ÉVALUATION THÉORIQUE

SRIO

1. D'après vous, Debian est-il mieux sécurisé qu'un système Windows ?

Debian est mieux sécurisé que Windows pour plusieurs raisons. Tout d'abord Debian est un OS open source ce qui diminue le risque de failles. De plus, les utilisateurs ne sont pas administrateurs sur Debian, il y a un système de gestion des droits d'accès. Enfin, il y a un système de privilège sur Debian, il est plus difficile d'exécuter des fichiers, la où sur Windows il peut y avoir des droits d'administrateur par défaut.

2. D'après votre compréhension du business de SRIOEYE, quelle sont les deux principes de sécurité que vous jugez plus délicat (commentez votre réponse)?

Entre la confidentialité, l'intégrité et la disponibilité, je pense que les deux principes les plus délicats sont la confidentialité et la disponibilité.

La confidentialité car un système ne doit pas être accessible par quelqu'un de non autorisé. Dans le cadre de SRIOEYE, il faut s'assurer qu'aucun attaquant puisse avoir accès au flux vidéo d'un particulier.

La disponibilité car lorsqu'on accède à une donnée, on doit avoir la certitude qu'elle n'a pas été altérée. Les enregistrements sont disponibles sur un serveur web, étant un service 24h/24, il faut s'assurer que les serveurs ne plantent pas, ou qu'ils ne soit pas victimes d'attaques.

3. Si vous devez résumer en un mot la raison pour laquelle les caméras de SRIOEYE sont vulnérables, quel mot utiliserez-vous ?

Je pense que nous pouvons utiliser le mot interconnexion. En effet, SRIOEYE met en connexion différents appareils, réseaux… Cependant, l'interconnexion engendre de nombreuses failles. On a des failles dans les échanges entre les équipements, entre la caméra et le cloud… Plein d'échanges qui peuvent engendrer des failles de sécurité.

4. Pour un équipement IOT comme une caméra connectée à faible puissance, quelles sont les conséquences sur les protocoles de sécurité ?

Un équipement à faible puissance a des conséquences sur les protocoles de sécurités. Certains sont trop gourmands et vont être négligés.

5. Quelle peut être l'avantage d'un protocole comme LORA pour SRIOEYE ?

LORA est un protocole de communication sans fil permettant à des appareils de communiquer à de longues distances. Il utilise les fréquences radio, n'est pas très rapide mais à une faible consommation d'énergie pour une bonne portée. Niveau sécurité, LORA utilise le chiffrement de bout en bout et des systèmes d'authentification. SRIOEYE a donc tout intérêt d'utiliser LORA puisqu'il renforce la sécurité, à une faible consommation énergétique.

Obtention de secret d'architecture

1. Comment appelle-t-on cette stratégie façon d'extirper les informations sur une cible ?

On appel ça du social engineering car on a manipuler quelqu'un afin d'obtenir des informations confidentielles.

Gestion de la base de données

1. Sachant qu'on sait que l'algorithme RC4 est utilisé, quelle est la longueur de la clé utilisée par les caméras ?

La caméra utilise une clé de longueur 512 bits. Avec un vecteur de chiffrement de 24, la clé de chiffrement est de 512-24= 488 bits.

2. Si les caméras doivent changer des clés, chaque une minute et garder les anciennes clés de la journée pour un souci de reporting, quelle est la taille en MBytes que consommerons le stockage des clés ?

Une journée dure 1440 minutes, on génère donc 1440 clés par jour. Une clé étant sur 512 bits, cela équivaut à 64 octets. On a donc 64*1440= 92160. Le stockage des clés consommera 92MBytes.

3. Quelle est le nombre combinaisons pour une attaque de type force brute si on veut deviner la clé utilisée par les caméras ?

Le nombre de combinaisons est de 128^512. 128 est la taille de la table ASCII.

4. Avec un CPU cadencé à 2.8 GHz par seconde, pourrais-je réussir à obtenir la clé utilisée par une caméra en moins d'une minute ? Sinon, quelle puissance de calcul me faudrait-il ?

Il n'est pas possible d'obtenir la clé en moins d'une minute.

5. En hexadécimal, réaliser la conversion des chiffres ci-dessous (en expliquant votre cheminement) : (1044), (481), (844), (24)

1044 = 16*65 + 4

65 = 16*4 + 1

4 = 16*0 + 4

1044 en décimal = 414 en Hexadécimal.

481 = 16*30 + 1
30 = 16*1 + 14
1 = 16*0 + 1
481 en décimal = 1E1 en hexadécimal

844 = 16*52 + 12

52 = 16*3 + 4

3 = 16*0 + 3

844 en décimal = 34C en Hexadécimal

24 = 16*1 + 8
1 = 16*0 + 1
24 en décimal = 18 en hexadécimal

7. Quelles sont les techniques dans la littérature pour contrer le type de problème que présentaient les deux codes précédents.

Afin de se protéger contre les Buffer Overflow il y a plusieurs techniques.

On peut utiliser le stack-smashing protector (SSP) qui est une extension de gcc qui permet de limiter les dégâts causés par un buffer Overflow. Gcc ajoute alors une valeur secrète sur la stack, appelé canari. C'est un morceau de mémoire ajoutée de façon adjacente et dont la valeur peut être vérifiée. Une vérification du canari est effectuée avant de sortir de la fonction, si elle a été modifiée alors le programme s'arrête brutalement.

On peut utiliser ASLR, cette technique consiste à déplacer aléatoirement les différents éléments chargés en mémoire, comme les bibliothèques de programmes et les données utilisateur, afin de rendre plus difficile la prédiction de l'emplacement des éléments en mémoire.