INSA – 5° R.T.

Sujet d'examen
26 janvier 2010

Sécurité des systèmes informatiques

2ème partie

Exercice 1 *(2 points)*
Une entreprise remarque que, statistiquement, elle souffre chaque année de cinq infections par des virus et de trois défigurations de son site web. La remise en état des machines après une infection par un virus nécessite deux jours de travail à l'administrateur, soit un coût total estimé à 2000 euros. Le site web peut être remis en état en quelques heures à partir des sauvegardes, soit un coût de 500 euros. La mise en place et la maintenance d'un produit antivirus et d'un système de protection du site web correspond à un coût annuel de 20 000 euros.

1. A partir des coûts ci-dessus, calculer la valeur du risque annuel dû aux virus et aux défigurations et juger l'utilité de la mise en place des mesures de sécurité énoncées.
2. Critiquer la manière dont le risque est calculé et les données limitées que l'on vous a fournies ; proposer une méthode plus adéquate.

Exercice 2 *(1 point)*
Face à une infection virale :

3. Quel est le rôle d'un système de diffusion de correctifs de sécurité ?
4. Quel est le rôle d'un logiciel antivirus ?

Exercice 3 *(3 points)*
Une vulnérabilité récente découverte dans le protocole TLS a été présentée comme suit (*Jake Edge*, LWN-2009-11-18, http://lwn.net/Articles/361697/) :

« Transport Layer Security (TLS), and its predecessor Secure Sockets Layer (SSL), are commonly used protocols for encrypting internet traffic, so TLS vulnerabilities can potentially affect a wide range of internet services. A recently disclosed flaw in the TLS protocol—though there is some dispute whether TLS is at fault—allows an "injected plaintext" attack against an encrypted session. This allows a "man in the middle" (MITM) attacker to prefix a victim's request with their own data, which gets interpreted by the server as if it came from the victim. *[…]*

TLS allows clients and servers to renegotiate various session parameters within the TLS connection. When the renegotiation is done, however, TLS applications still accept data that came in before the renegotiation as if it were in the new security context. That hole allows a MITM attack. By arranging that the last data received is from the attacker, then causing a renegotiation with the victim, the attack effectively prepends the attacker's payload to the victim's request. *[...]*

To attack a web-based application, the attacker typically would send their prefix to the server, then cause the renegotiation to occur. That renegotiation would actually be done between the victim's client and the server (with the MITM attacker just proxying the traffic). Due to the bug, the server would process the prefix in the new security context that gets established via the renegotiation. So, neither the client nor the server have any idea that this has occurred, and the attacker gets to insert his payload into the the client's secure session.

Eric Rescorla is one of those working on a long-term fix, but he also has a fairly straightforward example of the plaintext injection:

> E.g., the attacker would send:
>
> ```
> GET /pizza?toppings=pepperoni;address=attackersaddress HTTP/1.1
> X-Ignore-This:
> ```
>
> And leave the last line empty without a carriage return line feed. Then when the client makes his own request
>
> ```
> GET /pizza?toppings=sausage;address=victimssaddress HTTP/1.1
> Cookie: victimscookie
> ```
>
> the two requests get glued together into:
>
> ```
> GET /pizza?toppings=pepperoni;address=attackersaddress HTTP/1.1
> X-Ignore-This: GET /pizza?toppings=sausage;address=victimssaddress HTTP/1.1
> Cookie: victimscookie
> ```
>
> *[...]* »

Après avoir examiné ces informations et sachant que la faille concernée (CVE-2009-3555) affecte toutes les versions existantes du protocole SSL/TLS (SSL 2.0, SSL 3.0, TLS 1.0, TLS 1.1, TLS 1.2) :

1. Expliquez le résultat attendu par l'attaquant de cet exemple.
2. Envisagez (sommairement) des modifications du mécanisme de renégociation permettant d'éviter cette attaque.
3. Quelle va être la principale difficulté pour la mise en œuvre de ce type de protection ?

