Outils d'intrusion automatisée : risques et protections

Mathieu Blanc

Commissariat à l'Energie Atomique
DAM - Ile de France
Bruyères-le-Châtel
91297 Arpajon Cedex
mathieu.blanc@cea.fr

Résumé Les outils d'évaluation automatisée de la sécurité des systèmes d'information ont fortement évolué ces dernières années. Ils sont passés de l'outil d'audit non intrusif à l'exploitation de vulnérabilités clé-en-main et en profondeur. Du fait de la possibilité d'utilisation malveillante ou mal contrôlée, cette nouvelle génération d'outils représente une menace à prendre en compte.

Dans cet article, nous nous intéressons aux moyens de limiter les impacts de cette menace. En effet, une utilisation non maîtrisée de ces outils lors d'une évaluation peut dégrader le niveau de sécurité du système d'information audité. Ou encore, cette capacité intrusive utilisée à des fins malveillantes offre des moyens d'attaques puissants avec peu d'exigeances en termes de compétence.

C'est pourquoi cet article effectue l'étude de trois de ces outils parmi les plus connus, CORE IMPACT, Immunity CANVAS et Metasploit Framework. Après avoir mis en évidence les risques liés à ces outils, nous proposons des moyens de détecter leur emploi aux niveaux réseau et système. Nous envosageons enfin des techniques de contre-mesure, visant à bloquer ou à contenir des attaques effectuées par ces outils.

1 Introduction

1.1 Les outils de test d'intrusion automatisé

Sur le marché de la sécurité informatique, on trouve depuis longtemps des outils d'évaluation du niveau de sécurité des systèmes d'information. Ils ont pour but de trouver des failles de sécurité potentielles présentes sur le réseau, ainsi que de rapporter les services et systèmes dans des versions connues pour être vulnérables. Leur utilisation par des personnels de type RSSI participe, dans une approche quantitative, à l'évaluation des risques résiduels et à la prise de décision concernant d'éventuelles actions correctives.

Afin de minimiser les faux positifs, certains de ces outils vont jusqu'à une exploitation innofensive de ces failles. Toutefois, même pour ce type d'outil, l'utilisation n'est pas totalement sans risque et la possibilité d'introduire des faiblesses dans le système d'information a été évoquée [1].

Cependant, nous assistons depuis plusieurs années à l'apparition sur le marché d'outils, non plus seulement de recensement des vulnérabilités potentielles, mais bel et bien d'intrusion sur le réseau, les systèmes et les applications. Ils implémentent au plus près, potentiellement de manière automatisée, les actions que pourrait entreprendre un attaquant motivé et compétent, à l'exception peut-être de la furtivité qui n'est pas l'objectif de ces outils :

- Découverte du réseau
- Découverte des services

- Identification des vulnérabilités distantes résiduelles
- Tentatives d'exploitation de ces vulnérabilités
- Découverte des ressources présentes sur les machines compromises
- Elévation locale des privilèges
- Installation d'agents de rebond
- Répétition de ces étapes à partir de l'agent installé

Il existe des outils libres et des outils commerciaux. La particularité de ces derniers est de disposer de plus ou moins de *0Day exploits*[1].

L'amélioration des capacités de ces outils à automatiser les tests répond à un besoin légitime [2]. Cependant la fonctionnalité de "piratage automatisé en profondeur", facilitée par l'installation d'agents autorisant le rebond, ou par l'exploitation de failles d'applications clientes (pièces jointes et serveurs web piégés), est dévastatrice, particulièrement pour les "Crunchy Networks" [3].

De plus, parmi les outils commerciaux, on peut distinguer ceux qui sont fournis uniquement sous forme binaire, et ceux pour lesquels on dispose du code source de l'outil (souvent sous forme de scripts). Ces derniers intègrent généralement une plateforme de développement d'attaque et d'exploitation de vulnérabilités. Ils sont tout de même livrés avec leurs lots d'attaques préprogrammées prêtes à l'emploi. Plusieurs entreprises fournissent par ailleurs des lots d'attaques pour ces plateformes, correspondant à des vulnérabilités plus ou moins récentes, le prix allant de paire avec la "fraîcheur" de ce lot [4][5][6].

1.2 Un cadre d'utilisation sous contrôle

Ces outils automatiques de test d'intrusion sont utilisés dans le cadre d'audits de sécurité effectués par des équipes spécialisées, internes ou externes, mandatées pour cette évaluation. Notamment s'il s'agit d'outils commerciaux, coûtant plusieurs (dizaines de) milliers de dollars, leur usage est, en principe, limité à des entreprises spécialisées, ou ayant des besoins forts en sécurité. Tout cela peut mener à penser que de tels outils sont toujours utilisés dans un environnement parfaitement contrôlé. Pour les outils commerciaux, ce contrôle est également conditionné par la confiance accordée à l'éditeur, ou à l'éventuelle analyse de fichiers binaires.

Si dans le cas d'outils d'évaluation de vulnérabilités comme Nessus, la légitimité de leur emploi ne semble pas (trop) poser de problème, la doctrine d'emploi de cette nouvelle génération d'outil d'intrusion de systèmes d'information est moins immédiate. Premièrement, il existe des outils librement disponibles, notamment des outils Open Source. Ils peuvent être utilisés par n'importe qui possédant un minimum de compétences informatiques. Deuxièmement, par des moyens plus ou moins légaux (WareZ, P2P, sites Web mettant à disposition des logiciels piratés), on peut facilement se procurer certains des outils commerciaux de test d'intrusion. A partir de là, il devient difficile d'affirmer que ces outils sont utilisés dans un cadre défini... La présence de fontions de type "auto-hack" (découverte et attaque automatiques des vulnérabilités) et les interfaces "click and play" placent ces outils à la portée de beaucoup. Cela s'inscrit dans la tendance relevée par le CERT de la diminution du niveau de compétence requis pour lancer des attaques de plus en plus sophistiquées [7].

Ainsi on s'aperçoit que les barrières de la compétence et financière pour l'acquisition et la maîtrise de ces outils ne sont pas très épaisses. Il faut alors prendre en considération le développement des actes de cybercriminalité, la crainte d'attaques informatiques par des groupes armés ou terroristes (on évoque aussi le "cyberterrorisme") [8], et la propension de certains états à afficher

[1] Failles de sécurité résiduelles pour lesquelles aucun correctif n'a été publié.

une politique de lutte informatique militaire. Dans ces deux cas, des organisations terroristes ou étatiques peuvent financer l'achat de ces logiciels dans le but d'attaquer des entreprises ou des pays, directement ou en récupérant les codes d'attaques présents dans ces outils, comme cela a été exposé dans [9].

Par ailleurs, en novembre 2007, suite à une erreur dans la diffusion des mises à jour hors ligne d'un de ces outils, une partie de la liste de ses abonnés à été divulguée à certains d'entre eux. La répartition des domaines des adresses de ces abonnés est décrite dans le tableau 1. Même si cette liste n'est pas représentative de l'ensemble des clients de cet outil, force est de constater que beaucoup appartiennent à des organisations étatiques.

