Dossier

Sécuriser ses serveurs avec les tests de vulnérabilités

Jérémy Renard 

Degré de difficulté

Pour faire face aux différents types d'attaques, il est essentiel de se constituer de véritables remparts autour de nos serveurs. Malheureusement, nous savons aussi qu'aucun mur n'est imperméable... C'est pourquoi cet article se propose de renforcer notre château lui-même, soit nos machines, à l'aide des tests de vulnerabilités.

Le but de cet article est de trouver la bonne démarche pour, dans la mesure du possible, faire de nos systèmes des forteresses les plus inacessibles pour une personne malicieuse.

Nous commencerons par poser le décor de la bataille en explicitant les différents contextes possibles pour effectuer des tests de vulnérabilités pour, en fin de compte, adopter la bonne stratégie. Ensuite, nous devrons choisir les meilleures armes. Enfin, nous les affûterons comme il se doit pour obtenir le résultat escompté : la victoire sur l'attaquant en le devançant au mieux.

Les stratégies

Les tests de vulnérabilités peuvent être rencontrés dans deux cas principalement :

- en *stand-alone* ; nous cherchons à évaluer le niveau de sécurité d'un serveur,
- dans le cadre d'un test d'intrusion ; il s'agit alors de la phase d'identification des failles qui intervient après la recherche globale d'information mais avant l'exploitation et la compromission de la machine auditée.

Ensuite, nous pouvons distinguer trois stratégies pour dérouler un test :

- la mise en production d'un serveur ; effectivement, votre réseau a beau être le plus sûr du monde, y introduire une seule machine non fiable peut suffire à le compromettre totalement. Il est donc nécessaire dans un premier temps de placer la machine dans un environnement de test, d'effectuer les tests de sécurité et de la placer enfin sur le réseau

Cet article explique...

- Comment lancer, paramétrer et affiner un test de vulnérabilité.
- Comment analyser et interpréter les résultats d'un tel test.
- Comment faire évoluer son scanner.

Ce qu'il faut savoir...

- Le fonctionnement intrinsèque d'un scanner.
- En quoi consiste une vulnérabilité.
- Le langage C (bases).

de production si et seulement si son niveau de sécurité est satisfaisant,
- les tests récurrents ; il faut savoir qu'un test déroulé à un instant *t* ne peut assurer la sécurité de la machine qu'au temps *t* ! Nous comprenons alors pourquoi il est indispensable de réaliser des tests de manière récurrente. Nous limitons ainsi d'autant le risque facteur temps,
- les tests spontanés ; lancés par un attaquant lors d'une tentative ou plannifiés à une date quelconque sur décision de l'équipe sécurité de l'entreprise, un test spontané est un bon moyen de vérifier l'état de ses machines à un moment donné. Il n'est jamais possible de contrôler les actions qui ont eu lieu depuis la mise en marche d'un hôte – ou depuis le dernier test. Or, il suffit d'ajouter une application comportant une faille ou d'un manque de vigilance pen-dant ce laps de temps pour que le niveau de sécurité ne soit plus aussi haut que prévu.

Alors quelle stratégie adopter ? Certainement les trois ! En effet, combiner ces trois approches ne pourra être que bénéfique.

Quoiqu'il en soit, nous savons maintenant que si nous souhaitons avoir un niveau de sécurité correct sur nos machines, nous ne pourrons pas nous passer des tests de vulnérabilités car tout test de sécurité système se doit de rechercher les failles. Mais de quelle manière rechercher ces potentielles vulnérabilités ? Comme dans tout audit, la recherche peut reposer sur trois niveaux d'information différents :

- les tests de type *boîte noire* ; cela concernera certainement un pirate qui cherche à trouver la faille lui permettant d'atteindre notre système. Il peut aussi s'agir d'un auditeur externe à l'entreprise qui lance un test en aveugle pour simuler le comportement de l'attaquant. Le minimum d'information – voire pas du tout – sera fourni dans ce cas.
- les tests de type *boîte blanche* ; cette fois, soit l'attaquant est un ancien employé de l'entreprise ou le but de l'audit est de trouver le maximum de failles. Le testeur pourra alors avoir accès à toutes les données qu'il souhaite.
- les tests de type *boîte grise* ; seules les informations de base seront données mais celles-ci devraient permettre d'obtenir les résultats principaux.

Notre stratégie est prête mais l'aventure ne fait que commencer car il nous faut maintenant découvrir la nature des problèmes pouvant être rencontrés pour mieux savoir les combattre.

Attention à nos faiblesses !

Nous pensons en général à nous défendre du camp adverse mais ici,

Listing 1. *Exemple d'utilisation d'un scanner en mode passif, P0F*

```
p0f -i eth0 -vt
p0f - passive os fingerprinting utility, version 2.0.8
(C) M. Zalewski <lcamtuf@dione.cc>, W. Stearns <wstearns@pobox.com>
p0f: listening (SYN) on 'eth0', 262 sigs (14 generic, cksum 0F1F5CA2),
     rule: 'all'.

<Sat Apr 7 20:56:51 2007> 192.168.0.102:54222 - Linux 2.6 (newer, 1)
     (up: 5 hrs)
   -> 194.116.144.27:21 (distance 0, link: ethernet/modem)
<Sat Apr 7 20:58:15 2007> 192.168.0.102:49895 - Linux 2.6 (newer, 1)
     (up: 5 hrs)
   -> 64.233.183.99:80 (distance 0, link: ethernet/modem)
+++ Exiting on signal 2 +++
[+] Average packet ratio: 1.10 per minute.
```

Listing 2. *Exemple d'utilisation d'un scanner spécialisé, HTTPRINT (extrait)*

```
$ httprint -h 192.168.0.102 -s signatures.txt -P0
httprint v0.301 (beta) - web server fingerprinting tool
(c) 2003-2005 net-square solutions pvt. ltd. - see readme.txt
http://net-square.com/httprint/
httprint@net-square.com

Finger Printing on http://192.168.0.102:80/
Finger Printing Completed on http://192.168.0.102:80/
--------------------------------------------------

Host: 192.168.0.102
Derived Signature:
Apache/2.2.3 (Aurox)
811C9DC5E2CE6925811C9DC5811C9DC5811C9DC5505FCFE8811C9DC5811C9DC5
0D7645B5811C9DC5811C9DC5811C9DC5811C9DC5811C9DC5811C9DC5
E2CE6925E2CE6925E2CE6925811C9DC5E2CE6925811C9DC5E2CE6925811C9DC5
E2CE6925E2CE6925811C9DC5E2CE6925E2CE6925E2CE6925E2CE6925E2CE6923
E2CE6923E2CE6925811C9DC5E2CE6925E2CE6923

Banner Reported: Apache/2.2.3 (Aurox)
Banner Deduced: Apache/2.0.x, Apache/1.3.27, Apache/1.3.26,
     Apache/1.3.[4-24], Apache/1.3.[1-3], Apache/1.2.6
Score: 33
Confidence: 19.88
------------------------

Scores:
Apache/2.0.x: 33 19.88
Apache/1.3.27: 33 19.88
cisco-IOS: 30 14.15
Apache-Tomcat/4.1.29: 28 11.01
Microsoft-IIS/6.0: 20 2.92
------------------------
```

il ne faut pas oublier que la source du problème vient de l'intérieur même de notre réseau. Alors mieux vaut connaître la nature des maillons faibles car ils déterminent le niveau de sécurité globale de nos serveurs.

