Vulnérabilité des postes clients

Gaël Dellaleau and Renaud Feil

Ernst & Young
Technology and Security Risk Services
11 allée de l'arche
92037 Paris-La Défense Cedex – France {gael.dellaleau,renauld.feil}@fr.ey.com

Résumé Les postes de travail des utilisateurs sont des composants critiques du système d'information et doivent être protégés en conséquence. Le nombre croissant de vulnérabilités des applications clientes, en particulier les navigateurs Web, en font des cibles de choix pour les attaquants voulant gagner un accès au réseau interne d'une organisation.

Nous montrons que l'efficacité des protections couramment mises en oeuvre pour sécuriser les postes clients (antivirus, pare-feu, proxy Web) est souvent surestimée. Ces protections souffrent de limitations intrinsèques qui laissent la porte ouverte à certains types d'attaques. Il est ainsi possible à un attaquant d'identifier de façon précise la version des applications clientes utilisées pour pouvoir lancer ensuite une attaque ciblée, ou encore de contourner un filtrage antivirus.

Nous démontrons également qu'une application Web accessible uniquement depuis le réseau interne peut être attaquée à partir d'une simple page Web visionnée dans le navigateur d'un poste client, même si ce dernier n'est vulnérable à aucune faille de sécurité et que l'application nécessite une authentification.

Nous prévoyons enfin que des outils permettant d'automatiser l'identification et l'attaque des postes clients vont se développer. Nous présentons un « framework » de démonstration montrant la possibilité d'attaques automatisées auto-adaptables ciblant les postes de travail, ainsi que les parades permettant de protéger le système d'information face aux attaques présentées.

Cet article doit permettre de comprendre les nouveaux risques introduits par ce type d'outil afin de pouvoir proposer des contre-mesures adaptées, et de sensibiliser par des exemples concrets l'ensemble des personnes concernées par la sécurité du système d'information au sein des organisations.

1 Pourquoi s'intéresser à la sécurité des postes clients ?

Dans la plupart des organisations, l'essentiel des efforts de sécurisation porte sur les serveurs. Leur configuration est renforcée pour éviter les accès logiques non autorisés, ils sont protégés physiquement dans des salles à accès restreints, et leur sécurité est régulièrement auditée. Toutes les informations importantes y sont stockées (ou devraient l'être) et la plupart des traitements critiques pour l'entreprise y sont effectués. Le risque majeur est la compromission d'un serveur

contenant des données critiques, et le poste de travail de l'utilisateur est perçu parfois comme un composant plutôt secondaire dans l'architecture de sécurité de l'organisation.

Cette approche est justifiée sur les grands systèmes : des terminaux passifs se connectent à un mainframe, qui stocke les informations et réalise tous les traitements de façon centralisée. Le terminal dispose juste des capacités de traitement nécessaires pour assurer la connexion au mainframe, récupérer les entrées clavier de l'utilisateur et afficher les données sur l'écran. Dans ce contexte, les possibilités de compromission de la sécurité logique du terminal sont très limitées : toutes les applications et les données sont situées sur le mainframe. La sécurisation logique de l'ensemble se résume à la sécurisation du mainframe.

Cette vision évolue depuis la multiplication des failles de sécurité sur les applications clientes. Le poste client apparaît désormais comme un des composants les plus exposés du système d'information.

Pour introduire ce risque, cette première partie rappelle quelques constats simples liés aux attaques sur les postes client :

- Les *firewalls* et *proxies* n'offrent qu'une protection limitée
- La compromission d'un poste client donne à l'attaquant un accès au réseau interne
- Les applications clientes sont autant (voir plus) vulnérables que les applications serveurs
- La sensibilisation des utilisateurs ne garantit pas que les attaques échoueront

1.1 Les *firewalls*, *proxies* et *IDS/IPS* n'offrent qu'une protection limitée

Il est rare que les postes clients d'une organisation soient accessibles depuis un réseau externe : soit un *firewall* bloque la totalité des demandes de connexion leur étant destinées, soit un système d'adressage privé ne permet pas leur accès depuis Internet.

Cependant, la compromission d'un poste client ne nécessite pas de pouvoir s'y connecter directement, contrairement à l'attaque d'un serveur durant laquelle l'attaquant initie des connexions vers les services disponibles, pour tenter d'y exploiter des failles de sécurité. Lors de l'attaque d'un poste client, c'est la victime qui se connecte à un serveur dont le contenu est contrôlé par l'attaquant. Il suffit alors à l'attaquant de faire en sorte que l'utilisateur récupère un contenu « offensif », qui exploitera une faille de sécurité sur l'application cliente utilisée afin d'en prendre le contrôle.

Ce contenu offensif peut être déposé à différents endroits :

- sur un serveur situé sur Internet (*HTTP*, *streaming audio/vidéo*...) ;
- sur le serveur de messagerie de l'entreprise, dans la boite mail de l'utilisateur ciblé.

Première attaque : dépôt du contenu offensif sur un serveur de l'Internet La figure 1 montre le premier type d'attaque, durant laquelle un utilisateur

du réseau interne se connecte sur un serveur dont le contenu est contrôlé par l'attaquant pour récupérer à son insu un contenu offensif.

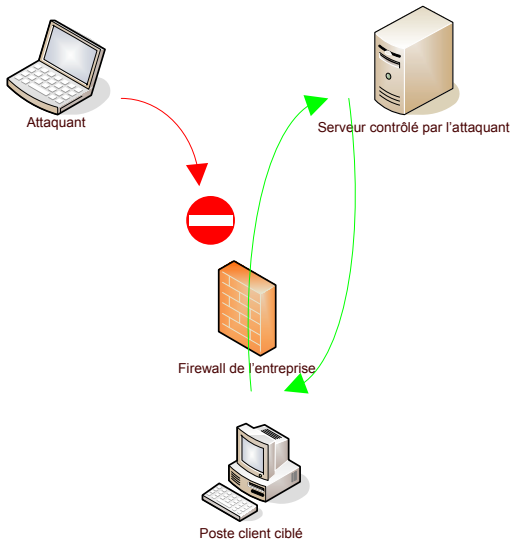

Fig. 1. Attaque par dépôt du contenu offensif sur un serveur de l'Internet

Le schéma 1 montre qu'une connexion directe entre l'attaquant et la victime est impossible, car elle est bloquée par le *firewall*. Mais si la victime initie une connexion vers un serveur de l'Internet sur un port autorisé, la réponse du serveur sera autorisée par les règles de filtrage (mécanisme *stateful*). Cette réponse contient le contenu offensif qui, en exploitant une faille sur l'application cliente, permet de prendre le contrôle du poste de travail de l'utilisateur et d'accéder ainsi au réseau interne.

Il est possible que les accès vers Internet des postes clients passent par un serveur *proxy*, dont le rôle est d'effectuer les connexions vers les sites de l'Internet pour le compte du poste client et de relayer la réponse. A lui seul, le serveur *proxy* n'empêche pas l'exploitation de failles sur les applications clientes, puisqu'il renvoie à l'identique le contenu applicatif de la réponse du serveur.

Si le serveur *proxy* est couplé à un système d'analyse et de filtrage de contenu dangereux (contrôles *ActiveX*, applets *Java*...), ou si un *IDS/IPS* est présent sur le réseau, leur contournement est souvent possible en utilisant un protocole chiffré. Ainsi, l'utilisation du chiffrement *SSL/TLS* permet d'amener jusqu'au poste client un contenu offensif sans qu'il soit détecté durant le transit.

Il est à noter également que le serveur utilisé pour l'attaque n'est pas obligatoirement sous le contrôle de l'attaquant. L'exemple des attaques par *XSS* (Cross-Site Scripting) montre qu'il est possible de déposer du contenu offensif sur un serveur dont l'attaquant n'a pas pris le contrôle.

Seconde attaque : dépôt du contenu offensif sur le serveur mail de l'entreprise Le second type d'attaque passe par l'envoi d'un courrier électronique à destination du ou des utilisateurs ciblés, comme présenté dans la figure 2.

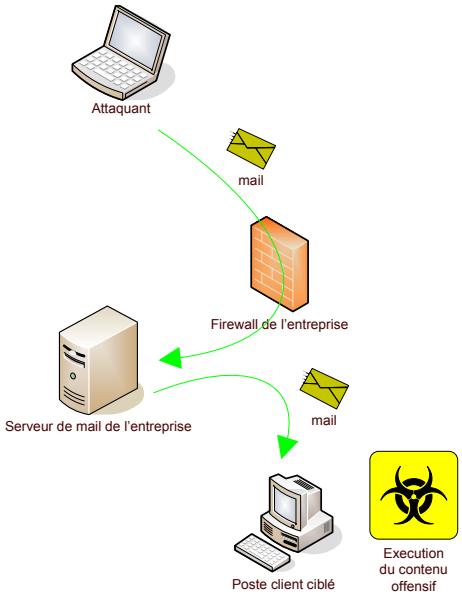

Fig. 2. Attaque par dépôt du contenu offensif sur un serveur de messagerie

Le mail est reçu par la passerelle *SMTP* de l'entreprise, puis stocké dans la boîte mail de l'utilisateur. Lorsque ce dernier récupère le mail, une faille de l'application de messagerie est exploitée pour prendre le contrôle du poste de travail de l'utilisateur. Une autre possibilité pour l'attaquant consiste à envoyer un contenu offensif sous forme de pièce jointe et à convaincre l'utilisateur d'exécuter cette pièce jointe en jouant sur sa curiosité.

La difficulté de ce type d'attaque est de contourner le filtrage antivirus mis en place sur les serveurs mail. Néanmoins un tel contournement est toujours possible, comme expliqué dans la suite de cet article.

1.2 La compromission d'un poste client donne à l'attaquant un accès au réseau interne

Le *firewall* et le serveur *proxy* ne protègent plus le réseau interne de l'entreprise lorsqu'un poste client est compromis. Il est facile de créer un canal de communication bidirectionnel entre le réseau interne de l'entreprise et l'Internet, et ce par différents moyens :

- utilisation de requêtes *HTTP*, avec si nécessaire *hijacking* du processus d'Internet Explorer pour réutiliser les éléments d'authentification demandés par le *proxy* Web ;
- utilisation de la méthode *CONNECT* du *proxy* Web ;
- *tunneling* dans des requêtes *DNS* (ou d'autres protocoles autorisés en sortie par le *firewall*).

Le canal de communication bidirectionnel ainsi créé peut permettre à un attaquant d'exécuter des commandes sur le poste client compromis ou de créer un tunnel *VPN* vers le réseau IP interne de l'entreprise, qui se retrouve ainsi exposé directement à l'attaquant. La détection de ces canaux de communication est complexe, car des techniques existent pour faire passer le trafic ainsi encapsulé pour un trafic légitime (principe du « *covert channel* ») D'une certaine façon, c'est désormais l'attaquant qui est assis derrière le poste de travail compromis, et qui peut continuer à lancer ses attaques depuis le réseau interne.