Domaines	Nombre
Etatiques	58 %
Commerciaux	35 %
Autre	8 %

TAB. 1: Exemple de clients de ces outils.

1.3 Comment faire face aux utilisations incorrectes ou abusives

Au vu des possibilités de subir des attaques provenant de ces outils automatisés d'intrusion, il nous a semblé très intéressant d'observer les fonctionnalités et le comportement de ceux-ci. Même si la première et plus fiable barrière contre une intrusion réussie reste l'installation de correctifs de sécurité, il ne faut pas négliger le fait que ces outils peuvent intégrer des vulnérabilités sans correctifs (0day).

L'objectif premier de cette étude est donc de contribuer à la compréhension du fonctionnement de ces outils, afin que l'utilisation de ces outils soit mieux maîtrisée. Ainsi on évitera l'introduction par erreur de failles dans le système d'information. En particulier, certains outils catégorisent des attaques comme étant "DOS safe", c'est-à-dire qu'elles sont censées ne jamais laisser un service attaqué dans un état indisponible.

Par exemple, sur une ancienne version de CORE IMPACT, l'exploitation de la vulnérabilité CVE-1999-0442 sur Solaris 7 passait entre autres par le déplacement temporaire de `/usr/bin/sh`. En cas d'erreur pendant l'exploitation, comme une rupture de connexion réseau, ce fichier n'était pas restauré, et alors non seulement la machine se retrouvait avec une porte dérobée (backdoor), mais de plus elle ne survivait pas au redémarrage.

Ensuite, le second objectif est de se protéger d'attaques provoquées par des utilisations abusives de ces outils. Pour cela nous allons rechercher des moyens de détecter une attaque effectuée à l'aide d'un de ces outils, à la fois d'un point de vue réseau et d'un point de vue système. Nous allons également tenter de proposer des mécanismes de contention et de contre-mesure contre ce type d'attaque.

Notre étude s'est porté sur 3 outils, qui nous semblent assez représentatifs du panel d'outils existants : CORE IMPACT [10], Immunity CANVAS Professional [11] et Metasploit Framework [12].

Cet article va d'abord présenter les différents types d'outils existants et leurs fonctionnalités respectives. Ensuite, nous verrons comment il est possible d'identifier ces différents outils, à partir de l'observation du trafic réseau via un NIDS[2], puis par l'observation d'évènements système sur une machine attaquée. Enfin nous aborderons brièvement différentes approches pour des tentatives de contre-mesures.

2 Caractéristiques des outils

La première partie de l'article va établir une comparaison des fonctionnalités des 3 outils considérés dans l'étude :

- CORE IMPACT 7.5 ;
- Immunity CANVAS Professional 6 ;
- Metasploit Framework 3.1.

CORE IMPACT est édité par la société CORE Security. L'interface principale de commande est celle présentée par la figure 1. Immunity CANVAS est édité par Immunity (voir figure 2). Metasploit est un outil Open Source développé principalement par un groupe d'expert en sécurité informatique. Il peut être manipulé soit par une interface shell, soit par une interface web, soit par une interface graphique développée récemment.

2.1 Systèmes supportés

	Hôtes		Cibles			
	Windows	Linux	Windows	Linux	Solaris	Autres Unix
IMPACT	X		X	X	X	X[a]
CANVAS	X	X	X	X	X	X
Metasploit	X	X	X	X	X	X

Tab. 2: Systèmes supportés en tant qu'hôtes et cibles.

Dans le tableau 2, on voit que CORE IMPACT est disponible seulement pour Windows, contrairement aux deux autres. Concernant CANVAS qui est entièrement en Python, même si l'installation est possible sous Windows, il est bien plus simple à installer sous n'importe quelle distribution Linux. Quant à Metasploit, il s'installe aussi bien sous les deux environnements.

Concernant les systèmes ciblés par les tests d'intrusion, les 3 outils sont équivalents. On notera toutefois que CANVAS, via les diffèrents pack d'exploits disponibles, peut cibler un plus grand nombre d'applications que les deux autres outils. Par exemple le pack Argeniss positionne très bien CANVAS pour l'évaluation de la sécurité d'un logiciel de gestion de bases de données.

2.2 Fonctionnalités

Le tableau 3 tente de recenser les caractéristiques les plus intéressantes des 3 outils.

[2] Système de détection d'intrusion réseau.

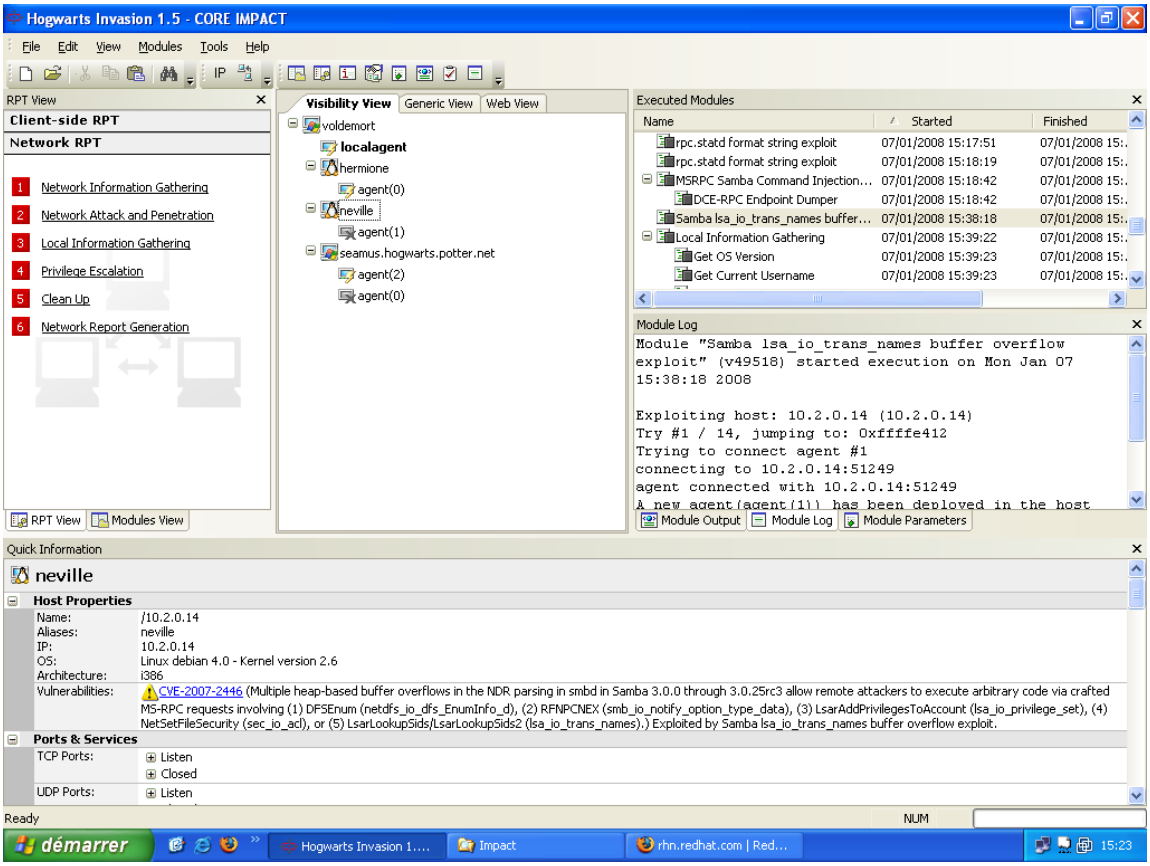


FIG. 1: Interface principale de CORE IMPACT.

