LaFoSec: Étude de la sécurité intrinsèque des langages fonctionnels [1]

Partie II sur IV

Recommandations au développeur OCaml

Damien Doligez, Christèle Faure, Thérèse Hardin, Manuel Maarek

JFLA - février 2013

SAFE RIVER
Safety & Security Forge

Cédric

Normation

1. Etude commanditée par l'Agence Nationale de la Sécurité des Systèmes d'Information

Pourquoi des recommandations

Besoin : accroître la résistance aux attaques et faciliter l'évaluation de sécurité. Obstacles :

Pourquoi des recommandations

Besoin : accroître la résistance aux attaques et faciliter l'évaluation de sécurité. Obstacles :

- Résistance aux attaques assez différente de la résistance aux évènements redoutés dans la sûreté

Pourquoi des recommandations

Besoin : accroître la résistance aux attaques et faciliter l'évaluation de sécurité. Obstacles :

- Résistance aux attaques assez différente de la résistance aux évènements redoutés dans la sûreté
- La méthodologie de développement suivie peut impacter fortement la sécurité

Pourquoi des recommandations

Besoin : accroître la résistance aux attaques et faciliter l'évaluation de sécurité. Obstacles :

- Résistance aux attaques assez différente de la résistance aux évènements redoutés dans la sûreté
- La méthodologie de développement suivie peut impacter fortement la sécurité
- Aucun trait de programmation ne peut, en général, à lui seul dégrader la sécurité mais ne peut non plus la garantir

Pourquoi des recommandations

Besoin : accroître la résistance aux attaques et faciliter l'évaluation de sécurité. Obstacles :

- Résistance aux attaques assez différente de la résistance aux évènements redoutés dans la sûreté
- La méthodologie de développement suivie peut impacter fortement la sécurité
- Aucun trait de programmation ne peut, en général, à lui seul dégrader la sécurité mais ne peut non plus la garantir
- Nécessité d'utiliser conjointement, de manière contrôlée, plusieurs traits du langage, des bibliothèques dédiées et certaines options du compilateur.

Pourquoi des recommandations

Besoin : accroître la résistance aux attaques et faciliter l'évaluation de sécurité. Obstacles :

- Résistance aux attaques assez différente de la résistance aux évènements redoutés dans la sûreté
- La méthodologie de développement suivie peut impacter fortement la sécurité
- Aucun trait de programmation ne peut, en général, à lui seul dégrader la sécurité mais ne peut non plus la garantir
- Nécessité d'utiliser conjointement, de manière contrôlée, plusieurs traits du langage, des bibliothèques dédiées et certaines options du compilateur.

Cadre des recommandations
Pourquoi des recommandations ?

On ne peut pas compter sur le fait que tout développeur connaît parfaitement :

- la sémantique des traits spécifiques aux langages fonctionnels typés : typage fort avec synthèse des types, fonctionnalité d'ordre supérieur, etc.
- la sémantique des traits impératifs fournis par ces mêmes langages : références, exceptions, chaînes de caractères, etc. et celle du GC
- la sémantique de la programmation "in the large" : modules, classes, compilation séparée
- les différentes options de compilation, édition de liens et exécution

Le développeur peut être maladroit ou malveillant : il faut faciliter le diagnostic des anomalies et des attaques.

Cadre des recommandations

Cadre des recommandations

Le développeur peut utiliser :

- n'importe quel trait du langage

Cadre des recommandations

Le développeur peut utiliser :

- n'importe quel trait du langage
- des bibliothèques externes OCaml qui ne sont pas nécessairement de confiance

Cadre des recommandations

Le développeur peut utiliser :

- n'importe quel trait du langage
- des bibliothèques externes OCaml qui ne sont pas nécessairement de confiance
- des bibliothèques C qui ne sont pas de confiance

Cadre des recommandations

Le développeur peut utiliser :

- n'importe quel trait du langage
- des bibliothèques externes OCaml qui ne sont pas nécessairement de confiance
- des bibliothèques C qui ne sont pas de confiance

Hypothèses sur l'exécution :

Cadre des recommandations

Le développeur peut utiliser :

- n'importe quel trait du langage
- des bibliothèques externes OCaml qui ne sont pas nécessairement de confiance
- des bibliothèques C qui ne sont pas de confiance