Il est à noter que même si l'attaquant ne continue pas ses attaques sur le réseau interne, il peut déjà avoir trouvé des données sensibles sur le poste client :

- L'essentiel du travail d'un employé (de la secrétaire au directeur), se fait à partir d'applications présentes sur son poste de travail (lecture et envoi d'emails, navigation Web, lecture et rédaction de documents Office, etc.). Des documents confidentiels peuvent être stockés sur le poste de travail, au moins temporairement, et souvent plus longtemps avec la multiplication des portables pour les nomades.
- Lorsque ces personnes se connectent vers d'autres systèmes, les mots de passe sont parfois stockés, au moins temporairement, sur le poste local.

1.3 Les applications clientes sont autant (voir plus) vulnérables que les applications serveurs

Les vulnérabilités touchant les applications clientes sont devenues de plus en plus populaires depuis la généralisation d'Internet à la fin des années 1990. Depuis, le rythme de découverte de failles de sécurité sur des applications clientes ne cesse de progresser. Lors de la 32ème conférence CSI en novembre 2005, le CTO (Chief Technical Officer) de Qualys, Gerhard Eschelbeck, a annoncé que plus de 60% des failles découvertes ces derniers mois étaient des failles « côté clients », et que la tendance allait encore s'affirmer [1].

Plusieurs raisons peuvent expliquer que les failles « côté client » se multiplient par rapport aux failles « côté serveur » :

- Les applications serveur répandues ont souvent déjà été auditées par des chercheurs de vulnérabilités. Même si elles contiennent encore des failles,

les plus évidentes ont été révélées. Ce n'est pas le cas de beaucoup d'applications clientes.
- Les nouvelles applications serveurs « majeures » ont intégré dans leurs processus de développement les bonnes pratiques de développement : les erreurs triviales sont plus rares qu'auparavant. Certains développeurs d'applications clientes ne prennent pas les mêmes précautions lors du traitement des informations récupérées, pensant à tort qu'une application cliente n'est pas une cible pour un attaquant (idée que « cette donnée est sûre, puisque c'est moi qui ai fait la requête pour la récupérer »).
- Il est aujourd'hui plus difficile pour une personne mal intentionnée d'exploiter les failles situées sur des serveurs. Les *firewalls* sont de mieux en mieux configurés et ne sont plus vulnérables aux attaques permettant de contourner leur filtrage de façon triviale. Les serveurs accessibles en *DMZ* font souvent l'objet d'une procédure de sécurisation. Les pirates cherchent d'autres moyens d'arriver à leurs fins, et étudient donc les applications clientes.

En particulier, les navigateurs Web et les moteurs de rendu *HTML* sont des éléments vulnérables :
- Le développement d'un outil de *parsing*, comme un interpréteur *HTML*, est loin d'être une tâche facile, surtout lorsque cet outil doit tolérer certaines erreurs dans le contenu interprété.
- Les navigateurs sont de plus en plus complexes. Ils intègrent des formats variés et pour améliorer « l'expérience utilisateur », permettent à du contenu actif de s'exécuter. Dans ce contexte, il n'est pas étonnant que les vulnérabilités soient plus nombreuses.

Tous les navigateurs sont potentiellement vulnérables, Internet Explorer le premier, mais Firefox et les autres contiennent également leur lot de vulnérabilités.

1.4 La sensibilisation des utilisateurs ne garantit pas que les attaques échoueront

Les attaques sur les postes clients nécessitent pour la plupart une interaction de l'utilisateur : celui-ci doit cliquer par exemple sur un lien *HTTP* pour se connecter sur un site Web précis. Il n'est pas certain que l'utilisateur décide de se connecter à ce site, surtout s'il a été sensibilisé aux risques liés aux attaques sur les postes clients. Cependant, diverses techniques permettent à l'attaquant d'orienter la décision de l'utilisateur et de multiplier le risque pour l'entreprise :
- Travail sur la rédaction et le design du mail, pour lui donner l'apparence d'un mail envoyé par une société ou un personne de confiance (principe du *phishing*).
- Utilisation de la syntaxe suivante pour dissimuler le nom du serveur auquel mène réellement le lien[1] :

[1] Cette syntaxe n'est cependant plus reconnue dans Internet Explorer et l'Explorateur Windows depuis la mise à jour MS04-004. (cf. http ://support.microsoft.com/default.aspx ?scid=834489)

`http://login:pass@site:port/chemin`

- Utilisation d'une faille liée à l'affichage du lien (failles liées à l'internationalisation par exemple), permettant d'afficher un lien de type « `www.site\_connu.com` » redirigeant en réalité vers « `www.attaquant.com` ».
- *Social engineering* (par téléphone...) incitant l'utilisateur à cliquer sur ce lien.
- Attaque sur les serveurs *DNS* (*pharming*) pour rediriger le trafic de certains sites légitimes, comme par exemple un moteur de recherche, vers le site contrôlé par l'attaquant.

Certaines de ces techniques nécessitent l'exploitation d'une vulnérabilité ou une absence de mise à jour des correctifs de sécurité.

La première technique est la plus simple, et ne suppose pas la présence d'une faille. Les nombreux cas de *phishing* montrent le risque que cette technique représente pour les entreprises. En effet, aidé par la loi des grands nombres, il suffit à l'attaquant d'envoyer à un large panel d'utilisateurs de l'entreprise un mail sur un sujet susceptible de les intéresser, à titre personnel ou dans le cadre de son travail, pour obtenir plusieurs connexions vers le serveur d'attaque.

Le choix du contenu du mail peut être modulé par l'attaquant en fonction de l'entreprise et des employés ciblés. Un contenu relatif à la profession de la cible pourrait attirer l'attention et encourager les utilisateurs à se connecter au site. L'attaquant peut faire des recherches sur du contenu intéressant, créer une page agrégeant ce contenu et rajouter un code d'exploitation offensif sur cette page. Pour ne pas attirer l'attention, les liens sur le site dirigeront vers d'autres sites légitimes au contenu abondant.

Certains contenus obtiendront de bons résultats, quels que soient les utilisateurs cibles :

- promotion pour des vacances ;
- proposition de chèque cadeau ou de réduction sur une grande enseigne ;
- invitation de la part d'un « ami » à mettre à jour son profil sur un site de mise en réseau ;
- demande de confirmation d'un achat sur un site de vente en ligne (avec un lien pour annuler l'achat) ;
- etc...

Même si une campagne de sensibilisation a été menée, il est fort probable qu'au moins un des utilisateurs parmi les centaines visées clique sur ce lien. Et il suffit d'une seule personne compromise pour donner l'occasion à un attaquant d'obtenir un accès vers l'ensemble du réseau interne.

2 Limites des technologies protégeant les postes clients

Conscientes de l'impact de la compromission d'un poste client, et alertées par les nombreuses publications de failles sur les applications clientes, certaines entreprises mettent en oeuvre des mécanismes additionnels pour protéger le poste client. Ces mécanismes viennent en complément des protections apportées par les *firewalls*, les *IDS/IPS* et les *proxies*.

Ces mécanismes complémentaires visent notamment à :
- analyser et bloquer l'exécution de programmes dangereux ;
- filtrer les informations envoyées par l'application client pouvant permettre d'identifier le type et la version de l'application cliente utilisée, pour rendre ainsi plus complexe la réalisation d'attaques ciblées.

L'efficacité réelle de l'ensemble de ces mécanismes de protection est souvent surestimée et plusieurs idées reçues à leur sujet procurent un faux sentiment de sécurité. Cette seconde partie présente les limites intrinsèques de certaines technologies protégeant les postes clients. Elle tord aussi le cou à une autre idée reçue, selon laquelle un attaquant doit exploiter une faille de sécurité sur le poste client avant de pouvoir rebondir et attaquer les applications situées sur des serveurs internes. En réalité, il est possible d'attaquer une application accessible uniquement depuis le réseau interne sans même exploiter une vulnérabilité sur le poste client.

2.1 « Une attaque peut-elle cibler avec précision les applications clientes que j'utilise dans mon environnement ? »

Idées reçues sur les protections contre les attaques ciblées des applications clientes

Quelques idées reçues... et largement diffusées Certains utilisateurs d'applications clientes alternatives (comme les navigateurs Firefox, Safari ou Opéra) et de systèmes d'exploitation autres que Windows (MacOS, Linux) pensent échapper aux attaques, qui sont généralement destinées aux utilisateurs du système d'exploitation Windows. Ils supposent que la découverte de l'application cliente qu'ils utilisent et de son environnement est difficile pour un attaquant. En particulier, la possibilité qu'un serveur hostile puisse détecter ces spécificités et envoyer de façon automatisée une attaque ciblée n'est pas prise au sérieux.

D'autres personnes ont mis en place des protections permettant de limiter les informations communiquées par leurs applications clientes, soit en modifiant les paramètres de configuration de ces applications, soit en utilisant un proxy filtrant les fuites d'information. Ils pensent ainsi éviter toute identification et rester « incognito ».

Enfin, certaines personnes pensent être protégées par les modifications qu'ils ont apportées dans la configuration par défaut de leurs applications clientes, par exemple en désactivant le *JavaScript* et les *cookies* dans leur navigateur Web.

C'est sans compter avec le fait que des techniques de *fingerprinting* évoluées peuvent détecter automatiquement leur environnement, comme nous allons le démontrer par la suite. Auparavant, détaillons un exemple de technique de *fingerprinting* « classique » et la protection qui permet habituellement de la rendre inefficace.

Des applications clientes bien « bavardes » La plupart des applications clientes envoient dans les requêtes faites aux serveurs des informations détaillées sur leur version et parfois même sur la plate-forme hôte (système d'exploitation

et architecture). Ces informations peuvent permettre à un serveur hostile de détecter le type et la version de l'application cliente, et de renvoyer une attaque ciblée en fonction des vulnérabilités de la version utilisée.

L'en-tête *HTTP* « User-agent » est un exemple typique : il contient le nom du navigateur utilisé, sa version, le système d'exploitation hôte, et souvent la langue choisie par l'utilisateur. Le tableau 1 permet de se rendre compte des informations renvoyées par plusieurs navigateurs.

