Hamac

RÉFÉRENCE

Modules livrés avec Hamac

Les modèles de sécurité, le module de placement et le patron par défaut installés avec le vocabulaire de Hamac.

Version 0.2 · 2026-10-31 · BROUILLON

OCamlPro SAS — forge.ocamlpro.com/OSS/hamac — AGPL-3.0

Sommaire

1 Objet 2
2 Le modèle `hamac_blp` 3
3 Le modèle `hamac_rbac` 4
4 Accès aux routes avec `hamac_blp` et `hamac_rbac` 5
5 Le patron `hamac_default` 6
6 Le module `hamac_ha` 7

SECTION 1

Objet

Hamac installe avec son vocabulaire (le module `hamac`) quelques modules LOAF prêts à l'emploi. Un patron ou un manifeste les charge par `use`, comme tout module, et en écrit les champs avec leur préfixe :

Module	Préfixe	Rôle
`hamac_blp`	`blp::`	modèle de sécurité Bell-LaPadula avec compartiments : niveaux, catégories, étiquettes des zones et des routes
`hamac_rbac`	`rbac::`	contrôle d'accès à base de rôles : les rôles que les routes exigent
`hamac_ha`	`ha::`	exigences de disponibilité et d'isolation
`hamac_default`		un patron prêt à l'emploi : quatre niveaux, une zone par niveau

La spécification des manifestes définit ce que ces modules emploient : les trois rôles des fichiers (les manifestes décrivent, les modules traduisent, Hamac agit), la relation `zone_leq` qui ordonne les zones, et les primitives de placement. Chaque module a sa propre version (0.2.0 pour tous ici) ; une organisation peut les remplacer par les siens, ou les compléter.

SECTION 2

Le modèle `hamac_blp`

`hamac_blp` est le modèle de Bell-LaPadula avec compartiments. Il ajoute :

Déclaration	Sens et contrôles
`blp::levels* [name]`, au patron	les niveaux, du moins au plus sensible : un ordre total
`blp::enclave_from? name`, au patron	ce niveau et au-delà : les zones sont des enclaves (primitive `enclave`) ; un niveau inconnu donne **E042**
`blp::category* name`, au patron	les catégories de besoin d'en connaître ; des catégories sans aucun niveau donnent **E006**
`blp::level? name`, à une zone	le niveau de la zone, l'un des niveaux (**E042**)
`blp::category* name`, à une zone	ses catégories, déclarées (**E042**)
`blp::classification? name`, à une route	niveau d'habilitation exigé de l'utilisateur
`blp::category* name`, à une route	catégories exigées de l'utilisateur

- Une **étiquette** est un couple (niveau, ensemble de catégories), et $(n_1, C_1) \leq (n_2, C_2)$ si et seulement si $n_1 \leq n_2$ et $C_1 \subseteq C_2$. Cet ordre définit un treillis.
- L'étiquette d'une zone est son niveau et ses catégories ; `zone_leq(A, B)` vaut si l'étiquette de `A` est inférieure ou égale à celle de `B`.
- Une route ne demande pas plus que l'étiquette de la zone de son service : sa classification ne **NE DOIT PAS** dépasser le niveau de la zone, et ses catégories **DOIT** être des catégories de la zone (**E044**). Un service ne sert donc pas de données plus sensibles que sa zone. Une classification qui n'est pas un niveau donne **E042**, et une route `public` ne **PEUT** pas exiger de classification ni de catégorie (**E043**).
- **Composition.** Les patrons qui déclarent des niveaux **DOIT** déclarer les mêmes, dans le même ordre (**E050**) ; un patron qui n'en déclare pas adopte ceux des autres. Les catégories s'unissent. Une zone étiquetée différemment par deux patrons donne **E051**.

Une étiquette dit à la fois *à quel point* une donnée est sensible (le niveau) et *à qui* elle est destinée (les catégories). Avec le patron de l'exemple de la spécification des manifestes (le patron `exemple_rh`), et deux zones `paie` et `dossiers` ajoutées pour l'illustration :

Zone	Étiquette	Comparaisons
`public`	(`public`, ∅)	inférieure à toutes les autres
`internal`	(`internal`, ∅)	`public` ≤ `internal` ≤ `rh`
`paie`	(`confidential`, {`finance`})	≤ `rh` ; incomparable avec `dossiers`
`dossiers`	(`confidential`, {`rh`})	≤ `rh` ; incomparable avec `paie`
`rh`	(`confidential`, {`rh`, `finance`})	supérieure à toutes les autres

`paie` et `dossiers` ont le même niveau, mais aucune des deux ne détient toutes les catégories de l'autre : un service de `paie` ne lit pas un service de `dossiers`, ni l'inverse. Un service de `rh`, qui détient les deux catégories, lit les deux.

