

Attaque solution

Philippe PRADOS

site@philippe.prados.name



*Préservez l'environnement,
n'imprimez pas ce document*

L'application est le maillon faible de la sécurité. Les développeurs d'application pour le Web, ne sont pas ou peu informés. Les pressions sur les délais empêchent généralement la prise en compte sérieuse des risques. Les pirates exploitent alors facilement ces faiblesses.

Est-ce que l'application exige une authentification ? La tableau 1 indique les différentes attaques possibles. Le tableau 2 évoque les autres vulnérabilités rencontrées fréquemment dans les applications WEB.

Avant de publier votre application sur le Net, imaginez toutes les utilisations détournées que vous pouvez imaginer. Si le risque est important, faites auditez votre application par des professionnels.

Philippe Prados

www.philippe.prados.name

Tableau 1

Attaque	Solution
Soumettre très rapidement des authentifications, avec différents mots de passes tirés de dictionnaires, ou produit par force brute.	Limiter le nombre de tentatives d'identification avant de bloquer un compte. Proposer un test, dont seul les humains, dans l'état actuel des technologies, est capable de résoudre (image avec texte).
Dénis de service en grillant tous les comptes.	Ne pas bloquer le compte, mais imposer un délai d'attente de plus en plus long avant une nouvelle tentative d'authentification.
À la longue pourtant, un mot de passe est vulnérable.	Imposer un changement de celui-ci régulièrement.
Approche inverse, chercher un compte possédant un mot de pass particulier. Si les identifiants sont prédictibles les uns des autres (numéro de compte bancaire, prénom suivit du nom, etc.) il est possible de chercher s'il n'existe pas un compte ayant un mot de passe simple.	Mémoriser les mots de passes en échec pendant une période, et, si un seuil est dépassé, refuser les authentifications avec ce mot de passe, même s'il est valide.
Exploiter le temps mis par le serveur pour détecter l'échec d'un mot de passe. En chronométrant le temps mis par le serveur pour refuser un mot de passe, il est possible de découvrir un à un les différents caractères du mot de passe.	Utiliser un algorithme à temps constant pour qualifier un mot de passe.
Le mot de passe initial d'ouverture du compte peut être toujours valide.	Celui-ci doit être immédiatement modifié après la première connexion et l'usage limité le temps.
Si un compte est découvert, il permet de prolonger l'attaque.	Limiter le nombre de connexions par jour et par compte, limiter à certaines tranches horaires pour certains jours.
Dénis de service par demande de reset d'un compte tous les cinq minutes.	Limiter le délai avant un nouveau reset de compte.
Regarder un utilisateur taper le mot de passe	Ajouter une contrainte biométrique, permettant de garantir qu'il s'agit bien de la bonne personne qui saisit le mot de passe. Une analyse du rythme de la saisie peut être un critère discriminant.
Découvrir les astuces mnémotechniques utilisées par les utilisateurs pour mémoriser les mots de passes.	Analyser la complexité des mots de passes avant de les accepter dans l'application. Cette analyse va au-delà des contraintes typographiques généralement imposées aux mots de passes.
Découvrir des informations par analyse des messages d'erreur.	Un message générique doit être diffusé, quelles que soient les causes de l'erreur (compte inexistant, mot de passe erroné, compte bloqué, utilisation hors délais, etc.).
Voler le cookie de session d'un utilisateur.	Associer à la session la version du navigateur utilisé et l'adresse IP source.
Découvrir les attaques	Une fois que l'utilisateur est authentifié correctement, il est pertinent de lui communiquer : <ul style="list-style-type: none"> le nombre d'échecs avant son authentification ; cela lui permet de découvrir une tentative de connexion ; la date et de l'heure de la connexion précédente ; Si cette information est inhabituelle, il peut prévenir l'administrateur ; son obligation de changer rapidement de mot de passe (d'initialisation ou standard).