Tab. 1. En-tête « User-agent » envoyé par plusieurs navigateurs

OS	Navigateur	"User-agent"
Windows 2000	IE 5.0	Mozilla/4.0 (compatible ; MSIE 5.01 ; Windows NT 5.0)
Windows 2000	Netscape 7.1	Mozilla/5.0 (Windows ; U ; Windows NT 5.0 ; fr-FR ; rv :1.4) Gecko/20030624 Netscape/7.1 (ax)
Windows 2000	Opéra 6.0	Mozilla/4.0 (compatible ; MSIE 5.0 ; Windows 2000) Opera 6.0 [fr]
Windows XP	IE 6	Mozilla/4.0 (compatible ; MSIE 6.0 ; Windows NT 5.1 ; SV1)
Windows XP	Firefox 1.5.0.1	Mozilla/5.0 (Windows ; U ; Windows NT 5.1 ; fr ; rv :1.8.0.1) Gecko/20060111 Firefox/1.5.0.1
Windows Longhorn	IE 7 Beta	Mozilla/4.0 (compatible ; MSIE 7.0b ; Windows NT 6.0)
Debian Sarge 2.6.8	Mozilla 5.0	Mozilla/5.0 (X11 ; U ; Linux i686 ; rv :1.7.8) Gecko/20050718 Debian/1.7.8-1sarge1

Les clients mails, *usenet* et *telnet* sont d'autres exemples d'applications clientes « bavardes » [3].

Alors que la diffusion d'une bannière permettant d'identifier la version précise est de plus en plus souvent désactivée dans les configurations par défaut des applications serveurs, la plupart des applications clientes continuent de divulguer des informations sensibles.

Les techniques de dissimulation des versions des applications clientes Pour diminuer les risques d'identification, il est possible de modifier ou de supprimer la bannière envoyée par l'application cliente. Sur certaines applications, des outils existent pour modifier la bannière, sur d'autres, les modifications demandent un peu plus d'efforts.

Pour le navigateur Firefox par exemple, l'option `general.useragent.override` (accessible en entrant l'adresse `about :config`) permet de modifier le contenu de l'en-tête « User-Agent » renvoyé au serveur. Sur le navigateur Internet Explorer, la manipulation est un peu plus complexe, puisqu'il faut modifier les entrées suivantes de la base de registre :

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet
Settings\5.0\User Agent]
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet
```

`Settings\User Agent\Post Platform]`

Pour les flux *HTTP*, cette dissimulation d'information peut aussi être effectuée par le serveur *proxy*. Ainsi, *Squid* supporte l'option « fake_user_agent », qui permet de modifier à la volée les en-têtes *HTTP* avant leur sortie sur Internet. A noter cependant que les en-têtes des flux *HTTPS* ne peuvent être modifiées par le *proxy*.

Il est ainsi possible de modifier l'en-tête envoyé par un navigateur Internet Explorer sous Windows XP pour le faire passer pour un navigateur Mozilla tournant sur une plate-forme Linux. Le contenu offensif ciblant Mozilla sur Linux aura peu de chance de fonctionner sur Internet Explorer. De même, une entreprise ayant migré vers Firefox peut continuer à envoyer aux attaquants éventuels l'en-tête correspondant à Internet Explorer pour leurrer les attaquants

Comme nous allons le voir, ces techniques de dissimulation n'offrent en pratique que peu de protection face à un attaquant déterminé.

Les raisons fondamentales qui font qu'il n'est pas possible de dissimuler la version d'une application cliente Il existe des raisons fondamentales qui mettent en défaut les idées reçues que nous avons évoquées précédemment, en particulier l'idée qu'un attaquant ne sera pas capable de cibler précisément, et de façon automatisée, l'environnement des postes clients d'une entreprise donnée. De même, les techniques de dissimulation consistant à filtrer les informations envoyées sur Internet n'apportent en réalité que peu de protection face à un attaquant désireux de récolter des informations sur l'application cliente et la plate-forme hôte.

Nous présentons plusieurs techniques de *fingerprinting* permettant de réaliser une identification efficace du type et de la version d'une application cliente, et cela malgré les éventuelles tactiques de dissimulation mises en oeuvre. Ces techniques détectent de plusieurs manières les variations permettant de distinguer les applications clientes.

Tout d'abord, lors du design d'une application, les responsables projet choisissent d'implémenter ou non certaines fonctionnalités prévues par les standards, et éventuellement de rajouter des fonctionnalités propriétaires. En testant si ces fonctionnalités sont présentes ou non, il est possible de distinguer les différentes applications et leurs versions.

De plus, même si les fonctionnalités implémentées par deux applications distinctes sont similaires, il est possible de tester les différences d'implémentation de ces fonctionnalités qui ont été introduites dans le code par les développeurs. En déterminant ces différences, parfois subtiles, il est encore une fois possible de retrouver la version et le type de l'application cliente.

Ainsi, tout comme les différences entre les piles *TCP/IP* permettent de distinguer différents systèmes d'exploitation à distance, il est possible de différencier différentes applications clientes. Cette possibilité existe pour toutes les applications clientes dont les requêtes tiennent compte des réponses du serveur, c'est-à-dire la quasi-totalité. Pour ces applications, le serveur peut renvoyer des réponses spécifiques et tester la réaction du client.

Pour illustrer le propos, nous présentons des techniques de *fingerprinting* « avancées » sur un type d'application cliente largement utilisé : les navigateurs Web.

Fingerprinting d'applications clientes : l'exemple des navigateurs Web

Les informations recherchées lors du fingerprinting d'un navigateur Web sont les suivantes :

- la famille et la version du navigateur ;
- les *plugins* disponibles sur ce navigateur et leur version (lecteur audio/vidéo, Flash...), pour pouvoir exploiter éventuellement des failles de sécurité présentes dans ces *plugins*.

D'autres informations permettent à un attaquant de lancer une attaque ciblée et efficace sur un poste client :

- le type de système d'exploitation utilisé par la plate-forme et sa version (notamment le numéro de Service Pack pour Windows) ;
- le niveau des correctifs de sécurité (*patches*) pour le navigateur et le système d'exploitation ;
- le paramétrage sécurité du navigateur et du système d'exploitation ;
- le nom et la version des outils de sécurité supplémentaires installés sur le poste client (antivirus, *firewall* personnel, *antispyware*).

Les principales techniques que nous avons explorées permettant de récupérer certaines de ces informations sont les suivantes :

- Observer les différences dans les en-têtes *HTTP*. Selon l'implémentation, les valeurs et l'ordre relatif des en-têtes comme « Accept », « If-Modified-Since », « Referer » ou « Cookie » diffèrent. Ces techniques ont déjà été explorées [2]. Cependant, la valeur et l'ordre des en-têtes peuvent être modifiés par le *proxy*, ce qui en fait une technique peu fiable pour identifier le navigateur.
- Utiliser des fonctions disponibles dans les moteurs de script et les différents *plugins* des navigateurs. Les navigateurs modernes disposent de nombreux moteurs et *plugins* permettant de lancer des morceaux de code : *JavaScript*, *VBScript*, *Java*, *ActionScript* (*Macromedia Flash*)... Ces *plugins* ont souvent accès à des informations sur le navigateur et la plate-forme, et peuvent renvoyer au serveur les informations récupérées.
- Observer les différences dans le support de certaines fonctionnalités. Selon le niveau de maturité du navigateur et son respect des standards, certains fonctionnalités sont implémentées ou non dans les navigateurs. Il est facile de tester sur une page Web le support de certaines fonctionnalités et d'en déduire des candidats possibles.
- Déterminer les différences d'implémentation de certaines fonctionnalités, comme le *parsing* du code HTML. Les navigateurs interprètent différemment certaines balises HTML volontairement malformées. Cette différence est détectable par le serveur, qui recevra ou non certaines requêtes en fonction de l'interprétation faite par le navigateur.

Test du parsing *HTML pour déterminer la famille du navigateur* Le test du *parsing* de *tags HTML* malformés permet de découvrir efficacement la famille du navigateur. Ce test consiste à insérer dans la page renvoyée au serveur une série de *tags HTML* malformés demandant au navigateur d'effectuer une requête vers le serveur. Si le navigateur réussit à interpréter l'un de ces *tags*, il effectue la requête correspondante, sinon, il ne fait rien. Le serveur qui reçoit ces requêtes peut ainsi construire le profil du navigateur.

Cette technique est fiable : elle peut être utilisée même dans des environnements défavorables (par exemple, cas où le langage de script JavaScript est désactivé sur le navigateur). Tous les *tags HTML* entraînant une requête vers le serveur peuvent être utilisés, par exemple le *tag* image `<img>`. Dans le cas (peu probable) où le téléchargement des images est désactivé sur le navigateur, il est possible d'utiliser d'autres *tags HTML*. Les fichiers référencés dans les *tags* `<script>` sont ainsi souvent récupérés par les navigateurs, même si le support du langage de script correspondant est désactivé. Les feuilles de styles *CSS* (*tag* `<style>` ou `<link>`) sont aussi intéressants.

Le tableau 2 montre l'interprétation de différents *tags* image malformés dans différents navigateurs. Les cases cochées indiquent les tags interprétés, c'est-à-dire ceux pour lesquels une requête d'une image est envoyée au serveur.

Tab. 2. Interprétation de *tags HTML* malformés par plusieurs navigateurs : Internet Explorer (IE), Firefox (FF), Mozilla (MZ), Netscape (NS) et Opéra (OP)

Balise HTML	IE 5	IE 6	FF 1.0.1	FF 1.0.5	FF 1.0.7	MZ 5.0	FF 1.5.0.1	NS 7.1	OP 6.0	OP 8.53
¡img/src=”/test0001”¿	X	X								
¡im\x00g src=”/test0002”¿	X	X								
¡img\x00src=”/test0003”¿			X	X	X	X	X			
¡img\x01src=”/test0004”¿										
¡img\x0Asrc=”/test0005”¿	X	X	X	X	X	X	X	X	X	X
¡im\x0Ag src=”/test0006”¿			X	X	X	X		X		
¡img”src=”/test0007”¿										
¡img dynsrc=”/test0008”¿	X	X							X	
¡img lowsrc=”/test0009”¿	X	X								

Ce test permet de discriminer les navigateurs selon plusieurs familles :
- Internet Explorer (toutes versions)
- Firefox version 1.0.X et Mozilla (version 5.0)
- Firefox version 1.5.x
- Netscape (version 7.1)
- Opéra (version 6.0)
- Opéra (version 8.53)

Utilisation des fonctions du moteur de script d'Internet Explorer pour déterminer sa version Le test du parsing des *tags HTML* permet de différencier plusieurs

familles de navigateurs. Par contre, il ne permet pas de distinguer entre elles les dernières versions d'Internet Explorer. Il est cependant possible d'utiliser pour cela certaines fonctions du moteur de script d'Internet Explorer. Ce moteur de script interprète et exécute des scripts pouvant être écrits en langage *Visual Basic Script*, *JScript* ou *JavaScript*. Il dispose d'une série de fonctions pouvant être appelées dans ces différents langages et permettant de récupérer la version exacte du navigateur.

Le script suivant utilise les fonctions `ScriptEngineMajorVersion()`, `ScriptEngineMinorVersion()` et `ScriptEngineBuildVersion()` pour déterminer la version du moteur de script utilisé. Le résultat est renvoyé au serveur dans une requête pour une (fausse) image *jpeg* :

```
<script language="javascript"> document.writeln("<img
src='ScriptEngineVersion_" + ScriptEngineMajorVersion() + "." +
ScriptEngineMinorVersion() + "." + ScriptEngineBuildVersion() +
"'.jpg>"); </script>
```

La version du moteur de script récupérée est corrélée avec la version d'Internet Explorer installée ainsi que le niveau des correctifs installés, comme le montre le tableau 3.

Tab. 3. Version du moteur de script pour différentes versions d'Internet Explorer

OS	Version IE	Script Engine Version
Windows 2000 sans SP	IE 5.0	5.1.4615
Windows 2000 SP4	IE 5.0 SP4	5.1.8513
Windows XP SP2	IE 6.0 SP2	5.6.8820

Utilisation des « Client Capabilities » d'Internet Explorer pour récupérer la version des plugins *installés* Outre la version du moteur de script, il est intéressant pour un attaquant de déterminer les versions des composants IE installés. Cette information lui permet en effet de lancer des attaques utilisant une faille dans un de ces composants.

Les *Client Capabilities* (ou « clientCaps ») permettant d'obtenir la liste et la version des composants Microsoft installés sur Internet Explorer. La méthode `isComponentInstalled()` permet de savoir si un composant précis est installé. La méthode `getComponentVersion()` permet d'en obtenir la version.

La page suivante permet de récupérer la version d'un composant installé. L'ID utilisé est celui de Windows Media Player. La version récupérée est renvoyée au serveur dans une requête pour une (fausse) image, par exemple « WMP_10,0,0,3646.jpg ».

