Université de Tlemcen Faculté des Sciences Département Informatique	Année universitaire 2017/2018 C. BEKARA bekarach@gmail.com

Examen Final S2 Module Sécurité Informatique L3, **1h30** (Documents et Téléphone non autorisés)

Exercice 1 (**8** pts)

Pour chaque question entourez **LA** ou **LES FAUSSES REPONSES/ FAUX CHOIX**

Attention : Une bonne réponse /Un bon choix coché → **-0.25**

Q1) Une fonction de hachage permet de (**0.75 pt**)

1) Retrouver un message à partir de son haché dans un temps raisonnable
3) Chiffrer un message

Q2) La notion de collision dans une fonction de hachage H signifie (**0.75**)

Si 3 coché pas tenir compte

2) H(M1) ≠ H(M2) avec M 1≠ M2
4) qu'étant donné m=H (M) il est impossible de retrouver M

Q3) L'impact d'une vulnérabilité affectant un logiciel (**0.75**)

3) Augmente tant qu'un exploit n'existe pas

Q4) Une attaque de type DoS vise à (**0.75**)

3) voler des informations confidentielles du disque dur de la victime

Q5) Le mécanisme de bourrage permet de (**0.75**)

3) garantir l'unicité du message chiffré
4) protéger contre une attaque de type rejeu (replay)

Q6) Dans l'authentification Windows LM Hash (**1**)

1) Le sel est utilisé pour garantir l'unicité des hachés des mots de passes
4) la taille du haché stocké dépend de la longueur du mot de passe

Q7) Dans l'authentification Unix basée sur DES (**0.75**)

1) On peut constater la présence de deux mots de passe identique à travers leur haché

3) On ne fait pas de distinction entre minuscules et majuscules : Les mots de passes NT?vKL12 et nt?vkl12 auront le même haché

4) Pour retrouver le mot de passe, il suffit de générer des mots de passes aléatoires et calculer leurs hachés en utilisant MD5 puis comparer le résultat avec le haché du mot de passe stocké

Q8) Un mot de passe est composé de 10 caractères, chacun pouvant avoir l'une des 62 valeurs suivantes (a..z, A..Z, 0..9). On suppose que le mot de passe d'une personne ne comporte que des minuscules et des chiffres non-nuls. Dans ce cas, le cardinale de l'espace de recherche de ce mot de passe est : (**0.75**)

1) 10^{62}
2) 62^{10}
4) 10^{52}

Explication : le mot de passe de notre utilisateur est constitué de 10 cacartère chacun pouvant être une lettre (26 lettres minuscule) ou un chiffre non-nul (1..9) donc chaque chiffre peut avoir 35 valeurs possible la taille de l'espace de recherche est donc 10^{35}

Q9) Soit H : $\{0,1\}^* \rightarrow \{0,1\}^{\mathbf{256}}$ une fonction de hachage, et F : $\{0,1\}^* \rightarrow \{0,1\}^{\mathbf{192}}$ une fonction de hachage, Pour espérer retrouver une collision $H(F(M_1))=H(F(M_2))$ avec M1≠M2, on doit effectuer au maximum (au pire) (**0.75**)

1) $\mathbf{2^{256}}$+2 opérations de hachage
3) $\mathbf{2^{255}}$ opérations de hachage
2) $\mathbf{2^{257}}$+2 opérations de hachage

Explication :

Au pire une collision a lieu dans F au bout de $\mathbf{2^{192}+1}$ messages. On supposera aussi que pour chaque valeur de F(M) on aura une valeur différente de H(F(M)), dans ce cas la collision H(F(M1))=H(F(M2)) aura lieu après $2^{192}+1$ messages et sera donc dû à la collision F. Comme chaque message est haché deux fois (H et F) on aura $\mathbf{2*(2^{192}+1)}=2^{193}+2$ opérations de hachage **au maximum**

Q10) Pour éviter une attaque par rejeu lors d'une authentification en réseau entre un client (prouveur) et un serveur (vérificateur) il faut (**1** pt) :

1) Envoyer le haché du mot de passe

4) Envoyer le mot de passe sur le réseau

Exercice 2 (**4** pts Justifiez vos réponses)

Q1) Soit une fonction de hachage H : $\{0,1\}^* \rightarrow \{0,1\}^{\mathbf{256}}$. Une collision sur H a été trouvée au-bout de $\mathbf{2^{127}}$ messages dans un temps de $\mathbf{2^{80}}$ secondes. Calculez le temps nécessaire pour réaliser une seule opération de hachage ?

R: Temps nécessaire= $2^{80}/2^{127}=2^{-47}$seconde (**1**)

Q2) Soit une fonction de hachage H: $\{0,1\}^* \rightarrow \{0,1\}^{\boldsymbol{t}}$. Une collision s'est produite après avoir essayé un nombre de messages égale à $\frac{1}{1024}$ de l'espace des valeurs hachés. Sachant qu'une opération de hachage consomme $\mathbf{2^{-20}}$ secondes et que le temps nécessaire pour la collision est de $\mathbf{2^{24}}$ secondes, calculer la valeur de ***t*** ? Que représente ***t*** ?

