv";
```

Immunity CANVAS

```
content :"|E577|kernel32.dll";
```

Metasploit Meterpreter

```
content :"metsrv.dll|00|MZ";
```

Attaques contre Linux (cf. article)

CORE IMPACT

```
content :"|89E5526801000000054E80000|";
```

Immunity CANVAS

```
content :"joh/zerh/dev";
```

Expérimentations

- Application sur l'analyse de trafic réel pendant 2 mois.

Résultats

- Les signatures n'ont jamais été déclenchées sur l'analyse du trafic réel.
- Sur un réseau d'expérimentation, nous avons vérifié que les signatures sont bien déclenchées lors d'une attaque réussie avec les outils.

Détection niveau hôte

Comme du point de vue réseau, l'agent est un élément fixe exécuté sur la machine victime :

- Intérêt de détecter un agent en cours d'exécution ;
- Signature spécifique à un des outils.

Détection de l'exécution d'un agent

- Traçage de l'exécution des processus ;
- Collecte et analyse centralisée des traces.

Collecte des événements

Deux parties dans la collecte :

- Agent de génération de traces d'exécution :
  - Déployé sur les machines à surveiller ;
  - Intercepte les appels aux services du système ;
  - Sérialise les appels effectués ;
  - Envoie les traces à un serveur réseau.

- Serveur de collecte :
  - Reçoit les traces d'exécution des systèmes surveillés ;
  - Enregistre dans une base de données ;
  - Réalise l'analyse des traces.

Analyse et détection des attaques

Signatures d'exécution des agents :

- Liste des appels effectués par un agent avec les arguments.
- Possibilité d'expression régulière dans les arguments.

Algorithme envisagé :

- Analyse en pseudo temps réel ;
- Un fil d'analyse pour chaque programme sur chaque système ;
- Correspondance exacte entre traces et signatures.

Traçage d'exécution sous Windows

Deux mécanismes différents sous Windows :

Interception des appels système dans le noyau

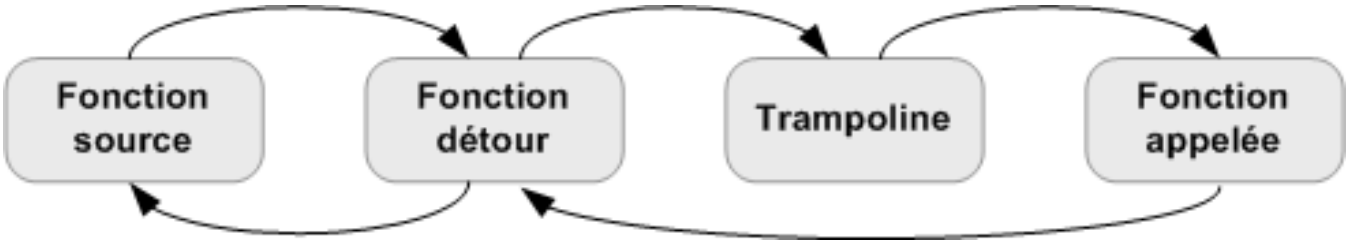
- Modification de la *System Service Descriptor Table* ;
- Traitement pré-appel (arguments) et post-appel (résultat).
- Outils existants : wormboy, Bindview Strace for NT.

Interception des appels à l'API Windows

- Détournement des appels aux DLL systèmes (KERNEL32, NTDLL...) ;
- Traitement pré-appel (arguments) et post-appel (résultat).
- Outils existants : NtTrace, StraceNT, Microsoft Detours.

Solution retenue : Microsoft Detours

- Traçage de l'API plutôt que des appels système ;
- Outil ancien, mais principe très général :
  - Détournement des fonctions de l'API en remplaçant le début de la fonction par un saut vers le code voulu.
  - Sauvegarde du code remplacé pour l'exécution normale de la fonction.



Exemple de fonction détournée

```
FARPROC __stdcall Mine_GetProcAddress(HMODULE a0,
                                      LPCSTR a1)
{
    _PrintEnter("GetProcAddress", "xs", a0, a1);

    FARPROC rv = 0;
    __try {
        rv = Real_GetProcAddress(a0, a1);
    } __finally {
        _PrintExit("GetProcAddress", (INT)rv);
    };
    return rv;
}
```

Traçage d'exécution sous Linux

Sous Linux, deux mécanismes de traçage également :

Interception des appels aux librairies

- Peu pratique dans notre cas : les agents effectuent directement des appels système.
- Outils existants : ltrace.

Interception des appels système

- Plusieurs méthodes possibles ;
- Une méthode légitime : `kprobe` ;
- Principe similaire à Detours.
- Traitement pré-appel (arguments) et post-appel (résultat).
- Outils existants : `systemtap`.

Exemple de fonction détournée

```
probe syscall.open {
  if (execname() != "staprun")
printf("%s:␣%s(%s)␣=␣", execname(), name, argstr)
}
probe syscall.open.return {
  if (execname() != "staprun")
printf("%s\n", returnstr(returnp))
}
```

Traçage d'exécution sous Solaris

Utiliser DTrace pour observer l'exécution de processus sous Solaris 10.

- Principe similaire à kprobe pour Linux (mais antérieur) ;
- Interpréteur de script fourni.

Article prometteur à BH Europe 2008

- Dtrace en tant que HIDS ;
- Dtrace pour l'analyse de malware.

Serveur de collecte de traces

Implémentation actuelle du serveur

- Langage python ;
- Sérialisation des appels système en `bencode` ;
- Stockage des logs dans une base SQLite ;

Serveur en cours d'implémentation, expérimentation à venir.

Contre-mesures face aux attaques

Comment se protéger des attaques menées grâce à ces outils ?

La première défense contre ces outils reste la mise à jour des logiciels :

- La plupart des exploits fiabilisés sortent après le correctif ;
- Les suppléments *O-day* sont généralement payants, et plutôt de type DoS ;
- Mais il est certain qu'il reste encore des failles non corrigées.

Blocage des attaques sur le réseau

Mise en place d'un système de prévention d'intrusion (IPS) :

- Avec les signatures définies précédemment ;
- Par exemple : `snort-inline`.

Remarque

L'attaquant pourra savoir que l'exploit a fonctionné.

Confinement des attaques

Objectif

Technique de leurre (honeypot) dynamique :

- Rediriger une attaque détectée sur une machine dédiée.
- Placer en environnement confiné (sandbox) un processus compromis.

Remarque

Le honeypot doit être capable d'émuler le comportement d'un agent.

Idées de contre-attaque

Les outils étudiés utilisent des briques logicielles existantes pouvant contenir des vulnérabilités.

Exemple

- CORE IMPACT utilise la librairie Winpcap, dont la version 4.0 comportait une vulnérabilité.
- On peut envisager d'exploiter cette vulnérabilité pour "contre-attaquer".
- Attention aux implications juridiques...

Conclusion

1. Etude de quelques outils d'intrusion automatisée :
   - Mécanismes de fonctionnement : agents, syscall proxying ;
   - Risques liés à l'utilisation ;
2. Définition de moyens de détection :
   - Réseau ;
   - Système (Windows, Linux, Solaris) ;
3. Proposition de contre-mesures.

Remerciements

- Mes collègues, en particulier Mathieu "George" et Pascal "Judas" ;
- Jonathan pour m'avoir aiguillé sur wormboy ;
- Le blog d'Ivan le fou, OpenRCE, Nynaeve ;
- La sagesse infinie de rstack, référence technique sans faille :)