Hypothèses sur l'exécution :

- Les entrées de l'exécutable ne sont pas contrôlées

Cadre des recommandations

Le développeur peut utiliser :

- n'importe quel trait du langage
- des bibliothèques externes OCaml qui ne sont pas nécessairement de confiance
- des bibliothèques C qui ne sont pas de confiance

Hypothèses sur l'exécution :

- Les entrées de l'exécutable ne sont pas contrôlées
- L'environnement d'exécution n'est pas à priori maîtrisé

Cadre des recommandations

Le développeur peut utiliser :

- n'importe quel trait du langage
- des bibliothèques externes OCaml qui ne sont pas nécessairement de confiance
- des bibliothèques C qui ne sont pas de confiance

Hypothèses sur l'exécution :

- Les entrées de l'exécutable ne sont pas contrôlées
- L'environnement d'exécution n'est pas à priori maîtrisé
- Seule certitude : les garanties du compilateur.

Cadre des recommandations

Le développeur peut utiliser :

- n'importe quel trait du langage
- des bibliothèques externes OCaml qui ne sont pas nécessairement de confiance
- des bibliothèques C qui ne sont pas de confiance

Hypothèses sur l'exécution :

- Les entrées de l'exécutable ne sont pas contrôlées
- L'environnement d'exécution n'est pas à priori maîtrisé
- Seule certitude : les garanties du compilateur.

Un recueil de règles

Un ensemble de règles :

Un recueil de règles

Un ensemble de règles :

- encadrant toute la production du source
- traitant des entrées du programme
- contraignant l'utilisation de bibliothèques OCaml
- contraignant l'utilisation de codes C
- contraignant l'environnement d'exécution

Un recueil de règles

Un ensemble de règles pour :

1. Défendre l'intégrité du comportement du code produit, donc augmenter la robustesse vis-à-vis d'attaques

Un recueil de règles

Un ensemble de règles pour :

1. Défendre l'intégrité du comportement du code produit, donc augmenter la robustesse vis-à-vis d'attaques
2. Compliquer la tâche du développeur malveillant

Un recueil de règles

Un ensemble de règles pour :

1. Défendre l'intégrité du comportement du code produit, donc augmenter la robustesse vis-à-vis d'attaques
2. Compliquer la tâche du développeur malveillant
3. Faciliter la préparation de l'évaluation sécurité

Un recueil de règles

Un ensemble de règles pour :

1. Défendre l'intégrité du comportement du code produit, donc augmenter la robustesse vis-à-vis d'attaques
2. Compliquer la tâche du développeur malveillant
3. Faciliter la préparation de l'évaluation sécurité
4. Faciliter le travail de l'évaluateur de la sécurité du logiciel

Un recueil de règles

Un ensemble de règles pour :

1. Défendre l'intégrité du comportement du code produit, donc augmenter la robustesse vis-à-vis d'attaques
2. Compliquer la tâche du développeur malveillant
3. Faciliter la préparation de l'évaluation sécurité
4. Faciliter le travail de l'évaluateur de la sécurité du logiciel

Un recueil de règles

Ensemble de règles sous forme de :

Un recueil de règles

Ensemble de règles sous forme de :

- recommandations (privilégier l'utilisation de ...) et des conseils (utiliser les types concrets ...)

Un recueil de règles

Ensemble de règles sous forme de :

- recommandations (privilégier l'utilisation de ...) et des conseils (utiliser les types concrets ...)
- mises en garde (ne pas confondre type de classe et type d'objet ...)

Un recueil de règles

Ensemble de règles sous forme de :

- recommandations (privilégier l'utilisation de ...) et des conseils (utiliser les types concrets ...)
- mises en garde (ne pas confondre type de classe et type d'objet ...)
- interdictions non-contournables (proscrire l'utilisation de Obj)

Un recueil de règles

Ensemble de règles sous forme de :

- recommandations (privilégier l'utilisation de ...) et des conseils (utiliser les types concrets ...)
- mises en garde (ne pas confondre type de classe et type d'objet ...)
- interdictions non-contournables (proscrire l'utilisation de Obj)
- interdictions avec échappement (ne pas utiliser ... sauf si) requérant alors une justification de l'innocuité de ce contournement dans la documentation du logiciel.

