Pour une qualification de la bibliothèque standard C dans des applications à sûreté critique

Les développeurs de logiciels ne doivent pas supposer que les outils et composants du commerce (COTS) tels que les compilateurs et bibliothèques standard sont exempts d'erreurs, ou que leur préqualification implique qu'ils le soient. En générant un rapport de qualification complet adapté aux besoins de l'organisme de certification, Solid Sands montre ici comment il est possible de démontrer l'intégrité des composants de bibliothèques utilisés dans les applications à sûreté critique.

AUTEUR

Marcel Beemster, Directeur Technique, Solid Sands.

On le sait, les solutions logicielles jouent un rôle de plus en plus important dans les systèmes à sûreté critique et les systèmes liés à la sûreté en général, dans lesquels les dysfonctionnements logiciels représentent des dangers potentiels et une menace réelle en termes de blessures, de décès, d'interruption de services essentiels ou de dommages environnementaux. Des organisations comme l'ISO et la Commission électrotechnique internationale (CEI) ont publié des normes largement adoptées grâce auxquelles la sûreté des logiciels peut être certifiée. Citons par exemple la norme ISO 26262 (véhicules routiers - sûreté fonctionnelle) pour l'automobile, la norme EN 50128 (systèmes de communication, de signalisation et de traitement - logiciels pour les systèmes de commande et de protection ferroviaires) pour le transport ferroviaire ou encore la norme CEI 61508 (sûreté fonctionnelle des systèmes électriques/ électroniques/ électroniques programmables) pour les applications industrielles.

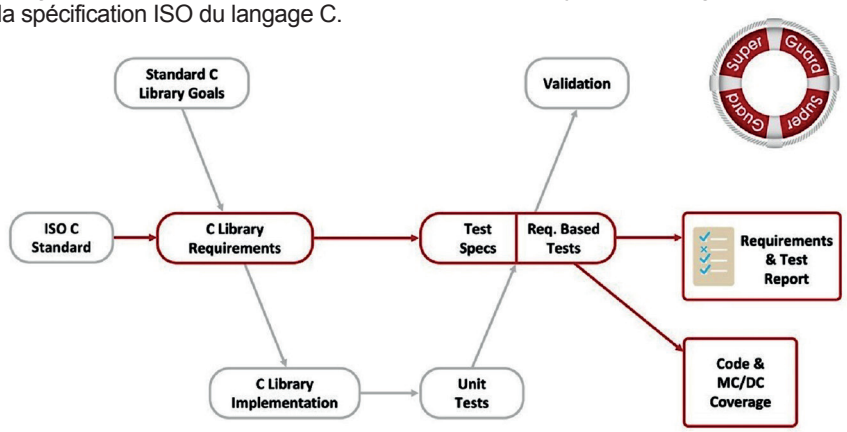
Dans ce cadre, les développeurs d'applications doivent démontrer que les logiciels, ainsi que les méthodes, les processus et les chaînes d'outils utilisés pour les développer sont conformes à ces normes. Cependant, une grande partie de la chaîne d'outils échappe au contrôle du développeur, ce qui rend vitale la validation du compilateur. Or peu de compilateurs sont exempts de bogues. Par conséquent, connaître les domaines dans lesquels ils fonctionnent mal permet d'éviter certaines erreurs.

Vers l'analyse détaillé d'un compilateur

Si on regarde dans le détail un projet, on s'aperçoit qu'une grande partie du code source qui se retrouve généralement sous la forme d'un binaire compilé ne passe jamais dans le compilateur dans des conditions identiques de cas d'utilisation, d'options de compilation et d'environnement matériel cible. Et une partie de ce code comprend des fonctions de bibliothèque précompilées telles que celles de la bibliothèque C standard (libc), souvent fournies au format binaire dans un kit de développement logiciel (SDK). Or, à ce niveau, l'inclusion de macros rend souvent les composants de la bibliothèque sensibles aux cas d'utilisation. Même dans une bibliothèque préqualifiée par le fournisseur du SDK à l'aide du même compilateur que celui livré avec le SDK, le même cas d'utilisation, les mêmes options de compilateur, et les mêmes exigences en matière d'environnement matériel cible ne sont pas forcément respectées, ce qui rend difficile la démonstration de la conformité. Pour surmonter cette difficulté, la suite de qualification de sûreté de bibliothèque C SuperGuard de Solid Sands fournit une traçabilité complète des résultats des tests individuels, jusqu'aux exigences dérivées de la spécification ISO du langage C. Cette suite peut prendre en charge la qualification des implantations de la bibliothèque C standard pour les applications à sûreté

1 RÔLE DU LOGICIEL SUPERGUARD DANS UN CYCLE DE DÉVELOPPEMENT EN V

La suite de qualification de sûreté de bibliothèque C SuperGuard de Solid Sands fournit une traçabilité complète des résultats des tests individuels, jusqu'aux exigences dérivées de la spécification ISO du langage C.