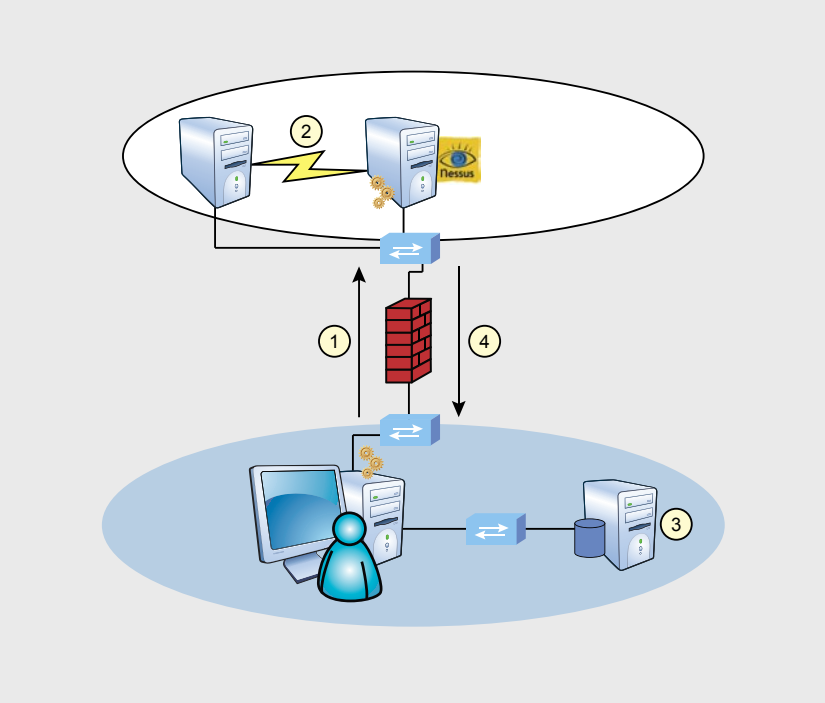


Figure 1. *Architecture de NESSUS et interactions*

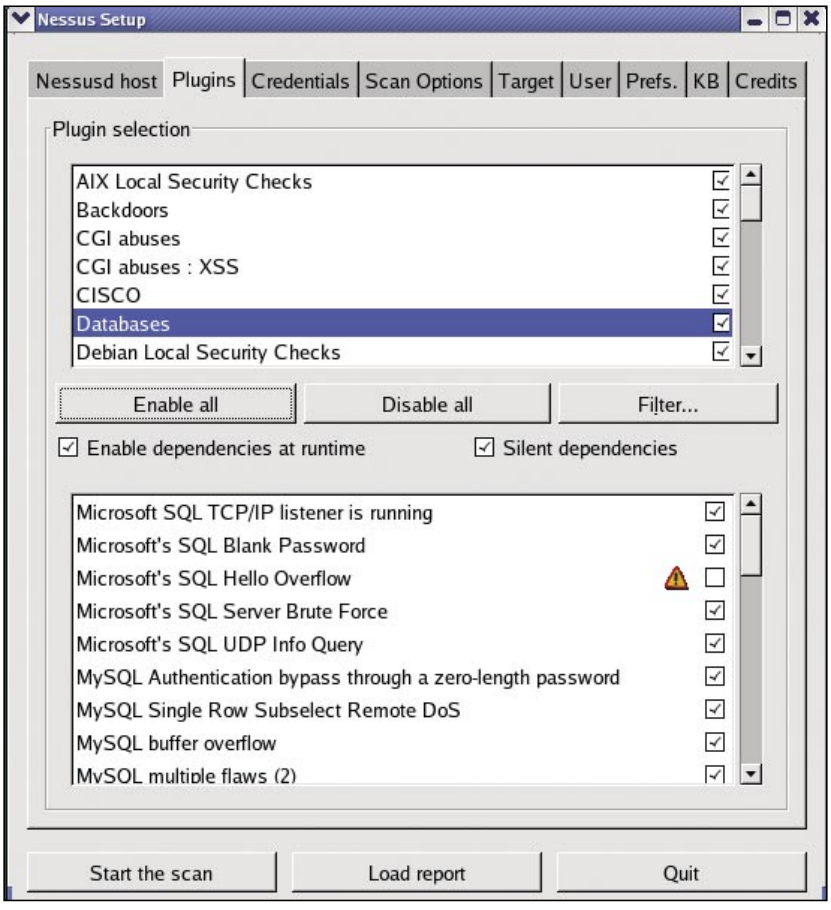


Figure 2. *Configurer NESSUS : bien choisir les plugins*

Ces vulnérabilités qui représentent nos points faibles naissent souvent d'une négligence d'un programmeur. Elles sont aussi souvent la conséquence d'une application qui devient obsolète et donc de moins en moins sécurisée avec le temps (par exemple, une vieille fonction).

Un algorithme peut finir par être la source de vulnérabilités lui aussi, notamment quand il s'agit de chiffrement (par exemple, un algorithme reposant sur une clé de 128 bits nous protégeait hier mais nous met en danger aujourd'hui). Nous en savons désormais plus sur les dangers mais de quoi s'agit-il exactement ? En fait, ils peuvent prendre plusieurs formes : un moyen de contournement, d'infiltration, de gain de privilèges, etc...

Mais heureusement, il existe plusieurs façons de le découvrir : lors des bulletins de sécurité émis par les éditeurs, grâce aux sites de veille de vulnérabilités (ex : CVE), par un *P0C*, un *full-disclosure*, etc... Nous remarquons qu'il est donc nécessaire de passer en revue régulièrement nos troupes.

La bataille peut donc prendre des formes très diverses et les découvertes peuvent être de différentes natures et surtout complexes. Ces premières connaissances en main, il nous faut cependant aller bien plus loin afin d'être parés à toute éventualité.

Épisode 1 : préparer son attaque

Avant d'étudier les vulnérabilités, nous avons besoin d'une arme capable de les trouver. Sans surprise, il s'agira d'un scanner de vulnérabilités. Mais il en existe de nombreux ! L'erreur serait de choisir le premier venu et/ou d'utiliser seulement son fonctionnement basique : alors gardons l'esprit logique qui nous ménera au but...

Le choix des armes...

Le terme de scanner de vulnérabilités désigne un grand choix de types d'outils. Il en existe de différentes sortes qui nous fourniront des résultats différents et surtout, d'une manière particulière. La liste qui suit ne prétend pas être exhaustive mais

ouvre un large horizon de ce qu'il est possible de faire.

Les scanners passifs ont l'avantage certain de ne pas générer de trafic. Ils sont juste en écoute sur une interface de notre machine pour récupérer des informations sur la cible. Mais conséquence, ils ne pourront relever que les données qu'ils voient passer. Un bon exemple est certainement P0F. Outil Open Source de prise d'empreintes, il est aussi assimilé à un premier niveau de scanner. Le Listing 1 montre un exemple de données récoltées à partir desquelles nous pourrons déduire certaines failles. Si nous pouvons voir le système d'exploitation et les ports scannés après avoir lancé un navigateur WEB (port 80) et un serveur FTP (port 21) sur la machine de test, il sera en revanche difficile de trouver bon nombre d'applications moins utilisées et encore moins leur version. C'est tout de même un premier pas car dans notre exemple, nous avons repéré un serveur FTP (port 21). Or, si nous trouvons un exploit générique aux serveurs FTP qui marche pour notre machine, c'est que la cible est vulnérable. Les scanners actifs, eux, lancent des tests pour déterminer la prise d'empreintes et nous fournissent ainsi un meilleur niveau d'information. Ce qui implique que nous trouverons plus facilement les vulnérabilités. L'exemple le plus connu est sans conteste NMAP. Largement utilisé et abordé dans différents articles, nous ne détaillerons pas son fonctionnement ici. À noter tout de même que NMAP peut effectuer une prise d'empreintes de quatre manières différentes :

- En récupérant la bannière de l'application,
- Sinon, en comparant la bannière avec un fichier de référence,
- Sinon, en testant une implémentation différente (Exemple, HTTP 1.1 au lieu d'HTTP 1.0),
- Sinon, en utilisant SSL.

La commande qui nous intéresse pourrait être tout simplement :

```
$ nmap 192.168.0.102 -p 1-65535
```

Nous cherchons ainsi à trouver les ports ouverts et toutes les applications présentées sur la machine auditée afin de trouver le plus grand nombre de vulnérablités possible.

L'intérêt d'un tel outil est de fournir un grand nombre d'options et ensuite de cibler notre scan sur les ports trouvées. Une autre commande utile serait alors (après avoir découvert les serveurs FTP, SMTP, HTTP, POP et HTTPS avec la première commande) :

```
$ nmap -A -T4 192.168.0.102 -p
      21,25,80,110,443
```

NMAP a aussi l'avantage d'être un outil bien connu en général et alors, les possibilités sont nombreuses pour affiner notre recherche et, par exemple, il a la capacité de se montrer furtif

Tableau 1. *Liste non exhaustive de scanners par type de machine*

Type	Exemples d'outils
Serveur WEB	NIKTO, Acunetix Web Vulnerability Scanner
Serveur WEB 2.0	JIKTO (non disponible à ce jour)
Serveur Base de Données	Shadow Database Scanner
Serveur sans fonction particulière	Shadow Security Scanner
Serveur WIFI	NETSTUMBLER, Wifi Scanner
Serveur Mail	MailScanner
Serveur FTP	FTP Scanner

Tableau 2. *Exemple de vulnérabilités couramment rencontrées*

Serveur	Vulnérabilité	Menace (exemple)
Serveur LINUX	Protocoles non chiffrés installés (rlogin, telnet, ...)	Récupération de mots de passe
Serveur WINDOWS 2003 Server	Vulnérabilité dans le service Terminal Service	Attaque de type *Man-in-the-middle*
Serveur SOLARIS	Configuration SNMP trop permissive par défaut	Accès à certaines données
Serveur MAIL (sendmail)	Vulnérabilité du code permettant un *buffer overflow* à distance	Gain de privilèges (droits *root*)
Serveur Base de données (ORACLE)	Pas de mot de passe par défaut pour le compte *tnslsnr*	Accès / corruption de la base
Serveur WEB	Portail d'authentification en clair (HTTP)	Récupération de mots de passe
Serveur FTP (Wu-FTPd)	Vulnérabilité du code permettant un *buffer overflow* à distance	Execution de code arbitraire en tant que *root*
Serveur DNS (en DMZ)	Le serveur autorise des requêtes DNS récursives	Attaque de type *cache poisoning*

contrairement à la plupart des scanners actifs.

Il existe aussi les scanners semi-passifs/semi-actifs. Ils représentent un bon compromis pour tirer partie des avantages d'un scanner passif (la furtivité) et d'un scanner actif (obtenir le maximum d'information). Nous pouvons prendre pour exemple sinFP.

Cet outil Open Source lance peu de tests de reconnaissance d'empreintes et seulement à travers des trames TCP/IP de base.

L'intérêt ? Éviter les filtrages des pare-feux et des différents équipements de détection d'intrusions (notamment les IDS et IPS). Ce qui peut s'avérer très utile pour un test externe.

Les scanners très actifs – comme AMAP – se basent sur le principe que peu importe la furtivité, le plus important est d'avoir les informations les plus exactes sur les applications tournant sur le serveur testé.