R:

Le nombre de message nécessaire pour avoir collision = $2^{\mathbf{t}}/1024=2^{\mathbf{t-10}}$

Le nombre de message nécessaire pour trouver la collision=$2^{24}/2^{-20}=2^{44}$

$2^{\mathbf{t-10}}=2^{44} \rightarrow \mathbf{t}=54$ (**0.75**), qui représente la longueur du haché/empreinte (**0.25**)

Q3) Vous avez un message **M** d'une taille ***t***, que vous voulez garantir la confidentialité en utilisant un chiffrement par bloc de taille **32** octets, avec un bourrage même si M a une taille multiple de la taille d'un bloc. Si la taille du message chiffré **C** résultant est de **1600** octets, indiquez :

- La taille minimale et la taille maximale de M en cas où la taille de M n'est pas un multiple de la taille d'un bloc
- La taille de M dans le cas où la taille de M est un multiple de bloc

R:

***** La taille minimale et la taille maximale de M en cas où la taille de M n'est pas un multiple de la taille d'un bloc
Comme la taille d'un bloc est de 32 octets, la taille minimale d'un bourrage est de 1 octet, et la taille maximale d'un bourrage est de 31 octets
Donc la taille minimal de M=**1600-31**=**1569** octets, et la taille maximal=**1600**-**1**=**1599** octets (**0.25**+**0.25**)
*La taille de M dans le cas où la taille de M est un multiple de bloc
Dans ce cas, le dernier bloc de C constitue le bourrage, les blocs précédents le message qui a une taille de **1600-32**=**1568** (**0.5**)

Q4) Soit une fonction de hachage H possédant la propriété suivante : $H(M_1||M_2) = H(M_1) \oplus H(M_2)$; où M_1 et M_2 sont deux messages différents de taille quelconque, ||: concaténation, $\oplus$: XOR. Est-ce que H est résistante aux collisions en pratique?

R: Non H n'est pas résistante aux collisions (**0.25**)
Soit M=M1||M2, on a H(M)= : $H(M_1||M_2) = H(M_1) \oplus H(M_2)= H(M_2) \oplus H(M_1)=H(M')$ avec M'=M2||M1
Donc H(M)=H(M') avec M≠M' (**0.75**)

Exercice 3 (**4** pts)

Vous disposez de **N** fichiers différents sur disque dur accessible en lecture/écriture, et vous voulez vous assurez de leurs intégrité à la fois contre des erreurs intentionnels (un attaquant) et non intentionnel (modification accidentel)

Q1) Comment allez-vous procéder pour garantir l'intégrité de chaque fichier ?
R: Calculer un MAC (Code intégrité) pour chaque fichier en utilisant une clé secrète (unique pour chaque fichier ou commune à tous les fichiers) et une fonction d'intégrité de données (CBC-MAC, HMAC etc.) puis joindre le MAC au fichier. Ce calcul se fait lors de la création du fichier et suite à chaque modification (**0.5**). Lors de la vérification si le MAC joint ne correspond pas au MAC calculé, on déduira qu'une modification non autorisé a peut-être été apporté au fichier (**0.5**)
On suppose maintenant que pour garantir l'intégrité de chaque fichier, vous avez les contraintes suivantes à satisfaire :

1) Ne pas générer un code d'intégrité (MAC) pour chaque fichier, mais uniquement un seul MAC pour l'ensemble des fichiers en utilisant CBC-MAC
2) Pouvoir vérifier l'intégrité de chaque fichier individuellement (pas de concaténation de fichiers pour former un seul fichier)

Q2) Comment allez-vous procéder pour satisfaire les deux contraintes précédentes (astuce : une fonction de hachage peux vous être utile !) (**Expliquez**)

R (2pt): en utilisant une fonction de hachage **H** on va générer un haché **hi**=**H**(**fi**) pour chaque fichier **fi**
On forme ensuite un message **M**=**h1**||**h2**||…||**hN** (concaténation des hachés générés) puis on calcule un seul MAC sur ce message $\mathbf{MAC_M}$ en utilisant CBC-MAC ainsi qu'une clé secrète
Au final nous avons N fichiers (f1…fN), le message M et $\mathbf{MAC_M}$
M est mis à jour à chaque modification de fichier **fi** en calculant le nouveau **hi** puis en calculant le nouveau $\mathbf{MAC_M}$
Pour vérifier l'intégrité d'un fichier **fi**, il nous faut

1) calculer hi=H(Fi)
2) vérifier l'intégrité de M
3) puis s'assurer que hi calculé est égale au hi contenu dans M, sinon ceci veut dire que fi a peut être modifié

Donc on voit bien ici qu'on a bien respecté les deux contraintes à savoir :
Générer/vérifier un seul MAC, et en même temps permettre de vérifier individuellement l'intégrité de chaque fichier, c'est-à-dire pour vérifier l'intégrité de fi je ne suis pas obligé d'inclure dans mes calcul d'autres fichiers

Maintenant on s'intéresse uniquement à la confidentialité de ces **N** fichiers. Vous voulez chiffrer chaque fichier en utilisant une clé unique $\mathbf{K_i}$ aléatoirement générée. Par peur d'oublis, vous souhaitez sauvegarder ces **N** clés sur un support amovible de façon sécurisée, puis utiliser la bonne clé depuis le support –donc sans avoir à la mémoriser en tête-lors du chiffrement/déchiffrement du fichier correspondant.

