Comment les directives populaires de l'industrie Misra C améliorent la sûreté et la sécurité des logiciels embarqués

L'annonce récente effectuée par le Misra Working Group de deux nouvelles mises à jour concernant les directives Misra C, Misra C:2012 Amendement 4 (AMD4) et de l'imminente Misra C:2023 marque une étape cruciale dans la prise en charge des besoins des développeurs de logiciels embarqués. Considérée depuis longtemps comme la norme industrielle de référence pour le développement de dispositifs à sûreté et sécurité critiques, Misra C apporte aujourd'hui aux développeurs des directives et règles de codage capables d'atténuer le risque de comportements inattendus ou indéfinis dans leurs applications. Explications en détail par la société LDRA.

AUTEURS

Andrew Banks, Technical Specialist, LDRA. Chairman du groupe de travail Misra C depuis 2013.

Avec la croissance régulière du nombre de dispositifs connectés et riches en fonctionnalités, les fabricants de produits à sécurité critique se sont tournés vers des techniques plus sophistiquées pour extraire davantage de fonctionnalités des microprocesseurs, microcontrôleurs et périphériques ayant des ressources limitées. Après des années de recherche et de développement dans des situations réelles, les nouvelles directives Misra C abordent ces techniques modernes avec des conseils sur le multithreading et les types atomiques afin de supporter les normes C ISO/CEI 9899:2011 et 2018 (connues respectivement sous le nom de « C11 » et « C18 »).

Pour rappel, Misra C n'est pas un guide de style de codage, mais plutôt un ensemble de règles et de directives visant à minimiser ou éliminer du langage des formules syntaxiques dont l'utilisation sémantique est réputée dangereuse. Dans le cadre d'un processus de développement logiciel complet, de nombreuses équipes de développement appliquent un guide de style de langage en plus des directives Misra C.

Misra C est aujourd'hui présent dans de nombreuses normes de sécurité fonctionnelle auxquelles les développeurs doivent se conformer. Les directives sont directement citées ou couramment appliquées dans les projets qui suivent les processus Autosar, CEI 62304, CEI 61508, ISO 26262 et DO-178C. Misra C constitue également la base de la norme de codage C++ du Joint Strike Fighter et de la norme de codage C de la NASA Jet Propulsion Library, entre autres.

Pourquoi les développeurs C ont besoin des directives Misra C

Les développeurs de systèmes embarqués incluent régulièrement des exigences en matière de sûreté, de sécurité et de fiabilité des logiciels dans leurs processus de construction de systèmes allant des unités de commande de moteur automobile aux machines industrielles. Si de tels systèmes se comportent de manière inattendue et dévient du processus strict prévu par le développeur, des problèmes de sécurité et des vulnérabilités risquent d'en résulter.

Néanmoins, les développeurs de systèmes embarqués sont confrontés à des difficultés pour y parvenir. Tout d'abord, le langage C permet aux développeurs de contrôler le comportement et la mémoire des applications d'une manière qui pourrait compromettre leur comportement prévu. De plus, C11 et C18 ne fournissent qu'une spécification incomplète de l'intention d'exécution, laissant certains aspects à l'appréciation du développeur et à son implantation. Ce qui conduit notamment à des particularités non déterministes du langage C, laissant une certaine liberté aux développeurs qui, de ce fait, pourraient les utiliser d'une manière potentiellement risquée pour le système en cours de développement. Et ce en raison :

- d'un comportement indéfini car la norme C ne spécifie pas d'exigence claire ;
- d'un comportement non spécifié car la norme C prévoit deux ou plusieurs possibilités et ne spécifie pas d'exigences claires concernant son utilisation ;
- d'une définition guidée par l'implantation car le compilateur et le runtime sont libres de définir leurs propres comportements et doivent les documenter eux-mêmes.