Exercice 4 *(4 points)*

Voici 3 exemples de méthode utilisables par un utilisateur pour choisir un mot de passe :

1. Utilisation d'un mot de passe de 8 symboles choisis au hasard parmi les symboles alphanumériques. On suppose (comme le font certains outils de génération automatique de mots de passe complexes) que le mot de passe contient exactement 1 lettre majuscule et 1 chiffre, les autres symboles étant des lettres minuscules.
   Exemple : `wei0Oone`, `Yei5rauk` ou `roh3Paht`
2. Utilisation de 2 mots du dictionnaire français courant accolés ou *éventuellement* séparés par un des caractères spéciaux choisis parmi : `, ; : ! ? . + - * / < > =`
   Exemples : `LUNE;MIEL` ou `ALLEZBLEUS`
3. Utilisation de la première lettre d'une phrase comptant au moins dix mots. On fera l'hypothèse que seules des minuscules sont utilisées.
   Exemple : `oflhqsdmsu` (2ème phrase ci-dessus)

Question 1 : Comparez ces méthodes en évaluant *quantitativement* le nombre de mots de passe différents associé à chaque méthode, en vous servant si besoin des informations indiquées ci-dessous. Il n'est pas indispensable de calculer de manière exacte la taille de l'espace considéré, mais essayez d'obtenir un ordre de grandeur *justifié* du nombre de mots de passe possibles (ou un minorant).

Question 2 : Ensuite, en vous appuyant sur les informations expérimentales fournies ci-dessous, calculez la durée espérée de résistance d'un mot de passe de type 1, 2 ou 3 face à un outil d'attaque par dictionnaire[1], suite à un vol de la forme chiffrée du mot de passe[2]. Considérez : un système Unix utilisant un chiffrement DES classique, un système Windows utilisant NT LM (DES) et un système OpenBSD utilisant (une variante de) Blowfish.

Quelques information utiles :

- On pourra considérer qu'un lexique de français courant compte environ 4000 mots[3] (méthode 2).
- Pour plus de réalisme, on pourra considérer (méthode 3) qu'une phrase est composée à 50% de « mots outils » qui ne représentent que 0,5% du lexique total (soit 20 mots au lieu de 4000), mais surtout qui ne correspondent qu'à 25% des lettres de l'alphabet (pour leur première lettre), soit seulement 6 lettres. Pour simplifier, on pourra considérer que la première lettre des autres mots correspond de manière équiprobable à l'ensemble des 26 lettres de l'alphabet.
- Le plus sûr est souvent d'évaluer expérimentalement le nombre de combinaisons par seconde (c/s) qu'une machine peut essayer[4]. Voici, sur une machine récente (Core2 Quad 2,4GHz) les performances obtenues par un outil librement disponible :

```
ortalo@mist:~$ john -test
Created directory: /home/ortalo/.john
Benchmarking: Traditional DES [64/64 BS MMX]... DONE
Many salts:  1026K c/s real, 1026K c/s virtual

Benchmarking: OpenBSD Blowfish (x32) [32/32]... DONE
Raw:  405 c/s real, 405 c/s virtual

Benchmarking: NT LM DES [64/64 BS MMX]... DONE
Raw:  8184K c/s real, 8184K c/s virtual
```

[1] Une attaque consistant à essayer systématiquement les mots d'un dictionnaire ou des combinaisons simples de ces mots entre eux ou avec des symboles courants.

[2] Contenue dans /etc/passwd ou /etc/shadow sur un système Unix, ou dans la base SAM d'une machine Windows, ou transitant sur le réseau pour certaines applications (VNC, etc.).

[3] Le lexique Dubois-Buyse des mots français courants (enfants entre 0 et 16 ans) compte 3726 mots.

[4] La quasi-totalité des systèmes d'exploitation utilisent un « grain de sel » (*salt*) aléatoire pour ralentir les attaques par dictionnaire (c'est une valeur concaténée au mot de passe de l'utilisateur qui multiplie immédiatement le nombre de combinaisons à essayer pour une attaque par dictionnaire). Voir : crypt(3). Le temps nécessaire pour chiffrer et comparer un mot du dictionnaire à la valeur dérobée dépend bien sûr du type d'algorithme (DES, MD5, MD4, Blowfish, etc.), mais dépend aussi fortement de la longueur du « *salt* » (10 bits, 12 bits, ou plus).