2 EXEMPLE DE RAPPORT AUTOMATIQUE D'UN TEST

SuperGuard intègre des capacités évoluées en matière de rapports et de documentation des exigences, de tests individuels et de résultats de tests, conformément aux principales normes.

Section	Title	Requirement	Description	Test Specification	Test	Result
	function			returned value corresponds to the destination buffer.	tspr3060.c	
C99:7.21.2.3	The strcpy function	REQ-C99:7.21.2.3-return	The strcpy function returns the value of s1.	Call the strcpy() function under two different scenarios (regular copy and zero length copy) and verify the returned value corresponds to the destination buffer in both cases.	4/11/2/3/t2.c	PASSED
C99:7.21.2.4	The strncpy function	DECL-C99:7.21.2.4	char *strncpy(char *s1, const char *s2, size_t n);			
C99:7.21.2.4	The strncpy function	UNDEF-C99:7.21.2.4	Behavior is undefined if the objects s1 and s2 overlap.			
C99:7.21.2.4	The strncpy function	UNDEF-C99:7.21.2.4	Behavior is undefined if the array s1 has a size less than n.			
C99:7.21.2.4	The strncpy function	UNDEF-C99:7.21.2.4	Behavior is undefined if s2 is not a string and has a size less than n.			
C99:7.21.2.4	The strncpy function	REQ-C99:7.21.2.4-copystring	If s2 points to a string with a length 'l2' that is less than n, strncpy() shall copy l2 characters from the array s2 into the array s1.	Call the strncpy() function with different values for the n parameter (including n==0) equal to and larger than the length of the origin string. Verify that the origin string is copied into the target array up to the terminating null character.	4/11/2/4/t1.c	PASSED
C99:7.21.2.4	The strncpy function	REQ-C99:7.21.2.4-copyn	If s2 does not point to a string with length less than n, strncpy() shall copy the first n characters from the array s2 into the array s1.	Call the strncpy() function with different values for the n parameter (including n==0), so that the first n characters of the origin array do not contain the null character. Verify that the first n characters of the origin string are copied into the target array.	4/11/2/4/t1.c	PASSED
C99:7.21.2.4	The strncpy function	REQ-C99:7.21.2.4-nomore	strncpy() shall not write into the target array s1 beyond the first n characters.	For all test cases, verify that the character with index n in the target array s1 is not modified. If that fits with the test, verify that also no characters beyond n are modified.	4/11/2/4/t1.c	PASSED
C99:7.21.2.4	The strncpy function	REQ-C99:7.21.2.4-nochange	strncpy() shall not modify array s2.	For all test case, verify that the array s2 is not modified.	4/11/2/4/t1.c	PASSED
C99:7.21.2.4	The strncpy function	REQ-C99:7.21.2.4-padding	If s2 points to a string with a length that is less than n, strncpy() shall append null characters after the copied characters in array s1 until n characters in total have been written.	Call the strncpy() function with strings that are shorter than the value passed as the n parameter. Verify that null characters are appended after the string copy in s1 until n characters are written.	4/11/2/4/t1.c	PASSED
C99:7.21.2.4	The strncpy function	REQ-C99:7.21.2.4-return	strncpy() shall return the value of s1.	Call the strncpy() function with different strings and values for the size (parameter n), and verify the returned value corresponds to the destination buffer in each case.	4/11/2/4/t1.c	PASSED

critique, tant pour les bibliothèques tierces non modifiées, que pour les implémentations auto-développées ou auto-maintenues.

Un objectif et un seul : la qualification

Cette qualification de la bibliothèque logicielle est essentielle puisque le code de cette bibliothèque est lié à l'application et est installé sur le dispositif cible. Un composant de bibliothèque défectueux met immédiatement en péril la sûreté fonctionnelle de l'ensemble de l'application. L'utilisation de bibliothèques logicielles dans les normes de sûreté fonctionnelle a en général un objectif : vérifier que l'implantation de la bibliothèque est conforme à sa spécification. Par exemple, l'ISO 26262 prévoit deux voies de qualification des bibliothèques, détaillées dans l'ISO 26262 Partie 8 et l'ISO 26262 Partie 6. L'outil SuperGuard fonctionne dans les deux cas de figure.

Ainsi, lors de la mise en œuvre des tests reposant sur les exigences recommandées dans les parties 8 et 6 de l'ISO 26262, le principal problème rencontré avec les spécifications de la bibliothèque C standard est que, si ces exigences fournissent une description comportementale détaillée pour chaque fonction, aucune d'elles ne définit d'exigences claires. Il faut donc créer ces exigences à partir des descriptions.

