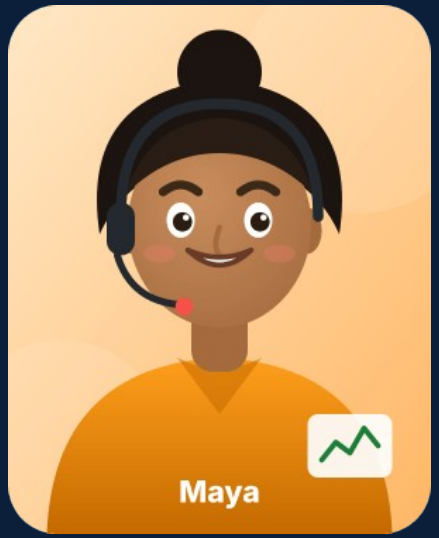
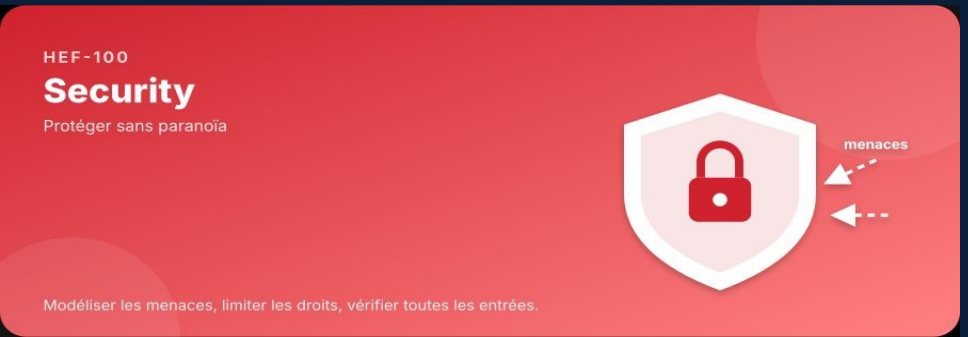
HEF-100 · MISSION 02 / 10

La modélisation des menaces

Avant de chercher des failles dans le code, cherche-les dans la conception : qui peut attaquer quoi, et comment ?



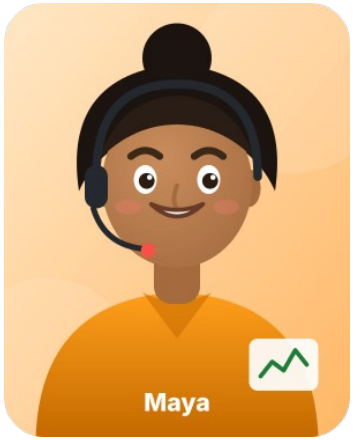


Badge : Stratège des menaces

90 min · intermédiaire

L'histoire

HashBox prépare un nouveau module de paiement. Kodi veut commencer à coder. Sam pose un papier sur la table et dessine quatre boîtes reliées par des flèches : le navigateur de l'utilisateur, le serveur web, le service de paiement, la base de données.



> *« Pour chaque flèche et chaque boîte, demande-toi : qui pourrait se faire passer pour l'autre ? modifier ? lire ? bloquer ? Tu trouveras la moitié des problèmes sans écrire une ligne de code. »*

Tu apprends à modéliser les menaces avec un schéma de flux de données et la méthode STRIDE.

Ta mission

À la fin, je serai capable de…

1. dessiner un diagramme de flux de données (DFD) avec frontières de confiance
2. appliquer STRIDE à chaque élément
3. évaluer et prioriser les risques (probabilité × impact)
4. choisir un traitement : réduire, transférer, accepter, éviter
5. documenter un modèle de menaces testable

Défi d'échauffement

Défi d'échauffement

Tu organises un grand événement. Avant de commencer, tu dresses la liste de **ce qui peut mal tourner** : une fausse invitation, un faux traiteur, une porte qui reste ouverte, un voisin qui coupe le courant, un invité qui perd ses documents. Pour chaque risque, tu notes la **probabilité** et la **gravité**, puis ce que tu fais. Écris trois risques avec leur traitement. Pourquoi est-il plus efficace de le faire **avant** l'événement ?

