

# La sécurité des applications

Philippe Prados  
[pp@philippe.prados.name](mailto:pp@philippe.prados.name)

## TABLE DES MATIERES

1.	L'authentification .....	3
2.	Traiter la navigation.....	3
3.	Qualifier les paramètres.....	3
4.	Flux de traitement.....	4
5.	Gestion des erreurs.....	4
6.	Prévention.....	4
7.	Déni de service.....	5

## **Avant-propos**

*Cet article évoque les différents pièges à éviter lors du développement d'un site Internet, afin de résister aux attaques les plus classiques, tentées par les pirates.*

Les développeurs de site Internet ne sont pas formés aux risques de sécurité. Il présente alors de nombreuses vulnérabilités. Le développeur doit avoir à l'esprit différentes familles de vulnérabilités afin de protéger son application. Plusieurs risques sont inhérents aux protocoles utilisés par un site WEB ou aux technologies couramment utilisées.

Lors du développement, il faut considérer

- Que toutes communications, si elle n'est pas cryptée, peut être manipulée. Cela concerne l'émission et la réception des données. Il ne faut donc jamais faire confiance aux données venant d'un utilisateur, il faut également interdire toutes les manipulations des données avant leurs affichages sur le navigateur de l'internaute.
- Aucune contrainte sur l'ordonnement des traitements n'est possible. Un pirate peut invoquer les pages dans l'ordre qu'il souhaite, voir envoyer simultanément plusieurs requêtes vers la même page avec des valeurs différentes pour les paramètres.
- Les données brutes, ne sont jamais dignes de confiance lorsqu'elles sont utilisées dans différentes technologies. Cela entraîne qu'il est impératif de nettoyer ces données suivant les contraintes d'usages qui en est fait, lors des différents traitements. Par exemple, il faut appliquer un encodage particulier lors de la génération d'une page HTML avec une donnée brute, un autre encodage lors de la génération d'un flux XML, et un troisième encodage lors de la génération d'une requête SQL. Il n'existe pas d'encodage générique, valide quelque soit les technologies utilisées.
- Le protocole http étant un protocole déconnecté, aucun traitement ne peut être effectué dans une transaction longue.
- Les limites d'un traitement humain ne s'imposent pas aux traitements automatisés. Par exemple, s'il est difficile pour un humain de tester de nombreuses valeurs, c'est un jeu d'enfant pour un programme.
- Les pirates ne cherchent pas seulement à exploiter l'application pour des usages non prévu, mais également à casser l'application, sans autre objectif.
- Les algorithmes sélectionnés par des années d'expériences pour améliorer les performances des traitements, fonctionnent efficacement pour les situations moyennes, mais sont très mauvais dans les cas limites. Les pirates savent générer les cas limites pour casser une application.
- Une attaque n'est pas uniquement dirigée vers le serveur. Elle peut également attaquer le client.
- De nombreuses informations apparemment anodines peuvent faciliter la mise en place d'une attaque.

## **1. L'AUTHENTIFICATION**

La première phase importante est l'authentification de l'utilisateur. Plusieurs précautions sont à prendre lors de cette phase. On peut les résumer ainsi :

- Crypter la page du formulaire
- Ne pas envoyer le mot de passe en clair. Il est préférable de le crypter à l'aide d'une valeur aléatoire et d'un algorithme MD5
- La valeur aléatoire doit être unique pour l'authentification. Elle ne doit pas pouvoir être réutilisée.

D'autres critères permettent de se protéger des attaques en forces brutes ou de contraindre une utilisation limitée de l'identité.

## **2. TRAITER LA NAVIGATION**

La phase suivante à traiter consiste à contrôler rigoureusement la navigation de l'utilisateur. C'est une phase difficile, car le serveur n'a aucun contrôle sur le navigateur de l'internaute. Il peut ouvrir simultanément plusieurs fenêtres, soumettre plusieurs requêtes en même temps, naviguer en suivant un chemin très éloigné que celui prévu par l'application.

Une première approche consiste à n'avoir qu'une seule URL pour l'application. Le serveur se charge de livrer la page correspondante, au fur et à mesure de la navigation. Cette approche interdit d'exploiter les techniques de cache du protocole http ou plusieurs fenêtres pour la même application. Il faut également que l'application interdise la soumission simultanée de plusieurs requêtes par le même utilisateur.

Une autre approche consiste à ajouter une valeur aléatoire dans chaque page. Celle-ci doit être présentée lors de la demande des nouvelles pages. Ainsi, une page n'est livrée que si elle vient d'une page précédemment générée par l'application. Cela complique fortement l'application, car chaque lien, chaque formulaire, doit intégrer cette valeur aléatoire. Il n'est plus possible de demander un « back » dans le navigateur.

## **3. QUALIFIER LES PARAMETRES**

Les paramètres des formulaires arrivent en forme brute dans l'application. Il peut y avoir de nombreux caractères étranges, plus ou moins attendus par l'application. Après une phase de nettoyage, les données brutes sont disponibles dans des variables. Ces données ne sont jamais dignes de confiance. Il n'est pas question de les exploiter sans y apporter des traitements spécifiques à chaque usage.