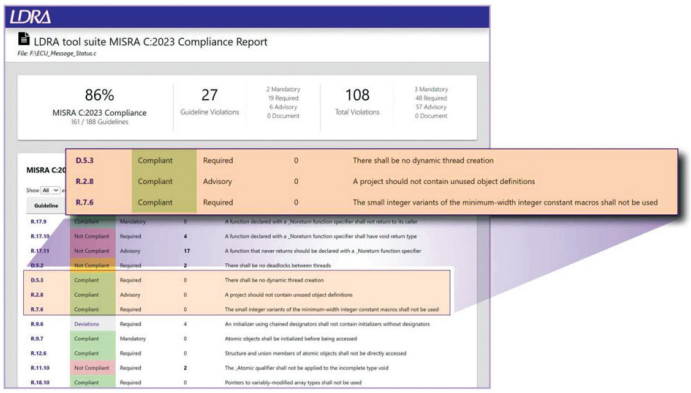
Un exemple de comportement indéfini consiste en particulier à laisser les variables de type tableau de grande taille non initialisées. Pour éviter le coût de traitement des appels de type memset() pour réinitialiser la mémoire, les développeurs renoncent parfois à initialiser ces types de données. Le standard C ne spécifiant pas comment les implantations doivent gérer les types de tableaux non initialisés, cette pratique peut avoir des conséquences imprévisibles lors de l'exécution. C'est pourquoi de nombreux compilateurs et analyseurs statiques le signaleront comme un problème.

Mais il ne s'agit là que d'un exemple extrêmement simple des lacunes que

les directives Misra C visent à éliminer. D'autres exemples montrent ces lacunes. A savoir, par exemple:
- L'écriture dans des fichiers ouverts en lecture seule, entraînant un comportement potentiellement indésirable;
- L'utilisation de fonctions récursives, entraînant un débordement potentiel de la pile;
- L'accès à la mémoire en dehors des limites d'une structure de données (par exemple, débordement de mémoire tampon), ce qui peut créer un angle d'attaque potentiel pour des pirates.

2 RAPPORT DE CONFORMITÉ MISRA

On voit ici un exemple de rapport de conformité Misra issu de la suite d'outils LDRA. *(Source: LDRA)*



Structure des directives Misra C

Les directives Misra C limitent en fait la syntaxe et la sémantique du langage C à un sous-ensemble prévisible qui répond aux besoins des systèmes critiques en matière de sûreté et de sécurité. En respectant des règles et des lignes directrices, ces directives minimisent, voire éliminent, les pratiques de codage réputées dangereuses ou peu sûres:
Règle: il s'agit d'une exigence de code source qui soit complet, objectif et sans ambiguïté. Les directives classent les règles comme décidables si elles peuvent être vérifiées de manière concluante par des techniques telles que l'analyse statique, et comme indécidables si aucune garantie de vérification n'est possible.
Exemple de règle: «La valeur d'un objet avec une durée de stockage automatique ne doit pas être lue avant d'avoir été définie». (Misra C:2012 règle 9.1).
Directive: il s'agit d'une ligne directrice qui peut être satisfaite par le biais d'un code, d'un processus, d'une documentation ou d'une exigence fonctionnelle. Les directives pouvant être sujettes à interprétation, des outils d'analyse aident à vérifier la conformité ; si ce n'est pas le cas, d'autres techniques de validation doivent être utilisées.
Exemple de directive: «Tout comportement défini par l'implémentation dont dépend la sortie du programme doit être documenté et compris». (Misra C:2012 Directive 1.1).
Prenons cet exemple qui utilise l'opérateur C sizeof() conformément à la norme C:

```
void foo(int32_t x)
{
size_t x;
s=sizeof(int32_t[x]); //Conforme à MISRA C
s = sizeof(int32_t[x++]);  //Non conforme à MISRA C
}
```

Comme sizeof()n'exécute pas les expressions qui lui sont transmises mais évalue simplement le type et la taille de l'expression résultante au moment de la compilation, ce code peut entraîner un comportement inattendu lors de l'exécution. En tant que tel, cet usage est limité par Misra C:2012, règle 13.6, indiquant que «l'opérande de l'opérateur sizeof ne doit contenir aucune expression ayant des effets secondaires potentiels».
Il s'agit d'une règle décidable, ce qui signifie qu'un développeur peut utiliser l'analyse statique pour trouver des occurrences dans le code où cette règle a été violée.
Une fois identifiée, les développeurs peuvent corriger l'erreur avec un simple ajustement:

```
void foo(int32_t x)
{
size_t x;
s=sizeof(int32_t[x]); //Conforme à MISRA C
++x;
s=sizeof(int32_t[x]); //Conforme à MISRA C
}
```

