Projet-SR2I203 : Etude et mise en œuvre des attaques sur les documents PDF

Format et sécurité des fichiers PDF

PDF est l'acronyme de *Portable Document Format*, qui a pour but de permettre la diffusion d'un document indépendamment du système d'exploitation, du *hardware* et du *software*, comme décrit dans ISO 32000-2. Pour cela, chaque document contient la description précise de tous les éléments qui le constitue : texte, image, mais également police de caractères et même du code exécutable qui pourrait être nécessaire à son affichage.

Un fichier PDF est constitué au minimum comme suit :

- un entête, contenant la version du format dans laquelle le document a été généré ;
- une liste d'objet, pouvant être de diverse nature, entre autre :
  - Booléen
  - Nombre réel
  - Entier
  - Chaine de caractères
  - Dictionnaires
  - *Streams*
- une table de référence, ou table de cross-reference (*x-ref table*), contenant l'*offset* en octet de chaque objet par rapport au début du document, ainsi que leur nombre et le nom de l'objet racine ;
- une balise de fin de fichier, appelé *Trailer*.

Le contenu affichable d'un PDF, texte et image, se trouve en général dans les objets de type *stream* sous forme binaire, et est en général compressé. Un document PDF ne contenant que du texte peut donc être plus ou moins volumineux que le fichier texte correspondant, ceci dépendant de la longueur du texte en question.

De ce fait, un document PDF n'est pas trivialement altérable via un éditeur de texte, c'est pour cela que ce format est privilégié pour la transmission de documents officiels et personnels. Il existe en outre des options supplémentaires permettant de renforcer la sécurité d'un fichier PDF.

On peut prendre pour exemple le fichier `hello.pdf` se trouvant dans `src`, qui contient simplement le texte "*Hello World !*". En ouvrant ce fichier dans un éditeur de texte, on trouvera à la fin du document une balise `/DocChecksum`. Comme on peut si attendre, la valeur suivant cette balise est la checksum calculé sur le fichier. Ceci assure que le document qui est affiché à l'utilisateur n'a pas été altéré a posteriori de sa génération, que ce soit accidentellement lors de son transport ou intentionnellement par un tiers naïf. Supprimer cette balise ne permet pas de contourner cette protection, le document apparait alors comme vierge.

Une checksum ne permet cependant que de garantir l'intégrité du documents vis-à-vis des erreurs de transmission lors de son échange ou de l'altération par un attaquant inexpériementé. Afin de garantir la confidentialité et l'authenticité d'un document, d'autres mécanismes de protection sont disponibles :

- **Chiffrement** : un fichier PDF peut être entièrement chiffré par un algorithme cryptographique, le standard défini dans PDF 2.0 étant un chiffrement par AES 256-bit, mais d'autres algorithmes sont envisageables. Afin d'accéder au contenu ou de modifier le PDF si celui est possible, il faudra dans ce cas fournir un mot de passe qui permettra de décrypter le PDF. La confidentialité des informations contenues dans le document dépend donc de la robustesse du schéma de chiffrement choisi.
- **Authenticité** : un PDF peut être signé par son créateur. La transmisision de son certificat permet alors au destinataire de s'assurer de la source de ce document. Couplé au chiffrement, cette mesure permet d'assurer que le document reçu a bien été envoyé par la personne en question, et les informations qu'il contient n'ont été ni altérées, ni divulguées.

Les possibilités du format PDF sont aujourd'hui multiples : lecture automatique, affichage dynamique, interactions avec l'utilisateur, etc. Néanmoins, tout cela a également complexifié le format, et crée de nouvelles surfaces d'attaque que nous tâcherons de décrire.

Attaque sur la sécurité d'un fichier PDF

Mot de passe Utilisateur vs Propriétaire

Un fichier PDF peut être protégé par deux types de mot de passe, chacun assurant un aspect de sécurité différent :

- Mot de passe utilisateur (*User password*): ce mot de passe permet de déchiffrer un document PDF qui est protégé par un algorithme de chiffrement.

- Mot de passe propriétaire (*Owner password*) : un document PDF peut avoir des restrictions embarquées, par exemple : impression, modification, copie de contenu, commentaire, signature, etc. Le mot de passe propriétaire permet de décider quelles fonctionnalités seront accessibles à l'utilisateur, et lesquelles ne le seront pas, ainsi que d'accéder à l'ensemble des fonctionnalités si fourni.

Un document peut être protégé par l'un ou par l'autre type de mot de passe, ou par les deux. Cependant, il n'offre pas le même niveau de protection.

