3.4 Corrections

Solution de l'exercice 30

1. Les erreurs :
   — *grave :* on ne s'assure pas les paramètres `username` et `password` sont sains : on est vulnérable à une injection SQL ;
   — *grave :* on inclue une page sans s'assurer que le paramètre page est sain : on est vulnérable à une inclusion de fichier ;
   — *modéré :* on ne limite pas le nombre de tentatives d'authentification ;
   — on différencie erreur de login de celle de mot de passe : c'est une fuite ;
   — en passant en `GET` on s'expose à ce qu'un autre utilisateur puisse lire l'URL saisie ou la « rejoue » en la recopiant
   — on teste si le nombre d'utilisateurs dans le résultat n'est pas 0 : il vaudrait mieux vérifier qu'il y en a un seul et unique ;

   Quant au fait que le mot de passe soit transmis et stocké en clair, il est clair que ce serait un problème majeur, mais le code donné ne permet pas de le déterminer avec certitude. En effet, ne connaissant pas le code côté client, il est possible et souhaitable que la variable `$password` soit un hashé du mot de passe saisi. Si la connexion n'est pas sécurisée, on reste toutefois vulnérable à une attaque au dictionnaire ou la force brute sur le hashé transmis.
2. Vu la qualité code, on peut se demander s'il ne faut tout recommencer, en s'appuyant sur une structure robuste pour l'authentification (e.g., le CAS, un OpenID). Ici, le fait que toute la gestion ait été faite à la main est la racine du mal... Quelques correctifs ou bonne pratiques pour répondre aux problèmes :
   — utiliser des requêtes paramétrées ou, au minimum nettoyer les chaînes saisies par l'utilisateur (prélude au TP, dans lequel on verra les attaques SQLi)
   — ne pas le faire : pas de branchement `if` et renvoyer un message « le login ou le mot de passe est incorrect »
   — mettre un état pour compter le nombre de répétitions, bloquer à $n = 3$ ou temporiser entre les tentatives
   — tester si égal à 1
   — pour l'inclusion de page, il faut revoir le code plus en profondeur et trouver un autre mécanisme, comme par exemple un système de liste blanche ou une libraire pour gérer des inclusions sûres.
   — passer en POST.

Solution de l'exercice 31

1. Elle permet à l'utilisateur de saisir une adresse (IP ou URL) qui sera ensuite scannée par le logiciel `nmap` avec l'option `-A` exécuté depuis le serveur.
2. La page est vulnérable à une grave injection de commande car le champs `ip` du formulaire n'est pas protégé, on peut terminer la commande avec « `;` » et injecter une commande de notre choix, par exemple `; ls /etc/passwd`. De plus, on peut saisir des adresses génériques ou des options de `nmap` : c'est alors le serveur qui va créer un trafic possiblement offensif.
3. Le mieux serait de supprimer cette fonctionnalité, ou si ou souhaite la garder, réserver le scan `nmap` à l'IP du client qui visite le site ou une machine particulière dédiées à ça, comme par exemple `scanme.nmap.org`.

Solution de l'exercice 32

1. On bénéficie ainsi des dernières et meilleures améliorations, en terme de performance, d'optimisation, de report d'erreurs et de corrections de bugs (qui peuvent conduire à des vulnérabilités).
2. Typiquement les fonctionnalités *Buffer security check*, *Safe exception handling* et *Compatibility with Data Execution Prevention* sont intégrées pour la sécurité, elles réduisent la surface d'attaque et sont complètement automatiques.
3. Permet de vérifier statiquement certaines motifs de programmation non robuste ou non-sûre, typiquement les fonctions du point suivant. C'est une façon de rendre effective l'application de la politique de code. Un des problèmes potentiel est le grand nombre de faux positifs des analyseurs statiques.
4. Il y a une liste de fonctions qui sont très souvent mal utilisée, que l'on sait être dangereuses et qui sont présentes pour retro-compatibilité comme `strcpy`, `strcat` ou `sprintf`.

Solution de l'exercice 33

1. La première va créer une queue de taille 1, la seconde une queue de taille 85 et la troisième supprimer la queue existante. *Il vaut mieux faire en sorte que l'ordre des paramètres ne crée pas d'ambiguïté sur leur utilisation et que les valeurs utilisées soient logiques ou explicites.*
2. C'est imprévisible car `free` sur `0` ne garantit rien et il n'y a aucune vérification de faites avant les appels à `free`. *Il faut vérifier que les paramètres passés sont sains*. Dans certains guides de style, l'usage des pointeurs est déconseillé car c'est une source d'erreurs trop importante.

Solution de l'exercice 34

1. Le serveur risque d'être indisponible si tous les sockets ont été alloués. C'est un déni de service.
2. 1. Increase the server's queue size : typically only 8 connections are allowed ; could consume considerable resources.
   2. Shorten the time-out period : might disallow connections by slower clients.
   3. Filter suspicious packets : if the return address does not match the apparent source, discard the packet. May be hard to determine.
   4. Change the algorithm : instead of storing the record in the queue, send the information encrypted along with the `SYN/ACK`. A legitimate client will send it back with the ACK.
3. 1. over-provisioning the network—have too many servers to be overwhelmed (expensive and unworkable) ;
   2. filtering attack packets—somehow distinguish the attack packets from regular packets (may not be possible) ;
   3. slow down processing—disadvantages all requestors, but perhaps disproportionately disadvantages attackers ;
   4. "Speak-up" solution (Mike Walfish)—request additional traffic from all requestors. Walfish's solution assumes that the attacker's bots are already maxed out. So this solution raises the proportion of valid to invalidrequests.