Les 25 années d'évolution des directives Misra C, ainsi que la complexité croissante des logiciels embarqués, ont permis aux outils de vérification de la conformité de progresser de manière spectaculaire en termes d'efficacité. Misra C lui-même recommande l'utilisation d'outils d'analyse statique pour assurer cette conformité.

Comment Misra C s'applique à la cybersécurité

La plupart des principes de sûreté du code s'appliquent également à la sécurité du code, simplement parce qu'un code bien écrit est à la fois sûr et sécurisé.
Un exemple courant et typique lié à la sécurité est le dépassement de mémoire tampon, où le volume de données écrites dans une zone de mémoire dépasse la taille réelle de cette allocation de mémoire. Un pirate informatique peut exploiter ce comportement pour écraser des zones protégées de la mémoire, modifiant potentiellement le chemin d›exécution des programmes ou introduisant un code malveillant de sa propre conception. Misra C couvre ce scénario dans la section 8.18 «Pointeurs et tableaux» avec des règles pertinentes à la fois pour la sûreté et la sécurité.
Par exemple, la règle Misra C:2012 18.2 stipule: «Un pointeur résultant d'opérations arithmétiques sur un opérande de pointeur doit adresser le même tableau que cet opérande de pointeur». Considérez l'exemple suivant dans ce contexte.
Ce code peut introduire une condition de dépassement de mémoire tampon en raison de l'absence de validation de la longueur du message entrant, pMessage.

```
extern uint8_t buffer[16];
/* pMessage pointe vers un message externe
* Le premier octet contient la longueur du message
*/
void processMessage (uint8_t const *pMessage)
{
uint8_t length = *pMessage; /* Longueur non validée */
for (uint8_t i = 0u; i < length; ++i)
{
++pMessage;
buffer[i] = *pMessage;
}
}
```

Un analyseur de code statique signalerait cela comme une violation de Misra C, conformément à la règle

2 FICHE DE DÉVIATION

On voit ici un exemple de fiche de déviation à Misra.

(Source : Conformité Misra : guide 2020)

Project	F10_BCM		
Deviation ID	D_00102	Status	Approved
Permit	Permit / Example / C:2012 / R.10.6.A.1		
Rule 10.6	The value of a composite expression shall not be assigned to an object with wider essential type		
Use case	The value of a composite expression is assigned to an object of wider essential type to avoid sub-optimal compiler code generation		
Reason	Code Quality (Time behaviour)	Scope	Project
Tracing tags	D_00102_1 to D_00102_10		

Raised by	E C Unwin	Approved by	D B Stevens
	Signature		*Signature*
Position	Software Team Leader	Position	Engineering Director
Date	14-Mar-2015	Date	12-Apr-2015

ci-dessus. Une fois identifié, les développeurs peuvent corriger ce comportement avec une vérification des limites et un gestionnaire d'erreurs sur pMessage.

Les directives Misra C incluent également des directives explicites pour le codage sécurisé, le cas échéant, afin d'aider les développeurs à éviter les pratiques de codage non sécurisées. De leur côté, les outils d'analyse statique mettent en œuvre une méthode d'analyse claire pour détecter les vulnérabilités potentielles.

Un exemple d'une telle directive est « La validité des valeurs reçues de sources externes doit être vérifiée » (Misra C:2012 Directive 4.14). La pratique la plus importante de ce codage sécurisé consiste à valider toutes les données provenant d'une source externe telle qu'un port RS232, une entrée utilisateur ou un domaine différent. Cela implique la présence de code dit "défensif" pour vérifier en permanence non seulement que les données reçues sont dans les limites attendues, mais aussi pour s'assurer que les données ont un sens. Par exemple, un historique des commandes reçues au fil du temps peut être utilisé pour vérifier des schémas d'utilisation anormaux.