	IMPACT	CANVAS	Metasploit
Découverte du réseau	X	X	
Maîtrise du payload		X	X
Syscall proxying	X	X	
Chiffrement du corps de l'agent			
Exploitation automatique	X	X	X

TAB. 3: Fonctionnalités comparées des différents outils.

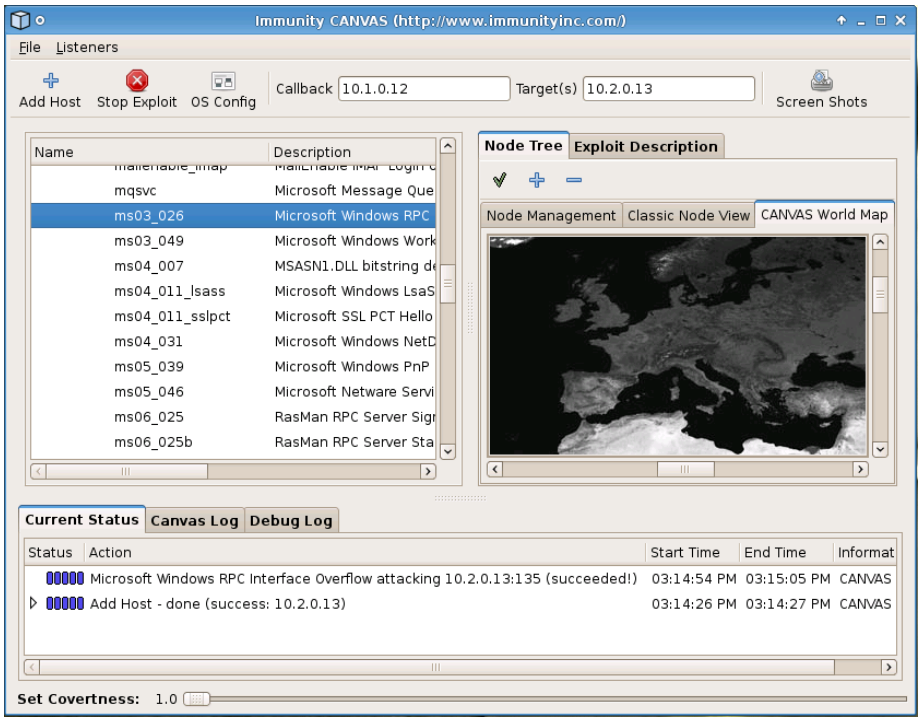


FIG. 2: Interface principale de CANVAS.

Deux données très intéressantes dans le tableau 3 : ces trois outils fournissent des moyens d'attaque automatique de systèmes, c'est d'ailleurs la raison pour laquelle ils sont traités dans cet article. En revanche, aucun d'entre eux ne chiffre le code de l'agent.

2.3 Caractéristiques techniques

Les outils d'exploitation automatisée de vulnérabilités se distinguent d'une exploitation de vulnérabilité "classique" par le fait qu'ils sont capables de déployer des agents sur les systèmes compromis. Ces agents sont de véritables satellites de l'outil, ils sont pilotables à distance et peuvent exécuter toutes les actions disponibles dans l'outil. Au niveau fonctionnel, l'injection d'un agent sur un hôte compromis est équivalente à l'installation de l'outil sur cet hôte.

Dans le but de déployer ces agents, l'exploitation d'une vulnérabilité va contenir deux phases clairement distinctes. La première phase vise à prendre le contrôle de l'exécution d'un programme vulnérable ciblé, c'est une phase qu'on retrouve dans n'importe quel code d'exploitation. La seconde phase, plus spécifique, vise à télécharger et exécuter le code de l'agent dans le contexte du programme ciblé.

On peut découper ces phases de la façon suivante :

1. Stage Loader :
   - Capture du flot d'exécution ;
   - Exécution du shellcode ;
   - Création d'un thread pour téléchargement et exécution de l'agent ;
   - Connexion de la console de l'outil (direct ou callback) ;
2. Agent :
   - Envoi du code de l'agent ;
   - Exécution du code de l'agent ;
   - Attente d'intructions.

Ces outils sont donc guidés par deux modèles de conceptions, d'une part l'utilisation de modules génériques, et d'autre part la séparation des "missions" du shellcode en plusieurs phases (*mission split* [13]). Nous allons en tirer parti pour définir des signatures visant à identifier les cas d'attaques réussies, ainsi que l'outil qui a été utilisé. En effet, en utilisant des signatures qui visent le code de l'agent téléchargé lors de la seconde phase, on s'assure, d'une part, que l'attaque a réussi, car cette phase ne se déclenche qu'à cette condition, et d'autre part de l'outil utilisé car il utilisera un agent spécifique.

3 Détection

Dans cette partie, nous allons chercher à détecter une attaque menée à l'aide de chacun des trois outils étudiés. Comme décrit dans la partie précédente, une telle attaque se déroule en plusieurs phases. L'observation de ces différentes phases est très importante, et nous amène a une orientation plus précise de cette étude. Précisément, la détection du code envoyé lors de la première phase de l'attaque n'est pas intéressante dans notre étude, car ce travail est déjà très bien réalisé par les IDS/IPS comme Snort ou Tipping Point. De plus, les techniques utilisées pour capturer le flot d'exécution peuvent être spécifiques aux vulnérabilités, et donc partagées par plusieurs outils. Dans tous les cas le *stage loader* est presque toujours différent d'une vulnérabilité à une autre.

Par conséquent, c'est plutôt la phase d'envoi du code de l'agent qui est intéressante pour nous, et ce pour deux raisons. D'abord, elle est synonyme de la réussite d'une attaque, et nous assure donc de l'absence de faux positifs. Ensuite, chacun de ces outils contient des briques génériques en termes de payload, en particulier le code de l'agent est indépendant de l'exploit utilisé, mais est spécifique à l'outil. En composant des signatures à partir du code générique contenu dans chaque outil, on détecte à coup sûr l'outil qui a été utilisé, et on obtient ainsi une information pertinente sur l'attaquant et sur sa détermination (critères de budget, de compétence, de formation, d'investissement en temps...).

Pour observer l'injection et le comportement de cet agent, nous avons mis en place une architecture virtuelle d'étude (cf. figure 3, avec notamment un point d'observation unique du trafic réseau entre les machines virtuelles contenant les différents outils d'audit et celles contenant les services vulnérables.

