TRUST IN SOFT

Garantie de déterminisme pour toutes les entrées pour les programmes C

Projet RAPID “ASECFLOD”, TRL 6

Pascal Cuoq

21 janvier 2022

1 Projet ASECFLOD

Le projet ASECFLOD (Analyses pour la Sécurité des Flots de Données) est un projet collaboratif entre la DGA et la société TrustInSoft, se déroulant d'avril 2019 à mars 2022, financé par le dispositif RAPID. Son objectif est d'étendre la plateforme TrustInSoft Analyzer, qui regroupe des outils d'analyse statique de code source de programmes en C et C++. Leur caractéristique commune est de reposer sur des méthodes formelles, en particulier l'Interprétation Abstraite, assurant que les résultats sont sémantiquement corrects, bien définis et exhaustifs.

Le premier objectif du projet est de permettre l'étude de la propagation d'information dans un programme. Cela consiste à calculer les flots de données globaux pendant l'analyse et à fournir des outils interactifs permettant l'exploration de ces informations pour aider les audits de code. Le second objectif est de faciliter l'analyse de logiciels communiquant avec des ressources matérielles, telles que des modules cryptographiques, via des opérations bas niveau (MMIO). Ces opérations rendaient l'analyse difficile, au point qu'il fallait modifier ces parties pour avoir une analyse pertinente sur le reste.

Dans ce contexte, il est apparu pendant la réalisation du projet que la pierre d'angle pour atteindre ce deuxième objectif était d'améliorer la précision de l'analyse de programmes C qui appliquent l'arithmétique entière ou bit-à-bit sur la représentation de pointeurs. Les travaux menés dans ce sens ont aussi conduit à une ré-expression plus rigoureuse des garanties apportées par l'analyse d'un programme C avec TrustInSoft Analyzer. Ces nouvelles garanties font partie de TrustInSoft Analyzer tel que fourni en 2022.

2 Comportements non spécifiés, comportements non définis

Le standard C laisse non défini (undefined) le comportement de certaines opérations. Cette formulation peut sembler anodine, mais en pratique "comportement non défini" est la catégorie des accès mémoire invalides qui interrompent brutalement l'exécution ("segmentation fault"), ou pire, laissent l'exécution continuer de manière erratique car l'accès à un pointeur invalide a modifié une autre variable qui se trouvait être là, et tout ceci de manière non reproductible d'une exécution à l'autre.

D'autres comportements non définis peuvent sembler sans danger. Un exemple qui prendra son importance dans cette présentation est `&a < &b`. Il est important d'éviter soigneusement aussi ces constructions, car les optimisations d'un compilateur moderne sont basées sur le fait que le programme compilé n'utilise pas de comportement non défini. Une illustration : le ticket 78420 sur le Bugzilla GCC.

Un type de constructions réellement plus anodines, mais occasionnellement surprenantes, est le comportement non spécifié (unspecified), qui à l'évaluation prend une valeur parmi un ensemble de valeurs possibles qui est clair suivant le contexte. Un exemple est `"foo"=="foo"`. Le compilateur est libre de partager les deux chaînes en mémoire (valeur `1`) ou de les représenter séparément (valeur `0`).

TrustInSoft Analyzer jusqu'en 2019, et son prédécesseur l'analyse de valeurs de Frama-C, interdisait les comportements non définis et aussi les comportements non spécifiés. Les comportements non spécifiés ne sont pas des erreurs graves et peuvent ne pas être des erreurs du tout, mais en les traitant comme des erreurs à éviter, ces analyseurs offraient presque la garantie que si un programme passe l'analyse, alors il est déterministe et ne peut produire qu'un seul résultat pour un vecteur d'entrée donné. Le projet ASECFLOD a permis de mettre le doigt sur deux lacunes liées à cette presque-garantie.

2.1 Les comportements non spécifiés ne sont pas tous des erreurs

En transformant les comportements non spécifiés en erreurs graves à absolument éviter, le TrustInSoft Analyzer de 2019 s'interdit d'analyser sans modifications les programmes qui s'appuient volontairement sur ce type de construction.