Une donnée brute sert généralement à plusieurs choses. Par exemple, une requête SQL, un flux XML, un flux XSL, un fichier plat, une requête LDAP ou un fragment HTML peut être généré. Dans chaque situation, il faut appliquer les règles spécifiques d'encodages. J'ai personnellement identifié X encodages différents dans les applications auxquelles je participe.

- Header HTTP. Cet encodage s'occupe de la génération d'en-tête HTTP.
- HTML. Celui-ci génère les entités nécessaires pour tous les caractères spéciaux d'un flux HTML.

- Javascript. Cet encodage supprime les caractères spécifiques lors de la génération d'un javascript présent dans une page HTML.
- String Javascript. Cet encodage traite spécifiquement des chaînes de caractères présents dans les javascripts.
- CSS. Celui-ci s'occupe de la génération des feuilles de styles ou des attributs style des tags HTML.
- Mime. Lors de la génération de type mime, il faut supprimer certaines valeurs.
- URL. Cet encodage traite des encodages spécifiques des URL.
- URL attribut. Celui-ci traite des attributs présents dans une URL.
- XML. Celui-ci s'occupe des spécificités des flux XML. Les flux XSL en font partie.
- LDAP. Cet encodage évite l'injection d'information lors de la génération de requête LDAP.
- LDAP attribut. Ce raffinement s'occupe spécifiquement des paramètres d'une requête LDAP.
- SQL. S'occupe de la génération des requêtes SQL.
- SQL Like. Traite spécifiquement de la syntaxe LIKE des requêtes SQL.
- Filename. Traite la génération d'un nom de fichier.

Il existe bien d'autres encodages suivant les technologies utilisées.

Les développeurs utilisent rarement qu'un seul langage. Il y a très souvent une génération d'un bout de langage à partir d'information brute. C'est le cas lorsque le programme génère une page HTML, une requête SQL, un flux XML ou une requête LDAP. Cette génération est souvent oubliée par les développeurs. Ils n'imaginent pas que toute la puissance du langage produit peut être exploité par un pirate. L'approche naïve consistant à supprimer tous les caractères illégaux dès l'entrée de l'information ne fonctionne pas. En effet, le programme doit être capable de traiter des utilisateurs s'appelant « O'Reilley ». Certains caractères sont trop suspects pour être honnêtes. C'est le cas du caractère de la valeur zéro. Il sert généralement de marqueur de fin de chaîne de caractère. Cela peut être exploité par un pirate pour contourner des règles de filtrages.

## 4. FLUX DE TRAITEMENT

Les paramètres étant correctement utilisés, il faut maintenant vérifier que les traitements s'effectuent comme il faut, et dans l'ordre prévu. Les applications complexes demandent des informations sur plusieurs pages, les manipulent, les agrègent, avant de demander le traitement unitaire ayant un impact sur le système d'information. Il ne faut jamais considérer que les vérifications effectuées précédemment empêchent un pirate d'agir. Il existe tellement de technique pour contourner les tests, qu'il est indispensable de reprendre toutes les vérifications depuis le début, et dans une transaction, avant de modifier le système d'information.

## 5. GESTION DES ERREURS

De nombreuses erreurs peuvent arriver dans une application. Certaines sont de véritable bogue, c'est-à-dire des situations non prévu par le développeur, d'autres sont des erreurs technologiques comme la perte de la communication avec la base de données, la saturation du disque dur, etc. Dans tous les cas, il ne faut pas transmettre d'information à l'utilisateur lui permettant d'identifier les composants sous-jacents de l'application. Retourner la requête SQL ayant échoué est une bonne idée pour le développeur, mais une très mauvaise si l'on souhaite se protéger des pirates.

## 6. PREVENTION

Des traitements supplémentaires permettent d'identifier une attaque de pirate.

Vous pouvez ajouter un script de gestion des erreurs 404 pour détecter les demandes de scripts réputés pour leurs failles. Lorsqu'un pirate demande une URL qui n'existe pas, vérifiez que celle-ci ne correspond pas à l'invocation d'un script connu. Si c'est le cas, il est probable qu'un pirate est en train d'analyser votre site. Vous pouvez envoyer un message automatiquement à l'administrateur pour qu'il réagisse rapidement. Des sites référencent des listes de scripts vulnérables.

```
ScriptAlias /abuse_trap /usr/lib/cgi-bin/abuse_trap.pl
<Directory /user/local/cgi-bin>
ErrorDocument 404 /abuse_trap
</Directory>
```

Le programme peut être enrichi des vérifications suivantes :

- L'utilisateur fait-il partie de la liste noire dynamique d'adresse IP et d'identification ?
- L'exception capturée est-elle conforme à ce que gère l'application ?
- Les champs des formulaires contiennent-ils des patterns douteux ?
- Leurs tailles respectent-elles les limites ?
- Les soumissions de formulaires correspondent-elles à celles attendues ?
- Les invocations en méthode GET ou POST sont-elles conformes à l'application ?
- Les noms des champs des formulaires sont-ils conformes ?
- Y a-t'il trop de requêtes par seconde venant du même utilisateur ou de la même adresse IP ?
- Les profils du navigateur et de l'adresse IP source sont-ils conformes tout au long de la session ?

