

TD – Eva

Rendu

Le but est de rendre un rapport présentant vos réponses aux questions de ce TD. Vous devez me rendre le rapport et **uniquement** le rapport. Il doit donc présenter **tous les éléments nécessaires** pour démontrer que vous avez bien compris la question et bien répondu à la question (et donc être illustré par des screenshots des éléments importants, par des copier/coller des morceaux de code importants, etc). **Mettez l'accent sur vos explications**, ce sont elles qui importent le plus. Le résultat n'a de valeur qu'en lien avec son côté explicable (comme lors d'un audit réel en fait).

Le rapport doit être dans l'un de ces formats :

- PDF (préféablement)
- DOCX (et surtout PAS DOC)
- ODT

Format du nom de fichier : `td-eva-Nom1-Nom2.pdf`

Envoyez votre rapport à l'adresse allan.blanchard@cea.fr au plus tard le **21/03 minuit** avec comme objet :

[INSA CVL] EVA

Information importante

Si vous n'utilisez pas la VM toutes les commandes `frama-c-gui` sont à remplacer par `ivette`.

1 Recherche et ... test ?

Le fichier `search.c` contient une fonction qui recherche un caractère dans une chaîne. Ainsi qu'une fonction `main` pour lancer la fonction.

1. Lancez la commande :

```
frama-c-gui search.c -eva
```

Des erreurs sont-elles remontées ?

2. Actuellement, ce programme n'est qu'un test : nous ne cherchons des erreurs que pour une entrée particulière. Nous voudrions faire en sorte de vérifier que le programme est valide pour n'importe quelle entrée. Écrire une fonction généralisée de lancement pour la fonction `search`. Pour cela, utiliser la fonction `Frama_C_interval(min, max)`, disponible en incluant `__fc_builtin.h`. Considérez un tableau de taille fixe pour stocker la chaîne d'entrée.

```
frama-c-gui search.c -main=fonction -eva
```

Si la fonction contient des erreurs, corrigez-les et vérifiez que la preuve passe.

3. Le lanceur couvre-t-il le cas où l'élément n'est pas présent dans le tableau ? Expliquez pourquoi.

2 La fonction “recherche si”

Le fichier `find_if.c` contient une fonction qui recherche une valeur dans un tableau telle que le prédicat transmis en entrée est vrai.

1. Lancez la commande :

```
frama-c-gui find_if.c -eva
```

Des erreurs sont-elles remontées ?

2. La fonction `Frama_C_show_each` peut être utilisée pour afficher au cours de l’analyse les valeurs prises par une variable. Par exemple `Frama_C_show_each(i)`. Utilisez cette fonction pour afficher les valeurs prises par `length` pendant l’analyse. Que constate-t-on ?
3. L’option `-slevel` permet de conserver plus d’états en parallèle pendant l’analyse. Choisissez une valeur pour cette option et recommencez l’analyse. Justifiez ce choix de valeur. Des erreurs sont-elles remontées ?
4. Une nouvelle fois, cette fonction n’est guère plus qu’un test. Généralisez le lanceur de la fonction. Pour cela, nous pouvons utiliser la fonction builtin `Frama_C_interval(min, max)`. Conservez un tableau de taille 10 et la fonction transmise en entrée de `find_if`. Relancez l’analyse, une erreur est-elle remontée ? Est-elle réelle ?
5. Si vous avez répondu oui à la question précédente, proposez une correction pour la fonction et vérifiez qu’elle passe l’analyse.

3 Tableau

On considère le code présent dans le fichier `array.c`. Cette fonction est tirée des règles de codage du CERT, qui montre des faiblesses pouvant entraîner des failles de sécurité.

1. Lancez Frama-C avec la commande suivante :

```
frama-c-gui array.c -eva
```

Que constatez-vous ?

2. Proposez une correction du code pour éviter l’alarme d’Eva, et vérifiez que votre solution ne provoque plus d’alarme.
3. On modifie la fonction `main` comme ceci :

```
int main () {
    insert_in_table(SIZE_MAX, Frama_C_interval(INT_MIN, INT_MAX));
    insert_in_table(100, Frama_C_interval(INT_MIN, INT_MAX));
    insert_in_table(101, Frama_C_interval(INT_MIN, INT_MAX));
}
```

Lancez Frama-C sur ce nouveau programme et corrigez le cas échéant `insert_in_table`.

4. On ajoute un quatrième appel à `insert_in_table` dans `main`.

```
insert_in_table(101, Frama_C_interval(INT_MIN, INT_MAX));
```

De nouvelles erreurs sont-elles levées ? Pouvez-vous expliquer ce qui se passe ?

5. L’option `-slevel` ne cherche pas à conserver les états séparés lors des appels de fonction. Nous pouvons configurer cela à l’aide de l’option `-eva-split-return-function` qui permet de spécifier sur quelle valeur séparer les états. Relancez l’analyse avec les options (on sépare sur la valeur 0) :

```
-slevel 10 -val-split-return-function realloc:0,insert_in_table:0
```

6. S’il reste des erreurs dans `insert_in_table`, corrigez-les, et relancez l’analyse pour assurer qu’il n’y en a plus.

7. Lorsque l'on investigate un bug, il peut être pratique de se concentrer sur une pile d'appel en particulier. Dans la fonction d'insertion, sélectionnez la variable de position au moment de l'insertion puis sélectionnez la pile d'appel correspondant au dernier appel. Que signifie le code surligné en rouge ?

4 Rice compression

Le fichier `rice.c` contient un algorithme de compression, l'entête fournit dans `rice.h` est le suivant :

```
/* *****  
 * Supported binary formats  
 * *****/  
  
/* These formats have the same endianness as the machine on which  
   the (de)coder is running on */  
#define RICE_FMT_INT8    1 /* signed 8-bit integer */  
#define RICE_FMT_UINT8   2 /* unsigned 8-bit integer */  
#define RICE_FMT_INT16   3 /* signed 16-bit integer */  
#define RICE_FMT_UINT16  4 /* unsigned 16-bit integer */  
#define RICE_FMT_INT32   7 /* signed 32-bit integer */  
#define RICE_FMT_UINT32  8 /* unsigned 32-bit integer */  
  
/* *****  
 * Function prototypes  
 * *****/  
  
int Rice_Compress( void *in, void *out, unsigned int insize,  
                  int format );  
void Rice_Uncompress( void *in, void *out, unsigned int insize,  
                    unsigned int outsize, int format );
```

Écrire une fonction main permettant de vérifier avec Frama-C si la fonction de compression `Rice_Compress` peut présenter des erreurs à l'exécution si on l'appelle avec un buffer d'entrée de 64 unsigned char.

Appelez votre fichier `test_rice.c`, le squelette est le suivant :

```
#include <limits.h>  
#include "__fc_builtin.h"  
#include "rice.h"  
  
int main(void) {  
    unsigned char buffer_in[64];  
    ...  
}
```

Lancez la commande :

```
frama-c-gui -eva -eva-precision 4 rice.c test_rice.c
```

Une erreur est-elle levée ? Quelles sont les valeurs des variables mises en jeu ? Quelle supposition fait visiblement la fonction de compression à propos du buffer de sortie ?

Assurez-vous d'initialiser votre buffer de sortie à 0 et relancez l'analyse. Une nouvelle erreur est-elle levée ? Si oui, inspectez les valeurs mises en jeu. Allez voir, dans `rice.c` la documentation de la fonction. Corrigez votre test en conséquence. L'analyse doit maintenant passer.