SECTION 3

Le modèle `hamac_rbac`

`hamac_rbac` est le contrôle d'accès à base de rôles. Il n'ordonne pas les zones ; il ajoute :

Déclaration	Sens et contrôles
`rbac::role* name` , au patron	les rôles que l'organisation attribue à ses utilisateurs
`rbac::role* name` , à une route	rôles dont l'utilisateur **DOIT** détenir au moins un ; un rôle qu'aucun patron de l'ensemble ne déclare donne **E045**, une route `public` qui en exige un **E043**

SECTION 4

Accès aux routes avec `hamac_blp` et `hamac_rbac`

Une requête vers une route est autorisée si le jeton présenté satisfait les conditions de chaque modèle. Avec ces deux modèles, il établit pour l'utilisateur :

niveau ≥ `blp::classification` **et** `blp::category` ⊆ catégories de l'utilisateur **et** (`rbac::role` vide **ou** un rôle de l'utilisateur ∈ `rbac::role`).

Avec les routes de `hr_api` dans l'exemple de la spécification des manifestes (`/employees/*`, en lecture : `internal`, catégorie `rh`, rôle `rh` ou `admin` ; `/employees/salaries` : `confidential`, catégories `rh` et `finance`, rôle `comptable` ou `admin`), et trois utilisateurs :

Utilisateur	Habilitation	`GET /employees/42`	`GET /employees/salaries`
Alice	`internal`, { `rh` }, rôle `rh`	autorisée	refusée : niveau et catégorie `finance` manquants
Bruno	`confidential`, { `rh`, `finance` }, rôle `comptable`	refusée : ni `rh` ni `admin`	autorisée
Chloé	`confidential`, { `rh`, `finance` }, rôle `admin`	autorisée	autorisée

`/employees/salaries` est couvert par les deux routes ; la plus spécifique, `/employees/salaries`, s'applique seule. Bruno lit donc les salaires sans pouvoir consulter la fiche d'un salarié : chaque route donne ses propres conditions, et aucune n'hérite d'une autre. `/health` est public : aucun jeton n'est demandé. Un chemin qu'aucune route ne couvre, `/admin` par exemple, est refusé à tous.

SECTION 5

Le patron `hamac_default`

`hamac_default` est un patron prêt à l'emploi, pour une organisation qui n'a pas encore le sien : le modèle `hamac_blp` avec les niveaux `public` < `default` < `internal` < `secure`, sans catégorie, et une zone par niveau, de même nom. Une stack le charge par `use hamac_default 0.2.` ; un patron qui ne déclare que des rôles peut le charger pour en garder les zones (`examples/enterprise-stack/organisation.loaf`).

SECTION 6

Le module `hamac_ha`

`hamac_ha` déclare des exigences de disponibilité et d'isolation, qu'il ramène aux primitives de placement `separate` et `min_domains` (et à leurs préférences) :

Déclaration	Sens et primitives
`ha::isolate_zones`, au patron	deux zones ne partagent jamais une machine : `separate` entre les services de zones différentes
`ha::stateful_domains? int`, au patron	un service à état (un volume `local` ou `replicated`) a des répliques sur au moins ce nombre de domaines de panne, au moins 1 (**E016**) : `min_domains`
`ha::prefer_stateful_domains? int`, au patron	la même exigence, en préférence : `prefer_min_domains`
`ha::critical`, à un service	deux services critiques ne partagent pas une machine : `separate`

```
use hamac_ha 0.2.

pattern exemple_rh {
  ha::isolate_zones.
  ha::prefer_stateful_domains 3.
}

service hr_api { ha::critical. }
```

Financement

« Ce projet a été financé par le Gouvernement dans le cadre du plan France 2030 opéré par Cap Digital et financé par l'Union européenne – NextGeneration EU »
« Ce projet a été financé par le gouvernement dans le cadre de France 2030 »
« Financé par l'Union européenne - Next Generation EU dans le cadre du plan France Relance »