Un recueil de règles

Ensemble de règles sous forme de :

- recommandations (privilégier l'utilisation de ...) et des conseils (utiliser les types concrets ...)
- mises en garde (ne pas confondre type de classe et type d'objet ...)
- interdictions non-contournables (proscrire l'utilisation de Obj)
- interdictions avec échappement (ne pas utiliser ... sauf si) requérant alors une justification de l'innocuité de ce contournement dans la documentation du logiciel.

Le recueil ne contient aucune règle sur la qualité du codage (taille des identifiants, longueur des modules, etc.)

Règles de développement

Utiliser les traits de OCaml à bon escient

Refléter l'architecture du logiciel

Guide de conception

- Associer un module à chaque sous-système et adapter strictement les interfaces aux besoins de communication entre sous-systèmes.
- Les interfaces doivent garantir l'encapsulation des données sensibles échangées entre sous-systèmes.
- Compiler séparément chacun de ces modules.

Refléter l'architecture du logiciel

Guide de conception

- Associer un module à chaque sous-système et adapter strictement les interfaces aux besoins de communication entre sous-systèmes.
- Les interfaces doivent garantir l'encapsulation des données sensibles échangées entre sous-systèmes.
- Compiler séparément chacun de ces modules.
- Ne pas utiliser les classes pour refléter l'architecture du système.

Style de programmation

OCaml est un langage fonctionnel, donc :

Style de programmation

OCaml est un langage fonctionnel, donc :

- Utiliser un **style fonctionnel pur**, encapsuler les effets de bord nécessaires.
- **fonctions récursives non terminales** : estimer le nombre d'appels récursifs
- Privilégier le **filtrage** comme moyen de définition de fonctions.
- Si la taille des données manipulées le nécessite, **partager des sous-structures** en les nommant ou grâce au filtrage.
- Choisir la **portée des identificateurs** de manière à ne jamais laisser visible un identificateur devenu inutile.

Modules, encapsulation

- Déterminer l'interface d'un module avant de le définir. Comparer cette interface avec celle inférée par la compilation du corps du module (option `-i`)
- Ne jamais masquer les noms des modules de la bibliothèque standard.
- Ne pas utiliser `open`

Classes

Le mécanisme des classes est puissant mais non dénué de risques :

- Ne pas utiliser les variables d'instance ou le marqueur `private` pour répondre à des exigences de sécurité.

Classes

Le mécanisme des classes est puissant mais non dénué de risques :

- Ne pas utiliser les variables d'instance ou le marqueur `private` pour répondre à des exigences de sécurité.
- Utiliser toujours la syntaxe explicite des redéfinitions ainsi que le contrôle de celles-ci.
- Utiliser les noms de méthode ou leurs types pour différencier des familles d'objets.
- Encapsuler toute classe contenant des données sensibles dans un module dont l'interface interdit tout héritage et ne donne accès qu'aux méthodes sûres.

Mutables et effets de bord

- Justifier l'utilisation de valeurs mutables. Les protéger par encapsulation.
- Adapter strictement la portée des variables mutables au besoin de l'application, en la justifiant.
- Vérifier que les variables mutables ne sont pas indûment partagées par un mécanisme d'*aliasing*.
- Adapter strictement l'utilisation de champs mutables dans les enregistrements et les objets aux besoins de l'application, en la justifiant.

Types de données

- Ne pas utiliser les classes pour définir des structures de données.
- Utiliser les types prédéfinis dans la bibliothèque standard (listes, piles, ..) pour représenter les structures de données utilisées en algorithmique.
- Sinon, définir la représentation des données manipulées par des types reflétant directement leur spécification.

Types de données

- Ne pas utiliser les classes pour définir des structures de données.
- Utiliser les types prédéfinis dans la bibliothèque standard (listes, piles, ..) pour représenter les structures de données utilisées en algorithmique.
- Sinon, définir la représentation des données manipulées par des types reflétant directement leur spécification.
- Séparer les sous-ensembles d'un même ensemble en introduisant un type somme ayant un constructeur de valeur par sous-ensemble.
- Représenter les données arborescentes par des types somme et les n-uplets par des enregistrements.
- Représenter l'absence de valeur significative à l'aide d'un type option.