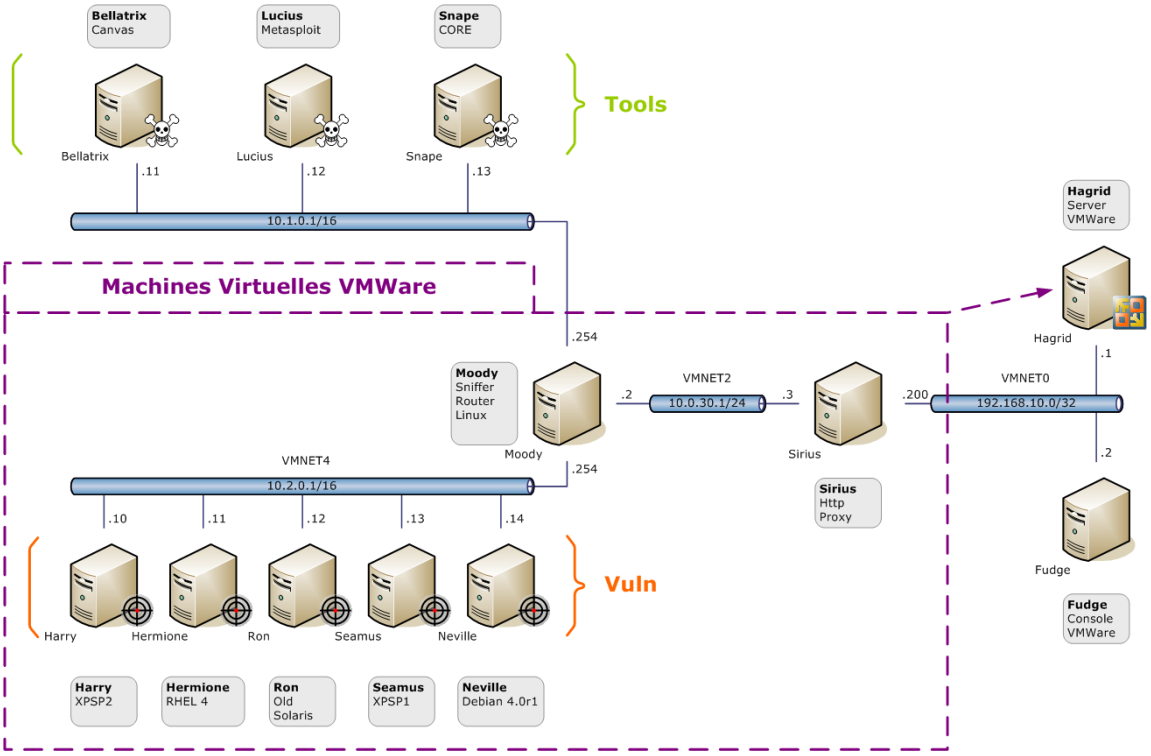


FIG. 3: Architecture du réseau virtuel d'étude.

La détection est envisagée suivant deux points de vue. Pour commencer, nous allons étudier la détection au niveau réseau via des techniques de types NIDS. Ensuite, nous continuerons sur la détection au niveau système. Ce point fait appel à des outils de type HIDS qui visent l'interception d'appels systèmes, et sont donc spécifiques à chaque système d'exploitation.

3.1 Détection niveau réseau

Comme souligné plus haut, l'injection du code des agents des différents outils est effectuée en clair (pas de chiffrement par défaut). Lors de l'observation du trafic réseau, c'est donc sans surprise que nous obtenons des communications présentant peu ou pas de caractéristiques aléatoires. Il semble donc possible d'en extraire des signatures pertinentes pour la détection niveau réseau.

Le tableau 4 recense l'ensemble des vulnérabilités qui ont été testées pour les trois outils.

	Windows		Linux	
	Serveur	Client	Serveur	Client
Metasploit	MS03-026	MS06-014 MS07-017	Non pertinent	Non pertinent
CORE IMPACT	MS03-026 MS06-040 MS06-066	MS07-017 CVE-2007-5659	CVE-2007-2446 CVE-2006-5276	CVE-2007-0245 CVE-2007-3387
CANVAS	MS03-026 MS06-040 MS06-066	MS07-004	CVE-2006-5276	CVE-2007-4575

TAB. 4: Vulnérabilités exploitées lors des tests des outils.

Metasploit 3 → Windows. Les fonctions de test d'intrusion automatique et d'installation d'agent sont relativement récentes dans Metasploit par rapport aux deux autres outils. La fonction de test automatique de vulnérabilité est liée à l'utilisation d'une base de donnée. Elle se fait à l'aide de la commande `db_autopwn`. Elle doit être couplée à une recherche de services ouverts effectuée par `db_nmap` [14].

L'agent est nommé le Meterpreter, il consiste principalement en une librairie dynamique, nommée `metsrv.dll`, qui est injectée dans le processus compromis sur la machine ciblée.

A l'aide de Wireshark, nous étudions la trace d'une attaque réussie par Metasploit. Le système ciblé est un Windows XP SP1 non patché, l'attaque est effectué sur la vulnérabilité MSRPC DCOM (MS03-026). Ce qui apparait immédiatement, c'est la transmission en clair du code de l'agent, comme on peut le voir sur la figure 4. En effet, on distingue clairement le nom `metsrv.dll`, ainsi que les lettres `MZ` caractéristiques de l'en-tête d'un exécutable Windows.

Nous avons également testé la détection des attaques côté client avec les vulnérabilités Internet Explorer COM CreateObject (MS06-014) et Windows ANI Stack Overflow (MS07-017).

Après nous être assuré que le mot `metsrv.dll` est bien contenu dans un seul paquet, nous pouvons l'utiliser dans une signature pour le NIDS Snort. Ce qui donne :