Pour ce faire, SuperGuard intègre la suite de tests de bibliothèque C standard éprouvée provenant de la suite de tests et de vérification de compilateur SuperTest de Solid Sands, mais avec des capacités accrues en matière de rapports et de documentation des exigences, de tests individuels et de résultats de tests, conformément aux principales normes.

Adaptés à un large éventail d'environnements de développement, les tests vérifient ici que le comportement de l'implantation est bien conforme à la spécification de la bibliothèque. Chacun d'eux exécute la construction ou la fonction à tester et compare les résultats de l'exécution aux résultats attendus (c'est-à-dire avec le « modèle ») définis dans la spécification de la bibliothèque. Pour vérifier le comportement de l'implantation, les tests sont compilés et exécutés dans un environnement d'exécution, ce qui signifie que toute la chaîne d'outils, y compris le processeur cible, est impliquée dans chaque test. Ce qui rend SuperGuard approprié pour la vérification des bibliothèques avec du matériel dans la boucle.

Quant aux tests de la partie autonome de la bibliothèque (généralement utilisée dans les systèmes dits « bare-metal », c'est-à-dire bruts sans aucune personnalisation), ils nécessitent des ressources minimales. Concrètement, la plupart des tests SuperGuard peuvent être exécutés sur des systèmes disposant de moins de 4 Ko de mémoire, ce qui rend la solution adaptée aux tout petits systèmes embarqués.

Pour mettre en œuvre les tests reposant sur des exigences, SuperGuard décompose les spécifications de la bibliothèque C standard en exigences d'implantation testables, ainsi qu'en spécifications de test décrivant comment chaque exigence est testée. En reliant les résultats individuels d'exécution des tests à la spécification de test, à l'exigence et à la fonction de bibliothèque standard correspondante, SuperGuard fournit une traçabilité complète pour les tests

EXEMPLE DE TEST EN DÉTAIL

Chaque test de la bibliothèque SuperGuard est développé selon une méthodologie cohérente. C'est la spécification de la section 7.21.2.4 de la définition du langage C99 :

7.21.2.4 Function strncpy

Synopsis

1 **#include <string.h>**
char * strncpy(char * restrict s1, const char * restrict s2, size_t n);

Description

2 La fonction **strncpy** copie au maximum **n** caractères (les caractères qui suivent un caractère nul ne sont pas copiés) du tableau pointé par **s2**, vers le tableau pointé par **s1**. Si la copie a lieu entre des objets qui se chevauchent, le comportement est indéfini.

3 Si le tableau pointé par **s2** est une chaîne plus courte que **n** caractères, des caractères nuls sont ajoutés à la copie dans le tableau pointé par **s1**, jusqu'à ce que **n** caractères en tout aient été écrits.

Renvoi de la fonction

4 La fonction **strncpy** renvoie la valeur de **s1**.

Le point clé ici est que le paragraphe 2 précise ce que **strncpy()** ne fait pas. Il se lit comme suit : « ne copie pas plus de **n** caractères. » Il n'est nulle part exigé que des caractères soient copiés de **s2** vers **s1**. Au paragraphe 3, est indiquée une action, à savoir que **s1** est complété par des caractères nuls. Une mise en œuvre strictement correcte consisterait à écrire **n** caractères nuls dans **s1**.

Cependant, ce n'est pas ce que la fonction doit faire. La compréhension générale est que la fonction copie autant de caractères que possible de **s2** à **s1** jusqu'à ce que la chaîne **s2** ou n soit épuisée. Mais pour définir les exigences, il faut être plus précis.

La première étape du développement de test consiste à extraire un ensemble d'exigences (REQ) de la description, en considérant ce que la fonction doit réellement faire. Ces exigences sont :

REQ-copystring : Si **s2** pointe vers une chaîne dont la longueur **l2** (définie par **strlen()**) est inférieure à **n**, **strncpy()** copie **l2** caractères, dans l'ordre, du tableau **s2** vers le tableau **s1**.

REQ-copyn : Si **s2** ne pointe pas vers une chaîne de caractères de longueur inférieure à **n**, **strncpy()** copiera les n premiers caractères, dans l'ordre, du tableau **s2** vers le tableau **s1**.

REQ-shorter : Si **s2** pointe vers une chaîne de caractères d'une longueur inférieure à **n**, **strncpy()** ajoutera des caractères nuls (**\0**) après les caractères copiés dans le tableau **s1**, jusqu'à ce que **n** caractères au total aient été écrits.

REQ-nomore : **strncpy()** ne doit pas écrire dans le tableau cible **s1** au-delà des **n** premiers caractères.

REQ-nochange : **strncpy()** ne doit pas modifier le tableau **s2**.

REQ-return : **strncpy()** doit retourner la valeur de **s1**.