Typage et types abstraits

- Encapsuler les représentations de données devant satisfaire des invariants de représentation dans un module muni de l'interface appropriée.

Typage et types abstraits

- Encapsuler les représentations de données devant satisfaire des invariants de représentation dans un module muni de l'interface appropriée.
- S'assurer que le type inféré correspond au type attendu.

Filtrage

- Privilégier les définitions par filtrage
- N'utiliser que des filtrages exhaustifs.
- Détecter les filtrages fragiles avant toute modification de la définition d'un type, en recompilant avec l'option `-w +4`.
- Justifier l'utilisation de gardes et ne pas mettre d'effet de bord dans les gardes.

Filtrage

- Privilégier les définitions par filtrage
- N'utiliser que des filtrages exhaustifs.
- Détecter les filtrages fragiles avant toute modification de la définition d'un type, en recompilant avec l'option `-w +4`.
- Justifier l'utilisation de gardes et ne pas mettre d'effet de bord dans les gardes.

Règles de développement

Eviter certains risques

Constructions prohibées dans le texte source de l'application

Des règles incontournables

- Proscrire l'utilisation des modules `Obj`, `Marshal` et `Printexc`.
- Si échange de données, définir un *printer* et un *parser* adaptés à l'aide des outils de la distribution OCaml.
- Ne pas utiliser les fonctions `unsafe_*` de la bibliothèque standard.
- Proscrire l'utilisation de `String.create` et la remplacer par `String.make`.

Traitement de données confidentielles

Bien cacher les données confidentielles :

- Ne pas se reposer sur les fonctions de finalisation du GC pour effacer les données confidentielles.

Traitement de données confidentielles

Bien cacher les données confidentielles :

- Ne pas se reposer sur les fonctions de finalisation du GC pour effacer les données confidentielles.
- Représenter les données confidentielles par des valeurs mutables simples (tableaux ou chaînes de caractères).
- Encapsuler dans un module la définition et les opérations (création, modification, effacement) de cette structure mutable.
- N'exporter le type de données que de manière abstraite et n'exporter aucune fonction de modification, sauf une fonction d'effacement.

Traitement de données confidentielles

Et ajouter encore une couche :

Traitement de données confidentielles

Et ajouter encore une couche :

- Compléter la protection des données confidentielles encapsulées dans des modules en les enveloppant dans un type fonctionnel (une fermeture).

Traitement de données confidentielles

Et ajouter encore une couche :

- Compléter la protection des données confidentielles encapsulées dans des modules en les enveloppant dans un type fonctionnel (une fermeture).
- Contrôler les utilisations des fonctions et opérations d'égalité, de comparaison et de hachage sur les données comportant des sous-structures sensibles.

Traitement de données confidentielles

La valeur d'un identificateur `x` ne peut fuir que par :

- Appel d'une fonction faisant fuir son paramètre avec `x` en argument
- Affichage de la valeur de `x`
- Affectation de la valeur de `x` dans un mutable
- Retour de `x` à la fin de la portée de `x`
- Levée d'une exception avec `x` comme argument
- Liaison d'un nouvel identificateur `y` à la valeur de `x` et fuite de `y` par l'une des méthodes de cette liste.

Traitement de données confidentielles

La valeur d'un identificateur `x` ne peut fuir que par :

- Appel d'une fonction faisant fuir son paramètre avec `x` en argument
- Affichage de la valeur de `x`
- Affectation de la valeur de `x` dans un mutable
- Retour de `x` à la fin de la portée de `x`
- Levée d'une exception avec `x` comme argument
- Liaison d'un nouvel identificateur `y` à la valeur de `x` et fuite de `y` par l'une des méthodes de cette liste.

Vérifier l'absence de fuite des données confidentielles par relecture du code.

Traitement de données confidentielles

Règles sur les exceptions et les données confidentielles

- Interdire le passage de données sensibles en paramètre d'exception, même si elles sont encapsulées.
- Vérifier que les bibliothèques utilisées ne font aucun passage de données sensibles en paramètre d'exception.
- Interdire le masquage des noms d'exception.