En effet, le *user password* implique que le fichier a été chiffré, et il est nécessaire ne serait ce que pour visualiser le document. Ce type de mot de passe vise à garantir la confidentialité du document. La protection atteinte par la présence de ce mot de passe peut être tout à fait significative. Comme toute méthode de chiffrement par mot de passe, sa robustesse dépend de celle de l'algorithme choisi, ainsi que de la complexité du mot de passe choisi. Si le chéma de chiffrement n'a pas de faille connu, la seule méthode restante reste des attaques par dictionnaire ou par force brute avec des outils comme *John the Ripper*.

Le *owner password* vise quant à lui à contrôler ce qui peut être fait avec le document une fois ouvert. Cependant, c'est au lecteur de PDF de garantir les restrictions que le propriétaire du document a mis en place. Un lecteur de PDF qui serait configuré pour ne pas mettre de restrictions quelle que soit la politique du document ouvert permettrait de contourner ces restrictions. En effet, un PDF visualisable avec des restrictions n'est pas réellement protégé, il existe des outils simples permettant de s'affranchir de la présence d'un *owner password* (http://freemypdf.com).

Falsification de signature

L'authenticité d'un document PDF est également sujette à des failles de sécruité. En effet, il est possible de compromettre un document possèdant une signature sans que cela affecte la perception de la signature : un lecteur de PDF ne détectera pas que la signature n'est pas valide pour le document qu'il est en train d'ouvrir.

Pour cela, il nous faut d'abord clarifier comment le mécanisme de signature d'un PDF fonctionne. Lors de l'ajout d'une signature au document, celle-ci est littéralement ajouté à la fin du document, grâce à un processus appelé *incremental updates*, ou *incremental savings*. Cette fonctionnalité permet de modifier un PDF sans toucher au contenu précédant. Cela est réalisé en ajoutant à la fin du *trailer* original une liste d'*updates* pour certains objets, suivi d'une table *x-ref* mise à jour, et d'un nouveau *trailer*. On passe alors d'un fichier HEADER | BODY | X-REF TABLE | TRAILER à un de la forme : HEADER | BODY | X-REF TABLE | TRAILER **| BODY UPDATES | X-REF TABLE UPDATES | TRAILER**. Dans la partie *Body updates* est alors

rajouté un objet décrivant le mécanisme de signature : algorithme utilisé, valeur de la signature et partie du document que la signature certifie.

Cependant ce mécanisme de signature n'est pas sans faille (*1 Trillion Dollar Refund - How To Spoof PDF Signatures*) :

- **Universal Signature Forgery** : l'idée de cette attaque est de désactiver la vérification de la signature alors que le lecteur de PDF utilisé affirme que la signature du docuemnt est valide. Cette attaque est réalisé en supprimant des informations présentes dans l'objet décrivant le mécanisme de signature. Le lecteur saute alors l'étape de vérification de la signature et la déclare valide par défaut, alors que l'attaquant peut avoir modifié le document.

- **Incremental Saving Attack** : cette attaque tire parti du mécanisme *incremental saving*, et du fait que la signature n'authentifie qu'une partie du document PDF. En effet, l'objet signature contient un champ `ByteRange` qui définit sur quelle portion du fichier la signature a été calculée. Or ce champ est constitué de quatre valeurs d'offsets absolus, la dernière correspondant à la fin du docuement une fois la signature rajouté (c'est à dire jusqu'à la fin du deuxième *trailer*). Cependant rien n'empêche un attaquant de faire une nouvelle *incremental update*, et de totalement redéfinir tous les objets dans une nouvelle *body update*, à la suite du *trailer* du bloc ajouté lors de la signature. En redéfinissant tous les objets, l'attaquant peut faire afficher ce qu'il veut, tandis que la vérification ne se fera que sur la partie originale du document et celle rajouté par la signature, mais pas sur celle rajouté par l'attaquant. Cette vulnérabilité vient du fait que la signature n'est pas vérifiée en prenant en compte l'ensemble du document, mais seulement la partie décrite par le champ `ByteRange` de l'objet, qui peut ne pas décrire la totalité du document si un *incremental update* est effectuée après la signature.

- **Signature Wrapping Attack** : tout comme l'attaque précédente, celle-ci exploite le fait que le champ `ByteRange` peut être manipulé par l'attaquant, et que la signature n'est pas calculée sur l'intégralité du document. En effet, la signature est stockée dans le document lui-même, plus précisement dans l'objet décrivant son mécanisme (`Perms`). Or, avant le calcul de la signature, on ne peut pas savoir ce qu'il y aura dans le champ qui stockera la signature. C'est pour cela que ce champ est exclu du calcul, et cela se traduit par la présence de 4 valeurs décrivant la portion du document couverte par la signature : `ByteRange = [a b c d]`. `a` correspond au début du document, `d` à la fin de l'*incremental update* dans laquelle se trouve l'objet `Perms`, `b` à la position où se trouve le début de la signature, et `c` à la position juste après le champ où se trouve la signature. La vérification se fait donc uniquement sur la partie du document décrite par `[a b] U [c d]`. L'attaquant peut alors changer `c` et `d` pour allonger la partie où se

trouve la signature, qui est celle qui n'est pas prise en compte dans la vérification de la dite signature, et y ajouter du code malicieux.