Le concept

Le principe

La **modélisation des menaces** (*threat modeling*) est un exercice de conception qui cherche, **avant** ou pendant le développement, les menaces possibles sur un système, puis décide quoi faire.

Les quatre questions (Shostack) :

1. **Sur quoi travaillons-nous ?** (le modèle du système)
2. **Qu'est-ce qui peut mal tourner ?** (les menaces)
3. **Que faisons-nous ?** (les contrôles)
4. **Avons-nous bien travaillé ?** (la vérification)

Le concept

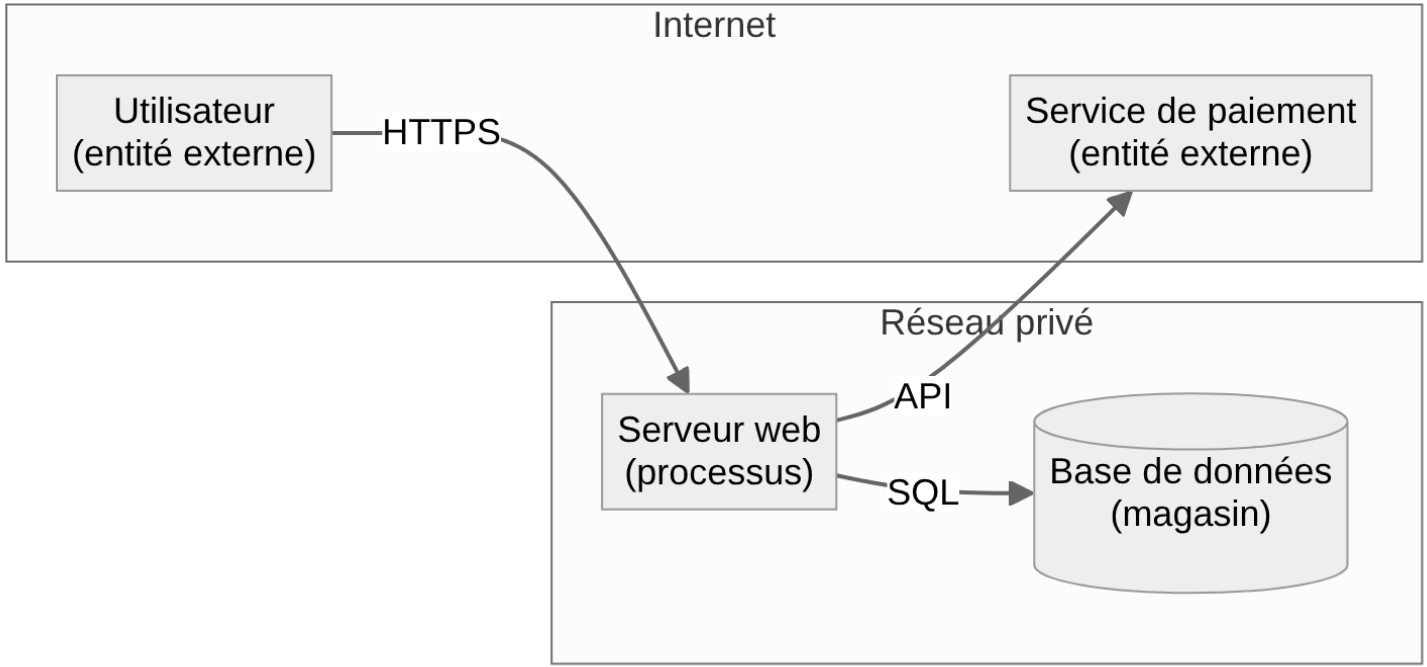
Etape 1 : le diagramme de flux de données (DFD)

On représente :

Élément	Forme habituelle	Exemple
Entité externe	rectangle	utilisateur, système tiers
Processus	cercle	serveur web, service de paiement
Magasin de données	deux lignes parallèles	base de données, fichier
Flux de données	flèche	requête HTTPS, requête SQL
Frontière de confiance	ligne pointillée	entre internet et réseau interne

Le concept

Etape 1 : le diagramme de flux de données (DFD) (suite)



Le concept

Etape 1 : le diagramme de flux de données (DFD) (suite)

Description du schéma : l'utilisateur envoie des requêtes HTTPS au serveur web. Le serveur web interroge la base de données en SQL et appelle le service de paiement externe par API. L'utilisateur et le service de paiement sont côté internet ; le serveur web et la base sont dans le réseau privé. Les flux qui traversent cette frontière sont les plus sensibles.

