

Sécurité Internet

Philippe PRADOS

pp@philippe.prados.name

Les applications Internet constituent des cibles privilégiées pour les pirates, car les attaques s'effectuent à distance et il est très difficile de remonter à l'auteur. Les attaques sont principalement de deux types : l'introduction dans le système pour y lire des informations sensibles ou y apporter des modifications préjudiciables à l'entreprise ou la mise hors-service de l'application (c'est le DoS: Denial of Service, « déni de service »). La première forme s'apparente au banditisme, la deuxième au vandalisme.

Une application présente sur le net doit essentiellement se protéger du banditisme, même si le vandalisme peut paralyser l'activité d'une entreprise dépendant exclusivement du net. L'attaque des sites comme Yahoo™ en février 2000 en a été une flagrante démonstration.

Afin d'obtenir la confiance des clients, il est indispensable de les protéger et de leur démontrer régulièrement. Les clients des sites Internet sont très volatiles. Ils sont à un clic du concurrent. Le frein principal du commerce électronique ou des accès bancaires est la crainte de l'internaute quant à la sécurité de ses informations confidentielles : les sites doivent offrir une garantie de sécurité pour l'internaute. Ce n'est généralement pas le cas : de nombreux sites protègent leurs serveurs mais négligent les clients. Une étude effectuée en février 2000 sur dix sites bancaires français faisait apparaître que cinq d'entre eux ne protégeaient pas les clients (Le Monde Informatique n° 843).

Pour proposer un site sécurisé, vous devez respecter un certain nombre de règles, justifiées par l'expérience. Il n'existe pas et il n'existera jamais de preuves de l'invulnérabilité d'un site. Il est nécessaire de se tenir régulièrement au courant des nouvelles attaques et des nouveaux « exploits » des hackers. Les sites devront être adaptés en conséquence. Un audit régulier du système complet doit être fait pour protéger le serveur et le client.

Quelle que soit la complexité de la partie cliente, elle utilisera le réseau pour communiquer avec le serveur. Plusieurs attaques peuvent alors être mises en place :

- Exploiter les bogues.
- Ecouter la communication entre le client et le serveur pour obtenir des informations confidentielles (sniffer).
- Se placer entre le client et le serveur afin de modifier partiellement les informations à la volée lors de la communication (« man in the middle » ou « homme au milieu »).
- Rédiger un programme qui se fait passer pour un client ou un serveur et simule leur comportement (spoofing).

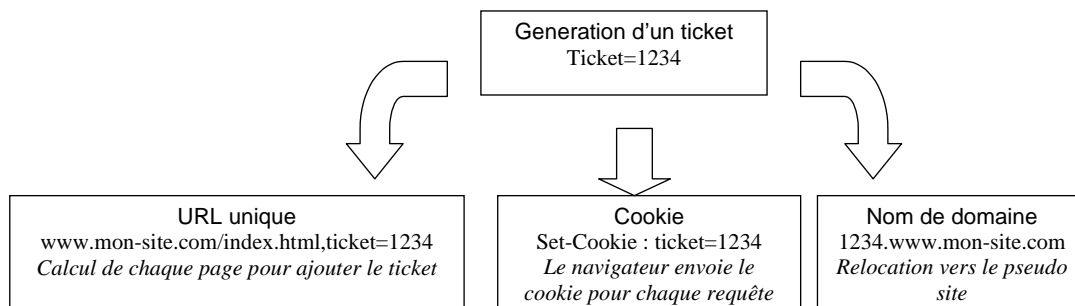
Il existe de nombreux outils pour faire cela. Comment réagir face à ces quatre situations ? Quelles situations permettent aux pirates de s'introduire par des failles de sécurité ?

Les architectes et les développeurs doivent intégrer très tôt la sécurité. Souvent, des choix d'architectures malheureux entraînent des impasses sécuritaires. Il n'est plus possible de corriger une faille car cela remet en cause l'ensemble de l'application. Les développeurs doivent résoudre de nombreuses difficultés, allant bien au-delà de l'objectif de l'application qu'ils conçoivent. Il est déjà difficile de rédiger un programme sans erreurs, alors comment rédiger un programme qui résiste aux pirates ? Ceux-ci feront tout ce qu'ils peuvent pour trouver les bugs et les exploiter à leurs profils.

De l'expérience, on peut dégager différents principes permettant d'améliorer la sécurité des sites Internets.