Attaque par un fichier PDF

Comme évoqué dans l'introduction, il est possible d'inclure des morceaux de code JavaScrypt dans un fichier PDF, initialement dans un but d'intercativité avec l'utilisateur. Cependant, rien n'empêche de faire exécuter des commandes à ce code, comme par exemple la connexion à site hébergeant un malware, puis son installation sur la cible, ou encore l'installation d'une back-door, etc.

Cette vulnérabilité passe par l'exécution d'un code arbitraire dissimulé au sein du PDF par l'attaquant, et dont la victime n'a même pas conscience de la présence. En effet, ce code n'est d'une part pas visible par l'intermédiaire du lecteur de PDF, mais s'exécute automatiquement dès l'ouverture du fichier.

Ceci est possible grâce à deux types d'objet : `/OpenAction` et `/AA`. Le premier décrit une action qui sera faite à l'ouverture du document, à laquelle on peut fournir une application à exécuter (cmd.exe par exemple), ainsi que des arguments optionnels à passer à cette application. Un attaquant peut ainsi relativement facilement faire exécuter un code arbitraire à sa cible, et lui faire installer un malware sans que celle-ci s'en rende compte.

Le deuxième type d'objet décrit une action automatique, qui peut être configurée pour s'exécuter selon un évènement particulier, comme l'ouverture du document par exemple.

L'attaquant n'a ensuite qu'à développer sa créativité pour créer le *payload* de son attaque, sachant que certaines actions pourraient être restreintes par le lecteur de PDF, ou nécessite que l'utilisateur clique au bon endroit pour autoriser des actions malicieuses. Il faut cependant noter que, bien que ces attaques fonctionnent souvent pour Adobe Acrobat Reader, le logiciel a tendance à demander la permission pour effectuer des actions automatiques. Néanmoins, si l'utilisateur a déjà autoriser par le passé ce genre d'action, et a permis au lecteur de se souvenir de son choix, alors ce type d'attaque peut être redoutablement efficace.

Le mode de transmission du PDF joue également un rôle important dans ce type d'attaque, puisque celles-ci recquièrent souvent de l'utilisateur qu'il consentent à ce que quelque chose se passe en devant cliquer sur `Accepter` dans une fenêtre pop-up. Les chances que cela arrive sont beaucoup plus élevées si l'utilisateur n'a pas de craintes quant à la provenance du PDF, ainsi que s'il doit juste regarder le document rapidement. L'aspect ingénierie sociale joue un rôle important dans une telle attaque : si l'utilisateur s'attend à ne pas passer beaucoup de temps sur ce document (il veut simplement l'imprimer), alors il prendra moins

le temps de lire et de se poser des questions sur ce que cliquer sur un bouton `Accepter` implique (comme pour les cookies par exemple).

Bonnes pratiques

Pour se protéger d'un fichier PDF potentiellement malveillant, les recommendations usuelles adaptables à n'importe quel fichier potentiellement malveillant s'appliquent :

- Ne pas ouvrir ce fichier ;
- N'ouvrir ce fichier que dans des environnements sûrs, par exemple :
  - un ordinateur dédié à cela ;
  - une VM ;
  - utiliser les lecteurs de PDF inclut dans les navigateurs internet (e.g. pdf.js) plutôt qu'une application de bureau. En effet, on peut espérer que même s'il y a une faille de sécurité, la sandbox du navigateur offrira plus de protection qu'une application, qui elle est directement en contact avec le système.
- Toujours utiliser des logiciels (dans le cas des docuemnts PDF, les lecteurs de PDF notamment) à jour ;
- Utiliser un lecteur de PDF plus léger, qui se contente simplement d'afficher le PDF lorsque les fonctionnalités spécifiques d'Adobe Acrobat Reader ne sont pas nécessaires ;
- Si les fonctionnalités d'Acrobat Reader sont nécessaires, ne jamais accepter que le logiciel se souviennent de vos préférences en matière d'écriture automatique de fichiers ou de connexions automatiques, afin de garder le contrôle sur les actions automatiques que le document cherche à exécuter.

Il existe également des outils permettant de se faire une idée si un document est potentiellement malveillant : on peut par exemple citer pdfid de Didier Stevens (https://blog.didierstevens.com/programs/pdf-tools/), qui permet d'énumérer les différents types d'objets présents dans un PDF. La présence d'objet de type `/JavaScrypt`, `/OpenAction` ou `/AA` peut être un indicateur que le document en question cache des intentions malicieuses.