Mise en œuvre de la conformité Misra C

Les directives Misra C ne spécifient pas de processus et d'outils spécifiques pour assurer la conformité, car de telles spécifications limiteraient injustement ce que les équipes de développement embarqué peuvent se procurer et mettre en œuvre. Le guide Misra Compliance 2000 fournit plutôt des définitions de ce qui doit être couvert dans le cadre du processus de développement logiciel lorsqu'il s'agit de revendiquer la conformité à Misra.

Ce guide précise ce qui suit :

Les directives MISRA sont destinées à être utilisées dans le cadre d'un processus documenté de développement logiciel. La conformité aux directives Misra doit faire partie intégrante de la phase de développement du code et les exigences de conformité doivent être satisfaites avant que le code ne soit soumis à l'examen ou aux tests unitaires.

Le guide explique en outre que les déclarations de conformité doivent inclure :

- l'utilisation d'un processus de développement logiciel rigoureux ;
- l'application de directives exactes doivent être appliquées ;
- la mise en œuvre de méthodes d'application efficaces ;
- la maîtrise de l'étendue de tout écart par rapport aux directives ;
- le respect du statut de tout composant logiciel développé en dehors du projet.

Les équipes de logiciels embarqués doivent donc décider de la meilleure façon d'appliquer les règles et directives Misra à leurs processus de développement et d'assurance qualité. Compte tenu de la complexité des applications embarquées contemporaines et de la rapidité des cycles de publication, la plupart des équipes intègrent désormais un outil de vérification automatisé, comme celui dévelppé par LDRA, dans leur stratégie de conformité afin de réduire les efforts manuels, d'améliorer la couverture et de minimiser les erreurs humaines. Le guide Misra Compliance:2020 recommande ainsi l'utilisation d'outils d'analyse statique, mais les directives avertissent également que l'automatisation ne peut pas vérifier complètement toutes les règles et directives.

Présentation du document Misra C:2023

Les dernières éditions AMD4 et Misra C:2023 couvrent le multithreading en C et les types atomiques pour répondre aux techniques de concurrence et interprocessus de plus en plus utilisées par les développeurs de systèmes embarqués actuels. Alors que de nombreuses bonnes pratiques se concentrent sur les performances des threads et l'optimisation de la mémoire, les directives Misra C visent exclusivement à éviter les comportements de concurrence non définis et imprévisibles qui sont susceptibles de compromettre la sûreté et la sécurité du système.

Les directives visent à minimiser les problèmes potentiels de concurrence en prenant en compte les fonctionnalités de multithreading du langage C, à savoir :

- restreindre la création dynamique de threads pour assurer une approche déterministe de la concurrence ;
- s'assurer que l'application crée les threads avant de leur associer des mutex ;
- minimiser le risque de blocages et de courses de données (Data Races)
- gérer l'utilisation sécurisée des objets et identifiants de thread.

Misra C comprend également des directives qui traitent des comportements non définis des types atomiques qui sont susceptibles de compromettre le système. Ces directives visent à :

- s'assurer que l'application configure correctement les types atomiques ;
- prévenir la suppression involontaire de l'atomicité lors du référencement de types atomiques par des pointeurs ;
- restreindre l'utilisation de plusieurs types atomiques dans la même instruction.

Les nouvelles éditions comprennent également d'autres règles qui reflètent les problèmes observés concrètement lors de l'utilisation du langage C ces dernières années. Ces règles incluent la restriction de l'utilisation de petites macros sur des entiers et de certains scénarios d'initialisation déterminés qui peuvent entraîner des initialisations multiples et conflictuelles du même élément.

Pour simplifier et rationaliser la gestion de la conformité et de la configuration, Misra C:2023 regroupe en fait toutes les éditions antérieures de Misra C et les améliorations AMD4 dans un document de référence unique et complet. Pour ceux qui ont l'habitude de gérer plusieurs versions et copies des éditions précédentes, ce changement est le bienvenu. ■