Le protocole http est déconnecté. Cela veut dire que chaque demande d'une page entraîne la connexion de l'utilisateur avec le serveur, la demande de la page et la fin de la communication. Entre deux demandes de pages, le serveur ne peut savoir qu'il travaille pour le même client. Il est nécessaire d'utiliser des technologies annexes pour regrouper les différentes requêtes d'un client. Diverses approches sont possibles :

- Utiliser des URL uniques pour chaque client. Cela consiste à travailler chaque page pour ajouter un identifiant à chaque URL. Lors de la demande de la première page, un ticket est créé. La page retournée possède le numéro du ticket dans chaque lien. Ainsi, lorsque l'utilisateur clic sur un lien, le numéro du ticket fait partie de l'URL. Le serveur est alors capable de relier la page initiale avec les nouvelles pages demandées. Cette technique oblige le serveur à calculer toutes les pages du site. Il n'est plus possible d'avoir des pages statiques. De plus, les caches des navigateurs ne peuvent plus fonctionner car chaque page est unique.
- Une autre approche, de loin la plus utilisée, consiste à générer un cookie de session. Un cookie de session est un en-tête particulier, qui n'est pas mémorisé sur le disque de l'utilisateur. Il est renvoyé au serveur pour toutes les demandes de pages du même domaine. Ainsi, le serveur peut regrouper les requêtes d'un utilisateur à partir de cette information.
- Une troisième approche consiste à utiliser un nom de domaine unique pour chaque client. Un sous domaine est associé à chaque client. Par exemple, l'internaute demande une page sur www.mon-site.org. Le serveur génère un ticket et renvoie l'utilisateur sur le site 12345.www.mon-site.org. Il préfixe le nom de domaine avec le numéro du ticket. En paramétrant correctement le serveur de nom, tous les sites terminant par www.mon-site.org seront renvoyés vers le même serveur. Identification utilisateur par DNS. Ajoutez la règle `*.www-server.example.org IN A 10.1.2.3`. Le préfixe permet d'identifier l'utilisateur. Cette approche évite de calculer chaque page mais les caches ne sont toujours pas disponibles.

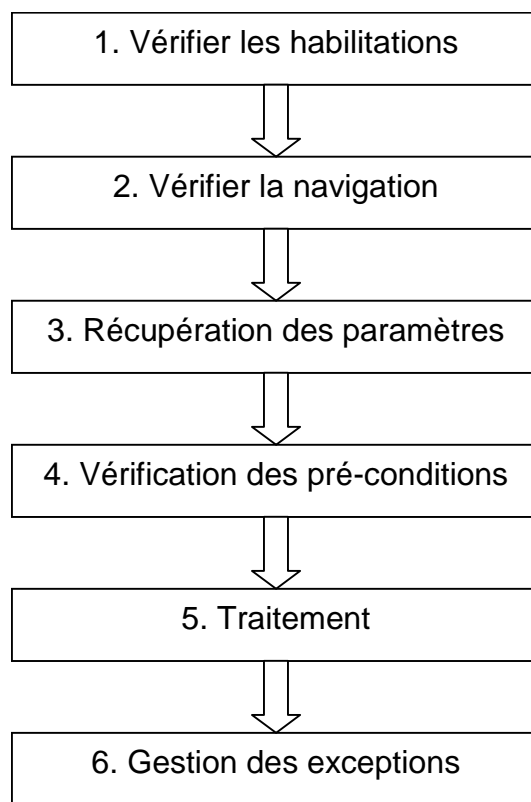


Ces différentes technologies permettent d'identifier les requêtes d'un utilisateur. Il s'agit d'une information primordiale. En effet, si un pirate arrive à voler cette information, il peut se faire passer pour un internaute. Il n'a pas besoin de connaître le mot de passe. Pour protéger cette information vitale, il faut impérativement utiliser des algorithmes de chiffrement. Le protocole SSL est là pour cela. Pour l'approche « URL

unique », il ne faut pas générer de page sur mesure si celle-ci n'est pas crypté. Pour l'approche Cookie, il faut utiliser un cookie sécurisé. Un cookie sécurisé n'est émit par le navigateur que pour les demandes de pages cryptées. Trop souvent, un cookie standard est utilisé avec le protocole SSL. Un pirate peut alors voler la session. L'approche « sous-domaine » ne peut pas être sécurisée car elle utilise un protocole additionnel, l'invocation du serveur DNS. Cette invocation n'étant pas sécurisée, un pirate peut connaître l'identifiant d'un internaute. En écoutant le réseau, il capture une demande de résolution du nom `12345.www.mon-site.org` et en déduit la valeur du ticket.

Si le pirate ne peut pas consulter le ticket, il peut en forger un ! Les algorithmes de génération de ticket de session sont parfois pauvres. En demandant quelques tickets, on déduit facilement l'algorithme du générateur. Il est alors facile de trouver un ticket valide. Le pirate injecte dans son navigateur, différentes valeurs de ticket crédibles, et regarde si cela correspond à une session en cours. Dans ce cas, tous les efforts précédant ne servent à rien. Le générateur de ticket doit utiliser un générateur aléatoire non prédictible. Les bibliothèques de cryptages possèdent généralement ce type d'algorithme de qualité sécurité. Cela prend plus de temps CPU, mais c'est le prix à payer pour interdire la génération de ticket.