Évidemment, AMAP ne se limite pas à cela mais c'est dans ce rôle qu'il se démarque. Son efficacité repose sur trois fichiers de référence :

- un fichier de tests (via les commandes de type `HELO` par exemple pour le service SMTP),
- un fichier de réponses (Exemple : comme ^220.*FTP pour le service FTP),
- un fichier dédié au services RPC connus (via les identifiants).

S'il est si important de connaître ces types de scanners, c'est qu'en fonction de la nature du test, l'un sera préféré à l'autre. Imaginons le cas d'un pirate, il cherchera à être plus furtif en général.

Alors, s'il a moins d'information, il prendra le temps sur son ordinateur de trouver la faille qui lui permettra de pénétrer sur le système.

A contrario, un auditeur aura tout intérêt à trouver la moindre vulnérabilité dans le but de ne pas être pris au dépourvu par une d'attaque. Toutes les portes doivent être scellées ! Le deuxième point à prendre en considération, c'est l'environnement du test. Effectivement, que nous soyons l'attaquant ou potentiellement attaqués, nous aurons peut être des équipements de sécurité à franchir pour atteindre la machine ou au contraire le champ libre si nous sommes sur le même sous-réseau. Nous devrons alors réagir différemment.

Cette première étape nous a donc normalement permis de trouver les applications qui peuvent être sources de vulnérabilités mais il est possible d'aller plus loin avec cette fois notre panoplie de scanners spécialisés. Selon le type d'applications et/ou de serveurs à auditer, des scanners particuliers vont nous apporter un niveau d'information encore plus fin.

L'exemple le plus facile est certainement le cas d'un serveur WEB. Un outil tel que HTTPRINT va nous permettre de mieux cerner un serveur HTTP. Le Listing 2 nous montre un extrait des informations recueillies et les résultats déduits et triés par ordre de probabilité (selon le nombre de tests réussis).

Concernant notre exemple de serveur WEB, nous sommes sur le bon chemin dans nos choix d'outil : après avoir découvert le type de ce serveur, l'application correspondante (ici, Apache 2), il nous reste à trouver le bon outil pour trouver les failles pertinentes sur le serveur HTTP testé. C'est NIKTO qui va nous aider. Ce scanner dédié aux serveurs WEB est très précis et comme le montre le Listing 3, de nombreuses vulnérabilités sont ainsi mises en évidence dont voici quelques exemples :

- Pour commencer, le scanner commence par rechercher les méthodes HTTP éventuellement utilisables par un attaquant et teste les restrictions associées. Ici, la méthode `TRACE` – même s'il ne s'agit pas de la plus dangereuse – semble être vulnérable.
- Après, nous voyons que NIKTO cherche à s'authentifier mais le site ne lui en laisse pas la possibilité puisqu'un filtrage IP a été mis

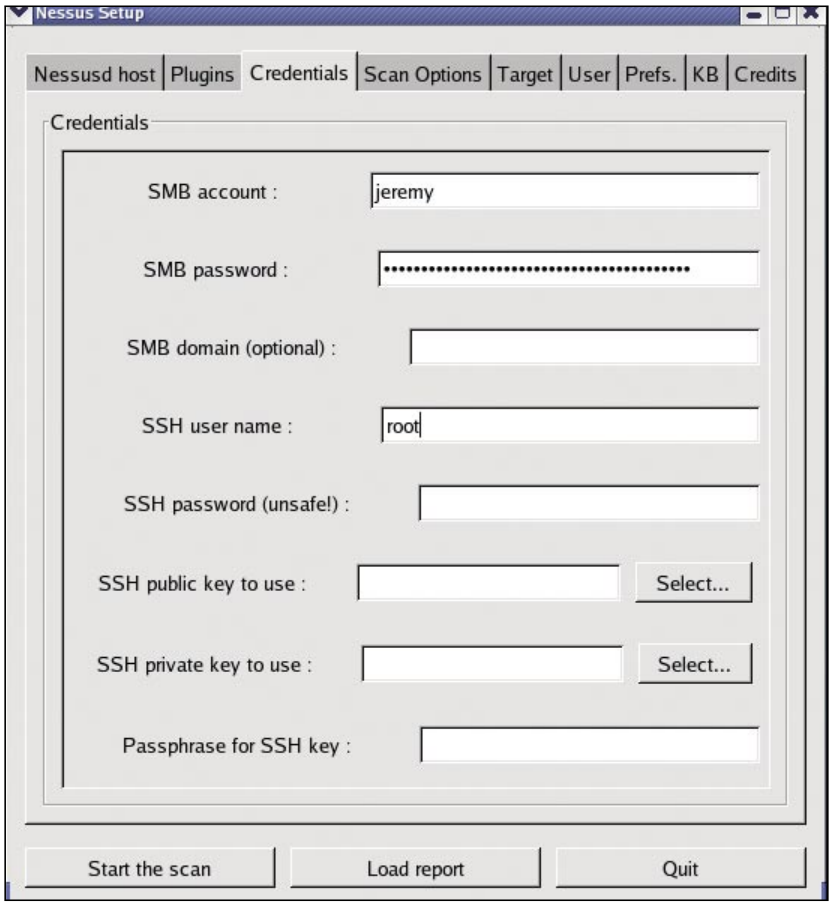


Figure 3. *Configurer NESSUS : simuler un compte interne*

en place. Cependant, un *spoofing* d'adresse IP devrait suffire à passer cet obstacle.

- Ensuite, nous voyons que de nombreuses applications sont reconnues avec, qui plus est, leur version. Il semble ici que PHP Image View 1.0 soit installé et vulnérable à une attaque de type Cross Site Scripting à cause d'un script JAVA trop permissif.
- Enfin, si cela n'apparaît pas ici, NIKTO peut aussi tenter d'accéder au maximum de répertoires de l'arborescence du site WEB pour trouver des informations confidentielles en explorant les CGI, les redirections de pages et autres configurations laxistes des pages HTML.

Pour ceux qui souhaiteraient aller encore plus loin, sachez qu'il existe une librairie nommée libWhisker qui permet de créer ses propres modules (en Perl) pour notamment la création de paquets.

Une idée est de constituer des requêtes malformées volontairement avec différents types de paquets car de nombreuses applications WEB ne supportent pas ce genre de requêtes ou du moins, elles ne sont généralement pas toutes gérées correctement.

Nous venons de voir un exemple de recherche d'outils assez complet mais il existe bien d'autres types de serveurs que nous pourrions rencontrer durant notre quête. Le Tableau 1 se propose de nous donner au moins un exemple d'outil que nous emploierons en fonction de la nature de la machine auditée.

Pour la plupart, la liste fournit soit des outils proposés gratuitement sur Internet (du moins, une version d'évaluation), soit des outils qui ont été proposés sur un CD hakin9 et que vous pourrez donc essayer. Parmi tous les outils que nous avons vu, il en reste un que nous ne pouvons pas oublier : NESSUS.

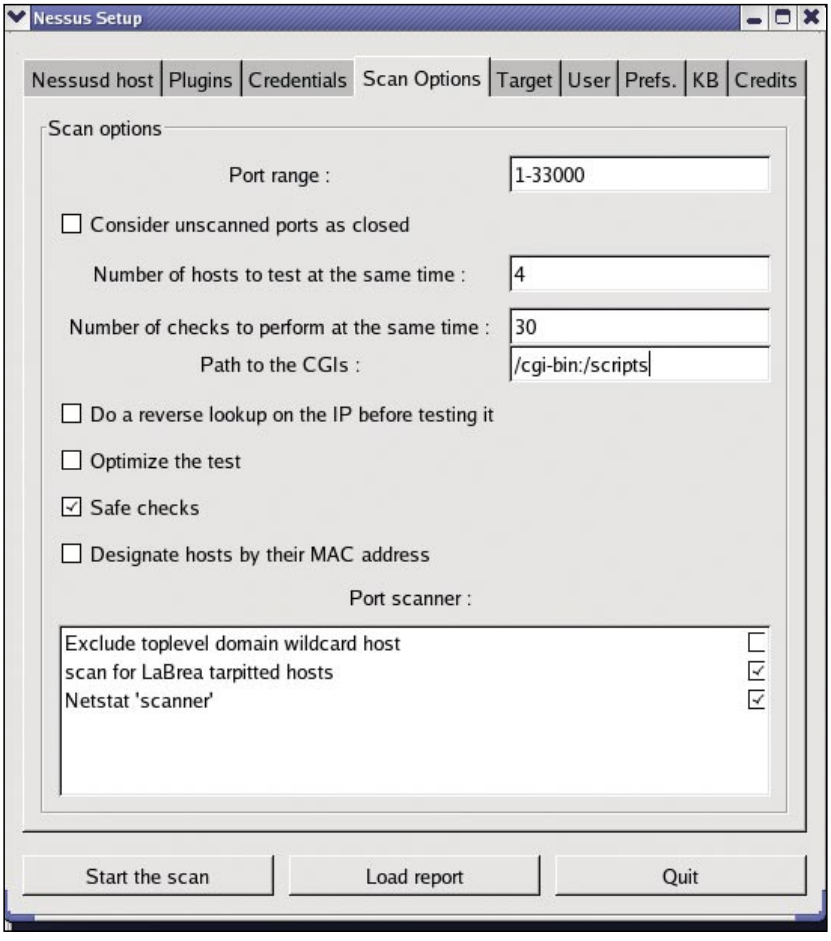


Figure 4. *Configurer NESSUS : affiner les options*

Il s'agit certainement du scanner de vulnérabilités le plus utilisé et il trouvera une place majeure dans notre armurerie. Pourquoi ?