Chaînes de caractères

En attendant des chaînes non mutables :

Chaînes de caractères

En attendant des chaînes non mutables :

- N'utiliser que des copies de chaînes de caractères littérales obtenues avec `String.copy`.
- Encapsuler les chaînes de caractères dont on souhaite préserver l'intégrité dans un type abstrait pour garantir leur non mutabilité

Entiers sur les architectures 32 bits

Les `int` sur une architecture 32 bits ne couvrent pas tous les besoins (taille de gros fichiers, ..)

- Préférer une architecture 64 bits pour restreindre le risque de débordement.
- Utiliser une bibliothèque de grands entiers si nécessaire.

Se rappeler que l'arithmétique sur les `int` est modulaire.

Règles de développement

Règles de réutilisation

Utilisation de codes externes

Cas de codes externes statiquement connus

- Ne pas utiliser de code externe, sauf pour des raisons dûment justifiées et documentées. Sinon ...

Utilisation de codes externes

Cas de codes externes statiquement connus

- Ne pas utiliser de code externe, sauf pour des raisons dûment justifiées et documentées. Sinon ...
- Faire une relecture critique du texte source du code C et démontrer son innocuité. Recompiler ce code.
- S'assurer de l'intégrité (avec une signature cryptographique par exemple) du code utilisé.
- Éviter le chargement dynamique de code. Sinon ..

Utilisation de codes externes

Chargement dynamique de code

- Proscrire les utilisations des fonctions `reset`, `add_interfaces` et `add_variable_units` de `Dynlink`.
- Retreindre, avec `Dynlink.allow_only`, les fonctions (sûres) utilisables et les regrouper dans un module dédié.
- Contrôler l'argument de `Dynlink.loadfile`
- Relire les sources des *plug-ins*. Les recompiler et contrôler leurs droits d'accès.

utilisation de bibliothèques OCaml pré-existantes

A partir des bibliothèques existantes, en reconstruire une dédiée à l'application, relire tous ses fichiers source en veillant aux points suivants.

utilisation de bibliothèques OCaml pré-existantes

A partir des bibliothèques existantes, en reconstruire une dédiée à l'application, relire tous ses fichiers source en veillant aux points suivants.

- Toute utilisation de `Obj`, de `Marshal`, de fonctions `unsafe_*` doit être justifiée, contrôlée et encapsulée.
- Tout appel à à `String.create` doit être contrôlé.
- S'il est impératif d'échanger des données par sérialisation/désérialisation, ajouter une signature cryptographique à la donnée sérialisée, de manière à contrôler l'origine de la valeur à désérialiser.
- Vérifier les utilisations des fonctions de la bibliothèque standard qui modifient l'environnement global d'exécution

Une installation de OCaml bien maîtrisée

Soigner l'installation du langage

- Prendre une version de confiance, et la rendre non modifiable après installation

Une installation de OCaml bien maîtrisée

Soigner l'installation du langage

- Prendre une version de confiance, et la rendre non modifiable après installation
- Ne pas donner de droits privilégiés aux compilateurs et exécutifs OCaml

Contrôler l'environnement d'exécution

- Vérifier les valeurs des variables d'environnement utilisées par OCaml `CAMLLIB`, `OCAMLLIB`, `CAML_LD_LIBRARY_PATH`, `OCAMLRUNPARAM`, etc.

Contrôler l'environnement d'exécution

- Vérifier les valeurs des variables d'environnement utilisées par OCaml `CAMLLIB`, `OCAMLLIB`, `CAML_LD_LIBRARY_PATH`, `OCAMLRUNPARAM`, etc.
- Interdire l'exécution en mode débogue de bytecode (non produit lui-même en mode débogue) en détruisant la variable `CAML_DEBUG_SOCKET`.

Contrôler l'environnement d'exécution

- Vérifier les valeurs des variables d'environnement utilisées par OCaml `CAMLLIB`, `OCAMLLIB`, `CAML_LD_LIBRARY_PATH`, `OCAMLRUNPARAM`, etc.
- Interdire l'exécution en mode débogue de bytecode (non produit lui-même en mode débogue) en détruisant la variable `CAML_DEBUG_SOCKET`.
- Ne pas donner accès à la pile de levée d'exceptions (mode backtrace) en interdisant l'exécution de codes (natif ou bytecode) compilés avec l'option -g.