Q3) Comment allez-vous procéder pour qu'au final vous n'aurez à retenir/mémoriser qu'une seule clé, et que les clés $\mathbf{K_i}$ soient protégées en cas de perte de la clé où accès par une personne autre que vous (**Expliquez**)

R: l'utilisateur va générer **N** clés aléatoires **Ki** puis les enregistrer dans un fichier **Mots_de_passe_chiffrés**, puis chiffrer le fichier en utilisant un algorithme de chiffrement et une clé secrète **K (différente des clés Ki)**, que l'utilisateur doit mémoriser (retenir en tête). Ensuite, l'utilisateur va stocker le fichier chiffré sur support de stockage amovible (clé usb). Ainsi, si un attaquant accède au support amovible il ne pourra pas accéder aux clés Ki car le fichier est protégé en confidentialité (**0.5 pt**)

Maintenant, lors du chiffrement/déchiffrement d'un fichier **fi**, l'utilisateur accède au fichier **Mots_de_passe_chiffrés,** le déchiffre en utilisant la clé **K** qu'il mémorise, récupère la clé **Ki** puis chiffre/déchiffre le fichier **fi (0.5 pt)**

Exercice 4 (**4** pts)
On propose d'étudier le protocole de communication **Y** se déroulant entre un Client (**A**) et un serveur (**S**) au-dessus de UDP comme le montre la figure suivante :

Messages échangées du protocole **Y**

Client (**A**)	Serveur (**S**)
(1): Req (@IP_A, @IP_S, **Requête**)→	
	← Req-Verif(@IP_S, @IP_A, R)
(2): Req-Reponse (@IP_A, @IP_S, R+1) →	A la réception du message **(2)** le serveur doit d'abord vérifier qu'il a bien reçu **R+1**, si c'est le cas il envoi : ← Rep (@IP_S, @IP_A, **Réponse**) Sinon il n'envoie rien

R: valeur aléatoire de **8** octets générée par S pour chaque Req reçu
Requête : la requête demandée par le client de taille **20** octets
Réponse : la réponse à la requête dans **(1)** envoyé par le serveur d'une taille de **160** octets
@IP_A, @IP_S : adresses IP de A et S respectivement
→, ← sens de la communication

Un attaquant veut exploiter le protocole **Y** pour lancer une attaque DoS par amplification contre le client **A**
Q1) Est-ce que l'attaque DoS va aboutir (se dérouler avec succès) dans le cas où l'attaquant n'est ni dans le sous réseau de A ni dans le sous-réseau de S ni dans le chemin liant A à S, et n'a pas un contrôle sur A ou S (**Justifiez**) ?
R : non l'attaque DoS ne va pas aboutir (**0.5**)
Pour qu'elle puisse aboutir, il faut que le serveur envoie une réponse de 160 octets à la victime dont l'@IP a été usurpé par l'attaquant dans le message (1). Toutefois, avant d'envoyer la réponse le serveur doit recevoir et valider le message (2) et en particulier s'assurer que le message (2) contient la valeur **R+1**, avec **R** la valeur généré par le serveur pour le client A (la cible dont l'identité a été usurpée par l'attaquant). La valeur R est envoyé à la victime et l'attaquant n'a aucun moyen d'intercepter R, à moins de deviner le bon nombre R parmi un ensemble de 2^{64} nombre (la probabilité est infiniment petite) (**1.5**)

On suppose maintenant qu'un attaquant veut lancer une attaque de type DoS contre le serveur. On suppose que le serveur alloue un espace mémoire de 100 Méga Octets pour contenir dans une table Connexion les informations suivantes : **(@IP_client**, **R_client)**, qu'il met à jour à la réception du message **(1)** en provenance de chaque nouveau client et puis supprime l'entrée de la table à la réception puis vérification avec succès du message **(2).** Le serveur ne crée une entrée dans la table connexion que si l'@IP client n'existe pas, autrement la requête du client est ignorée.

Q2) Expliquez comment un attaquant va procéder pour lancer l'attaque DoS contre le serveur ?
Rappel : une attaque DoS vise entre autres à saturer les ressources stockage, calcul, bande passante, etc. de la victime (qui est dans notre cas le serveur), ou l'empêcher de fournir le service voulu
R:
Cette fois-ci l'attaque de DoS aura lieu par saturation mémoire, et en particulier saturer la tableau Connexion du Serveur afin qu'il ne puisse pas servir des requêtes légitimes (**0.5**)
Dans ce cas l'attaquant va envoyer une suite de requêtes (message (1)) en usurpant une @IP différente pour chaque requête, sans pour autant jamais envoyer de message (2). (**1**)
Ceci aura par conséquence au bout d'un certain nombre de requêtes envoyés la saturation de la table connexion au bout de 100 méga octets/12=10^8/12=8 333 333 requêtes (**0.5**)