L'exigence **REQ-nochange** découle de la déclaration de **s2** comme tableau constant, mais cela ne garantit pas qu'une implémentation de **strncpy()** n'écrira pas dans **s2**.

Pour chaque exigence, une spécification de test est ensuite développée, en définissant comment le test vérifie que l'exigence est « Vrai ». Une seule spécification de test conduit généralement à plusieurs cas de test différents, couvrant les différents domaines d'entrée et de sortie de la fonction. Les cas de test sont mis en œuvre par le test. La spécification de test relie l'exigence aux tests.

Par exemple, les spécifications de test pour les exigences **REQ-copystring** et **REQ-nomore** sont les suivantes :

Spécification de test pour **REQ-copystring** : Appelez la fonction **strncpy()** avec différentes valeurs du paramètre n (y compris **n==0**) égales et supérieures à la longueur de la chaîne d'origine. Vérifiez que la chaîne d'origine est bien copiée dans le tableau cible jusqu'au caractère nul de fin.

Spécification de test pour **REQ-nomore** : Pour tous les cas de test, vérifier que le caractère d'indice n dans le tableau cible **s1** n'est pas modifié. Si cela correspond au test, vérifiez également qu'aucun caractère au-delà de **n** n'est modifié.

Ici, la spécification de test **REQ-nomore** est implantée dans le même fichier de test que les autres cas de test pour **strncpy()**. Etant donné que l'exigence doit de toute façon être inconditionnelle pour chaque appel à **strncpy()**, au lieu de créer de nouveaux tests pour cette spécification de test, elle est implantée simplement en se greffant sur une vérification supplémentaire de chaque test de cas pour les autres exigences, plutôt que de créer de nouveaux tests.

reposant sur des exigences. Pour prouver son exhaustivité, il offre une couverture de code structurel de près de 100 % pour plus de 80 % des fonctions, avec une couverture MC/DC (Modified Condition/Decision Coverage) élevée.

Attention toutefois, toutes les fonctions de la bibliothèque C standard ne sont pas implantées uniquement sous forme de binaires précompilés. Beaucoup dépendent aussi fortement des informations contenues dans les fichiers d'en-tête source. Ces derniers définissent les types, les variables globales et les macros et font autant partie de la bibliothèque que les fonctions (précompilées) de la bibliothèque. De nombreuses fonctions sont implantées à la fois comme des fonctions réelles et comme des macros. Pour des raisons de rapidité et d'efficacité, il est donc courant d'utiliser la mise en œuvre de macros. SuperGuard teste les deux.

Contrairement aux binaires correspondants, les macros de type fonction ne sont pas ici précompilées, mais générées par le compilateur du SDK avec le code source de l'application. Il est donc essentiel qu'elles soient vérifiées, ainsi que le contenu des autres fichiers d'en-tête, pour le cas d'utilisation spécifique d'une application donnée.

L'analyse de la couverture du code, incontournable

Dans SuperGuard, une attention particulière est accordée à la couverture de code obtenue pour une implantation de bibliothèque C standard open source, mature et renommée, afin de répondre à l'exigence ASIL D de la clause 12.4.2.3. Pour de nombreuses fonctions de cette bibliothèque, SuperGuard atteint une couverture de 100 % et une couverture MC/DC élevée. Les tests SuperGuard traitent aussi les cas dits « anormaux » de deux manières. La première concerne le comportement défini résultant d'une entrée anormale – par exemple, le passage d'un nombre négatif à la fonction sqrt() qui doit renvoyer la valeur NaN (en se basant sur l'arithmétique CEI 60559). Bien qu'anormal, le comportement de la fonction est parfaitement défini et peut être vérifié.

La seconde concerne les exigences que le compilateur peut vérifier. Par exemple, si une fonction doit avoir un type de retour void, un test peut essayer d'utiliser la valeur de retour en s'attendant à ce que cela génère une erreur de compilation. SuperGuard appelle un tel test un X-Test. Les X-Tests donnent une réponse positive (PASS) si le compilateur génère une erreur à la compilation, et une réponse négative (FAIL) dans le cas contraire. Les X-Tests ne sont jamais exécutés.

Enfin, SuperGuard ajoute la traçabilité nécessaire pour faire le lien entre les résultats des tests reposant sur les exigences, et la spécification de la bibliothèque C standard. Une traçabilité assurée en décomposant les spécifications fonctionnelles de la bibliothèque standard ISO C en exigences clairement définies, en développant des spécifications de test appropriées, et en les mettant en œuvre conformément à la norme requise. Une approche qui permet aux développeurs d'effectuer des tests dans le même environnement de développement, dans les mêmes conditions d'utilisation et sur le même matériel cible que leur application, avec une couverture du code structurel proche de 100 %.