- L'en-tête `referer` respecte-t-il les contraintes de déploiement ?
- La session a-t-elle une durée de vie raisonnable ?
- L'adresse IP source correspond-elle aux contraintes réseaux de déploiement ?
- Le passage de l'application en SSL est-il sécurisé ?
- Détection de plusieurs échecs d'authentification à partir de la même adresse IP.
- Détection d'un volume important d'identification à deux échecs.
- Détection d'un nombre de reset de mot passe important pour un utilisateur.

Des produits commerciaux permettent d'automatiser cela.

## 7. DENI DE SERVICE

L'application peut être un chemin pour obtenir un déni de service et rendre ainsi le serveur inutilisable. Indépendamment des attaques réseaux, Plusieurs techniques sont possibles :

- Bloquer tous les comptes des utilisateurs en forçant un échec de connexion
- Bloquer le compte de l'utilisateur d'exécution
- Consommation de toute la mémoire
- Consommation de toute la CPU

Les algorithmes sont prévus pour être efficace dans le cas moyen, mais pas pour le pire. Un pirate peut volontairement utiliser des informations plaçant celui-ci dans les pires conditions. Avec peu de volume, il peut alors avoir un impact très important sur l'application.

Par exemple :

- Il peut demander l'exécution de requêtes SQL particulièrement longue.
- Une valeur pour une expression régulière male écrite entraînant une consommation de la CPU de 100% pendant plusieurs siècles ([http://www.cs.rice.edu/~scrosby/hash/CrosbyWallach\\_UsenixSec2003.pdf](http://www.cs.rice.edu/~scrosby/hash/CrosbyWallach_UsenixSec2003.pdf)).
- Une valorisation d'un tableau de hash avec des valeurs particulières entraînant un temps excessif pour l'ajout d'une nouvelle valeur (<http://www.cs.rice.edu/~scrosby/hash/>)

Par exemple, pour analyser des expressions régulières, plusieurs familles de moteurs peuvent être utilisés : Les NFA ou les DFA. Il s'agit d'une différence d'approche dans l'analyse de l'expression. Un moteur NFA analyse tous les chemins possibles de l'expression, et retourne en arrière lors d'un échec. Un moteur DFA maintient une liste des candidats encore possible, et l'élague au fur et à mesure des caractères rencontrés. Chaque caractère est analysé qu'une seule fois. Un moteur DFA est plus complexe à compiler, mais plus rapide à l'exécution. Un moteur DFA est plus rapide à compiler mais moins rapide à l'exécution. Il est fortement dépendant de la rédaction de l'expression régulière, alors qu'un moteur DFA en est indifférent.

Par exemple, une expression `(int|info)` peut être optimisée en `in(t|fo)` pour un moteur NFA. Ainsi, en cas de retour arrière, les deux premières lettres peuvent être considérées comme possibles. Il n'est pas nécessaire d'analyser une nouvelle fois l'ensemble des caractères.

L'application utilise un moteur NFA, sensible à la syntaxe de l'expression régulière. Suivant les cas, le temps d'analyse d'un paramètre peut être linéaire par rapport à la taille de la chaîne  $O(n)$ , quadratique  $O(n^2)$ , cubique  $O(n^3)$  ou exponentielle  $O(n^n)$ .

Exemple	Type
<code>a*</code>	Linéaire
<code>a*[ab]*0</code>	Quadratique
<code>a*[ab]*[ac]*0</code>	Cubique
<code>(a aa)*0</code>	Exponentielle

Un déni de service peut être obtenu en envoyant, en grande quantité, des valeurs suffisamment longues pour entraîner un travail excessif du serveur. Cela occasionne de nombreux retours arrière. Par exemple, dans le cas d'une expression exponentielle, une chaîne de trente caractères peut être analysée en une minute en consommant 100% de la CPU.

Pour vous en convaincre, consultez la démonstration ici : <http://jakarta.apache.org/oro/demo.html>.

Indiquez l'expression AWK (moteur DFA) `(a|aa)*0`, demandez `matches()`, indiquez la valeur `aaaaaaaaaaaaaaaaaaaaaZ` et lancez le traitement. Vous obtenez rapidement un résultat. Modifiez alors l'algorithme utilisé en Perl5 (moteur NFA), et lancez à nouveau le traitement. Suivant la taille de la chaîne, le résultat peut mettre un temps très important pour analyser l'expression. Notez que le moteur DFA proposé par ORO ne gère pas les caractères unicodes.

Il faut limiter la taille des données avant d'appliquer une expression régulière. Optimisez également les expressions régulières pour éviter au maximum les retours arrière lors de l'analyse d'une chaîne.

Les requêtes non déterministes doivent avoir un temps maximum d'exécutions. Il faut limiter le temps limite d'exécutions d'une requête à la base de donnée à une valeur raisonnable, compatible avec les autres time-outs du serveur d'application.

Philippe PRADOS – Avril 2004  
press@philippe.prados.name