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any \
  (msg:"Metasploit agent injection"; \
    content:"metsrv.dll|00|MZ"; )
```

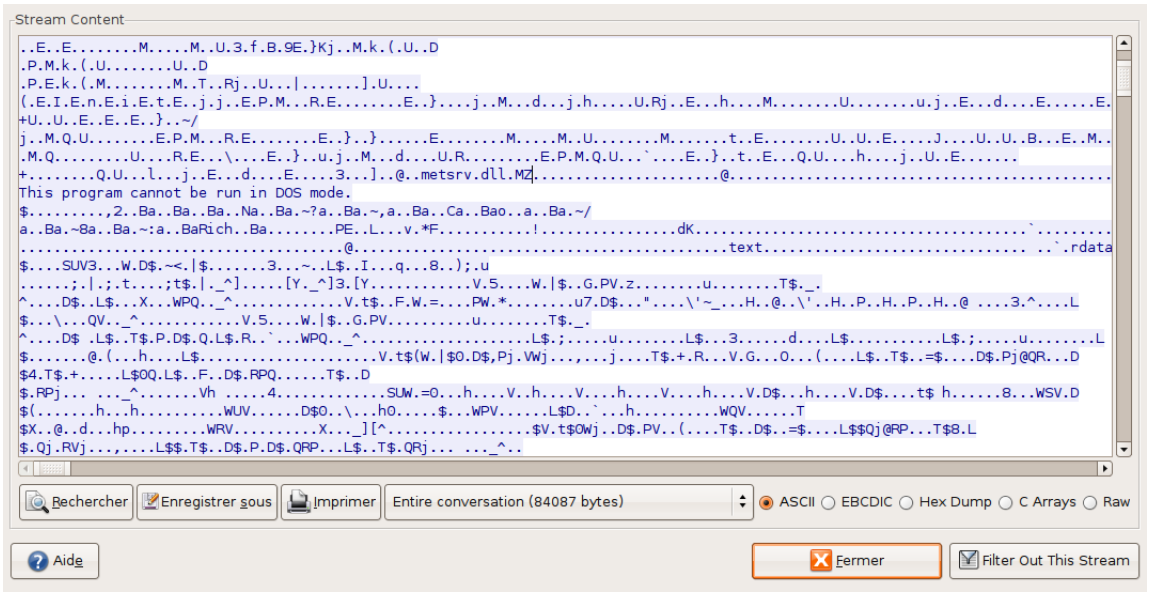


FIG. 4: Capture de `metsrv.dll`.

Metasploit 3 → Linux. A ce jour, Metasploit contient très peu d'exploit pour Linux. De plus, il n'existe pas d'agent du type Meterpreter pour Linux, cette fonctionnalité ne concerne que des cibles Windows. Nous avons donc pas étudié ce cas.

IMPACT 7.5 → Windows. CORE IMPACT est l'outil le plus "simple" à utiliser des trois outils étudiés, dans le sens où l'automatisation est poussée au maximum, et quelques clics de souris conduisent au résultat. A aucun moment il n'est nécessaire de comprendre ce que fait l'outil, alors que CANVAS et surtout Metasploit demandent une plus grande maîtrise de la sécurité informatique. Il est même assez inquiétant de voir à quel point l'accès à un tel outil d'attaque est simplifié à l'extrême !

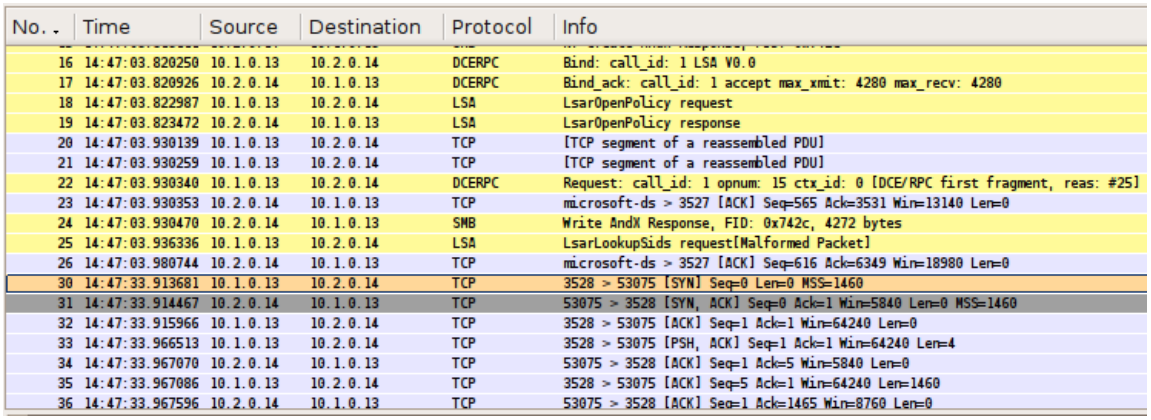
Lors d'une attaque réussie, CORE IMPACT installe un agent qui par la suite s'incorpore dans l'interface graphique, et peut être sélectionné comme point de départ de nouvelles découvertes et attaques. A l'aide de Wireshark, nous avons observé des attaques de CORE IMPACT contre des systèmes Windows XP SP1 et SP2. Par exemple, nous avons testé les failles MS03-026 (MSRPC DCOM) et MS06-040 (MSRPC SRVSVC NetrpPathCanonicalize) contre XP SP1 et la faille MS06-066 (MSRPC Netware Client) contre XP SP2. Côté client, nous avons testé les failles Windows ANI (MS07-017) et Buffer Overflow dans Adobe Reader 8.1.0 (CVE-2007-5659).

Après l'attaque réussie contre le service vulnérable, une connexion est établie par le stage loader vers le client CORE IMPACT. Ensuite commence le chargement du code de l'agent par le service compromis. Ces deux phases se retrouvent sur la capture d'écran de wireshark présentée en figure 5, produite lors d'une attaque de l'hôte Windows XP SP2 sur la vulnérabilité MS06-066.

En étudiant le flux TCP capturé par Wireshark, on observe le code de l'agent en clair, on peut voir une série de messages de diagnostique interne de l'agent. Toutefois ces messages ne sont pas spécifiques à l'agent CORE IMPACT, mais plutôt des messages standards de fonctions C. Nous

allons donc chercher une signature spécifique, et il semble que le tout début de l'injection soit un bon candidat :

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any \
  (msg:"CORE IMPACT agent loader (WIN32)"; \
    content:"|51525033D25268|send|8BC45268|recv"; )
```



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

FIG. 5: Capture d'une attaque de CORE IMPACT.

IMPACT 7.5 → Linux. De la même façon que lors d'une attaque contre un hôte Windows, CORE IMPACT déploie un agent lors de l'exploitation réussie d'une vulnérabilité sur un hôte Linux. Cet agent est évidemment différent de celui injecté dans le système Windows. C'est pourquoi il est nécessaire de définir une signature différente du cas précédent.