Les menaces se concentrent aux frontières de confiance : c'est là que des données passent d'un monde peu sûr à un monde plus sûr.

Étape 2 : STRIDE

STRIDE est un moyen mnémotechnique de six catégories de menaces :

Lettre	Menace	Propriété violée	Exemple
S	**Spoofing** (usurpation d'identité)	authenticité	se faire passer pour un administrateur

Le concept

Étape 2 : STRIDE (suite)

Lettre	Menace	Propriété violée	Exemple
T	**Tampering** (altération)	intégrité	modifier le montant d'une commande en transit
R	**Repudiation** (répudiation)	non-répudiation	nier avoir validé un paiement, sans trace
I	**Information disclosure** (divulgation)	confidentialité	fuite de la base d'utilisateurs
D	**Denial of service** (déni de service)	disponibilité	saturer le serveur
E	**Elevation of privilege** (élévation de privilèges)	autorisation	un utilisateur devient administrateur

STRIDE par élément : tous les éléments ne sont pas exposés à toutes les menaces.

Le concept

Étape 2 : STRIDE (suite)

Type d'élément	S	T	R	I	D	E
Entité externe	✔		✔			
Processus	✔	✔	✔	✔	✔	✔
Magasin de données		✔	✔	✔	✔	
Flux de données		✔		✔	✔	

Étape 3 : évaluer le risque

On estime pour chaque menace :

- la **probabilité** (1 à 5) ;
- l'**impact** (1 à 5) ;
- le **risque = probabilité × impact** (1 à 25).

Le concept

Étape 3 : évaluer le risque (suite)

Il existe des méthodes plus fines (DREAD, CVSS pour les vulnérabilités), mais un tableau simple suffit pour décider quoi traiter en premier.

Étape 4 : traiter

Stratégie	Signification	Exemple
Réduire	ajouter un contrôle	authentification forte
Éviter	supprimer la fonction ou l'exposition	ne plus stocker les numéros de carte
Transférer	partager le risque	assurance, prestataire certifié
Accepter	assumer en connaissance de cause	risque faible, coût du contrôle trop élevé

Le concept

Arbres d'attaque et cas d'abus

- Un **arbre d'attaque** part d'un objectif de l'attaquant (« voler des données ») et le décompose en moyens alternatifs (ET/OU).
- Un **cas d'abus** (*abuse case*) est un cas d'utilisation vu par l'attaquant : « En tant que concurrent, je veux télécharger tout le catalogue de cours. »

Quand le faire ?

À la conception d'une nouvelle fonctionnalité, quand l'architecture change, après un incident. Le modèle est un **document vivant**, versionné avec le code.

🧠 ***Astuce** : un modèle de menaces imparfait fait gagner plus qu'une absence de modèle. Commence petit, par les flux qui traversent une frontière de confiance.*

Exemple pas à pas

Problème : automatiser la première passe de STRIDE sur un DFD et prioriser les risques.

```
STRIDE = {"S": "Usurpation", "T": "Altération", "R": "Répudiation", "I": "Divulgat
   ion", "D": "Déni de service", "E": "Élévation de privilèges"}
PAR_TYPE = {
    "externe": "SR",
    "processus": "STRIDE",
    "magasin": "TRID",
    "flux": "TID",
}

DFD = {
    "elements": {
        "Utilisateur": "externe",
        "Serveur web": "processus",
        "Service de paiement": "externe",
        "Base de données": "magasin",
        "Flux navigateur→web": "flux",
```

Exemple pas à pas

Exemple pas à pas (suite)

```
        "Flux web→base": "flux",
        "Flux web→paiement": "flux",
    },
    "frontieres": {"Flux navigateur→web", "Flux web→paiement"},     # flux qui tra
   versent une frontière de confiance
}

def generer_menaces(dfd):
    menaces = []
    for nom, type_ in dfd["elements"].items():
        for lettre in PAR_TYPE[type_]:
            menaces.append({"element": nom, "stride": lettre, "frontiere": nom in
   dfd["frontieres"]})
    return menaces

menaces = generer_menaces(DFD)
print(len(menaces), "menaces candidates")

# Évaluation (probabilité, impact) pour quelques menaces ; les autres reçoivent un
```