Contrôler l'environnement d'exécution

- Vérifier les valeurs des variables d'environnement utilisées par OCaml `CAMLLIB`, `OCAMLLIB`, `CAML_LD_LIBRARY_PATH`, `OCAMLRUNPARAM`, etc.
- Interdire l'exécution en mode débogue de bytecode (non produit lui-même en mode débogue) en détruisant la variable `CAML_DEBUG_SOCKET`.
- Ne pas donner accès à la pile de levée d'exceptions (mode backtrace) en interdisant l'exécution de codes (natif ou bytecode) compilés avec l'option -g.
- Contrôler la configuration des signaux

Code cible

Cible de la compilation

- Préférer une implantation en code natif.

Code cible

Cible de la compilation

- Préférer une implantation en code natif.
- Justifier toute implantation en bytecode. Utiliser alors le mode `-custom` et sinon, justifier le choix.

Code cible

Cible de la compilation

- Préférer une implantation en code natif.
- Justifier toute implantation en bytecode. Utiliser alors le mode `-custom` et sinon, justifier le choix.
- Produire un exécutable complet en n'utilisant (si possible) que du code chargé statiquement, que ce soit en mode natif ou en bytecode (options `-custom` ou `-output-obj`).
- En exploitation, ne jamais utiliser la boucle interactive ni les options de débogue et de profilage du compilateur. Ne pas compiler avec l'option `-g`.

Code cible

Cible de la compilation

- Préférer une implantation en code natif.
- Justifier toute implantation en bytecode. Utiliser alors le mode `-custom` et sinon, justifier le choix.
- Produire un exécutable complet en n'utilisant (si possible) que du code chargé statiquement, que ce soit en mode natif ou en bytecode (options `-custom` ou `-output-obj`).
- En exploitation, ne jamais utiliser la boucle interactive ni les options de débogue et de profilage du compilateur. Ne pas compiler avec l'option `-g`.
- Règles sur les options de compilation à utiliser ou proscrire.

Conclusion

Un recueil de 143 règles réparties dans 26 fiches :

- qui fournit un ensemble de prescriptions et de guides pour le développement du source, de l'exécutable et de la documentation

Conclusion

Un recueil de 143 règles réparties dans 26 fiches :

- qui fournit un ensemble de prescriptions et de guides pour le développement du source, de l'exécutable et de la documentation
- qui procure une trame pour la préparation de l'évaluation en structurant le dialogue avec les développeurs

Conclusion

Un recueil de 143 règles réparties dans 26 fiches :

- qui fournit un ensemble de prescriptions et de guides pour le développement du source, de l'exécutable et de la documentation
- qui procure une trame pour la préparation de l'évaluation en structurant le dialogue avec les développeurs
- qui allège le travail des évaluateurs : le respect de ces règles élimine certains risques.

Conclusion

Un recueil de 143 règles réparties dans 26 fiches :

- qui fournit un ensemble de prescriptions et de guides pour le développement du source, de l'exécutable et de la documentation
- qui procure une trame pour la préparation de l'évaluation en structurant le dialogue avec les développeurs
- qui allège le travail des évaluateurs : le respect de ces règles élimine certains risques.

Bénéfices :

- Texte source mieux architecturé et plus lisible

Conclusion

Un recueil de 143 règles réparties dans 26 fiches :

- qui fournit un ensemble de prescriptions et de guides pour le développement du source, de l'exécutable et de la documentation
- qui procure une trame pour la préparation de l'évaluation en structurant le dialogue avec les développeurs
- qui allège le travail des évaluateurs : le respect de ces règles élimine certains risques.

Bénéfices :

- Texte source mieux architecturé et plus lisible
- Documentation ciblant mieux les propriétés de sécurité

Conclusion

Un recueil de 143 règles réparties dans 26 fiches :

- qui fournit un ensemble de prescriptions et de guides pour le développement du source, de l'exécutable et de la documentation
- qui procure une trame pour la préparation de l'évaluation en structurant le dialogue avec les développeurs
- qui allège le travail des évaluateurs : le respect de ces règles élimine certains risques.

Bénéfices :

- Texte source mieux architecturé et plus lisible
- Documentation ciblant mieux les propriétés de sécurité
- Donc, maintenance et évolution facilitées

Contacts

- Véronique Delebarre : veronique.delebarre@safe-river.com
- Christèle Faure : christele.faure@safe-river.com