Nous avons observé à l'aide de Wireshark les deux phases d'une attaque contre un hôte Linux (Debian 4.0r1) lors d'attaques contre les failles dans le préprocesseur DCE/RPC de Snort 2.6.1.2 (CVE-2006-5276) et dans la fonction lsa_io_trans_names de Samba 3.0.24 (CVE-2007-2446). Nous avons également testé les exploits contre les failles CVE-2007-0245 (OpenOffice RTF Prtdata tag) et CVE-2007-3387 (Xpdf-libpoppler StreamPredictor). Ici aussi on observe un code binaire en clair, contenant des messages de diagnostique. Nous allons utiliser les premiers octets du code de l'agent pour la signature de ce cas d'attaque.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any \
  (msg:"CORE IMPACT agent loader (LINX86)"; \
    content:"|89E552680100000054E80000|"; )
```

CANVAS → Windows. CANVAS contient des fonctions de test automatique d'intrusion, et un système d'agent très intéressant. En effet, celui-ci est basé sur MOSDEF (pour Most Definitely) [13], un assembleur multi-plateforme implémenté en Python. Ainsi, lors d'une attaque réussie, c'est un

serveur MOSDEF qui est injecté sur la machine compromise, adapté au système d'exploitation (Windows, Linux, BSD...). Ensuite, le client dialogue avec ce serveur MOSDEF et peut donc en théorie faire exécuter n'importe quel code assembleur sur la machine compromise. En particulier, le serveur MOSDEF peut servir de rebond pour utiliser toutes les fonctionnalités de CANVAS à partir d'une machine compromise.

Avec Wireshark, nous étudions les paquets échangés immédiatement après une compromission d'hôtes sous Windows XP SP1 et SP2 (mêmes failles que dans le cas de CORE IMPACT : MS03-026, MS06-040, MS06-066). Côté client, nous avons testé l'exploit contre MS07-004 (Windows VML RecolorInfo). Dans les traces, nous observons des noms de librairies dynamiques et de fonctions de l'API Windows (figure 6). C'est la compilation effectuée à la volée par MOSDEF qui provoque le chargement de différentes fonctions de l'API Windows au fur et à mesure de l'exécution de commande à partir de l'interface de CANVAS.

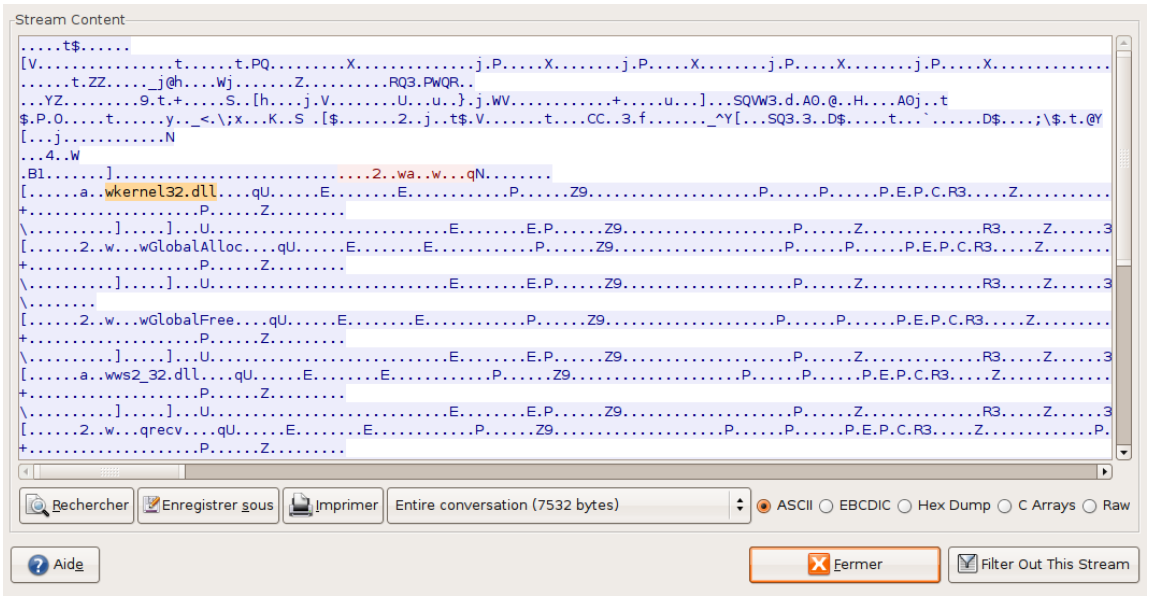


FIG. 6: Capture d'une conversation MOSDEF Win32.

Il semble que les noms de fonctions apparaissant ici soient utilisés lors de l'initialisation du serveur MOSDEF. Il est donc certain qu'ils apparaitront à chaque compromission réussie d'une cible sous Windows. Nous allons donc définir une signature Snort à partir des mots-clés observés dans la capture.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any \
  (msg:"CANVAS MOSDEF server initialisation"; \
    content:"|E577|kernel32.dll"; )
```

CANVAS → Linux. Nous avons observé les échanges ayant lieu à la suite d'une attaque contre Debian 4.0r1 exploitant la faille DDCE/RPC dans Snort 2.6.1.2 (nous avons dû modifier l'exploit pour qu'il fonctionne sur une Debian), et côté client l'exploit CVE-2007-4575 (OpenOffice Database 2.3.0 static java execution). L'initialisation du serveur MOSDEF pour Linux est moins évidente que le pendant pour Windows. En effet, plutôt que de faire appel à des librairies comme dans le cas de Windows, ici le code injecté effectue directement des appels système et il est donc difficile d'identifier des chaînes de caractères. Nous avons donc choisi une chaine extraite des premiers octets échangés entre la console et la machine compromise qui correspond à une ouverture de `/dev/zero`.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any \
  (msg:"CANVAS MOSDEF server initialisation"; \
    content:"joh/zerh/dev"; )
```

Détection avec Bro IDS. En plus de Snort, nous avons également envisagé la détection de ces outils avec le NIDS Bro. Le script suivant intègre les signatures fournies dans les paragraphes précédents pour les attaques contre les systèmes Windows :

```
@load conn
module hot_stuff;

redef tcp_content_deliver_all_orig = T;
redef tcp_content_deliver_all_resp = T;

global log_file = open_log_file("hot_stuff") &redef;

const hot_stuff: table[string] of pattern = {
  ["CORE IMPACT agent loader"] =
    /\x51\x52\x50\x33\xd2\x52\x68send\x8b\xc4\x52\x68recv/,
  ["CANVAS MOSDEF server initialisation"] =
    /\xe5\x77kernel32.dll/,
  ["Metasploit agent injection"] = /metsrv.dll\x00MZ/,
} &redef;

