Fondements de la sécurité (FOSE)

Master *Sécurité des systèmes informatiques*

Cours 1. Anatol Slissenko. Version du 24/10/2005

1 Notions de base

Le livre de Bruce Schneier [Sch00] donne une bonne introduction (sans formules, très compréhensible) dans le problème de sécurité des systèmes informatiques. La thèse de Bruce Schneier est que *l'espace social physique* et *cyberespace (cyberspace)* sont *similaires*. Le cyberespace est peuplé par des individus, on y trouve des communautés grandes et petites, il est rempli par le commerce. Les menaces et les crimes dans le cyberespace reflètent les menaces et les crimes du monde social physique. Les types de crimes sont pareils mais les méthodes sont différentes. Les attaques peuvent être automatisées (e. g. "salami attack" – voler une petite somme d'un nombre énorme de comptes), elles peuvent être réalisées à distance (l'intrusion d'un lycéen anglais dans un ordinateur du Pentagone), et leur technique peut être propagée super rapidement (sites de hackers qui exposent des clichés pour faire des virus).

Le problème de la sécurité est un problème *système*. Par exemple, votre ordinateur peut être super-protégé, mais une information très sensible peut se trouver sur un papier qui est déplacé entre des bureaux sans aucune protection. Et on peut imaginer et trouver en pratique beaucoup d'autres situations de ce genre.

1.1 Disponibilité, intégrité, confidentialité

Un système informatique, comme n'importe quel autre système utilisé par l'homme, doit vérifier certaines propriétés formulées explicitement (comme spécification de requis) ou implicitement (comme présentes dans notre tête). En pratique c'est très difficile (et même impossible) de formuler explicitement toutes les propriétés désirées. Cela provient du fait que notre compréhension d'un système complexe est habituellement incomplète. Rappelons que les requis habituellement sont divisés en *requis de fonctionnement* et *requis d'environnement*.

Quand même, on veut que notre système fonctionne 'correctement'. Autrement dit, on veut que le système soit *fiable* (*reliable* en anglais). La *fiabilité* caractérise dans quelle mesure le système fonctionne correctement. Une notion plus précise définit la fiabilité comme la probabilité de fonctionner selon la spécification des requis durant un intervalle de temps donné.

Maintenant supposons que le système dévie du comportement spécifié. Quel genre de conséquences peut impliquer un tel comportement ?

Cela peut impliquer des conséquences graves pour l'environnement ou pour les hommes, plus généralement, pour la société. Une telle déviation peut ne pas avoir de conséquences graves. Dans le cas de conséquences graves on dit que *le système n'est pas sûr*. Donc ***la sûreté*** est l'absence de conséquences graves impliquées par le fonctionnement du système.

Le problème de la sûreté est typique pour les systèmes de contrôle. Les deux classes de propriétés qu'on demande d'un système de contrôle sont la sûreté (safety) et la vivacité (liveness). La dernière

propriété précise les aspects positifs du système ; i.e. les fonctions qui doivent être assurées par le système.

Le problème de sécurité comprend évidemment les requis de sécurité. Parmi les propriétés de sécurité on distingue les 3 propriétés suivantes qui sont aujourd'hui considérées comme les plus pertinentes :

- disponibilité (availability)
- intégrité (integrity)
- confidentialité (confidentiality)

Les 3 premières lettres des mots anglais donne la triade ***A-I-C*** souvent formulée comme le but de la gestion de la sécurité.

On peut imaginer d'autres propriétés et d'autres classifications des propriétés de sécurité. Par exemple, la propriété d'*exclusivité* qui dit qu'un concurrent ne peut pas faire profit des informations du système qu'il a obtenues, peut être d'une façon légitime.

L'ordre des lettres dans la triade **A-I-C** vient de [HBH04] (Une autre source du même genre est [BK02]). Ces 2 livres ne sont pas des manuels pour apprendre les problèmes de la sécurité. Ils sont utiles pour vérifier la complétude des connaissances et pour l'organisation d'une gestion pratique de sécurité.

D'une façon informelle on peut expliquer les notions de la triade comme il suit ci-dessous.

La *Disponibilité* est un analogue de la vivacité. Elle dit que le système assure ses fonctions spécifiées pour les utilisateurs autorisés. La différence avec la vivacité est dans le fait que l'ensemble des entrées possibles dans le cas de la sécurité contient des comportements malveillants.

L'Intégrité dit que le système ne permet pas de changement non autorisés du système.

La Confidentialité dit que le système ne permet pas d'accès non autorisé au système.