```
<html> <body id=oCC style="behavior:url('#default#clientCaps')">
<script language="javascript"> document.write("<img src='WMP_" +
```

```
oCC.getComponentVersion('{22d6f312-b0f6-11d0-94ab-0080c74c7e95}',
'componentid') + "'.jpg>"); </script> </body></html>
```

Les composants dont il est possible de retrouver la version sont les suivants :
- Address Book
- Windows Desktop Update NT
- DirectAnimation
- DirectAnimation Java Classes
- DirectShow
- Dynamic HTML Data Binding
- Dynamic HTML Data Binding for Java
- Internet Connection Wizard
- Internet Explorer 5 Browser
- Internet Explorer Classes for Java
- Internet Explorer Help
- Internet Explorer Help Engine
- Windows Media Player
- NetMeeting NT
- Offline Browsing Pack
- Outlook Express
- Task Scheduler
- Microsoft virtual machine

Cette liste contient des composants dont la liste des vulnérabilités connues est longue.

Utilisation des propriétés de l'objet « navigator » pour récupérer les versions du système d'exploitation et de l'architecture Interroger l'objet « navigator » est un autre moyen d'obtenir des informations sensibles. Il contient plusieurs propriétés stockant notamment des informations sur le système d'exploitation de la plate-forme et l'architecture matérielle. Ces propriétés peuvent être récupérées par des techniques similaires à celles évoquées précédemment.

Voici quelques-unes des propriétés pouvant intéresser un attaquant :
- `navigator.userAgent` : elle contient la valeur de l'en-tête HTTP « User-Agent » renvoyé par le navigateur. Sa valeur peut-être modifiée dans la configuration. Si elle ne l'est pas, cette propriété permet de récupérer la valeur de l'en-tête « User-Agent » en évitant le filtrage du *proxy*.
- `navigator.appMinorVersion` : disponible sur Internet Explorer, cette propriété indique la version du Service Pack de Windows qui est installée.
- `navigator.plateform` : elle indique la famille de système d'exploitation installé sur la plate-forme.
- `navigator.cpuClass` : disponible sur Internet Explorer, elle indique la famille du microprocesseur.
- `navigator.oscpu` : disponible sur la plupart des navigateurs hors Internet Explorer, elle indique la famille du système d'exploitation utilisé.

Ces propriétés ne sont pas disponibles sur tous les navigateurs, et lorsqu'elles le sont, leur valeur peut être différente selon le navigateur. Cela constitue un autre moyen de réaliser un *fingerprinting* de la famille du navigateur.

Le tableau 4 montre le contenu de quelques propriétés selon le navigateur.

Tab. 4. Propriétés de l'objet « navigator » selon le navigateur et la plate-forme

Propriété	IE 6 (Win XP SP2)	FF 1.5.0.1 (Win XP SP2)	MZ 5.0 (Debian Sarge 2.6.8)
navigator.userAgent	Mozilla/4.0 (compatible ; MSIE 6.0 ; Windows NT 5.1 ; SV1)	Mozilla/5.0 (Windows ; U ; Windows NT 5.1 ; fr ; rv :1.8.0.1) Gecko/20060111 Firefox/1.5.0.1	Mozilla/5.0 (X11 ; U ; Linux i686 ; rv :1.7.8) Gecko/20050718 Debian/1.7.8-1sarge1
navigator.appMinorVersion	;SP2 ;		
navigator.platform	Win32	Win32	Linux i686
navigator.cpuClass	x86		
navigator.oscpu		Windows NT 5.1	Linux i686

Utilisation des objets « navigator.plugins » et « mimeTypes » pour récupérer la version des *plugins* installés La récupération d'informations sur les *plugins* disponibles sur le navigateur est intéressante pour un serveur hostile, puisque ces *plugins* peuvent contenir différentes failles de sécurité pouvant être utilisées pour prendre le contrôle du poste client.

L'objet « navigator.plugins » est disponible sur la plupart des navigateurs hors Internet Explorer, et contient la liste des *plugins* installés, une description de ces *plugins* ainsi que le nom du fichier utilisé par le *plugin*. Le nom du fichier permet entre autre de distinguer différents systèmes d'exploitation : sur les systèmes Windows on remarque l'extension « .dll », alors que sur les systèmes Linux, ces fichiers ont une extension « .so ». Dans certains cas (navigateur Opéra), le chemin complet vers le fichier est indiqué, par exemple :