Solution de l'exercice 35

1. 134.214.142.10, ports 22, 80, 443, 631 et 8080 pour respectivement ssh, hhtp, https, imprimantes via ipp et serveurs d'appli ou proxy.
2. 134.214.143.195, port 53. C'est le port standard des services DNS, il y a ainsi des chances qu'un firewall autorise des paquets entrant depuis ce port.

3. on envoie des SYN, il s'agit donc d'un SYN scan. Par défaut, si l'on est pas privilégié root, on ne peut pas taper si bas dans la pile TCP/IP et on doit impérativement utiliser les primitives bind(), connect() qui respecteraient le protocole SYN, SYN/ACK, ACK. Avec un SYN scan, dès qu'un SYN/ACK ou un RST/ACK est reçu, on interrompt la connexion avec un RST et on ne la laisse pas s'établir complètement avant de la fermer. La ligne de commande utilisée est sudo nmap -sS -PN -p 22,80,443,631,8080 -e eth0 --packet_trace liris.cnrs.fr. nslookup 134.214.142.10 indique que c'est la machine liris.univ-lyon1.fr.
4. 22, 80, 443 et 8080 ouverts car on reçoit SYN+ACK. 631 est fermé car on reçoit RESET+ACK.
5. 80, 443 et 8080 ouverts car on reçoit SYN+ACK. Pour 22 et 631 on ne reçoit rien en retour, y compris après une deuxième tentative : ces ports sont sans doute filtrés.
6. C'est la même machine mais avec des interfaces différentes : ce ne sont pas les mêmes règles de filtrage qui s'appliquent selon le médium. Il ne semble pas y avoir de firewall sur l'interface filaire et le wifi est plus sévèrement filtré.

Solution de l'exercice 36

1. Il n'y a ni dé-spécialisation ni vérification des chaînes contenues dans le cookie, si on le modifie on va pouvoir spécifier un chemin arbitraire, potentiellement normalement inaccessible.
2. Si on met le bon nombre de ../ (ou éventuellement plus que nécessaire pour être sûr), on peut accéder à tous les fichiers lisible avec le compte du serveur qui s'exécute (e.g., www-data) :

```
1 GET /vulnerable.php HTTP/1.0
2 Cookie: TEMPLATE=../../../../../../../../../../etc/passwd
```

3. — on dé-spécialise les caractères spéciaux des chemins, mais attention, on peut utiliser l'encodage des URLs pour déjouer une protection naive, e.g., %2e%2e/ qui va être transformé en ../. On peut également définir une *liste blanche* des fichiers autorisés et enfin, éviter d'utiliser cette méthode d'inclusion dynamique qui est catastrophique.
   — on peut par exemple chroot pour créer un *bac à sable* et tout du moins minimiser les accès de www-data.

Solution de l'exercice 37

1. chmod() opère sur un *nom* et open() sur un *descripteur* : rien ne garantit donc que le fichier dont les droits modifiés soit celui ouvert.
2. si on entre dans un état ou le fichier pointé est invalide, alors la ligne 341 supprimera le fichier. On pourra alors en mettre un autre à la place : celui sur lequel P1 effectuera chmod().
3. le temps est très serré pour ces exploits : un polling brutal serait inefficace car pas assez précis, le taux de réussite de l'exploit serait quasi nul.
4. idem que précédent, mais avec moindre importance.
5. la correction la plus simple est d'utiliser fchmod(). De manière générale, on peut aussi repenser l'utilisation des locks dans /tmp.
6. avec /etc/shadow on accède aux hashés des mots de passes des utilisateurs, pour pouvoir ensuite les attaquer (cf. question 3 de l'exercice 1). Avec cet exploit, on peut en fait accéder *en lecture* à n'importe quel fichier du système, y compris des partitions entières que l'on pourra dumper, des certificats, des clefs, des fichiers de configurations etc.

Solution de l'exercice 38

1. c'est une attaque à la seconde pré-image (pour une fonction de hashage f, connaissant $h = f(m)$, trouver $m \neq m'$ tel que $f(m) = f(m')$). C'est possible car CRC32 n'est pas un hash cryptographique : il n'est pas fait pour résister à des attaques (et l'espace de recherche est trop petit). Pour construire m', on peut s'appuyer sur m modifié et remplir les commentaire xml avec les bons caractères pour faire coïncider les hashés. Utiliser un vrai hash cryptographique comme SHA (ou MD5).
2. On est dans *Porous Defense* : le mécanisme choisi ici est inadapté, plus particulièrement *Use of a Broken or Risky Cryptographic Algorithm*. On peut aussi dire que c'est une augmentation locale de privilège (*local privilege escalation*). C'est l'exploitation d'une faille jusqu'ici inconnue (*0-days*) utilisé par le vers Stuxnet pour se propager, vers de haute technologie ciblant les infrastructures industrielles, parmi les 4 qu'il comptait.