Parce qu'en plus d'être capable de nous fournir les applications et leur version, il peut aussi tester les vulnérabilités associées en jouant certains exploits. De plus, il est capable d'intégrer des modules comme ceux fournis par NIKTO et THC-HYDRA.

Le premier nous fournira, comme nous l'avons vu ci-dessus, des informations essentielles sur un serveur WEB et le second nous permettra de casser les mots de passe trop faibles. Son rôle ne s'arrête pas là : il est aussi en mesure de nous donner des solutions et des recommandations pour corriger les failles.

Nous verrons dans la deuxième phase que les choses ne se passeront pas si simplement mais cet outil nous sera d'une grande utilité dans de très nombreux cas. Pour résumer cette partie, voici une liste de questions à se poser pour préparer correctement son test de vulnérabilités :

- S'agit-il d'un test de mise en production ? Récurrent ? Ou spontané ?
- S'agit-il de tests internes ou externes ? (réseau local, DMZ, Internet)
- S'agit-il d'un test de type *boîte noire* ? *Boite blanche* ? Ou *boîte grise* ?
- Avons-nous surtout besoin d'être furtif ? Exhaustif ? Ou de trouver un compromis ?
- Dans quel environnement travaillons-nous ? (très critique, production, développement, ...)
- À partir des fonctions principales du serveur, quels outils spécialisés sont à notre dispositon ?

Cette check-list remplie, nous devrions de cette façon couvrir la majeure partie des cas et démarrer sur la bonne piste.

La formation

Nous faisons donc ici le choix de détailler NESSUS pour répondre aux besoins de cet article. Dans un premier

Are You

Sure

You're

Secure?

Comprehensive PC security solutions from Ashampoo®

Are you sure
that you're *not* being spied on?

Are you surprised
that your computer often doesn't do what *you* want?

Do you have the feeling
that something is *slowing down* your PC?

Ashampoo® AntiVirus

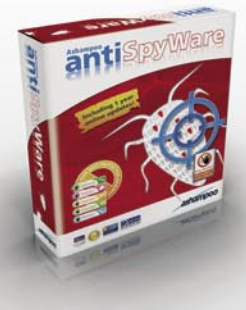
Complete virus protection without system slowdown!
Simple and reliable. Just install it and forget it.

Ashampoo® AntiVirus gives you comprehensive protection against viruses, worms, Trojans and dialers. And it also uses minimum memory and system resources, so that you won't even notice that it's there during your regular everyday work.

Tough on viruses. Easy on users.

29.99 US$

Ashampoo® AntiSpyWare



Zero tolerance for spyware.
Regain full control over your computer.

Featuring new technologies and additional security tools, Ashampoo® AntiSpyWare protects you against the entire spectrum of new malware threats you are exposed to on the Internet, including hijackers, dialers, spyware, worms, adware, Trojans, key loggers and even the treacherous new rootkits.

Blocks threats before they can do any damage.

29.99 US$

Ashampoo® FireWall FREE & FireWall PRO

Full security without gobbledygook for novices and pros.
Our ultimate protection against Internet attacks on your computer.

Ashampoo® FireWall monitors your active Internet connection and automatically blocks the activity of viruses and spyware programs. Among other things, this prevents Trojan Horse programs from turning your computer into a "zombie PC" that hackers can use for sending millions of spam mails with your account.

FREE
FREE

29.99 US$
PRO

www.ashampoo-security.com

temps, commençons par en expliquer son fonctionnement si spécial. Ses capacités évoluées nous permettront de préparer un test très fin pour des résultats d'autant plus intéressants. Voici la liste des particularités de NESSUS :

- un mode client / serveur ; rares sont les scanners qui reposent sur une telle architecture et c'est un point fort de NESSUS. Pourquoi ? Parce que la plupart du temps, la machine à auditer se trouve sur un réseau, une zone ou même une localisation différente de la machine qui nous sert à effectuer le test. Concrètement, cela signifie que l'auditeur a sur son poste la partie client et qu'il suffit de placer la partie serveur au bon endroit. Ainsi, on peut éviter les problèmes de pare-feu ou de localisation puisqu'au pire, il ne faudra ouvrir qu'un seul port sur le pare-feu (le port par défaut étant le 1241) puisque les tests vont s'effectuer intra-zone. Au mieux, le port utilisé sera un port déjà ouvert.
- l'authentification et le chiffrement ; les connexions entre le client et le serveur ne peuvent se faire qu'après connexion avec un mot de passe ou mieux, après validation des certificats mis en place. En effet, NESSUS propose de chiffrer les communications via SSL/TLS. Il est aussi possible d'utiliser sa propre PKI pour la gestion des certificats,
- les plugins ; NESSUS est constitué de nombreux modules d'attaque ou plugins. Chacun correspond à un test spécifique pour une vulnérabilité donnée. Il consiste en général à se connecter à un port, récupérer une bannière et même tester l'exploit associé à la vulnérabilité en fonction du paramétrage du scanner. Un plugin renvoie des résultats sur le test mais, aussi, inscrit des informations utiles dans la base de connaissances,
- la base de connaissances ; lors des tests, un plugin trouvera par exemple que le serveur WEB utilisé est Apache 1.3.29/OpenSSL 0.9.7a. Alors, il entre cette information dans la base de connaissances. Ensuite, cette information pourra soit être utilisée par un autre plugin ou corrélée. Par exemple, si un autre plugin teste les vulnérabilités d'OpenSSL, il saura que la version installée est dépassée et porteuse de failles. Deuxième exemple : imaginons maintenant qu'un script cherche les instances d'un serveur FTP et les trouve sur les ports 21 et 1499. La base va servir à enregistrer ces informations. Ensuite, pour chaque test concernant FTP, les tests seront lancés deux fois et seulement deux fois (une fois par instance donc). L'avantage de la base est de gagner en performance en évitant de rejouer certains tests et aussi d'obtenir des informations plus pertinentes.

Pour se rendre compte de l'interaction entre tous ces éléments de NESSUS, regardons la Figure 1. Pour commencer, l'auditeur se connecte au serveur NESSUS à partir de sa machine où le client NESSUS est installé (1). On sup-

Listing 3. *Cas d'un serveur WEB avec NIKTO*

```
$ perl nikto.pl -host victim.com
-***** SSL support not available (see docs for SSL install instructions)
      *****
---------------------------------------------------------------------------
- Nikto 1.36/1.37 - www.cirt.net
+ Target IP: 192.168.0.102
+ Target Hostname: victim.com
+ Target Port: 80
+ Start Time: Sat Apr 7 19:32:16 2007
---------------------------------------------------------------------------
- Scan is dependent on "Server" string which can be faked, use -g to override
+ Server: Apache
+ Server does not respond with '404' for error messages (uses '301').
+ This may increase false-positives.
+ Not found files redirect to: http://www.victim.com/
      Nikto-1.36-VhLbgLPTjj2.htm
+ The root file (/) redirects to: http://www.victim.com/
+ All CGI directories 'found', use '-C none' to test none
+ / - Redirects to http://www.victim.com/ , Default EMC Cellera manager
      server is running.

+ Over 20 "Moved" messages, this may be a by-product of the server
+ answering all requests with a "302" or "301" Moved message. You should
+ manually verify your results or use the "-404" option.
+ / - TRACE option appears to allow XSS or credential theft. See http:
      //www.cgisecurity.com/whitehat-mirror/WhitePaper_screen.pdf for details
      (TRACE)
+ /admin.php?en_log_id=0&action=users - Needs Auth: (realm "Forbidden access
      with this IP address!")
+ /members.asp?SF=%22;}alert('Vulnerable');function%20x(){v%20=%22 - Web Wiz
      Forums ver. 7.01 and below is vulnerable to Cross Site Scripting (XSS).
      CA-2000-02. (GET)
+ /phpimageview.php?pic=javascript:alert('Vulnerable') - PHP Image View 1.0
      is vulnerable to Cross Site Scripting (XSS). CA-2000-02. (GET)
+ /user.php?op=confirmnewuser&module=NS-NewUser&uname=%22%3E%3Cimg%20src=%22
      javascript:alert(document.cookie);%22%3E&email=test@test.com - Post
      Nuke 0.7.2.3-Phoenix is vulnerable to Cross Site Scripting (XSS).
      CA-2000-02. (GET)
+ /admin.php - Needs Auth: (realm "Forbidden access with this IP address!")

+ Over 20 "Moved" messages, this may be a by-product of the server
+ answering all requests with a "302" or "301" Moved message. You should
+ manually verify your results or use the "-404" option.
+ 2673 items checked - 7 item(s) found on remote host(s)
+ End Time: Sat Apr 7 19:37:23 2007 (307 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

pose que la connexion est acceptée et le test paramétré. Le testeur lance son scan sur la machine à auditer (2). Le scan commence par un scan TCP puis continue avec le lancement des plugins (nous verrons lesquels plus tard) pendant lequel la base de connaissances est utilisée (3). Finalement, NESSUS nous renvoie les résultats (4) et peut en générer un rapport au format HTML ou PDF par exemple.

Listing 4. *Extrait des informations fournies par la commande nasl*

```
[29208]() NASL> [00285c18] <- 1
[29208]() NASL> [00285c40] <- 0
[29208]() NASL> [00285c68] <- 5
[29208]() NASL> [00285c98] <- 6
[29208]() NASL> [00285cc0] <- 17

....