```
"C:\Program Files\Opera\Program\Plugins\PlugDef.dll"
```

L'objet « navigator.mimeTypes » est lui aussi disponible sur la plupart des navigateurs hors Internet Explorer, et contient la liste des type MIME supportés par le navigateur, ainsi que le nom du *plugin* nécessaire pour lire le type correspondant.

Conclusion Comme le montrent ces quelques exemples, un serveur hostile peut déterminer suffisamment d'informations sur le navigateur, ses *plugins* et l'OS pour lancer une attaque ciblée. Les utilisateurs de navigateurs alternatifs et/ou

de systèmes d'exploitation moins répandus ne sont ainsi pas à l'abri d'un serveur capable de les reconnaître et d'envoyer un contenu offensif adapté.

D'autres technologies peuvent être utilisées pour obtenir des informations sur le client : les *applets Java*, les *ActionScript* de *Flash*, le support de certains formats d'image, les composants *ActiveX*, le support des *xmlHTTPRequest*... Les techniques de dissimulation de la version du navigateur ne peuvent contenir toutes les fuites d'informations engendrées par ces nombreuses fonctionnalités.

Pour ces raisons, le problème ne peut pas être résolu simplement. Il est plus sage de partir du principe qu'un serveur hostile sera capable de déterminer l'environnement exact du poste client, et pourra tenter d'exploiter une faille non corrigée dans un des composants du navigateur.

2.2 Limites des dispositifs de protection contre l'exécution de code malveillant

2.3 Idée reçue : « **De multiples lignes de défense (antivirus, antispyware, pare-feu, proxy...) rendent impossible la prise de contrôle d'un poste de travail depuis Internet** »

Même s'il est possible de cibler précisément une attaque sur une application cliente, cela a-t-il une importance puisque des protections permettent, d'une part, de détecter et d'éradiquer les programmes exécutables malveillants, et d'autre part, d'empêcher un programme exécutable non autorisé de se connecter à Internet ? Ainsi, certaines personnes considèrent que même si un attaquant arrive à attaquer le poste client, il ne pourra pas exécuter son cheval de Troie et l'utiliser pour accéder au réseau interne de l'entreprise depuis Internet.

L'antivirus est considéré aujourd'hui comme l'une des protections les plus efficaces pour garantir la sécurité du système d'information d'une organisation. Suite aux pertes engendrées par les premières vagues massives de virus se propageant par Internet, de nombreux projets antivirus ont été lancés et la majorité des organisations disposent aujourd'hui d'une infrastructure antivirus solide.

Alors que les premiers antivirus se présentaient comme des outils permettant de bloquer des programmes malveillants bien identifiés, les discours marketing actuels tendent à en faire une solution miracle, permettant de lutter contre toutes les formes de menaces informatiques. Il est ainsi présenté comme un moyen de lutter contre les *spywares*, les virus, les outils de *hacking* comme les chevaux de Troie, et de façon générale contre tous *malwares* pouvant causer du tort à l'utilisateur.

Il est vrai que l'apparition des antivirus à moteur heuristique appuie ce discours. Ces moteurs heuristiques ne comparent plus le code des programmes suspects à une liste de signature prédéterminées, mais analysent son code binaire pour y rechercher les signes caractéristiques d'une activité virale (tentative de réplication, présence d'une routine de déchiffrement ou de polymorphisme, envoi au carnet d'adresses, écriture dans des fichiers sensibles...). Il devient ainsi théoriquement possible de détecter certains nouveaux programmes malveillants,

pour lesquels aucune signature n'est présente dans la base de l'antivirus. C'est ainsi que se présentent certains antivirus à moteur heuristique.

Les organisations qui ne veulent cependant pas faire reposer leur sécurité sur un seul éditeur ont trouvé une sécurité supplémentaire : pour augmenter la probabilité de détection des codes malveillants, elles installent les antivirus de deux éditeurs différents sur les serveurs et sur les postes clients. En cas de défaillance du moteur d'un éditeur, qui ne détecterait pas un programme malveillant particulier, ces organisations comptent sur le second pour bloquer l'attaque à temps. Certains produits proposent même d'intégrer (par exemple sur une même passerelle de messagerie) plusieurs moteurs antivirus, provenant là aussi d'éditeurs différents. Le contenu suspect est analysé successivement par tous les moteurs disponibles : si rien n'est détecté, c'est vraiment que le code est inoffensif !

C'est du moins l'idée reçue que l'on entend souvent... En réalité, les antivirus restent un outil créé pour lutter contre une menace « standard », et à ce titre là, ne détectent généralement que les menaces « standard ». Ce que détectent les antivirus, c'est le bruit de fond de l'Internet que représente chaque nouvelle vague de virus, attaquant aveuglement des adresses IP aléatoires, ou les outils de *hacking* disponibles sur Internet depuis tellement longtemps qu'ils ont eu droit à leur propre signature dans les bases de données des éditeurs.

Nous montrerons qu'une attaque ciblée, construite par une personne mal intentionnée dans le but de compromettre le système d'information d'une organisation précise, n'est pas une menace « standard ». Cette attaque réfléchie peut contourner plus facilement qu'on ne le pense le filtrage effectué par les antivirus situés en périmètre du réseau interne et sur les postes de travail.

Il est bien connu qu'un *malware* programmé pour une porter atteinte à une cible précise peut ne pas être détecté par l'antivirus. Nous présentons dans cette partie une autre limite inhérente au fonctionnement des antivirus. Cette limite permet d'utiliser un contenu offensif connu, c'est-à-dire dont la signature est présente dans la base de l'antivirus, pour réaliser une attaque qui ne sera pas détectée et arrêtée par cet antivirus. Nous présenterons aussi les autres dispositifs de protection censés empêcher la connexion à Internet d'un programme malveillant en mettant l'accent sur les limites intrinsèques de ces dispositifs.

Limites fondamentales des dispositifs de protection contre l'exécution de code malveillant Le champ d'action des protections apportées par les antivirus et les filtres réseau périmétriques est limité par certaines causes fondamentales, que nous allons détailler. Nous montrons que, malgré l'existence de ces dispositifs de protection, un code malveillant peut atteindre un poste de travail, y être exécuté sans être détecté, et établir un canal de communication entre Internet et le réseau interne.

Antivirus et logiciels assimilés

Description du dispositif de protection Les logiciels dédiés à la détection et à l'éradication des contenus exécutables malveillants se diversifient sur le plan des dénominations marketing (antivirus, anti-*spyware*, anti-*trojan*,

anti-*malware*, filtrage de sécurité Web...) tout en restant basés sur les mêmes grandes techniques de détection :
- base de signature, qui reste l'outil principal ;
- émulation du code ;
- exécution du code dans une *sandbox* ou une machine virtuelle ;
- analyse heuristique du code.

Limites intrinsèques de ces dispositifs de protection La suite d'octets composant la « signature » de tout programme malveillant connu, comme un cheval de Troie, peut être éliminée en compressant ou chiffrant le programme à l'aide d'une routine choisie par l'attaquant. De ce fait, un antivirus ne peut détecter la présence du code malveillant qu'en émulant la routine de déchiffrement ou de décompression du programme, ou en l'exécutant dans un environnement contrôlé (*sanbox*).

D'habitude, un programme s'exécute au sein du microprocesseur sous le contrôle du système d'exploitation de l'utilisateur. Un code émulé ou exécuté dans une *sandbox* tourne dans un environnement très différent de l'environnement normal d'exécution. Ces différences d'environnement peuvent être détectées par le code soumis à l'analyse, lequel est donc capable de modifier son comportement en fonction de l'environnement qu'il « voit ». Il est donc possible à tout programme utilisant ce principe d'exhiber un comportement inoffensif lorsqu'il est analysé par un antivirus, et un comportement hostile lorsqu'il est exécuté normalement sur un système.

A titre d'exemple, on peut imaginer une routine de déchiffrement d'un cheval de Troie qui utilise comme clé secrète une valeur obtenue dynamiquement, qui dépend :
- des données retournées par certains appels systèmes ;
- du contenu de certaines zones de la mémoire du processus ou du système ;
- du temps d'exécution de certaines instructions assembleur ;
- du contenu de certains fichiers ;
- du contenu d'une page Web donnée, récupérée en lançant en tâche de fond le navigateur de l'utilisateur ;
- ...

Dans le cadre d'une exécution émulée par un antivirus, ce dernier ne saura pas renvoyer les valeurs exactes correspondant à une exécution normale, et la clé de déchiffrement générée sera invalide. Le code malveillant (cheval de Troie) n'étant pas déchiffré, l'antivirus ne le détectera pas. Par contre, lorsque le programme sera effectivement exécuté sur le poste de travail, la clé de déchiffrement correcte sera obtenue et le programme hostile sera exécuté.

En effet, il est impossible pour un logiciel de filtrage de contenu exécutable d'émuler parfaitement l'environnement d'exécution réel d'un programme sur le poste de travail. C'est bien évidemment le cas pour les analyses qui ont lieu au niveau du filtrage périmétrique sur une autre machine et parfois un autre système d'exploitation (serveur de messagerie, serveur *proxy* Web). Mais c'est également le cas si l'émulation a lieu sur la même

machine, ne serait-ce que du fait de ses interactions avec l'extérieur via le réseau.
Remarquons qu'un logiciel antivirus qui tenterait naïvement de contourner ce problème en autorisant l'exécution du fichier à analyser, puis en stoppant temporairement son exécution pour « scanner » le contenu de la mémoire du processus, n'aurait pas plus de succès. En effet, le code hostile aurait alors le choix, pour être indétectable, entre une exécution très rapide de son action (par exemple une injection de code dans un autre processus), ou bien une inactivité de quelques minutes avant le déclenchement de la routine de déchiffrement du contenu hostile.
Il est donc possible pour un attaquant d'utiliser n'importe quel logiciel « cheval de Troie » fonctionnel, déjà connu, pour effectuer ses oeuvres malveillantes. Il lui suffit pour cela de le protéger de la détection en lui adjoignant une petite routine dépendant de l'environnement d'exécution. Par la suite, en cas de détection par un antivirus de cette routine (ajout de la routine dans la base de signature de l'antivirus), l'attaquant n'a pas à ré-écrire un nouveau cheval de Troie complet : il lui suffit pour éviter la détection de modifier la petite partie initial de code en interrogeant une autre partie de l'environnement d'exécution.
En conséquence, un logiciel antivirus ne peut fournir non seulement aucune assurance sur la détection des logiciels malveillants qui ne sont pas déjà connus (c'est-à-dire qui ne sont pas référencés dans sa base de signature), ni sur la détection des logiciels malveillants connus mais qui auraient été « protégés » selon les principes exposés précédemment. Il s'agit là d'une limite intrinsèque des mécanismes de détection de code malveillant, à laquelle il n'est pas possible d'apporter une réponse simple.

Filtrage réseau périmétrique

Description du dispositif de protection Il est courant qu'un pare-feu interdise toute connexion directe depuis le réseau interne de l'entreprise vers Internet. Seules les connexions Web (protocoles *HTTP* et *HTTPS*) et de transfert de fichier (protocole *FTP*) sont généralement autorisées, mais pas directement : le poste client doit passer nécessairement par un relais (serveur *proxy*) pour accéder à Internet. Ce relais, dans l'idéal, est configuré pour ne relayer que les requêtes correctement authentifiées par l'envoi d'un identifiant et d'un mot de passe valides.
Par ailleurs, le relais peut intégrer des fonctionnalités de sécurité additionnelles, comme par exemple :
- contrôle d'accès restreignant les sites Web dont la visite est autorisée ;
- analyse et filtrage du contenu transitant par le relais (*JavaScript*, *ActiveX*, *applets Java*, fichiers exécutables...) pouvant inclure la vérification par un logiciel antivirus des fichiers téléchargés et des pages Web accédées.

Ces différentes protections ne sont néanmoins pas efficaces pour empêcher qu'un code malveillant, exécuté sur le poste de travail, n'établisse un canal de communication avec l'attaquant situé sur Internet.

Limites intrinsèques de ces dispositifs Les limites de ces dispositifs de protection ont été présentées lors d'éditions antérieures du SSTIC. Nous nous bornerons à les rappeler.

Ainsi, l'article [4] détaillait plusieurs méthodologies d'attaque d'un poste client situé au sein d'un réseau entreprise et protégé par l'ensemble de ces dispositifs. Il présentait une « backdoor » qui s'injectait au sein du processus du navigateur Web pour utiliser son paramétrage et contourner ainsi les problèmes liés à l'existence de proxies, d'authentification, et de pare-feu.

Précédemment, une implémentation d'un cheval de Troie utilisant les objets OLE pour contrôler Internet Explorer et contourner ainsi ces protections avait été détaillée dans [5].

Par ailleurs, les dispositifs périmétriques de filtrage de contenu Web (*JavaScript*, *ActiveX*, *applets Java*, fichiers exécutables...) sont soumis aux mêmes limites intrinsèques que les logiciels antivirus. Ces limites ont été décrites dans la section précédente.

Enfin, quelles sont les limites de l'interdiction d'accès à certains sites, qui pourrait par exemple empêcher un cheval de Troie d'effectuer une communication de type « reverse connect » vers Internet (technique du « tunneling HTTP » vers un serveur contrôlé par l'attaquant) ? Il faut distinguer deux cas selon les modes de filtrage possibles :

- liste noire[2] : le site du pirate ne sera pas situé dans la liste dans le cas d'une attaque ciblée, ou dans le cas d'une attaque massive, il n'y sera intégré que lorsqu'il sera trop tard ;
- liste blanche[3] : ce type de filtrage est rarement présent en entreprise car il est difficile à mettre en place. De plus, il ne protège pas contre le « reverse connect » d'un cheval de Troie s'il y a une possibilité de poster des messages sur l'un des sites autorisés : ces messages peuvent servir de moyens de communication avec l'extérieur si le forum est accessible à la fois depuis le réseau interne et depuis Internet. De même, si une faille de sécurité de type XSS (Cross-Site Scripting) est présente sur l'un des sites autorisés, toutes les attaques clientes sur le navigateur Web redeviennent possibles. De même en cas de faille de sécurité dans un serveur de la liste blanche, si cette faille entraîne sa compromission et la possibilité pour un attaquant d'y déposer du code offensif.

Filtrage réseau sur l'hôte

Description du dispositif de protection Sur l'hôte, un pare-feu dit « *firewall* personnel » peut être configuré. Ce type de pare-feu peut interdire :

- d'une part, toute connexion depuis le réseau vers le poste de travail ;
- d'autre part, toute connexion vers le réseau de tout programme exécuté localement qui ne serait pas contenu dans une liste blanche d'applications autorisées.

[2] Principe de filtrage consistant à autoriser tout ce qui n'est pas explicitement interdit.
[3] Principe de filtrage consistant à interdire tout ce qui n'est pas explicitement autorisé.

Limites intrinsèques de ce dispositif Comme expliqué dans la section précédente, il est possible d'injecter du code dans le processus du navigateur Web, ou de contrôler ce dernier par différents moyens (les objets OLE n'étant qu'un exemple), pour être en mesure de réaliser des requêtes réseau arbitraires vers des sites Web quelconques. Ces requêtes seront autorisées par le pare-feu personnel.

Démonstration des limites des antivirus

Concept Nous avons développé un code de démonstration utilisant le principe général que nous avons présenté, à savoir qu'il est possible de réaliser un code qui exécute des actions différentes en fonction de son environnement d'exécution (émulation ou exécution normale).

L'article [6] montre que certaines techniques anti-émulation ont été utilisées dans le passé par des développeurs de virus (instructions assembleur inconnues des émulateurs, mauvaise gestion des exceptions, des threads...). Ce sont néanmoins des techniques que les antivirus sont en mesure de contrer, à la différence de la technique plus universelle que nous présentons.

Remarque. Nous avons basé cette démonstration sur une modification du logiciel de compression d'exécutables UPX[7], dont le code source est disponible sous une licence libre. Ce choix pragmatique de réutilisation d'un code existant avait pour objectif d'éviter la charge de travail liée à la reprogrammation d'un code assembleur chargeant les sections d'un programme exécutable en mémoire. Un attaquant motivé pourrait s'affranchir totalement d'une telle réutilisation de code.

Implémentation Le contournement que nous avons implémenté repose sur la réalisation d'une routine de détection de l'environnement d'exécution par mesure du temps d'exécution de certaines instructions assembleur. Cette approche est similaire aux techniques de détection de machines virtuelles présentées au SSTIC 2004 [8].

Si cette routine détecte qu'elle s'exécute dans l'environnement contrôlé par l'antivirus (*sandbox*), le code boucle en attendant la fin de l'analyse ou saute à une adresse contenant du code invalide. Lorsque la routine détecte qu'elle s'exécute directement sur le microprocesseur au sein du système d'exploitation du poste de travail, le code s'oriente vers la routine de décompression et d'exécution du contenu offensif connu. Ce contenu offensif est alors exécuté, sans avoir été bloqué par l'antivirus alors qu'il est pourtant présent dans sa base de signatures.

Voici un exemple de code assembleur que nous avons pu ajouter au *stub* de lancement d'UPX pour interdire aux antivirus d'analyser le programme compressé par UPX :