Exemple pas à pas

Exemple pas à pas (suite)

```
   e note par défaut
EVALUATIONS = {
    ("Serveur web", "E"): (3, 5),
    ("Base de données", "I"): (3, 5),
    ("Flux navigateur→web", "I"): (4, 4),
    ("Flux web→paiement", "T"): (2, 5),
    ("Utilisateur", "S"): (4, 4),
    ("Serveur web", "D"): (4, 3),
}

def risque(m):
    p, i = EVALUATIONS.get((m["element"], m["stride"]), (2, 2))
    return p * i + (3 if m["frontiere"] else 0)          # bonus pour les frontiè
   res de confiance

classement = sorted(menaces, key=risque, reverse=True)[:6]
for m in classement:
    print(f"risque {risque(m):2d}  {m['element']:22s} {m['stride']} {STRIDE[m['str
   ide']]}")
```

Exemple pas à pas (suite)

Résultat : le programme génère automatiquement une liste de 23 menaces candidates à partir de la structure du DFD, puis les classe. Les premières (élévation de privilèges sur le serveur web, fuite de la base, divulgation sur le flux navigateur→serveur) sont celles à traiter d'abord.
Remarque : STRIDE produit des **candidats** : l'humain doit encore juger lesquels sont réalistes et comment les traiter.

Un registre des risques :

Menace	Contrôle choisi	Stratégie	Responsable	Vérification
Élévation de privilèges sur le serveur web	moindre privilège, revue du contrôle d'accès, tests	réduire	équipe backend	tests d'autorisation automatisés
Divulgation de la base	chiffrement, accès restreint, sauvegardes chiffrées	réduire	infra	audit trimestriel

Exemple pas à pas

Exemple pas à pas (suite)

Menace	Contrôle choisi	Stratégie	Responsable	Vérification
Altération du flux de paiement	signature des requêtes, TLS, contrôle d'intégrité	réduire	backend	test d'intégration
Perte du numéro de carte	ne pas le stocker	**éviter**	produit	revue de conception

À toi, avec aide

Modélise le DFD d'une fonction « réinitialisation de mot de passe » :

- l'utilisateur envoie son e-mail au serveur web ;
- le serveur génère un jeton, l'enregistre en base et envoie un lien par e-mail via un service de messagerie externe ;
- l'utilisateur clique sur le lien et envoie un nouveau mot de passe.

1. Liste les éléments (entités, processus, magasins, flux) et les frontières de confiance.
2. Applique STRIDE à au moins quatre éléments et trouve **8 menaces concrètes**.
3. Pour les trois plus graves, propose un contrôle.

À toi, avec aide · réponse

Indices progressifs

Indice 1

Menace S : quelqu'un demande la réinitialisation pour le compte d'un autre.

Indice 2

Menace I : un message d'erreur qui révèle si une adresse e-mail est inscrite.

Indice 3

Menace E/T : un jeton prévisible, réutilisable ou sans expiration.

À toi, avec aide · réponse

Réponse

Éléments : Utilisateur (externe), Serveur web (processus), Base (magasin), Service e-mail (externe), flux « demande », « jeton → base », « e-mail avec lien », « nouveau mot de passe ». Frontières : utilisateur ↔ serveur ; serveur ↔ service e-mail.

Menaces (exemples) :

1. **S** : demander la réinitialisation du compte d'une autre personne (spam, harcèlement).
2. **I** : message « adresse inconnue » qui permet de **lister les comptes** existants (énumération).
3. **T/E** : jeton **prévisible** (horodatage, compteur) que l'attaquant devine pour prendre le compte.
4. **T** : jeton **réutilisable** ou sans expiration (volé dans un historique ou un e-mail transféré).
5. **I** : lien envoyé en clair (HTTP) ou jeton écrit dans les journaux.
6. **D** : demandes massives qui saturent la messagerie ou bloquent légitimement un compte.
7. **R** : aucun journal des réinitialisations : impossible d'enquêter.