[29208]() NASL> [00286288] <- "2.2.8"
[29208]() NASL> [002862d8] <- 1
[29208]() NASL> [00286300] <- 0
[29208]() NASL> [00286328] <- "smb_kb921883.nasl"
NASL:0058> if (description) { ... }
[29208](/tools/nessus/lib/nessus/plugins/smb_kb921883.nasl) NASL> [00286300]
      -> 0
NASL:0026> supported_protocol=6;
[29208]() NASL> [00286350] <- 6
NASL:0028> protocol[0]="PC NETWORK PROGRAM 1.0";
[29208]() NASL> [000a34a0] <- "PC NETWORK PROGRAM 1.0"
NASL:0029> protocol[1]="LANMAN1.0";
[29208]() NASL> [000a34e0] <- "LANMAN1.0"
NASL:0030> protocol[2]="Windows for Workgroups 3.1a";
[29208]() NASL> [000a3500] <- "Windows for Workgroups 3.1a"
NASL:0031> protocol[3]="LM1.2X002";
[29208]() NASL> [000a3520] <- "LM1.2X002"
NASL:0032> protocol[4]="LANMAN2.1";
[29208]() NASL> [000a3560] <- "LANMAN2.1"
NASL:0033> protocol[5]="NT LM 0.12";
[29208]() NASL> [000a3580] <- "NT LM 0.12"
NASL:0035> nes_native_os="Windows 2002 Service Pack 2 2600";
[29208]() NASL> [002863a0] <- "Windows 2002 Service Pack 2 2600"
NASL:0036> nes_native_lanman="Windows 2002 5.1";
[29208]() NASL> [002863f0] <- "Windows 2002 5.1"
NASL:0043> SMB_HDR_SIZE=32;
[29208]() NASL> [00286418] <- 32
NASL:0046> SMB_FLAGS_SERVER_TO_REDIR=128;
[29208]() NASL> [00286440] <- 128
NASL:0047> SMB_FLAGS_REQUEST_BATCH_OPLOCK=64;
[29208]() NASL> [00286490] <- 64
NASL:0048> SMB_FLAGS_REQUEST_OPLOCK=32;
[29208]() NASL> [002864e0] <- 32
NASL:0049> SMB_FLAGS_CANONICAL_PATHNAMES=16;
[29208]() NASL> [00286530] <- 16
NASL:0050> SMB_FLAGS_CASELESS_PATHNAMES=8;
[29208]() NASL> [00286580] <- 8
NASL:0051> SMB_FLAGS_RESERVED=4;
[29208]() NASL> [002865d0] <- 4
NASL:0052> SMB_FLAGS_CLIENT_BUF_AVAIL=2;
[29208]() NASL> [002865f8] <- 2
NASL:0053> SMB_FLAGS_SUPPORT_LOCKREAD=1;
[29208]() NASL> [00286648] <- 1
NASL:0054> SMB_FLAGS_MASK=251;

...