```
PUSH EDI        ; GAEL: start of environment-dependent code
PUSH ESI
PUSH EBP
```

```
        PUSH EAX
        PUSH EBX
        PUSH ECX
        PUSH EDX
        CALL envdep_geteip
envdep_geteip:
        POP ECX           ; get current value of the EIP register

        dw 0x310F         ; RDTSC : mesure du temps
        MOV EBX,EAX
        dw 0x310F         ; RDTSC : mesure du temps
        SUB EAX,EBX       ; delta : temps d'exécution des instructions
        SHR EAX,4
        ADD EAX, ECX
        db 0x83
        db 0xC0
        db 0x13           ; ADD EAX, 0x13
envdep_loop:
        JMP EAX           ; <--- saut à une adresse dépendant du temps
        JMP SHORT envdep_loop
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
```

```
MOV DWORD ['ADR0'], 'VAL0'  ; modification de la partie compressée
                            ; par UPX pour se protéger des antivirus
                            ; utilisant une décompression directe
                            ; au lieu d'une émulation
POP EDX
POP ECX
POP EBX
POP EAX
POP EBP
POP ESI
POP EDI         ; GAEL: end of environment-dependent code
```

Nous avons réalisé des tests en protégeant ainsi un logiciel classique de type « cheval de Troie » détecté par 24 antivirus différents. Les résultats de ces tests sont synthétisés dans le tableau 5.

Tab. 5. Taux de détection d'un cheval de Troie

Contenu de l'exécutable	Virus détecté	Programme « suspicieux »	Pas de détection
Cheval de Troie (CDT) seul	24	0	0
CDT compressé par UPX	16	8	0
CDT compressé par UPX et protégé	0	2	22

2.4 Exposition des applications internes aux attaques provenant d'Internet)

Idée reçue :
« **Les applications internes de l'entreprise sont protégées des attaques externes puisqu'elles ne sont pas accessibles depuis Internet** » Les applications Web sont à la mode aujourd'hui. Plus simples à déployer et à maintenir, sans impact sur les postes clients, elles ont de nombreux avantages. Certes, elles nécessitent de prendre en compte certains risques spécifiques en matière de sécurité. En effet, l'attaquant n'a plus besoin de disposer d'une application cliente spécifique (excepté un navigateur Web) pour se connecter et essayer d'attaquer l'application. De plus, les protocoles de l'Internet et les technologies du Web, lorsqu'ils sont mal utilisés, présentent des failles qui sont pour la plupart bien connues par les personnes mal intentionnées.

Pour ces raisons, les organisations soucieuses de leur sécurité informatique choisissent généralement :

- de limiter lorsque cela est possible l'accès de leurs applications Web aux postes du réseau interne (principe de l'Intranet) ;
- de mettre en place une authentification forte pour protéger l'accès aux applications devant être accessibles depuis Internet ;

– d'appliquer les principes de développement sécurisé dans leurs projets Web (quoique ce constat soit plus sous notre plume un souhait qu'une réalité).

Une idée reçue veut que ces mesures de protection soient suffisantes pour garantir la sécurité d'une application Web.

Cependant, nous montrons qu'un type méconnu d'attaque permet d'accéder depuis Internet à une application Web :

– accessible uniquement depuis le réseau interne ;
– même protégée par une authentification forte ;
– ne souffrant d'aucune « faille » de sécurité (selon les bonnes pratiques actuelles de développement).

Pour réussir cette attaque, l'attaquant a besoin d'un relais qui lui permettra de rentrer dans le réseau interne et d'envoyer des requêtes vers l'application web. Ce relais, c'est le navigateur d'un poste client ayant visité au préalable une page Web contrôlée par l'attaquant.

Limites fondamentales des dispositifs d'isolement entre Internet et les applications internes L'accès simultané des utilisateurs à Internet et aux applications internes introduit l'existence d'un canal de communication entre Internet et les applications internes, introduisant ainsi une sorte de routage ou relais involontaire. Ce relais peut se faire de plusieurs manières.

D'une part un attaquant ayant pris le contrôle d'un poste de travail a accès à une partie du réseau interne de l'entreprise à partir d'Internet ainsi qu'à l'ensemble des applications et ressources d'infrastructures auxquelles l'utilisateur a accès, avec les droits de l'utilisateur (utilisation d'un cheval de Troie avec reverse connect).

D'autre part, il n'est pas nécessaire pour un attaquant de prendre le contrôle complet d'un poste de travail situé sur le réseau interne pour accéder à certaines applications internes ou externes. En effet, certaines attaques sont rendues possibles par deux faits :

– le processus « navigateur Web » tournant sur le poste de travail est autorisé à réaliser des requêtes Web à la fois vers Internet et vers le réseau interne ;
– le modèle de sécurité des navigateurs Web autorise toute page Web à réaliser des requêtes vers toutes autres pages Web, pouvant être situées sur des serveurs arbitraires, qui peuvent être internes ou externes.

En conséquence, un code actif malveillant inclus dans une page *HTML* visionnée sur Internet par un utilisateur, comme un script *JavaScript*, pourrait effectuer des requêtes vers les applications internes. Il n'est pas nécessaire pour cela d'exploiter une faille de sécurité dans le navigateur ou dans les dispositifs de protection du réseau de l'entreprise.

Il suffit qu'une page Web contrôlée par l'attaquant reste d'une manière ou d'une autre « présente » (*popup* invisible...) et qu'elle lance à intervalles réguliers des requêtes à destination de l'application Web interne visée. Si cette application nécessite une authentification (y compris une authentification forte), il faudra que l'utilisateur du poste de travail ciblé s'authentifie pour que l'attaque fonctionne.

Une fois que l'utilisateur est authentifié, si la page « d'attaque » est toujours « présente », les requêtes qu'elle envoie à destination de l'application seront interprétées par le serveur interne comme des demandes légitimes provenant de l'utilisateur authentifié. Les actions correspondantes seront réalisées avec les droits de l'utilisateurs (envoi par le navigateur de son cookie de session à l'application Web), et sembleront provenir du poste client de cet utilisateur.

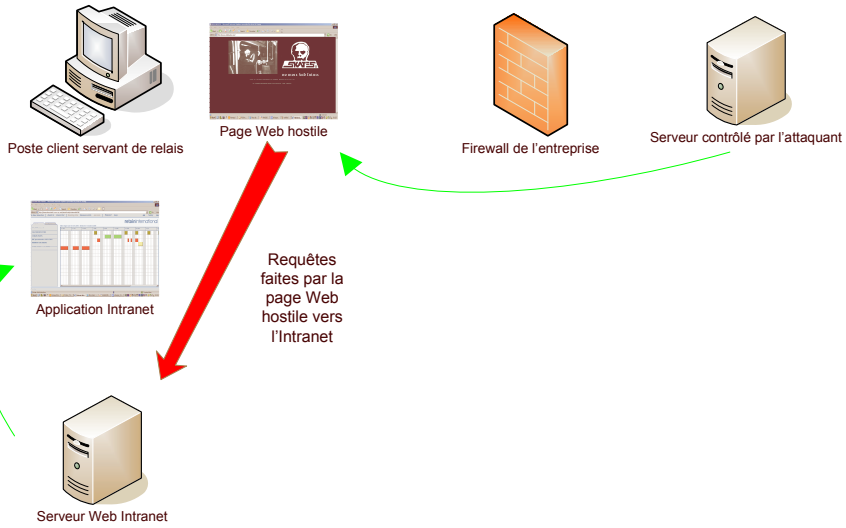

Fig. 3. Attaque d'une application Web interne par une page HTML visionnée sur Internet

3 Automatisation des attaques sur les postes clients : framework de démonstration

Cette partie présente un framework[4] de démonstration permettant de prouver le risque d'automatisation des attaques sur les postes clients. Il utilise les techniques de *fingerprinting* et d'exploitation présentées précédemment, ainsi que d'autres techniques déjà publiées. Ce framework a été développé par Renaud Feil et Gaël Delalleau en langage *Python*. Le code source des différents

[4] Ce développement est un « framework », car il définit la relation entre différents modules. Si pour l'instant, seuls les protocoles HTTP et HTTPS sont développés, la structure du « framework » permet de rajouter facilement des modules supportant d'autres protocoles comme vecteur d'attaque.

scripts de ce framework de démonstration n'est pas publié, néanmoins il est possible que des outils similaires soient déjà utilisés ou en cours de développement par des personnes mal intentionnées.

3.1 Difficulté des attaques sur les postes clients et intérêt du framework de démonstration

Les vulnérabilités présentées dans la partie précédente démontrent les possibilités de compromission d'un poste client et des applications Web du réseau interne, et ce malgré les technologies de protection mises en oeuvre. Cependant, la réalisation d'une attaque sur un poste client n'est pas triviale. En effet, cette attaque se fait sur un laps de temps très court : les quelques secondes durant lesquelles l'utilisateur ciblé se connecte au serveur d'attaque et récupère la réponse du serveur.

Lorsqu'il dépose un code offensif sur un serveur, l'attaquant ne peut savoir de façon précise :

- à quel moment les utilisateurs ciblés se connecteront
- quelles seront les applications clientes utilisées et leurs versions
- quels seront les systèmes d'exploitation sous-jacents
- quels seront les filtrages et les mécanismes de sécurité pouvant bloquer certains types d'attaques.

L'attaquant peut faire des suppositions sur ces paramètres, mais il n'est pas certain de deviner juste. De même, la réalisation d'une reconnaissance préalable est fastidieuse et pas toujours efficace.

A priori, les cibles ne se connecteront qu'une fois au contenu offensif. Si l'attaque cible plusieurs centaines d'utilisateurs dans une entreprise, il est fort possible que la plupart des connexions aient lieu très rapidement (par exemple, quelques minutes à quelques heures après la réception d'un courrier électronique). Les utilisateurs n'ayant pas lu le contenu d'un courrier électronique et cliqué sur le lien ne sont probablement pas intéressés. Ainsi, si les premières attaques échouent, il n'est pas facile pour l'attaquant de modifier le contenu offensif à temps pour pouvoir compromettre les postes clients. Il devra relancer une nouvelle vague d'envoi de courriers électroniques pour encourager les cibles à se connecter au contenu offensif. Ces répétitions peuvent être nombreuses, coûteuses en temps et de moins en moins efficaces si les utilisateurs et le département sécurité de l'organisation ciblée remarquent les courriers électroniques suspects.

Pour augmenter la probabilité de réussite de leurs attaques, des personnes mal intentionnées sont susceptibles de développer des outils permettant d'automatiser l'attaque des postes clients. Ces outils leur permettent de détecter les caractéristiques de l'environnement ciblé, de le soumettre aux attaques les plus pertinentes et de maintenir un canal de communication avec les postes clients compromis (et situés sur le réseau interne de l'entreprise).

Ainsi, tout comme on a vu apparaître il y bien longtemps des outils automatisés comme SATAN[9] (et aujourd'hui Nessus[10] ou Core Impact[11]), permettant d'automatiser les tests de vulnérabilités sur des applications serveurs, il est