event tcp_contents(c: connection, is_orig: bool,
  seq: count, contents: string)
  {
    for ( pa in hot_stuff )
      if ( hot_stuff[pa] in contents )
        print log_file, fmt("Hot stuff: %.6f %s : %s",
c$start_time,id_string(c$id), pa);
  }
```

Test en conditions réelles Nous avons ajouté les règles de détection définies pour Snort à une sonde opérant sur une interconnection de réseaux. Après une observation d'un mois et demi, nos règles n'ont jamais été déclenchées. Ceci conforte notre postulat de départ selon lequel il est possible d'écrire des règles spécifiques aux agents déployés par les outils de test d'exploitation automatisé, qui ne produisent donc pas de faux positifs.

3.2 Détection niveau hôte

La détection niveau hôte s'appuie sur la collecte d'évènements système (typiquement des appels système) pour repérer des traces d'exécution d'un agent. Comme expliqué plus haut, il est pertinent de s'attacher à détecter la phase d'exécution d'un agent car celui-ci est à la fois révélateur d'une intrusion réussie et spécifique à l'outil employé pour l'attaque.

Les paragraphes suivant vont détailler pour chaque système d'exploitation considéré le mécanisme de collecte de traces système envisagé, puis les séquences d'évènements révélatrices de l'exécution d'un agent CORE IMPACT ou CANVAS.

Système hôte Windows La collecte de traces d'exécution sous Windows se fait généralement au niveau de l'API Windows [15], c'est-à-dire l'ensemble des librairies contenant les fonctions standard utilisées par les programmes de type Win32 pour accéder aux appels systèmes. En effet, les appels systèmes sont rarement utilisés directement dans le code exécutable. De plus, il est beaucoup plus facile d'instrumenter l'API Windows que le noyau, en effet le noyau du système de Microsoft est très peu documenté publiquement, et les appels systèmes reçus par le noyau reçoivent généralement un grand nombre d'arguments difficilement interprétables sans le point de vue de l'API Win32 [16].

L'outil que nous avons choisi pour tracer l'exécution des processus sous Windows et collecter les évènements intéressants est Microsoft Detours [17]. D'autres outils existent, comme par exemple NtTrace [18], straceNT [19], strace for NT [20], mais ces outils ont l'inconvénient de ne tracer qu'un seul processus à la fois, tandis que Detours peut être configuré de façon à être chargé pour tous les processus. Un autre avantage de Detours est d'être fourni directement par l'éditeur du système d'exploitation, on peut donc envisager son déploiement sans trop inquiéter les administrateurs.

La principale difficulté est de déterminer l'ensemble des fonctions à surveiller. En effet, il n'est pas envisageable de tracer l'ensemble de l'API Win32 car le mécanisme de Detours utilise alors toute la ressource processeur et la machine ne répond plus. De plus le code source correspondant à une surveillance de la totalité des fonctions est vraiment très long. Il est donc essentiel de restreindre la portée de la collecte d'évènements. L'ensemble de librairies dynamiques retenu est assez typique : NTDLL, KERNEL32, ADVAPI, USER32, WS2_32, WININET. Il a été déterminé à partir des librairies utilisées par les agents CORE IMPACT et CANVAS. Ensuite, la liste des fonctions que nous avons tracées est indiqué en annexe.

Système hôte Linux Dans le cas d'un hôte Linux, on dispose de davantage d'outils pour l'instrumentation du noyau par rapport à un hôte Windows. Nous envisageons donc la collecte d'évènements système directement au niveau du noyau, par l'observation des *syscalls*. De plus, ceci est motivé par le fait que les *shellcodes* effectuent directement des appels système, sans passer par des fonctions de librairies.

Un ensemble de fonctions d'instrumentation du noyau Linux, appelé `kprobes`, est directement intégré depuis la version 2.6.9. L'avantage de ces outils est que l'on dispose d'un langage de haut niveau, `systemtap` [21], pour écrire des scripts d'instrumentation des différents sous-systèmes du noyau [22]. Par rapport à Microsoft Detours, ceci est très intéressant car il n'est plus nécessaire de produire un code complet en langage C.

Une autre possibilité existe pour tracer les appels système effectués, il s'agit du système d'audit du noyau Linux [23]. Il est accompagné d'un daemon auditd qui accepte des règles sur les évènements à auditer, notamment concernant les appels système. Ce système d'audit a été développé initiale-

ment en rapport avec SELinux, il est donc également possible de le piloter via la configuration SELinux pour ce qui concerne les appels système.

Ces deux types de solutions, `kprobes` ou `auditd`, sont intégrées par les distributions Linux majeures comme RedHat, Suse et Debian/Ubuntu. Il est donc tout à fait envisageable de les mettre en oeuvre sans remettre en cause les contrat de maintenance.

Comme pour le système Windows, une surveillance de l'ensemble des appels système avec kprobes serait beaucoup trop lourde. Il faut donc sélectionner une liste suffisante d'appels à tracer pour pouvoir détecter l'exécution d'un agent CORE IMPACT ou CANVAS.

Système hôte Solaris Pour la détection au niveau hôte sous Solaris, nous avons envisagé d'utiliser l'outil `dtrace` [24]. Il rend possible l'écriture de scripts pour tracer les appels aux fonctions du noyau Solaris. Il est disponible dans le système Solaris 10, et les sources peuvent être consultées dans le projet Opensolaris. En outre, l'utilisation de `dtrace` à des fins de détection d'activités malicieuses a déjà été envisagée, par exemple dans [25].

Analyse des traces Il reste à déterminer des signatures correspondant à des parties significatives de l'exécution des agents CORE IMPACT et CANVAS. La solution retenue pour le moment est d'utiliser une signature correspondant à l'initialisation de l'agent. Toutefois cette partie n'a pas encore été finalisée.

L'utilisation de signatures "statiques" présente l'inconvénient de correspondre à un cas particulier d'exécution, alors qu'il se pourrait très bien que l'exécution d'un même code dans deux contextes différents produise des séquences d'appels de fonctions différents. Cette détection pourrait être améliorée en utilisant les techniques actuelles de détection de comportements malicieux via l'analyse dynamique de binaire, par exemple les techniques d'analyse de N. Netercote reposant sur l'outil Valgrind [26].

4 Contre-mesures

La première méthode pour mettre en échec ces outils de test d'exploitation automatisé reste bien évidemment la mise à jour des systèmes et des applications. La proportion d'exploits 0day au sein de ces outils (notamment CANVAS) reste mineure.

Ensuite il est envisageable de bloquer ces attaques à partir de leur détection, via des mécanismes de type IPS (prévention d'intrusion). En effet, nous avons défini des signatures au format de Snort pour ces différents outils, elles peuvent être mises en oeuvre dans un IPS au niveau réseau. De la même façon, les moyens de détection au niveau système peuvent être utilisés afin de bloquer ou de confiner l'exécution d'un agent.

Par exemple, on peut envisager, avec une détection en temps réel de l'exécution d'un agent, le déroutage de son exécution dans un environnement de type sandbox. A partir de là, l'agent continuerai de s'exécuter mais dans un environnement simulé.

Une dernière possibilité envisageable est celle de la contre attaque visant à provoquer un déni de service sur l'outil de test d'intrusion. Ceci est d'autant plus envisageable que ces outils ne sont pas exempts de vulnérabilités. Par exemple, la vulnérabilité récente dans la librairie Winpcap [27] a impacté directement tous les outils qui l'utilisent.

De façon plus générale, on peut considérer qu'un ordinateur exécutant un outil de test d'intrusion est bien plus exposé aux attaques qu'un autre. En particulier, le pare-feu et l'antivirus devront être désactivé pour les utiliser.

Bien sûr, dans le cas où la source de l'attaque est externe au domaine de responsabilité du responsable sécurité, cette possibilité soulève des difficultés juridiques rédhibitoires [28].

5 Limites

Cet article présente des moyens de détection et de protection contre une utilisation "naïve" de ces outils. En effet, un auditeur expérimenté ou un attaquant soucieux de furtivité ne sera pas impacté par ces mesures :

- Les signatures réseau sont potentiellement contournables pour un auditeur expérimenté. Toutefois, cela demande de toucher aux fichiers internes des outils, qui peuvent être du code source ou des parties binaires. Si l'on est capable de faire ce type de modifications, alors on peut échapper à la détection par signature ;
- L'auditeur expérimenté n'utilise pas forcément ces outils, il en a bien d'autres à sa disposition voire même des outils développés par ses propres soins et en fonction du système d'information à auditer.

Concernant la détection au niveau système, elle pose très certainement des problèmes de performances et d'administrabilité. Elle pourrait même remettre en cause le maintien du support éditeur suivant la solution retenue, notamment dans le cas de modifications du noyau du système. Mais comme cela a été signalé, les solutions retenues dans cet article, soit Microsoft Detours pour Windows, systemtap ou SELinux pour Linux et dtrace pour Solaris, sont totalement supportées par leurs éditeurs.

Enfin, des évolutions de ces produits peuvent remettre en question les résultats de l'article, notamment les signatures : l'intégration de chiffrement du code, de polymorphisme ou de toute technique d'obfuscation utilisée par les virus informatiques. Mais étant donné que le but de ces outils n'est pas d'être furtif, il est peu probable que ce type d'évolution soit envisagé.

6 Conclusion

Le marché du test d'intrusion automatique, voire clé-en-main, s'est fortement développé ces dernières années. Si ces outils sont au départ conçus pour l'évaluation du niveau de sécurité d'un réseau, ils représentent toutefois une menace importante : d'une part les risques liés aux utilisations légitimes mais incorrectes, qui introduisent des faiblesses dans le système d'information, et d'autre part lorsqu'ils sont acquis par des utilisateurs malveillants pour une utilisation offensive.

Nous avons donc recherché des moyens de protection contre ces risques introduits par l'utilisation de tels outils. Afin de faciliter l'enrichissement de la bibliothèque d'attaques, ces outils intègrent des plateformes génériques de développement de code d'exploitation de vulnérabilités. Ce modèle de conception rend envisageables des méthodes de détection génériques de ces outils. De plus, ces outils présentent différentes phases dans l'exploitation d'une vulnérabilité, nous avons donc considéré la phase de téléchargement du code de l'agent, qui ne se déclenche que dans le cas où l'exploitation est réussie. Nous avons ainsi pu proposer des signatures qui, en théorie, ne déclenchent pas de faux-positifs. Enfin, ces signatures donnent une information fiable sur l'outil qui a été utilisé par l'attaquant.

Dans un second temps, nous avons envisagé des approches possibles pour mettre en place des contre-mesures. Enfin, nous avons considéré les limites de notre approche. Si les solutions proposées dans notre étude ne peuvent bloquer un auditeur (ou un agresseur) expérimenté, elles l'obligeront

toutefois à ne pas se contenter des possibilités de ces outils. Dans le cas d'un auditeur, son évaluation du niveau de sécurité n'en sera que plus pertinente [29].

Aujourd'hui, la disponibilité de tels outils d'intrusion met à la portée d'un public large des attaques extrêmement sophistiquées et en profondeur. Il nous parait nécessaire que les acteurs de la sécurité soient conscients des risques qu'ils représentent.

Références

1. Renaud Deraison : Compromettre son réseau en l'auditant ? In : SSTIC05. (June 2005)
2. Renaud Bidou : Outils de scan. MISC Hors-Série 1 (October 2007)
3. Bill Cheswick : The Design of a Secure Internet Gateway. In : USENIX Summer 1990 Conference. (1990)
4. Immunity : Immunity CANVAS Early Updates. `http://www.immunitysec.com/partners-index.shtml` (2008)
5. Argeniss : Argeniss 0day Pack. `http://www.argeniss.com/` (2008)
6. DSquare Security : D2 Exploitation Pack. `http://d2sec.com/products.htm` (2008)
7. Jeffrey J. Carpenter : Computer Security Issues that Affect Federal, State, and Local Governments and the Code Red Worm. `http://www.cert.org/congressional_testimony/Carpenter_testimony_Aug29.html` (August 2001)
8. Eric Filiol : Guerre de l'information : guerre, guérilla et terrorisme informatique : fiction ou réalité ? MISC 33 (September 2007)
9. Robert Graham, David Maynor : A Simpler Way of Finding 0day. In : Black Hat USA 2007. (July 2007)
10. Core Security : CORE IMPACT. `http://www.coresecurity.com/products/coreimpact/` (2008)
11. Immunity : Immunity Canvas. `http://www.immunitysec.com/products-canvas.shtml` (2008)
12. Metasploit : The Metasploit Framework. `http://framework.metasploit.com/` (2008)
13. David Aitel : MOSDEF. In : Black Hat USA 2004. (July 2004)
14. H D Moore : Metasploit 3.0 Automated Exploitation. `http://blog.metasploit.com/2006/09/metasploit-30-automated-exploitation.html` (2006)
15. Ivo Ivanov : CodeProject : API hooking revealed. `http://www.codeproject.com/KB/system/hooksys.aspx` (April 2002)
16. Mark Russinovich, Bryce Cogswell : Windows NT System-Call Hooking. `http://www.ddj.com/184410109` (January 1997)
17. Galen Hunt, Doug Brubacher : Detours : Binary Interception of Win32 Functions. In : 3rd USENIX Windows NT Symposium, Seattle, USENIX (July 1999) 135–143
18. Roger Orr : NtTrace - Native API tracing for Windows. `http://www.howzatt.demon.co.uk/NtTrace/`
19. Pankaj Garg : StraceNT - A System Call Tracer for Windows. `http://www.intellectualheaven.com/default.asp?BH=projects&H=strace.htm`
20. Bindview Security Research : Strace for NT. `http://packetstormsecurity.org/NT/strace-0.3.zip`
21. Red Hat, IBM, Intel, Hitachi : SystemTap. `http://sourceware.org/systemtap/`
22. Nicolas Greneche, Mathieu Blanc : Systemtap. MISC **27** (September 2006)
23. Steve Grubb : Linux Audit System. `http://people.redhat.com/sgrubb/audit/`
24. Sun Microsystems : Dtrace. `http://www.sun.com/bigadmin/content/dtrace/`

25. David Weston, Tiller Beauchamp : DTRACE : The Reverse Engineer's Unexpected Swiss Army Knife. In : Black Hat Europe 2008. (March 2008)
26. Nicholas Nethercote : Dynamic Binary Analysis and Instrumentation. Doctor of philosophy, University of Cambridge (2004)
27. CVE : WinPcap NPF.SYS bpf_filter_init Arbitrary Array Indexing Vulnerability. `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5756` (2007)
28. Marie Barel : Fraude informatique et preuve : la quadrature du cercle ? In : SSTIC05. (June 2005)
29. H D Moore, Valsmith : Tactical Exploitation. In : Black Hat USA 2007. (July 2007)

A Listings des fonctions tracées

A.1 API Windows