NASL:0072> SMB_FLAGS2_KNOWS_LONG_NAMES=1;
[29208]() NASL> [00287070] <- 1
NASL:0073> SMB_FLAGS2_MASK=63559;
[29208]() NASL> [002870c0] <- 63559
...
```

L'entraînement

À ce point, nous avons correctement déterminé la nature de notre test et avons choisi d'utiliser NESSUS en tant que scanner. S'il peut suffir de cliquer sur le bouton *start* comme beaucoup le pensent, on passerait alors à côté de beaucoup de possibilités et surtout, la pertinence de notre scan en pâtirait.

Après avoir installé la dernière version, lancé le serveur NESSUS, installé le client sur notre poste, créé un utilisateur, s'être enregistré sur le site de NESSUS, téléchargé les derniers plugins et lancé le client NESSUS, il nous reste la partie la plus intéressante à effectuer, le paramétrage. Juste avant, il est possible d'utiliser NMAP pour connaître la plage de ports à étudier par exemple, avoir une idée/confirmer des informations déjà obtenues sur le système d'exploitation et les applications, etc... Nous nous basons pour le moment sur le client NESSUS généralement trouvé sous Linux.

- Le premier onglet qui apparaît est l'écran de connexion. Il suffit d'entrer son nom d'utilisateur et son mot de passe.
- Ensuite, sélectionnons les bons plugins comme le montre la Figure 2. Nous nous rendons compte pourquoi les premières étapes étaient si importantes. En fonction du système d'exploitation et des applications, nous choisirons les modules correspondants. Aussi, il ne faut pas oublier de prendre en compte l'environnement dans lequel nous travaillons. Ici, il s'agira de choisir ou non les plugins de type attaques *DoS*. Évidemment, le mieux est de tenir compte de ce type d'attaque car une personne malicieuse, elle, n'hésiterait certainement pas. Le but est de sélectionner uniquement les familles de modules dont nous avons besoin pour éviter les

faux-positifs. Une idée peut aussi être de sélectionner des plugins *a priori* incompatibles. Par exemple, il arrive de voir qu'un module destiné initialement pour Linux soit capable de rendre hors-service un service Windows. Cela peut toujours être intéressant même si la quantité de travail en est accrue pour trier les alertes pertinentes des fausses alertes. De plus, on se rendra compte que l'on ne pourra pas apporter d'actions correctives dans la plupart de ces cas.

- Le troisième onglet nommé *credentials* – que nous voyons en Figure 3 – n'est à surtout pas négliger. Il va en effet nouspermettre de simuler un compte interne. Si l'auditeur ou l'attaquant a obtenu suffisamment d'information lors d'un premier scan, il pourra recommencer le test de vulnérabilités mais cette fois avec le compte obtenu pour tenter d'aller plus loin. Ainsi, entrer ses *login/password* pour le compte SMB pourra permettre sur une machine Windows de trouver des partages de fichiers et autres accès qui devraient nous être interdits. Cela est souvent dû à un réseau de confiance trop permissif.
- C'est l'onglet *Scan Options* qui est représenté en Figure 4. Il est lui aussi très utile à l'affinage de notre préparation. On commence par entrer la plage de port trouvée (à l'aide de NMAP par exemple). Ensuite, tout dépend de l'environnement et de ce que l'on souhaite faire. Pour de meilleurs résultats, nous aurions plutôt intérêt à ne pas cocher *Safe check* afin de jouer les plugins jusqu'au bout. Aussi, il peut être intéressant de lancer le maximum de tests simultanés pour tester la robustesse de la machine auditée (Ex : 25 ou plus). Dans le cas contraire, si nous ne pouvons pas nous permettre de faire tomber la machine, on prendra toutes les précautions en cochant *Safe check* et en lançant moins de tests simultanément (Ex : 3). Quant à l'option *Optimize the test*, nous devrions la désactiver pour des résultats plus exhaustifs. En effet, si cocher ce paramètre permet de gagner en rapidité (interaction avec la base de connaissances privilégiée), cela empêche l'exécution de certains tests qui pourraient pourtant nous révéler des informations supplémentaires.
- L'onglet suivant est juste le moyen de donner l'adresse IP ou le nom de la cible. Attention cependant à s'assurer qu'une mauvaise configuration du serveur DNS ne nous trompera pas sur la cible. Une vérification rapide à l'aide de `nslookup` par exemple nous évitera bien des ennuis. Notons qu'il est possible d'écrire l'ensemble des adresses à auditer dans un fichier texte, ce qui s'évérera très pratique pour scanner un grand nombre de machines. Il sera aussi utile d'enregistrer ici la session pour ne pas avoir à paramétrer de nouveau le scan la prochaine fois que nous souhaiterons le faire.
- La Figure 5 nous montre les préférences – *Prefs* – qui nous permettront notamment de simuler des comptes tout comme dans l'onglet *credentials* mais pour les comptes applicatifs cette fois. Imaginons qu'à travers un scan AMAP ou avec les informations fournies, nous avons déterminé qu'un serveur POP3 était présent sur la ma-chine. Alors, afin de simuler un tel compte et de pousser nos tests plus loin, nous allons entrer ces informations. En fonction des connaissances que nous avons sur la cible à ce moment, c'est dans cet onglet que nous pourrons au mieux personnaliser le scan.
- Pour terminer, l'onglet *KB* pour *Knowledge base* (base de connaissances) est là pour nous permettre d'activer ou non celle-ci. Nous aurions tort de nous en priver et nous faisons le choix par conséquent de cocher l'option correspondante : *Enable KB Saving.*

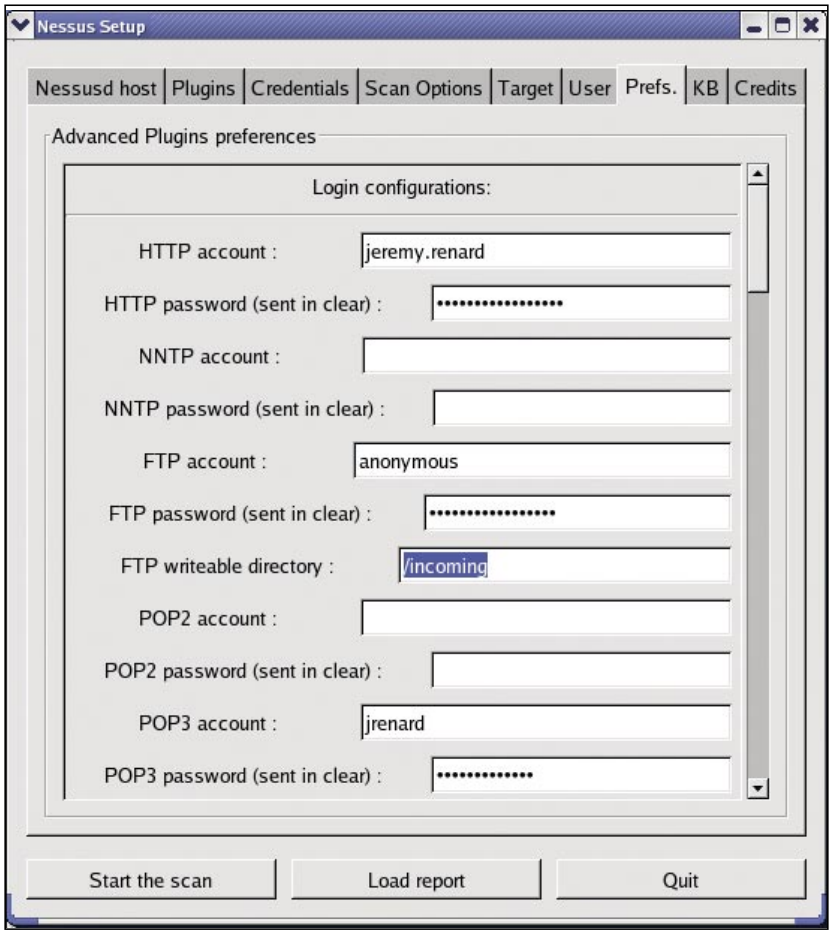


Figure 5. *Configurer NESSUS : simuler les comptes applicatifs*

Voilà, nous savons maintenant correctement paramétrer notre scanner même s'il ne s'agit que d'une méthode parmi d'autres.

Quoiqu'il en soit, nous devrions voir la différence au niveau des résultats. En effet, il suffit de lancer le scan et de générer un rapport pour s'en apercevoir. Nous pourrions nous arrêter là. Cependant, je vous propose d'aller plus loin pour ne pas nous contenter de données brutes mais d'une interprétation réfléchie. Pour cela, encore bien des choses sont à apprendre comme savoir éliminer les faux-positifs.

Ce critère pris en compte en plus d'une analyse et d'une interprétation fine nous permettra de générer nous-mêmes un rapport pertinent. C'est justement l'objet de la deuxième partie. Les choses sérieuses commencent...

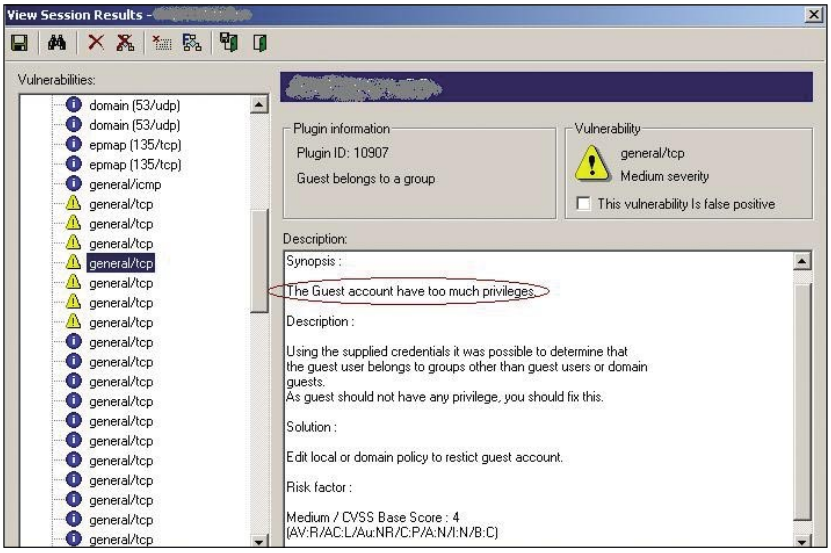


Figure 6. *Alerte NESSUS : exemple d'incohérence*

Épisode 2 : la bataille commence

Nous devons maintenons mettre en place une bataille intelligente en exploitant les informations collectées lors de l'épisode 1. Pour gagner à l'issu de ce deuxième épisode, nous devrons cerner et éliminer dans un premier temps les vraies failles en tirant partie de notre esprit d'analyse et d'interprétation. Ensuite, nous devrons nous assurer la victoire totale en éliminant les dernières vulnérabilités à l'aide de notre technicité.