prévisible que des outils permettant d'automatiser l'attaque des postes clients soient développés et se généralisent.

Afin d'évaluer le niveau de risque qui pèse sur les postes clients, et de participer à la prise de conscience de ce risque, nous avons développé un programme de démonstration du principe d'attaque automatisée contre les postes clients. Ce framework permet de mieux comprendre la gravité de la menace contre laquelle il s'agit de protéger les postes clients.

3.2 Architecture du démonstrateur

L'objectif de ce démonstrateur est de présenter les risques actuels qui pèsent sur les applications clientes. Pour cela, le démonstrateur intègre plusieurs techniques de *fingerprinting* et d'attaque et les assemble comme le ferait un attaquant voulant compromettre un grand nombre de postes clients.

Ce framework est développé en langage *Python*, avec un *SGBDR*[5] *MySQL*.

Il est découpé en différents modules :

- Le module « fakemail » : il génère les courriers électroniques encourageant les utilisateurs ciblés à se connecter au serveur d'attaque.
- Le module « track » : il assure le suivi des attaques en cours, des succès et des échecs.
- Le module « core » : il gère les connexions entrantes des postes clients sur différents ports et fait appel aux différents modules intervenant dans l'attaque.
- Le module « fingerprinting » : il réalise l'identification de l'application cliente et du système d'exploitation.
- Le module « content » : il consolide le contenu à renvoyer à l'utilisateur à partir des bases de données « d'exploits » (code offensif permettant d'exploiter une faille de sécurité sur l'application cliente), « fakecontent » (contenu permettant de ne pas éveiller l'attention de l'utilisateur) et « payload » (programme dont l'exécution sera le résultat concret de l'attaque réalisée sur le poste client)

Les modules s'articulent de la façon décrite dans la figure 4.

Dans sa première version, le framework gère uniquement les protocoles *HTTP* et *HTTPS*. Le support d'autres protocoles (par exemple *streaming audio/vidéo*, *telnet*, *SMTP*, *POP3* et *IMAP4*...) peut être rajouté en dupliquant le module « core » et en intégrant dans les modules « fingerprinting » et « content » les techniques de détection de l'environnement et les attaques adaptées.

Module « fakemail » : génération des courriers électroniques envoyés aux utilisateurs Le module « fakemail » génère les courriers électroniques encourageant les utilisateurs ciblés à se connecter au serveur d'attaque.

Le module attend en entrée une liste contenant les champs suivants :

- adresse de courrier électronique ;

[5] Système de Gestion de Base de Données Relationnel

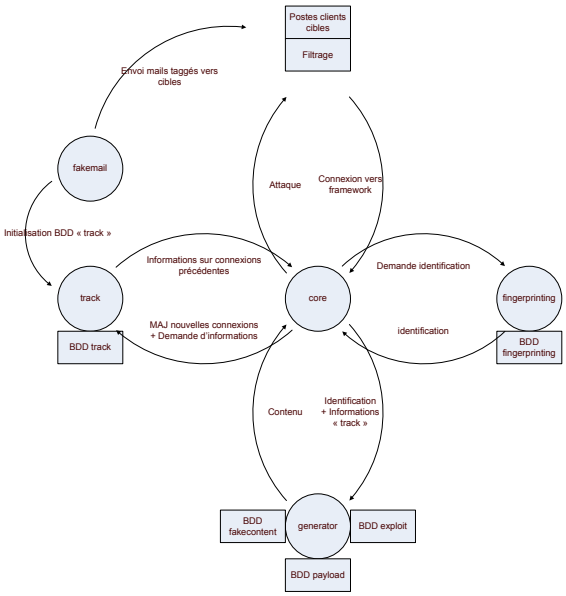

Fig. 4. Architecture du framework de démonstration

- thème à utiliser pour le contenu du courrier électronique (parmi des thèmes prédéfinis), c'est-à-dire le texte et les images à utiliser pour attirer la curiosité de l'utilisateur et le pousser à cliquer sur l'un des liens du courrier électronique (principe du *phishing*) ;
- objectif de l'attaque sur le poste client de cet utilisateur (parmi une liste d'objectifs possibles) ;
- nom de l'organisation ciblée et plage d'adresse IP permettant de restreindre la diffusion de l'attaque (si l'attaque provient d'une adresse IP ne faisant pas partie de la plage définie, un contenu inoffensif est renvoyé à l'application cliente).

Ce module génère un *tag* aléatoire pour chaque entrée de cette liste. Ce *tag* est stocké avec les informations fournies en entrée dans la base de données du module « track ». Le *tag* est aussi intégré dans les liens envoyés par courrier électronique à l'utilisateur correspondant. Il permettra ainsi d'identifier l'utilisateur lorsqu'il se connecte au serveur.

Différents thèmes peuvent être créés, selon une logique présente dans certains kits de *phishing* disponibles sur Internet. Des exemples de thèmes sont :

- chèque cadeau dans une grande enseigne ;
- achat en ligne à annuler ;
- promotion pour des vacances ;
- site de mise en réseau.

Une fois la totalité des courriers électroniques envoyés, le module « fakemail » n'intervient plus dans le déroulement de l'attaque.

Module « track » : suivi des attaques en cours Le module « track » réalise le suivi des attaques en cours. Il enregistre les informations sur les courriers électroniques envoyés, les connexions, les succès et les échecs des différentes attaques.

La base de données est initialisée à la demande du module « fakemail ». Elle est ensuite mise à jour tout au long du déroulement de l'attaque à la demande du module « core » lors des différents événements :

- connexion d'une application cliente ;
- *fingerprinting* réussi d'une application cliente et de son environnement ;
- envoi d'un contenu offensif au client ;
- réussite ou échec de l'attaque envoyée au client.

En particulier, ce module stocke les échecs de certaines attaques. De cette façon, le module « content » peut adapter progressivement son attaque lors des connexions d'autres applications clientes de la même organisation cible. Si un mécanisme de sécurité ou une erreur quelconque fait échouer un certain type d'attaque, une attaque différente sera lancée contre les prochains postes clients se connectant.

Module « core » : le chef d'orchestre de l'attaque Le module « core » attend les connexions des applications clientes sur plusieurs ports TCP et UDP. Lorsqu'une connexion arrive, il informe le module « track » de cette connexion,

et demande une identification de l'application cliente et de son environnement au module « fingerprinting ». Une fois l'identification réalisée, il envoie les informations au module « content ». Il récupère le contenu offensif qu'il renvoie alors à l'application cliente ciblée.

Module « fingerprinting » : identification de l'environnement du poste client Le module « fingerprinting », à la demande du module « core », renvoie le contenu des tests à effectuer pour déterminer l'environnement du poste client. Il traite les informations renvoyées par l'application cliente, et à l'aide de sa base de signature, détermine les candidats les plus probables pour différents éléments :

- la famille et la version de l'application cliente et des éventuels *plugins* disponibles ;
- le système d'exploitation utilisé et sa version (notamment les Service Pack pour Windows) ;
- le niveau probable des correctifs de sécurité pour le navigateur et le système d'exploitation ;
- le paramétrage sécurité du navigateur et du système d'exploitation ;
- si possible, le nom et la version des outils de sécurité supplémentaires installés sur le poste client (antivirus, *firewall* personnel, *antispyware* et autres *antimalwares*).

Les tests permettant de récupérer ces informations sont en partie présentés dans la partie précédente. L'en-tête « User-agent » du protocole HTTP n'est pas utilisée, puisque cette information peut être modifiée ou supprimée facilement.

A noter que la présence d'un serveur *proxy* peut être détectée par l'observation de certains en-têtes HTTP, ainsi que par un interfaçage avec l'outil « p0f »[12]. Si les en-têtes caractéristiques de l'utilisation d'un *proxy* ne sont pas présents, l'interfaçage avec p0f peut permettre de détecter des incohérences entre le système d'exploitation détecté par le *fingerprinting* applicatif, et celui trouvé au niveau de la pile *TCP/IP* par p0f. Une incohérence peut indiquer la présence d'un serveur proxy en coupure.

module « content » : génération du contenu renvoyé à l'application cliente Le module « content » génère un contenu offensif à la demande du module « core ». Ce contenu est modulé en fonction de plusieurs informations :

- thème correspondant à celui envoyé dans le courrier électronique ;
- objectif de l'attaque sur cet utilisateur ;
- résultat du *fingerprinting* effectué précédemment ;
- résultat des attaques précédentes sur d'autres applications clientes de l'organisation ciblée.

Les attaques implémentées permettent soit l'exécution à distance de code à l'aide de vulnérabilités connues, soit l'attaque d'une application Web selon la technique présentée précédemment.

Conclusion Ce framework démontre le danger que représente l'automatisation des attaques sur les applications clientes. Cette automatisation permet de lancer efficacement des attaques à grande échelle.

C'est « le sens de l'histoire » que d'assister à la généralisation de ce type d'outil, contre lesquels les entreprises vont devoir se protéger.

4 Solutions face à la menace grandissante sur les postes clients

Les outils automatisés d'attaque des applications clientes représentent, comme nous l'avons vu dans les parties précédentes, un risque réel. Les découvertes de plus en plus fréquentes de vulnérabilités dans les applications clientes, le risque toujours présent de contournement du filtrage antivirus, et la possibilité d'attaquer une application Web interne sans même profiter d'une faille dans le navigateur, remettent en question la solidité du périmètre de l'entreprise. A tout moment, un attaquant peut se retrouver « assis » derrière le poste d'un utilisateur légitime.

Il existe cependant des parades et des solutions qui permettent :
- soit de se protéger contre ces outils automatisés et leurs attaques ;
- soit d'en limiter l'impact sur le système d'information lorsqu'une attaque sur un poste client réussit.

Nous présentons quelques-unes de ces solutions :
- sensibilisation des utilisateurs ;
- renforcement du filtrage de périmètre ;
- renforcement de la configuration du poste de travail ;
- détection et prévention d'intrusion ;
- cloisonnement des segments réseau contenant des postes clients ;
- renforcement du suivi des connexions dans les applications Web.

4.1 Sensibilisation des utilisateurs