Réponse (suite)

8. **S** : prise de contrôle du compte e-mail de la victime → la réinitialisation devient un moyen de prise de contrôle.

Contrôles prioritaires : jeton **aléatoire** (`secrets`, 128 bits), **à usage unique**, **expirant** (15 minutes) et stocké **haché** ; réponse **identique** que l'adresse existe ou non ; **limitation de débit** + journalisation + notification à l'utilisateur.

À toi, en solo

Moteur de modèle de menaces. Écris un petit module qui :

1. définit un DFD (éléments, types, flux, frontières de confiance) pour une application de ton choix (au moins 7 éléments) ;
2. génère les menaces STRIDE par élément (comme dans l'exemple) ;
3. permet de saisir une évaluation `(probabilité, impact)` et un traitement (`reduire`, `eviter`, `transferer`, `accepter`) pour chaque menace ;
4. produit un rapport Markdown : tableau trié par risque, menaces non traitées en tête, statistiques par lettre STRIDE ;
5. refuse de considérer le modèle « terminé » s'il reste des menaces de risque ≥ 15 sans traitement.

À toi, en solo · réponse

Pistes de correction

```
PAR_TYPE = {"externe": "SR", "processus": "STRIDE", "magasin": "TRID", "flux": "TI
   D"}
NOMS = {"S": "Usurpation", "T": "Altération", "R": "Répudiation", "I": "Divulgatio
   n", "D": "Déni de service", "E": "Élévation"}

class ModeleMenaces:
    def __init__(self, elements, frontieres):
        self.elements, self.frontieres = elements, set(frontieres)
        self.menaces = [{"element": n, "stride": l, "p": 2, "i": 2, "traitement":
   None, "controle": ""}
                        for n, t in elements.items() for l in PAR_TYPE[t]]

    def evaluer(self, element, lettre, p, i, traitement=None, controle=""):
        for m in self.menaces:
            if m["element"] == element and m["stride"] == lettre:
                m.update(p=p, i=i, traitement=traitement, controle=controle)
                return
        raise KeyError((element, lettre))
```

À toi, en solo · réponse

Pistes de correction (suite)

```
    @staticmethod
    def risque(m): return m["p"] * m["i"]

    def non_traitees_graves(self, seuil=15):
        return [m for m in self.menaces if self.risque(m) >= seuil and not m["trai
tement"]]

    def termine(self):
        return not self.non_traitees_graves()

    def rapport(self):
        lignes = ["| Risque | Élément | Menace | Traitement | Contrôle |", "|---|-
--|---|---|---|"]
        tri = sorted(self.menaces, key=lambda m: (m["traitement"] is not None, -se
lf.risque(m)))
        for m in tri[:15]:
            lignes.append(f"| {self.risque(m)} | {m['element']} | {m['stride']} {N
OMS[m['stride']]} | {m['traitement'] or 'à décider'} | {m['controle']} |")
        stats = {}
```

À toi, en solo · réponse

Pistes de correction (suite)

```
        for m in self.menaces:
            stats[m["stride"]] = stats.get(m["stride"], 0) + 1
        lignes.append("\nMenaces par lettre : " + ", ".join(f"{k}={v}" for k, v in
    sorted(stats.items())))
        return "\n".join(lignes)

modele = ModeleMenaces(
    {"Client": "externe", "API": "processus", "Base": "magasin", "Flux client→API"
   : "flux", "Flux API→base": "flux",
     "Paiement": "externe", "Flux API→paiement": "flux"},
    frontieres={"Flux client→API", "Flux API→paiement"})
modele.evaluer("API", "E", 3, 5)
modele.evaluer("Base", "I", 3, 5)
assert not modele.termine()                              # des menaces graves rest
   ent sans traitement
modele.evaluer("API", "E", 3, 5, "reduire", "contrôle d'accès testé")
modele.evaluer("Base", "I", 3, 5, "reduire", "chiffrement + accès restreint")
assert modele.termine()
print(modele.rapport())
```

À toi, en solo · réponse

Pistes de correction (suite)