Garder son sang froid

À travers le rapport généré, nous verrons – avec la plupart des scanners comme NESSUS et de différents éditeurs – une liste de vulnérabilités classées par importance (*info*, *warning* ou *high*), une solution de correction et une référence CVE.

L'objectif ici est de s'y retrouver parmi toutes les informations

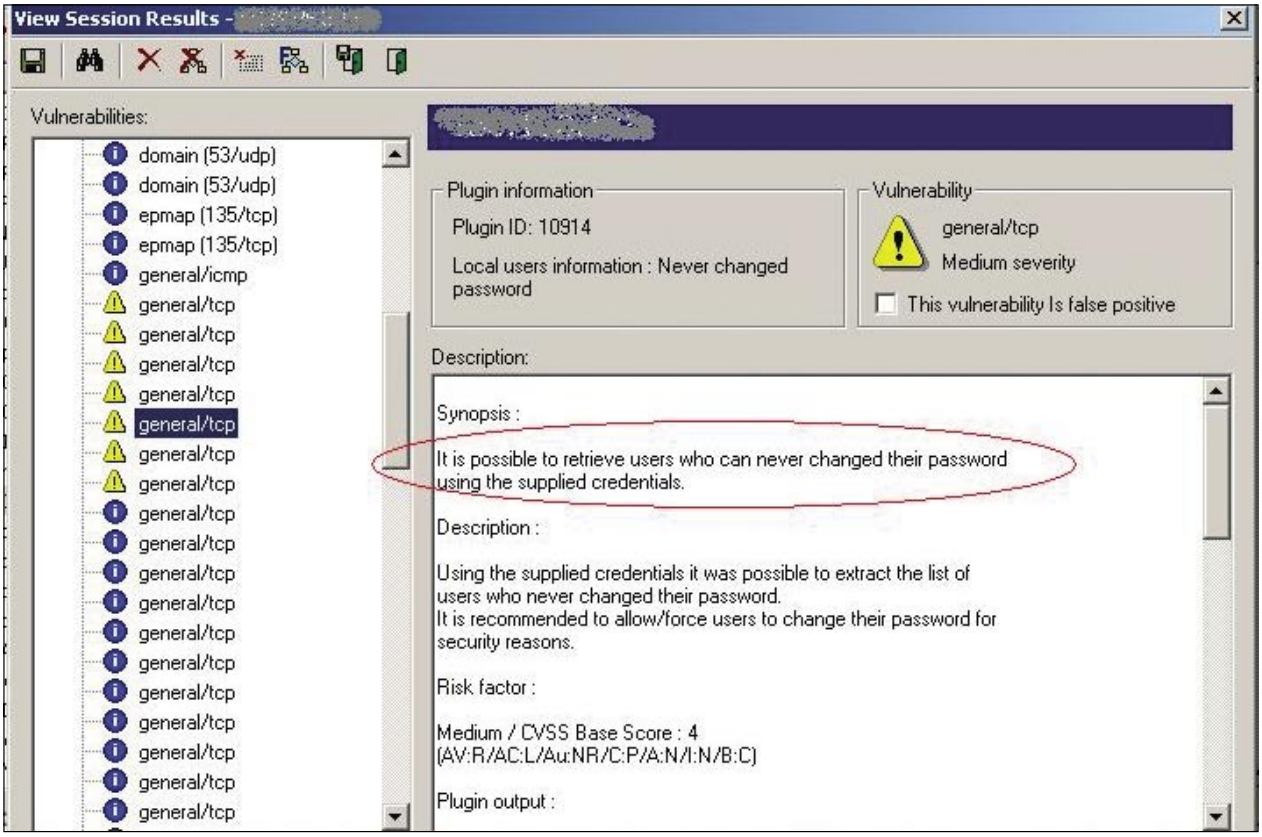


Figure 7. *Alerte NESSUS : exemple d'interaction (1/2)*

collectées. La première chose à faire est de séparer les informations de la façon suivante :

- celles qui nous paraissent évidentes et pertinentes (ex : une vulnérabilité sur notre serveur sendmail alors que nous sommes sûrs d'avoir installé cette solution),
- celles qui nous paraissent incohérentes (ex : une vulnéra-bilité pour IIS alors que notre serveur WEB est sous Linux),
- celles où nous avons encore un doute (mais un peu de recherche ou un simple `rpm -qa | grep application` pourra parfois suffire).

Concernant la première liste, nous allons nous-mêmes déterminer le niveau de criticité en nous aidant toutefois de l'appréciation du scanner. Il faut prendre en compte les facteurs suivants :

- facteur criticité ; si nous sommes dans une DMZ par exemple, la moindre faille pourrait être fatale. Même chose s'il s'agit d'un serveur financier de l'entreprise. Nous aurons donc tendance à sur-évaluer l'importance de chaque vulnérabilité trouvée.
- facteur risque ; une vulnérabilité importante mais dont l'exploit est quasiment impossible dans notre environnement sera finalement sous-évaluée. Inversement, une vulnérabilité dont un ver vient de voir le jour – et qui risque d'infecter une bonne partie de notre réseau – sera cette fois sur-évaluée.
- facteur impact ; si la machine vulnérable se trouve sur un réseau coupée du monde ou au contraire sur un réseau de confiance, l'impact pourra être sous-évalué dans le premier cas et sera sur-évalué dans le deuxième.

Pour faire notre propre appréciation, nous devons prendre en compte les conditions du test (localisation, criticité de la zone) et les paramètres (avons-nous pu tester tous les flux ? Lancer tous les plugins voulus ? Avons-nous des comptes internes ou applicatifs ?).

Il serait bien entendu impossible d'énumérer ici toutes les vulnérabilités que nous pourrions trouver. Toutefois, je vous propose de voir ensemble des exemples récurrents selon le système d'exploitation, le type ou la zone du serveur dans le Tableau 2.

La chasse aux faux-positifs

Les vulnérabilités relevées qui nous paraissent moins évidentes ou encore sans importance doivent être étudiées de plus près. Pour montrer au lecteur un autre environnement proposé par NESSUS, nous allons travailler cette fois sous Windows avec le client associé NessusWX :

- les incohérences ; un cas classique est de nous signaler que le compte *Guest* d'une machine Windows a trop de privilèges (Cf. Figure 6). Pourtant, nous pourrions voir avec un outil comme Caïn&Abel que ce compte est désactivé,
- les interactions entre les informations ; une première alerte nous signale que le mot de passe d'un compte X n'expire jamais (Figure 7) alors que pourtant une autre *info* montre que la politique de mots de passe demande de changer celui-ci tous les 90 jours (Figure 8),
- le code du plugin ; en cas de doute, la lecture du code source d'un plugin peut nous donner la

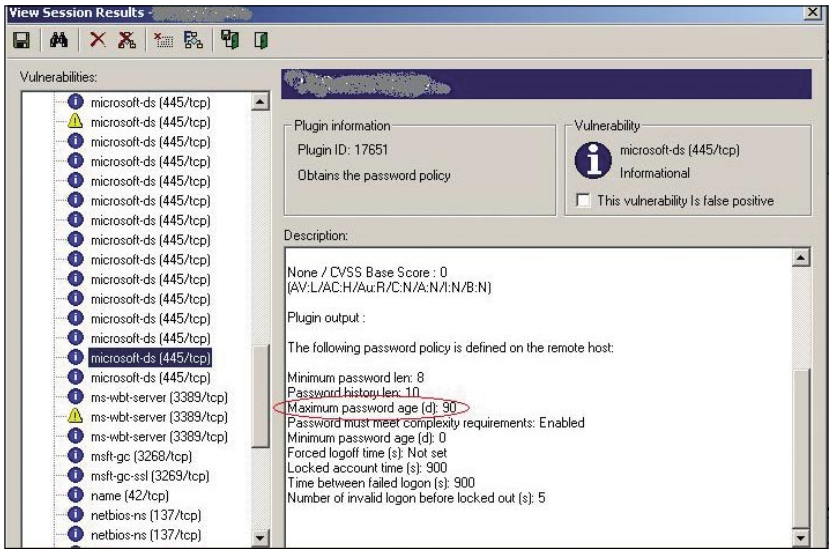


Figure 8. *Alerte NESSUS : exemple d'interaction (2/2)*

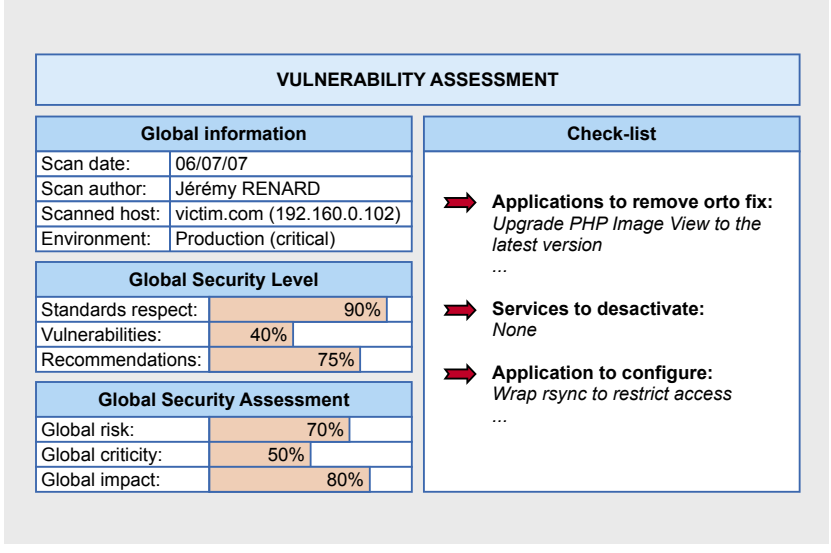


Figure 9. *Tableau de bord : synthèse des résultats*

réponse. Ils sont tous disponibles sur le site de NESSUS. La plupart du temps, ce sont les plus vieux modules qui génèrent des faux-positifs car les tests ne sont plus valables,

- les commandes de NASL ; il est possible d'utiliser NASL en ligne de commande pour rejouer un ou plusieurs plugins et en étudier les sorties. Pour cela, nous allons lancer la commande suivante : `nasl -t victim.com -T /home/jer/nessus/out.ms06-035 ~/plugins/smb_kb921883.nasl`. Juste pour voir à quoi cela ressemble, regardons un extrait de ce que nous pourrions obtenir avec le Listing 4. Chaque étape du test est explicitée et nous pouvons remarquer toutes les entrées et sorties. Evidemment, cela suppose de connaître suffisamment bien la vulnérabilité et l'exploit pour réussir à trouver les informations que l'on recherche. Une astuce consiste à lancer un `tcpdump` dans une première session pendant que nous déroulons notre commande NASL dans une deuxième session afin de vérifier les informations que permet de laisser passer la vulnérabilité jouée par le plugin,
- utiliser un deuxième scanner pour confirmer la vulnérabilité trouvée. Le tableau 1 devrait nous aider à trouver notre graal,
- utiliser des outils complémentaires ; ce ne sont pas les outils qui manquent pour vérifier une faille en l'exploitant. Un bon exemple est certainement Metasploit. Ensuite, tout dépendra de la vulnérabilité à étudier.

On n'a pas le droit à l'erreur !

Notre analyse et notre interprétation nous ont normalement permis d'éliminer le maximum de faux-positifs d'une part et d'évaluer au mieux les vulnérabilités restantes d'autre part.

En premier lieu, nous ne pourrons être satisfaits de notre scan que si nous sommes sûrs d'avoir pris en compte toutes les failles. En oublier une seule pourrait suffir à offrir la porte recherchée à l'attaquant. La base des plugins est certes très importante mais elle n'est pas forcément exhaustive. Lors de la découverte d'une nouvelle vulnérabilité par exemple, il sera nécessaire d'écrire nous-même le plugin correspondant avec NASL.

Heureusement pour nous, l'écriture d'un tel plugin nous a été simplifiée par leurs auteurs. Le Listing 5 nous en montre le squelette. À nous ensuite de le compléter pour qu'il réponde à nos besoins. Pour cela, il existe des fonctions prédéfinies qui sont regroupées par catégories. En voici les principales :

- *Networking functions* ; pour ouvrir/fermer une *socket* ou encore, pour récupérer des informations sur la machine cible, ...
- *HTTP functions* ; pour récupérer les entêtes HTTP, obtenir les résultats des commandes (ex : `GET`), ...
- *Packet manipulation functions* ; sert notamment à forger des paquets,
- *String manipulation functions* ; opérations usuelles sur les chaînes de caractères,
- *Cryptographic manipulation functions* ; on retrouve ici les méthodes de hashs et d'authen-tifications usuelles.

Listing 5. *Squelette d'un plugin écrit avec NASL*