Tableau 2

Attaque	Solution
La technologie http, permet à l'internaute de naviguer comme il le souhaite sur un site. Il peut voyager de page en page en suivant des liens, mais il peut également aller directement à une page, sans suivre le cheminement prévu par le développeur. Cela permet à des pirates, dans certaines situations, d'abuser des services du site. Par	Pour contrer cela, il faut suivre précisément le cheminement de l'utilisateur, et interdire toutes déviations.

Attaque	Solution
<p>exemple, imaginez un site de commerce suivant le processus suivant :</p> <ul style="list-style-type: none"> • Remplissage du panier, • Ouverture du processus de commande, • Demande des informations bancaires, • Validation de la commande. <p>Si le site n'est pas protégé suffisamment, le pirate peut tenter de remplir le panier après la saisie des informations bancaires ou de sauter cette étape pour confirmer sa commande.</p>	
<p>Naïvement, les développeurs pensent que les valeurs données dans les différents champs de l'application sont bonnes. Les pirates exploitent cela pour :</p> <ul style="list-style-type: none"> • Injecter du code SQL, permettant de consulter toute la base, de la modifier, voir de la détruire. • Injecter du code LDAP, permettant de contourner les authentifications ou de voler des informations personnelles. • Injecter du code HTML, permettant le vole du cookie de session d'un utilisateur. • Injecter des valeurs hors-limites, permettant l'exécution de code arbitraire par débordement de tampon, la modification des prix d'une commande, la modification des champs cachés, • Injecter des paramètres entraînant des traitements excessivement gourmand en CPU (via des expressions régulières mal écrite, des requêtes à la base de données trop complexes, ...) 	<p>Il faut vérifier précisément, et systématiquement, tous les paramètres manipulables. Cela concerne, bien entendu, les champs des formulaires, mais également les champs cachés, les valeurs des cookies, les en-têtes du navigateur, etc.</p>
<p>Injection de traitement dans les valeurs.</p>	<p>Les informations venant de l'utilisateur sont utilisés pour générer différents traitements. Avant de les exploiter, il faut éventuellement les encoder pour respecter les contraintes d'intégration. Trop souvent, les développeurs imaginent que les données sont saines, car elles ont été nettoyer au début du traitement. C'est possible, mais comme il existe tellement de possibilité d'utiliser une même donnée, il est probable que le nettoyage ne soit pas suffisant dans tous les cas. Sauf si l'application n'accepte que des valeurs alphanumériques, sans espaces, sans accents, sans caractères de ponctuations. Cela n'est concrètement pas possible. Les données utilisateurs sont généralement exploitées pour générer des requêtes SQL, LDAP, pour des expressions régulières, des fichiers XML, XSL, produire des pages HTML, etc. Suivant les cas, il faut modifier les données pour éviter une mauvaise interprétation par ces différents autres langages. Certains caractères doivent être présenté avec un encodage particulier, des mots-clefs doivent être systématiquement supprimé, etc. Par exemple, pour injecter une donnée dans une page HTML, il existe six encodages différents, suivant la localisation de la donnée dans la page.</p>
<p>Condition de course.</p>	<p>Les applications WEB sont, par nature, multi-tâches. Les développeurs imaginent qu'un utilisateur particulier ne peut faire qu'une seule chose à la fois. L'application est alors écrite comme si elle était mono-tâche. Un pirate peut exploiter cela pour demander simultanément plusieurs traitements, dont l'inconsistance permet de casser la base de données ou de contourner des protections. C'est ce que l'on appelle des « conditions de course » (race condition).</p> <p>L'application doit garantir que l'exécution simultanée de plusieurs traitements pour un même compte n'aura pas d'impact sur la qualité des données manipulées. Il faut également qu'une seule transaction soit ouverte lors du calcul d'une page. Si l'application utilise plusieurs petites transactions, il y a un risque d'inconsistance des données.</p>
<p>Les erreurs peuvent révéler de nombreuses informations à l'utilisateur.</p>	<p>Il est important que qualifier précisément les erreurs pouvant être transmises à l'utilisateur de celles ne le pouvant pas. À défaut, il est préférable d'afficher un message générique que de transmettre la requête SQL ayant échouée.</p>