1.2 CISSP Exam.

Le livre [HBH04] concerne l'examen pour obtenir un certificat CISSP.

CISSP est une abréviation pour "Certified Information Systems Security Professional" (Professionnel Certifié de la Sécurité des Systèmes d'Information). Cette certification est coordonnée par $(ISC)^2$. $(ISC)^2$ est une abréviation pour "International Information Systems Security Certification Consortium" (Consortium International de Certification de la Sécurité des Systèmes d'Information)

Le Corps Commun de Connaissance (Common Body of Knowledge (CBK)) pour CISSP consiste en **10 domaines** :

1. Gestion de la sécurité de l'information (Information Security Management)
2. Architecture et modèles de sécurité (Security Architecture and Models)
3. Systèmes de contrôle d'accès (Access Control Systems and Methodology)
4. Développement d'applications et de systèmes (Applications and Systems Development)
5. Sécurité d'opérations (Operations Security)
6. Cryptographie (Cryptography)
7. Sécurité physique (Physical Security)
8. Sécurité des télécommunications, des réseaux et d'internet (Telecommunications, Network, and Internet Security)
9. Planification de la continuité de l'activité d'entreprise (Business Continuity Planning)

10. Loi, investigation[1] et éthique (Law, Investigations, and Ethics) □

En 1999 le Comité du CBK du (ISC)[2] a défini ***5 domaines*** (areas) ***fonctionnels*** de CBK (Five Functional Areas of CBK) définies :

1. Requis de protection de l'information (Information Protection Requirements) : comment formuler les propriétés de sécurité.
2. Environnements de protection de l'information (Information Protection Environments) : fonctionnement des systèmes informatiques et de l'environnement de leur fonctionnement.
3. Technologie et outils de la sécurité (Security Technology and Tools) : avec quels outils on assure les requis.
4. Mécanismes d'assurance, de confiance et de confidentialité (Assurance, Trust, and Confidence Mechanisms) : comment on vérifie que les outils de protection marchent comme il faut.
5. Services de protection et de gestion de l'information (Information Protection and Management Services) : fonctions de business et services qui sont sous le contrôle direct du gestionnaire de la sécurité. □

Le classement ci-dessus donne une certaine orientation dans l'ensemble des connaissances qui concernent la sécurité des systèmes informatiques.

1.3 Menace, vulnérabilité, risque

Une attaque est une réalisation d'une menace (threat). On appelle *attaque* tout ensemble d'actions dont le but ou la conséquence est ou peut être une violation des propriétés de sécurité.

Une menace est "un signe qui laisse prévoir un danger". Autrement dit, une menace est une source d' attaque potentielle. Une menace peut provenir de l'environnement naturel, physique ou peut être le résultat d'actions humaines. Les actions humaines peuvent être préméditées ou non, malveillantes ou non. Le gestionnaire de la sécurité doit prendre en compte toutes les menaces possibles.

Exemples de menaces :

- Coupure d'électricité (power loss)
- Panne du matériel (hardware failure)
- Défaillance (débâcle) du logiciel (software crash)
- Erreur d'opérateur (operator error)
- Action malveillante intérieure (malicious inside action)
- Logiciel malveillant (malware)
- Incendie, explosion, inondation, séisme (fire, explosion, flood, earthquake)
- Sabotage
- Panne de disque (disk failure)
- Corruptions des données (data corruption)
- Données inexactes (inaccurate data)
- Accès non autorisé (unauthorized access)
- Cracking
- Fuite de média (media leakage)
- Vol du média (theft of media)
- Interception d'émissions
- Communications défaillantes ou corrompues

[1] enquête

La Vulnérabilité caractérise les composants du système susceptibles d'être attaqués avec succès.

Exemples de vulnérabilité :

- Faibles mots de passe
- Adressage IP statique pour accéder à l'intranet
- Ports non standards ouverts
- Réseau sans fil mal configuré
- Absence de sauvegarde (backup)

Risque. Le mot "risque" a des significations différentes (autour du même sujet) :

(1) La probabilité qu'une menace exploitera une vulnérabilité du système. Donc, c'est une fonction qui a 2 arguments, menace et vulnérabilité, et donne comme valeur une probabilité.

(2) L'espérance de la perte obtenue dans le cas de l'exploitation d'une vulnérabilité par une menace donnée.