Nous avons expliqué dans la première partie de cet article que la sensibilisation des utilisateurs ne garantissait pas que les attaques visant les postes clients échouent. C'est vrai, car dans une organisation composée de nombreuses personnes, il se trouvera toujours un utilisateur pour agir en dépit des règles de sécurité. Néanmoins, une sensibilisation efficace permet de diminuer le nombre des utilisateurs agissant ainsi, par une présentation didactique des risques.

La sensibilisation sécurité doit leur rappeler que le lancement d'un exécutable non autorisé par l'entreprise peut compromettre le poste de travail, menace d'autant plus actuelle que la partie précédente rappelle les limites des antivirus. Cette sensibilisation sécurité doit aussi inclure une partie présentant les attaques récentes contre les applications clientes. Il s'agit de faire passer l'idée que cliquer sur un lien n'est pas une action anodine pour la sécurité du poste de travail, et peut permettre à un attaquant d'en prendre le contrôle.

En cas de réception d'un courrier électronique suspect, ou en cas d'événements « bizarres » suite à la visite d'un site Web, les utilisateurs ne doivent pas hésiter à prévenir leur correspondant sécurité informatique, qui fera escalader si nécessaire l'incident à des experts pour analyser les risques réels (sans se faire compromettre eux-mêmes, nous l'espérons, par une visite naïve du site Web indiqué par le lien).

4.2 Renforcement du filtrage de périmètre

Nous avons expliqué dans les deux premières parties de cet article que le filtrage de périmètre ne protégeait pas contre les attaques sur les applications clientes. C'est vrai, car les mécanismes de filtrage comme les *firewalls* et les *proxies* ne bloquent pas les attaques contenues dans des flux applicatifs autorisés.

L'ajout sur le serveur *proxy* d'un moteur de filtrage de contenu permet de réaliser une analyse des flux renvoyés à l'application cliente et de rendre plus complexe sa compromission. Le paramétrage de ce moteur de filtrage devrait interdire le téléchargement des contenus les plus dangereux, comme les fichiers exécutables. Il en est de même pour la passerelle de messagerie, qui doit tirer les conséquences des limitations intrinsèques des moteurs d'analyse antivirus et ne pas laisser passer les contenus les plus dangereux. Néanmoins, interdire les exécutables est une limitation de fonctionnalité qui n'est pas toujours possible d'imposer, et cela ne protège pas contre les diverses failles de sécurité des logiciels clients qui permettent l'exécution de code arbitraire.

Ces moteurs de filtrage ne peuvent pas détecter et bloquer tous les flux risqués. En particulier, ces mécanismes ne peuvent pas analyser les flux chiffrés (SSL/TLS en particulier). Pour ces raisons, les organisations en ayant la possibilité devraient restreindre l'accès à des serveurs utilisant des flux chiffrés selon un principe de liste blanche : seuls les serveurs de confiance devraient pouvoir être accédés sans que leur contenu soit analysé par le filtre de contenu.

Les organisations véritablement soucieuses de leur sécurité ne devraient autoriser les connexions à partir de postes clients que vers des serveurs situés dans une liste blanche. Le filtrage selon une liste blanche empêche les utilisateurs de se connecter à des serveurs hostiles... à moins qu'un incident de sécurité ne vienne transformer un serveur de confiance en serveur hostile (par exemple une faille de type Cross Site Scripting sur une page Web de ce serveur, permettant à du code hostile de s'exécuter dans le contexte du site de confiance).

4.3 Renforcement de la configuration du poste de travail

La protection du poste de travail contre ces attaques, par des moyens technique, peut être payante bien qu'elle ne soit pas facile à mettre en oeuvre dans un contexte réel.

Ainsi, les techniques de protection de type « EndPoint », les *HIDS*, les techniques de *sandboxing*, le *hardening* des applications et du noyau, les différents paramètres des politiques Windows, etc. . . sont à étudier.

Il serait également potentiellement possible de limiter les attaques d'Internet vers l'interne en tirant partie du concept de zones de sécurité d'Internet Explorer. Ainsi il pourrait être possible de configurer les paramètres de sécurité pour interdire aux pages Web provenant d'un site Internet d'accéder à l'Intranet et aux sites de confiance.

4.4 Détection et prévention d'intrusion

Les systèmes détecteurs d'intrusion ou *IDS* peuvent détecter certaines attaques connues. Pour améliorer leurs capacités d'alerte, les *IDS* devraient aussi détecter les tentatives évidentes de *fingerprinting* d'une application cliente. Ainsi, si les alertes émises par ces équipements sont supervisées, elles peuvent permettre de détecter les traces d'activités suspectes. En particulier, les attaques en provenance de postes clients du réseau interne doivent faire l'objet d'une attention particulière.

Les systèmes de prévention d'intrusion ou *IPS* peuvent avoir un rôle particulier à jouer. Outre le blocage des attaques renvoyées dans les flux Web, il serait sage que ces équipements rajoutent dynamiquement les serveurs identifiés comme hostiles dans une liste noire. Cela éviterait que d'autres postes clients s'y connectent et que les attaques finissent par aboutir.

Les tunnels *HTTP* utilisés par les attaquants pour communiquer avec les postes clients compromis pourraient être détectés par une analyse statistique sur la fréquence des requêtes, le type d'URL ou encore le ratio entre la taille des paquets correspondant aux envois et aux réceptions de données (en effet, dans le cas de requêtes *HTTP* classiques, la taille des requêtes est statistiquement plus faible que la taille de réponse, ce qui n'est plus vérifié quand un tunnel entrant est établi via un flux *HTTP*). Des techniques existent cependant pour dissimuler le transfert d'informations. En encapsulant la communication dans des requêtes *HTTP* ressemblant de très près à des requêtes légitimes et conservant un ratio de téléchargement similaire à celui d'une utilisation standard, il est très difficile de détecter le tunnel.

4.5 Cloisonnement des segments réseau contenant des postes clients

Une des pensées les plus pragmatiques concernant la sécurité des postes clients consiste à dire qu'ils seront compromis tôt ou tard. Partant de ce principe, il est plus sage de considérer les segments réseau contenant des postes clients comme des réseaux potentiellement hostiles.

Il en résulte un cloisonnement selon le principe du moindre privilège. Seules les applications nécessaires sont accessibles aux utilisateurs.

Ce cloisonnement rend plus complexe la compromission en cascade des serveurs internes, mais demande dans les grandes organisations des efforts importants de déploiement, car il faut identifier au préalable les besoins de communication entre les utilisateurs et les serveurs. Plusieurs organisations ont cependant réussi à déployer ce type de cloisonnement sur leur réseau interne.

4.6 Renforcement du suivi des connexions dans les applications Web

Il est possible de diminuer le risque de compromission d'une application Web par le type d'attaque présenté dans cet article en renforçant le suivi des connexions au niveau de l'application Web.

Une fois que l'utilisateur s'est authentifié sur l'application, un identifiant de session généré aléatoirement devrait être inséré dans chaque page renvoyée au navigateur. Cet identifiant devrait être placé dans un champ caché des formulaires de la page Web, ou dans les liens menant aux autres pages, et non pas dans un *cookie* comme c'est actuellement la norme. L'envoi de cet identifiant de session au sein de la requête demandant la page Web suivante prouve que cette requête provient bien de la fenêtre du navigateur où était affichée la page précédente. Cet identifiant de session doit être systématiquement vérifié avant d'autoriser l'accès aux pages Web nécessitant une authentification.

Par ailleurs, il est nécessaire de s'assurer que chaque page sensible des applications Web internes est bien protégée par un mécanisme d'authentification, notamment les pages Intranet afin d'éviter le vol d'informations par une page Web hostile qui tenterait de s'y connecter et renverrait le contenu du site sur un serveur Internet. L'authentification ne doit pas être transparente pour l'utilisateur, puisque cela la rendrait également transparente pour une page Web hostile qui tenterait de se connecter à l'application. Ainsi les mécanismes de type Single Sign On, par exemple l'utilisation d'une authentification *HTTP Basic NTLM* basée sur l'identifiant et le mot de passe Windows de l'utilisateur, qui est envoyée automatiquement par Internet Explorer, seraient à proscrire.

Références

1. Qualys (2006), Laws of Vulnerabilities, `http://www.qualys.com/company/newsroom/newsreleases/france/pr.php/2005-11-15`
2. Shreeraj Shah, Browser Identification for web application `http://www.net-square.com/whitepapers/browser_ident.pdf`
3. Jose Nazario, Passive System Fingerprinting using Network Client Applications `http://www.crimelabs.net/docs/passive.html`
4. Benjamin Caillat, Eric Detoisien (2005), Compromettre le système d'information d'une entreprise via ses utilisateurs, SSTIC 2005. `http://actes.sstic.org/SSTIC05/Compromettre_un_SI_via_ses_utilisateurs/`
5. Nicolas Grégoire (2003), JAB, une backdoor pour réeseau Win32 inconnu. SSTIC 2003. `http://actes.sstic.org/SSTIC03/JAB/`
6. Nicolas Brulez (2004), Virus : Détections Heuristiques en environnement Win32. SSTIC 2004. `http://actes.sstic.org/SSTIC04/Detection_virale_heuristique/`
7. UPX : Ultimate Packer for Xecutables `http://upx.sourceforge.net/`
8. Gaël Delalleau (2004), Mesure locale des temps d'exécution : application au contrôle d'intégrité et au fingerprinting. `http://actes.sstic.org/SSTIC04/Fingerprinting_integrite_par_timing/`

9. CIAC : Note sur SATAN `http://www.ciac.org/ciac/notes/Notes07.shtml`
10. Nessus Vulnerability Scannner `http://www.nessus.org/`
11. Core Impact Automated Penetration Testing Product `http://www.coresecurity.com/products/coreimpact/index.php`
12. p0f : Outil d'OS fingerprinting passif `http://lcamtuf.coredump.cx/p0f.shtml`