Un modèle de menaces utile est **vérifiable** : ici, une règle automatique empêche de le déclarer terminé tant qu'il reste un risque grave sans décision.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Modéliser trop tard	Corrections coûteuses	Dès la conception
Lister sans prioriser	Cent menaces, aucune action	Probabilité × impact, frontières de confiance d'abord
Penser uniquement attaquant externe	On oublie initiés et erreurs	Inclure employés, fournisseurs, erreurs
Modèle jamais mis à jour	Il ne correspond plus au système	Document vivant, revu à chaque changement important

Mini-boss

Mini-boss

Modèle de menaces du module de paiement de HashBox. Les flux : navigateur → serveur web (HTTPS), serveur web → API du fournisseur de paiement, fournisseur → serveur web (webhook de confirmation), serveur web → base de données, administrateur → console d'administration.

1. Dessine le DFD (Mermaid) avec les frontières de confiance et sa description en texte.
2. Génère les menaces STRIDE et retiens les 15 plus pertinentes avec une description **concrète** de chacune.
3. Pour le **webhook** (le fournisseur appelle ton serveur), détaille spécifiquement : comment un attaquant peut-il forger un faux « paiement confirmé » ? Quels contrôles (signature HMAC, liste d'IP, idempotence, vérification côté fournisseur) ?
4. Évalue le risque, choisis un traitement, désigne un responsable et un test de vérification pour chaque menace de risque ≥ 12.
5. Écris le code du **contrôle du webhook** : vérification de signature HMAC en temps constant, tolérance d'horloge, rejeu refusé (identifiant d'événement déjà vu), avec 6 tests.

Mini-boss · réponse

Pistes de correction

Un bon dossier : un DFD à 5 éléments et 5 flux avec deux frontières ; au moins 15 menaces concrètes (par exemple « un attaquant envoie un faux webhook `payment.succeeded` pour obtenir un cours gratuit » ; « le journal contient des numéros de carte » ; « un administrateur compromis modifie des montants sans trace »). Le code du webhook : `signature = HMAC-SHA256(secret, horodatage + "." + corps)`, comparée avec `hmac.compare_digest`, refus si l'horodatage dépasse 5 minutes, ensemble des identifiants d'événements déjà traités pour refuser le rejeu, réponse 200 idempotente. Les tests couvrent : signature valide, corps modifié, horodatage trop ancien, rejeu, signature absente, secret erroné. Chaque menace ≥ 12 a un traitement, un responsable et un test (ou une vérification manuelle planifiée).

Quiz éclair

Quiz éclair

1. STRIDE est :

A. un algorithme de chiffrement
B. un protocole réseau
C. une méthode de classification des menaces en six catégories

2. Les menaces se concentrent particulièrement :

A. aux frontières de confiance
B. au centre des bases de données
C. dans les commentaires du code

3. Le risque se calcule souvent comme :

A. coût + durée

Quiz éclair

Quiz éclair (suite)

B. probabilité × impact
C. nombre d'utilisateurs × nombre de serveurs

4. 🔁 *Révision mission 01* : La triade CIA désigne :

A. contrôle, identité, accès
B. chiffrement, intégrité, authentification
C. confidentialité, intégrité, disponibilité

5. 🔁 *Révision mission 01* : Le principe de moindre privilège dit de :

A. n'accorder que les droits nécessaires
B. donner tous les droits aux administrateurs
C. supprimer les mots de passe

Quiz éclair · réponse

Réponses

1. **C**.
2. **A**.
3. **B**.
4. **C**.
5. **A**.

Je réfléchis

Je réfléchis

1. Quel projet que tu as mené aurait gagné à un modèle de menaces fait au début ?
2. Qu'est-ce qui rend STRIDE utile même pour quelqu'un qui n'est pas spécialiste de sécurité ?

À retenir

1. Modéliser avant de coder : DFD + frontières de confiance.
2. STRIDE : Spoofing, Tampering, Repudiation, Information disclosure, DoS, Elevation.
3. Risque = probabilité × impact.
4. Traiter : réduire, éviter, transférer, accepter.
5. Le modèle est un document vivant et vérifiable.

MISSION ACCOMPLIE

+100 XP

Badge débloqué : Stratège des menaces

Prochaine mission : Identité, authentification et autorisation