Estimation de risque : somme sur les menaces pour une certaine durée de temps de :

(probabilité de réalisation d'une menace) × (coût de la perte impliquée par l'attaque correspondante réussie)

C'est plutôt une estimation des pertes liées à la sécurité. En fait, pour estimer la probabilité de réalisation d'une menace, on prend en compte les vulnérabilités pertinentes. Dans le contexte de sécurité, au moins aujourd'hui, c'est difficile d'évaluer les probabilités à partir des données statistiques appropriées. Cependant, on peut analyser des distribution de probabilités qu'on trouve plausibles et on peut faire des conclusions sur la base de notre expérience.

1.4 Attaques

Pourquoi l'une ou l'autre propriété de sécurité peut-ell être violée ? Cela peut arriver à cause de certaines actions, par exemple incendie, vol, virus, interception de l'émission électro-magnétique etc. Toutes ces possibilités doivent être prises en compte.

Habituellement on exclut de la notion d'attaque les actions naturelles accidentelles comme séisme, incendie accidentel, inondation etc. Cependant l'incendie montre qu'une même action qui viole la sécurité peut être accidentelle ou préméditée et malveillante.

Il est clair qu'on ne peut pas définir les propriétés de sécurité sans donner une classe d'attaques à prendre en considération. La description des attaques à prendre en compte est la partie la plus difficile de l'analyse et de la gestion de sécurité.

Qui attaque (exemples) :

- Hackers (crackers)
- Criminels solitaires
- Initiés malveillants
- Espionnage industriel
- Presse
- Crime organisé
- Police
- Terroristes
- Services de renseignements
- Info-guerriers (infowarriors)

Une petite taxonomie des attaques.

Attaques : physiques, sociologiques (humaines), informatiques.
Attaques physiques : mécaniques, chimiques, électromagnétiques, radioactives, ...
Attaques sociologiques : juridiques, ingénierie sociale, ...
Attaques informatiques : contre les processus, contre les communications, contre les informations.
Attaque : destructive et non destructive.
Attaque : préméditée ou accidentelle (intentionnelle ou par inadvertance).
Attaque : issue de la nature ou humaine.

Les attaques les plus répandues aujourd'hui sont des attaques informatiques. Les attaques les plus efficaces sont sociologiques (ingénierie sociale, vol, pot de vin). Parmi les attaques coûteuses et difficiles à tracer on trouve les attaques de déni de service et les attaques qui exploitent le dépassement du tampon (buffer overflow ou buffer overrun en anglais).

Souvent une attaque a comme but l'insertion d'un code malveillant dans le système.

Codes malveillants.

Les code de ce genre peuvent avoir des comportements différents. Les exemples les plus connus :

– bombe logique (logic bomb) : un programme qui fait quelque-chose de malveillant dans certaines conditions, e. g. à une date et une heure particulière,

– cheval de Troie (Trojan horse) : un programme qui apparemment manifeste un certain comportement utile, mais secrètement fait autre chose,

– virus (virus) : un cas particulier de cheval de Troie, qui de plus se reproduit et propage l'infection aux programmes et ordinateurs,

– ver (worm) : pareil à un virus, mais avec le but de consommer la quantité maximale possible des ressources de calcul,

– porte secrète (trapdoor, backdoor) : programme qui possède un point d'entrée secret,

– fuites d'information (information leaks) : points dans un programme qui rendent l'information traitée par le programme accessible à des personnes non autorisées.

1.5 Lignes de protection

Protection d'accès.
Protection de composants : Processus. Canal. Information.
Protection d'information : cryptographie, stéganographie, tatouage, signatures digitales, obfuscation.

1.6 Exercices

Les questions suivantes sont prises des questions de l'examen CISSP. Un exercice propose une question et plusieurs phrases comme réponses possibles. La question peut avoir plusieurs réponses, dans ce cas il faut aussi choisir la ou les meilleure(s) réponse(s).

1. Les règles et les procédures de sécurité doivent protéger des valeurs importantes et de plus :
   a. Être efficaces en coût
   b. Être justifiées par l'analyse de risque
   c. Doivent assurer la fonctionnement de l'organisation
   d. Être applicable à chacun dans l'organisation

2. Le "Masquerading" (déguisement) est :

a. Une tentative de faire une intrusion dans le système via une porte secrète (backdoor)
b. Se présenter comme un utilisateur légitime
c. Toujours fait à l'aide de spoofing
d. Fait par une application d'une masque de sous-réseau à un domaine interne d'IP.

Cette liste sera complétée. Les réponses seront aussi données.

2 Modèles de système

Un modèle peut être ciblé sur des faces différentes de la sécurité. Considérons un exemple qui modélise une exécution d'une politique de sécurité et permet de préciser les notions introduites ci-dessus.