```
kernel32.dll
    Sleep
    QueryPerformanceCounter
    LoadLibraryA
    LoadLibraryW
    LoadLibraryExA
    LoadLivraryExW
    GetProcAddress
    GetLastError
    ExitProcess
    ExitThread
    ReadFile
    WriteFile
    CreateFileA
    CloseHandle
    SetFilePointer
    GetCurrentProcess
    DuplicateHandle
    DeleteFileA
    MoveFileA
    SetCurrentDirectoryA
    CreateDirectoryA
    RemoveDirectoryA
    GetCurrentDirectoryA
    OpenProcess
    TerminateProcess
    GetCurrentProcessId
    VirtualAlloc
    VirtualFree
    GetStdHandle
    GetLocalTime
    SetLocalTime
    SystemTimeToFileTime
```

```
    FileTimeToSystemTime
    GetTimeZoneInformation
    FindFirstFileA
    FindNextFileA
    FindClose
    GetFileAttributesExA
    CreateThread
    CreateProcessA
    OutputDebugStringA
    WaitForSingleObject
    InitializeCriticalSection
    DeleteCriticalSection
    EnterCriticalSection
    LeaveCriticalSection

ws2_32.dll
    WSAStartup
    socket
    WSASocketA
    bind
    connect
    listen
    accept
    send
    recv
    sendto
    recvfrom
    getpeername
    getsockname
    getsockopt
    setsockopt
    ioctlsocket
    closesocket
    shutdown
    gethostname
    select

advapi32.dll
    GetUserNameA
```

A.2 Linux syscalls

```
3   : read
4   : write
5   : open
6   : close
```

```
90  : mmap
102 : socketcall
```