Une fois l'utilisateur identifié et sa session gérée, il faut analyser ses requêtes. Trop souvent, les développeurs font confiance aux clients. Grave erreur ! Les pirates auront tôt fait d'abuser de cette confiance. Chaque paramètre doit être analysé scrupuleusement. Les développements que j'ai audité ne sont généralement pas structurés. Les paramètres sont récupérés au fur et à mesure des besoins. Le premier sert à valoriser un paramètre d'un traitement qui est immédiatement exécuté. Le deuxième est convertie en entier avant d'alimenter une propriété, etc. Que se passe-t-il si le deuxième paramètre n'est pas un entier comme prévu ? Pas de problème, les développeurs ont ajouté les tests de surface pour ne soumettre le formulaire qu'à la condition que le deuxième paramètre soit bien un entier. Quelle naïveté ! Les pirates vont se débarrasser rapidement des tests de surfaces, et vont tester la qualité des développeurs en envoyant précisément une donnée alphabétique dans un champ numérique. Le premier paramètre est utilisé pour exécuter le traitement. Le deuxième paramètre interrompt le traitement. La page est alors à moitié calculée. Cette situation n'est pas envisagée par le développeur. L'application devient instable.



Il est impératif de structurer les développements. Chaque demande de formulaire doit suivre plusieurs étapes précises. Il faut dans un premier temps vérifier les habilitations de l'utilisateur. A-t-il le droit de demander cette page ? Il faut ensuite vérifier la navigation. Est-ce normal de demander cette page à ce moment de la navigation ? L'utilisateur n'a-t-il pas contourné la navigation prévue par le développeur ? Ce point est très difficile à maîtriser car l'utilisateur peut ouvrir plusieurs fenêtres, mémoriser des signets où il le souhaite, invoquer le bouton « back », etc. Il faut ensuite récupérer les paramètres. Sont-ils tous présents ? Peut-on les convertir comme il se doit ? Respectent-ils toutes les pré-conditions (tailles des champs, caractères valides, etc.) ? Ensuite, et seulement ensuite, il est possible d'utiliser ces valeurs pour les traitements. Au terme de ceux-ci, est-ce que tout c'est bien passé ? Une exception non-prévue n'a-t-elle pas été émise ? Les messages d'erreurs associés ne doivent pas donner d'informations aux pirates. On retrouve trop souvent des noms de fichiers, des noms de tables SQL ou d'autres informations sensibles. Cela permet aux pirates d'analyser l'architecture du serveur. Fonctionne-t-il sous Windows ou Unix ? Quel base de donnée utilise-t-il, MySQL, Oracle, DB2 ? Quel serveur http gère les requêtes, IIS, Apache ? Quel est la technologie utilisée pour calculer les pages, WebSphere, BEA, PHP, ASP ?

Cette structure de développement n'est qu'un début pour sécuriser une application. Il est nécessaire d'ajouter de nombreux codes défensifs à des points stratégiques de l'application. Un code spécifique peut également détecter les comportements douteux d'un utilisateur. Cela peut alerter les administrateurs sur les correctifs à apporter. L'application connaît des informations qu'aucun outil générique de détection d'intrusion ne peut connaître. Ces informations peuvent être exploitées avec une grande efficacité afin d'arrêter ou de ralentir la progression des pirates. Des informations pertinentes peuvent être mémorisées pour servir de preuve lors d'un procès. Il est parfois nécessaire de vérifier des évidences. Par exemple, si un ticket est généré pour un navigateur particulier, il n'y a pas de raison que la version évolue lors de la session. Si c'est le cas, un pirate a certainement réussi l'exploit de voler ou de générer un ticket. Un code spécifique peut vérifier cela et

enregistrer l'utilisateur ou l'adresse IP source dans une liste-noire pendant vingt minutes. Cela ralentira considérablement les attaques du pirate. Il perdra sûrement patience et préférera dépenser son temps vers d'autres cibles.

La sécurité est un problème global qui ne peut être résolu avec des solutions génériques. Chaque maillon de la chaîne doit être étudié. L'application est malheureusement le *maillon faible* de la sécurité. C'est pour cela que les pirates entrent généralement par ce point. Chaque application est unique. Il n'est pas possible de bénéficier des informations présentes dans les listes de diffusions pour corriger une application. Les patchs des différents produits utilisés sont rapidement disponibles, mais qui peut écrire un patch pour votre application ? Qui connaît suffisamment le code pour pouvoir le modifier sans risque ? Quel processus rapide permet le déploiement de la version corrigée ?

Les entreprises doivent revoir tous leurs processus de développement et de déploiement. Les pirates sont de plus en plus nombreux, les failles de plus en plus conséquentes, mais les développeurs ne sont toujours pas formés. Il existe pourtant des formations spécifiques pour eux. Le time to market est trop souvent privilégié à la qualité du code et à sa sécurité. Il ne faut pas s'étonner alors que la presse relate régulièrement les exploits des pirates.

En ces temps troubles, espérons que les directeurs informatiques, les chefs de projets et les développeurs sauront convaincre leurs clients et leurs dirigeants afin d'améliorer rapidement cette faiblesse des nouvelles technologies.