On peut traiter un système comme un algorithme distribué, i. e. comme une collection de processus qui interagissent entre eux. À part des processus il y a des unités d'information. On peut distinguer dans un système un ensemble de processus, un média de communication (ou canaux) et des unités d'information. De plus un système possède des entrées et des sorties. Ces composants constituent notre système.

Dans le cadre général des systèmes d'états-transitions on distingue les états sécurisés (secure) ou non sécurisé (insecure). Le système est sécurisé (secure) si à partir d'un état admissible initial il n'atteint jamais un état non sécurisé. Autrement dit, aucune exécution n'arrive à un état non sécurisé. À partir de cette idée on peut développer des modèles d'états-transitions.

Voici un exemple de notion de système.

La classe des systèmes que nous introduisons est définie pour les ***univers*** de :

• Composants $\mathcal{C}$. Cet univers consiste en des ensembles finis avec une relation $\in$. Un composant peut contenir des autres composants, ces derniers peuvent encore contenir des composants etc. Pourtant chaque composant est fini et bien fondé (well founded), i. e. l'arbre d'appartenances $\in$ est fini.

• États $\mathcal{S}$ de composants. Un état n'est pas atomique et contient une description des accès (voir ci-dessous). Un état fait partie d'un état global.

• Entrées $\mathcal{X}$ (inputs). Les entrées servent à décrire l'interaction du système avec le monde extérieur.

• Actions $\mathcal{A}$. Les actions décrivent l'utilisation d'accès, les types d'actions sont concrétisés ci-dessous.

Comme variables pour les éléments des ensembles introduits ci-dessus nous utilisons les lettres, avec des indices ou non : c pour $\mathcal{C}$, s pour $\mathcal{S}$, x pour $\mathcal{X}$ et α pour $\mathcal{A}$.

Un composant modélise, par exemple, un ordinateur ou un compte ou un ensemble de services ou un ensemble de fichiers etc. Par exemple, un administrateur gère les comptes d'utilisateurs, des accès à des programmes et des fichiers. Un utilisateur peut gérer des accès d'autres utilisateurs. Et ainsi de suite. Nous ne supposons pas que 2 composants différents soient disjoints, ils peuvent avoir des sous-composants communs.

Une *action* est un élément important du modèle. Une action réalise un accès, et ce dernier peut changer des autres accès et états.

Une action peut être des 3 types :

Type 1 :

$$(\mathcal{C} \times \mathcal{S} \times \mathcal{X}) \times (\mathcal{C} \times \mathcal{S}) \to \mathcal{S} \times \mathcal{S}$$

qui signifie que α agit à partir d'un composant c_1, dont l'état est s_1 et l'entrée est x, sur un composant c_2, dont l'état est s_2, et change les états de ces composants :

$\alpha(c_1, s_1, x, c_2, s_2) = (s_1', s_2')$.

Type 2 :

$$(\mathcal{C} \times \mathcal{S} \times \mathcal{X}) \times (\mathcal{C} \times \mathcal{S}) \to \mathcal{S}$$

qui signifie que α agit à partir d'un composant c_1, dont l'état est s_1 et l'entrés est x, sur un

composant c_2, dont l'état est s_2, change l'état de c_1 et *supprime* c_2.

Type 3 :

$\mathcal{C} \times \mathcal{S} \times \mathcal{X} \to \mathcal{S} \times (\mathcal{C} \times \mathcal{S})$

qui signifie que α agit à partir d'un composant c_1, dont l'état est s_1 et l'entrés est x, change l'état de c_1 et *crée* un composant c_2 dont l'état est s_2.

Une action de 2 premiers types peut être *reflexive*, cela veut dire qu'elle peut agir d'un composant sur lui-même.

A ce niveau d'abstraction nous introduisons 3 actions : changer un état, supprimer un composant et ajouter un composant. A des niveaux plus bas d'abstraction nous pouvions introduire des actions plus concrètes et diversifiées qui prennent en compte les entrées, la lecture de fichiers, l'écriture dans fichiers, l'exécution de programmes etc.

Nos états sont structurés. Un état s d'un composant c contient un ensemble d'accès qu'on dénote $Access(c, s)$. Un élément de $Access(c, s)$ est un uplet (c', s', α') qui dit que le composant c dans un état s est autorisé d'appliquer une action α' à un composant c' dont l'état est s'. En particulier, c' peut être un nouveau composant. Cette permission peut dépendre de l'entrée ou non. En fait, une permission peut impliquer une obligation d'appliquer l'action permise mais c'est un autre niveau d'abstraction.