```
…# Script title and description

if (description) {
   script_id(XXXX);
   script_cve_id("CVE-200X-XXXX");
   script_version("$Revision:1.0$");
   name["english"] = "script_name";
   script_name(English:name["english"]);
   desc["english"] = "description complete…";
   summary["english"] = "description courte";
   script_summary(english:summary["english"]);
   script_category(ACT_XXX);
   script_copyright(english:"No copyright");
   family["english"] = "nom_famille" ;
   script_family(english:family["english"]);
   script_dependencies("XXX.nes", "XXX.nasl");
   script_require_ports("Service/XXX",nb_port);
   exit(0) ;
}

# check begins here

include(«XXX.inc »);

# (ex : http_func.inc, voir la liste des fonctions ci-dessus)
port = get_kb_item("Service/XXX");
      # ou get_http_port(default:80) if ( !port) port = 80;
if ( !get_port_state(port)) exit(0) ;
if (safe_check()) {
   # sera exécuté seulement si safe-check a été coché dans les paramètres
   banner = get_http_banner(port: port);
   …
   ;
   Risk factor : "Low";
       # ou "Medium" ou "High" security_hole(port: nb_port, data: report);
   exit(0);
} else {
   payload ="…";
   … # lancer attaque (DoS ?)
   … # récupérer bannière
   … # test si la machine répond toujours ou non
}
}
```

Malheureusement, il nous faudrait bien plus d'un article pour étudier réellement ce langage et pour voir toutes ses possibilités.

C'est pour cela que nous n'irons pas plus loin à ce sujet mais je vous recommande vivement les manuels de référence de Renaud Deraison (le créateur de NESSUS) et M. Arboi (l'auteur de NASLv2) pour découvrir les subtilités de NASL.

Mission accomplie ?

Considérer avoir réussi notre mission revient à éliminer l'ensemble des dangers. Cela suppose donc de corriger toutes les vulnérabilités réelles. Pour parvenir à cela, nous devrons :

- lire les informations fournies par NESSUS,
- tenir compte de la référence donnée (CVE ou documentde référence) pour obtenir plus d'information,
- utiliser ses propres connaissances pour trouver le bon moyen de combler la faille.

Puis, concrétement, l'action corrective pourra être :

- l'application d'un correctif (ex : un correctif Windows MS07-XXX),
- une meilleure configuration (ex : supprimer la communauté *Public* de SNMP),
- renforcer la politique de mots de passe (ex : imposer un mot de passe qui expire tous les trois mois),
- chiffrer les communication (ex : imposer l'utilisation de SSL pour un site WEB),
- un filtrage de ports, d'adresses IP ou d'utilisateurs de l'*Active Directory*,
- un moyen palliatif spécifique à l'application (ex : changer une valeur de sa clé de registre).

À ce stade, nous disposons des vulnérabilités pertinentes, des références pour les illustrer et des recommandations à suivre pour corriger la faille : tous les éléments sont là pour écrire un bon tableau de bord.

Épisode 3 : bilan de la bataille

Tout notre travail serait vain s'il n'était pas suivi pas les bonnes personnes qui devront appliquer les recommandations et suivre nos conseils afin d'éviter de rencontrer de nouveau un problème récurrent. Pour cela, nous allons voir un exemple de tableau de bord qu'il faudra personnaliser en fonction de vos besoins. Cependant, le contexte d'une entreprise implique souvent des contraintes supplémentaires qu'il faut commencer par prendre en considération.

Dans le même temps, cela représentent des faiblesses sur lesquelles un attaquant peut compter alors mieux vaut en être averti car le danger est réel...

Dans le camp des pros

Appliquer un correctif est une solution fréquente pour combler une faille. Cependant, plusieurs raisons peuvent l'empêcher :

- l'entreprise doit garder sur ses serveurs des applications non sécurisées car elle est en partenariat avec l'éditeur en question, malgré que celui-ci ne soit pas en mesure de proposer des logiciels sans failles,
- l'entreprise n'a pas les licences nécessaires pour avoir les protections suffisantes ou pas sur l'ensemble de son parc,
- le correctif n'a pas encore été développé pour le logiciel utilisé par l'entreprise,

De même, il n'est pas toujours facile de pouvoir apporter une action corrective :

- les exigences du client font que l'entreprise ne peut pas apporter certaines modifications (par exemple, la mise en place d'une authentification forte basée sur SSH et clés asymétriques au lieu d'un telnet reposant sur un mot de passe),
- l'application d'un correctif ou le renforcement d'une configuration aurait un trop grand impact dans un environnement critique. L'entreprise privilégie alors le rendement de la production de ses serveurs à leur sécurité,
- la correction pourrait avoir des conséquences inadmissibles pour les utilisateurs ou l'administrateur du serveur incriminé (dégradation notable de l'ergonomie, des fonctionnalités, ...).

Au rapport !

Le tableau de bord proposé s'écrira en quatre parties (ici, quatre feuilles

Terminologie

- *Faux-positif* : émission d'une fausse alerte par le scanner. Le cas le plus fréquent se produit quand la vulnérabilité est déclenchée d'après une bannière qui n'a pas été mise à jour.
- *Full-diclosure* : il s'agit d'une démarche technique qui consiste à publier l'ensemble des éléments techniques relatif à la découverte d'une vulnérabilité.
- *Nessus Attack Scripting Language* (NASL) : langage, inspiré du C, créé par l'auteur de Nessus, Renaud DERAISON en 1998, et amélioré par Michel ARBOI avec la version 2 en 2003. Ce langage a pour but de simplifier l'écriture de scripts d'attaques de Nessus – que l'on retrouve ainsi dans les *plugins*.
- *Prise d'empreintes* (ou *fingerprinting*) : technique qui vise à récupérer le nom du système d'exploitation et des applications démarrées sur une machine en se connectant sur le serveur cible sur les différents ports et en analysant les paquets reçus.
- *Proof of Concept* (POC) : il s'agit d'une démonstration de faisabilité d'un certain concept, comme une attaque à partir d'une certaine vulnérabilité.
- *Réseau de confiance* : dans une entreprise, un réseau de confiance représente l'ensemble des machines auxquelles nous pouvons avoir accès à partir d'un compte local sur ce même réseau. En général, il s'agit d'une branche complète d'un *Active Directory*.

dans un tableur) qui auront chacune un rôle déterminant. La première étant la conséquence des trois autres, commençons par la deuxième. Il s'agira du respect des standards de sécurité.
En effet, imaginons que notre scanner ne nous révéle aucune faille sur les protocoles `rlogin` ou `xdmcp`, nous savons maintenant aller plus loin et que ces protocoles sont potentiellement dangereux. On devra donc vérifier dans cette partie que nous utilisons seulement des protocoles de communications chiffrés comme `ssh`.

De même, si nous avons réussi à obtenir suffisamment d'information sur la politique de mots de passe, nous pourrons l'évaluer par rapport à nos exigences.

La suite des critères dépend de nos besoins et des standards plus ou moins restrictifs qui sont mis en place dans l'entreprise. Une autre idée pourrait être de vérifier que le serveur est synchronisé avec le serveur de temps NTP.

En effet, nos logs seront ainsi exploitables pour une investigation post-attaque. La troisième partie de notre tableau de bord contiendra la liste ordonnée des vulnérabilités pertinentes trouvées. Nous devrons trouver les informations issues de notre analyse, des corrélations entre les résultats. Ici, figureront alors la description de la faille, le niveau d'importance attribué, les références additionnelles et enfin, les actions correctives agrémentées de nos conseils.

C'est de cette manière claire et synthétique que l'administrateur sera en mesure de savoir de quoi il s'agit, de quelle manière cela impacte son serveur, quels sont les risques et comment remédier aux problèmes de sécurité. Ensuite, nous trouverons dans la dernière partie l'ensemble des recommandations pour sécuriser le serveur à partir des informations récupérées et interprétées.

Cela ne veut pas dire que les recommandations sont moins im-portantes que les vulnérabilités données. Par exemple, imaginons le cas d'un serveur de base de don-nées Oracle dont l'authentification peut se faire à travers un portail WEB. Si l'administrateur a pensé à mettre en place SSL pour se connecter en HTTPS (port 443), très bien ! Mais s'il a laissé la possibilité de s'authentifier en HTTP (port 80) sans redirection automatique vers le port 443 qui plus est, c'est beaucoup moins bien !

Un attaquant en profitera pour écouter les communications en clair sur le port 80 et récupérer les *credentials* des personnes ayant fait le mauvais choix de s'authentifier en HTTP. D'autres recommandations pourront être la restriction par filtrage (ex : seul le groupe *SAP* pourra accéder à la solution de même nom). Notre bon sens est largement mis à contribution ici.

Revenons à la première partie. du tableau de bord. Ici, nous inscrirons les informations basiques du scan (machine testée, date, lieu, environnement, ...). Nous résumerons aussi l'état de la machine pour les trois critères pris en compte précedemment : respect des standards, présence des vulnérabilités sur la machine et nécessité d'appliquer certaines recommandations.

Enfin, nous pourrions ajouter une *check-list* des actions correctives à apporter sur le serveur avec leur ordre d'importance. La Figure 9 montre un exemple de tableau de bord que nous aurions pu obtenir suite à un test de vulnérabilités. Ceci est seulement une idée de présentation. Je laisse le soin au lecteur de compléter le tableau.

Conclusion

Un test de vulnérabilités bien préparé et bien interprété grâce au savoir-faire d'un expert permet d'augmenter de manière significative le niveau de sécurité des machines. Sans surprise, cette technique est aussi largement utilisée par les attaquants pour trouver la ou les porte(s) d'entrée sur les serveurs.

En effet, une fois les remparts transpercés, la recherche de vulnérabilités et leur exploitation vont donner le dernier assaut dirigé directement contre les pièces les plus importantes de notre réseau, nos précieux serveurs.

C'est donc à l'entreprise d'avoir une stratégie préventive en amont avec des guides de *hardening*. L'étape suivante est d'élever le niveau de sécurité par les tests. En aval, il faudra mettre en place une gestion des correctifs (ou *patch management*) pour maintenir ce niveau dans le temps.

La sécurisation d'un serveur passe donc par tout un cycle de vie mais à coup sûr, il vaut mieux que ce soit nous qui fassions le travail plutôt que l'attaquant... ●

À propos de l'auteur

Jérémy Renard est Consultant Sécurité des Systèmes d'Information. Actuellement, il a pour mission de maintenir et d'améliorer le niveau de sécurité du Datacenter d'un grand groupe français au sein de l'équipe sécurité dédiée, grâce notamment aux tests de vulnérablités. Diplômé d'un Master Sûreté et Sécurité à l'Université de Jussieu (Paris), il s'intéresse depuis longtemps aux tests d'intrusion.

Sur Internet

- *http://cve.mitre.org/* – Site de veille,
- *http://www.nessus.org/* – Site officiel de NESSUS,
- *http://www.virtualblueness.net/nasl.html* – Site de référence de NASLv1, par R. DERAISON,
- *http://michel.arboi.free.fr/nasl2ref/* – Site de référence de NASLv2, par M. ARBOI,
- *http://www.gomor.org/cgi-bin/sinfp.pl* – sinFP, un scanner actif et furtif,
- *http://www.cirt.net* – Nikto, un scanner WEB très spécialisé,
- *http://www.foundstone.com/* – Chercher Attacker, un anti-scanner.