On peut modéliser des modes d'exécution différents. Un simple mode correspond à l'exécution des algorithmes distribués asynchrones. Chaque transition fait un changement local selon une action accomplie. Et ce changement définit un changement de l'état global.

Un *état global* g d'un système S est représenté par un ensemble de composants $\mathcal{C}_0 \subseteq \mathcal{C}$ et par une application $state : \mathcal{C}_0 \to \mathcal{S}$ qui attribue un état à chaque composant. On peut supposer qu'il y a une fonction act qui indique, pour des c, s et x donnés, quelle action de $Access(c, s)$ à exécuter. On suppose qu'une entrée arrive comme une suite de paires (c_i, x_i), où $c_i \in \mathcal{C}_0$ et $x_i \in \mathcal{X}$. Une paire (c_i, x_i) déclenche une transition en appliquant $act(c_i, state(c_i), x_i)$ à partir de c_i.

C'est une manière de représenter un *système réactif*.

Une *exécution* (un *run*) d'un système à partir d'un état global g_0, dont l'ensemble de composants est C_0, est déterminée par une entrée. Le premier élément (c_0, x_0) de l'entrée dit quelle action à appliquer à C_0. Cette application d'action amène le système dans un état g_1. L'élément suivant de l'entrée transforme g_1 en g_2 et et ainsi de suite.

Dans les systèmes réels il y a des *dépendances prédefinies* entre les accès. Nous supposons que c'est vrai pour nos systèmes. Cela veut dire que certains accès impliquent des autres accès. Une dépendance traditionnelle est transitive : si (c_0, s_0) a un accès à (c_1, s_1) via une action α, et (c_1, s_1) a un accès à (c_2, s_2) via la même action alors (c_0, s_0) a un accès à (c_2, s_2) via α. Cependant cet accès dérivé peut être ne pas explicité dans la définition du système. Dans notre cas la transitivité peut ne pas avoir lieu.

Les notions introduites ci-dessus permettent de comparer la puissance de systèmes, et donc de parler de preservation de sécurité lors d'une extension du système.

On compare les systèmes définis sur un même ensemble d'univers.

Un état global g_1 avec l'application d'états $state_1$ *inclut* g_0 avec l'application d'états $state_0$ si chaque composant de g_0 est un composant de g_1, et pour chaque composant c_0 de g_0 on a $state_0(c) = state_1(c)$, $Access(c_0, state_0(c_0)) \subseteq Access(c_1, state_1(c_1))$.

Un système S_1 dans un état g_1 est une *extension* d'un système S_0 dans un état g_0 si g_1 inclut

g_0.

Le comportement d'un système S_1 à partir d'un état g_1 est une *extension du comportement* d'un système S_0 à partir d'un état g_0, si pour tout run ρ_0 de S_0 à partir de g_0 il existe un run ρ_1 de S_1 à partir de g_1 qui conserve l'extension d'états, i. e. le $k$ème état $\rho_0(k)$ de ρ_0 est inclus dans le the $k$ème état $\rho_1(k)$ de ρ_1.

Un système S_1 est une *extension* de S_0 si pour tout état g_0 de S_0 il existe un état g_1 de S_1 tel que le comportement de S_1 à partir de cet état est une extension du comportement de $\mathcal{S}_0$ à partir de g_0.

Deux systèmes sont *équivalents* si l'un est une extension de l'autre.

On suppose que des requis d'un système qu'on considère sont donnés. Les requis spécifient quels accès doivent être permis et les quels interdits pour toute exécution du système.

Ce modèle permet de faire des précisions des notions introduites ci-dessus.

Disponibilité. On a une classe de requêtes à des composants de la forme

Trouver x ***tel que*** $F(x, c)$,

et on a des contraintes temporelles d'attente d'une réponse. Disponibilité veut dire que pour chaque requête de la classe le système fournit une réponse correcte dans la limite de temps prévue.

Intégrité. Aucune action non spécifiée ne peut être appliquée à une composante.

Confidentialité. Aucune composant non spécifié ne peut avoir un accès à un autre composant.

C'est une abstraction des systèmes réels qui permet quand même de formuler les propriétés de sécurité d'une façon plus concrète.

Pour aller plus loin il faut structurer les éléments de système. Autrement dit, introduire un vocabulaire détaillé.

Par exemple, un composant peut être un fichier. Il a un nom. Donc on a une sorte *fichier*, et des constantes de cette sorte. On a un prédicat d'accès *c peut lire le fichier f* du système et le prédicat correspondant des requis *c peut lire le fichier f à l'instant t*. L'action de lecture permet de récupérer le fichier dans le composant c.