Un exemple récurrent a été l'analyse de fonctions de copie de mémoire de type `memmove`, qui doit fonctionner même si source et destination se recouvrent. Typiquement, avant analyse cette fonction est écrite :

```
void my_memmove(char *dest, char *src, size_t len) {
  if (dest <= src) { /* copy from the beginning */ ...
```

Si cette fonction est appliquée à des pointeurs vers des blocs mémoire différents, TrustInSoft Analyzer 2019 avertit à juste titre que `<=` est non défini dans ce cas, ce qui est en principe très dangereux.

Le correctif pragmatique dans cette situation est de s'appuyer sur la conversion de pointeur vers entier, que tous les compilateurs C courants visant des architectures ordinaires avec une vision à plat de la mémoire proposent, avec les garanties auxquelles les développeurs s'attendent :

```
void my_memmove(char *dest, char *src, size_t len) {
  if ((uintptr_t)dest <= (uintptr_t)src) { /* copy from the beginning */ ...
```

Cette précaution élimine le comportement non défini qui était présent, mais TrustInSoft Analyzer continuait de rejeter les appels à `my_memmove` qui passent en argument des blocs différents, car il considérait le comportement non spécifié qui reste après correctif comme une erreur de la même nature que le comportement non défini initial. C'est regrettable parce qu'il est loin d'être aussi grave : la version avec correctif pragmatique fonctionne et continuera de fonctionner, au contraire de la version dangereuse initiale. Si on souhaite implémenter une fonction de type `memmove` en C sans passer par une copie intermédiaire, le test utilisant des conversions vers `uintptr_t` est la seule solution raisonnable.

2.2 Le déterminisme n'est pas réellement garanti

Malgré ses efforts sur le diagnostic des comportements non spécifiés, TrustInSoft Analyzer 2019 ne garantit pas réellement qu'un programme analysé sans avertissement est déterministe. Des adresses (fixées en pratique par le compilateur, le linker et l'implémentation des fonctions d'allocation) peuvent être incorporées dans des calculs et rendre le résultat dépendant du layout en mémoire. Un exemple parmi d'autres est le suivant :

```
int programme(unsigned char c) {
  int x = 0, y = 0, t[3] = {3, 5, 7};
  if ((uintptr_t)&x % 17 == 0) x = 1; else y = 2;
  return c + t[x + y];
}
```

TrustInSoft Analyzer 2019 analyse ce programme sans avertissement, ce qui garantit que pour toutes les valeurs d'entrée possibles (variable `c`), il n'y a pas d'accès hors bornes au tableau `t` ni d'autre comportement non défini. TrustInSoft Analyzer 2019 garantit aussi, correctement, que pour toutes les entrées le résultat retourné est compris entre les bornes 5 et 262 (255 + 7). Par contre, ce que ne garantit pas le fait que le programme s'analyse sans avertissement, c'est que le résultat soit indépendant du layout, et en effet, il ne l'est pas : le résultat dépend de si l'adresse choisie par le compilateur pour la variable `x` est multiple de 17 ou pas, et cet aspect non-déterministe de l'exécution du programme, hors du contrôle du développeur ou de l'utilisateur du programme, n'est pas visible dans les résultats de l'analyse.

3 Garantie apportée par TrustInSoft Analyzer à partir de 2022

Grâce aux travaux réalisés, l'interpréteur abstrait au sein de TrustInSoft Analyzer marque maintenant sans omissions les calculs qui dépendent du layout. Pour prendre en compte le fait que ce non-déterminisme peut être volontaire et inévitable, ces marques ne sont pas traitées comme les erreurs graves que sont les comportements non définis. Il est donc possible d'une part d'analyser sans modification plus de programmes qui manipulent les représentations de certaines adresses. Et d'autre part, l'analyseur offre une vraie garantie sur le déterminisme des programmes analysés :

Quand un programme s'analyse sans avertissements, alors pour tout vecteur d'entrée, pour chaque calcul du programme marqué comme ne dépendant pas du layout, il existe une valeur unique que prend ce calcul pendant l'exécution, et ce pour tous les layouts résultant de choix internes du compilateur, du linker ou des fonctions d'allocation. Et cette valeur appartient à l'ensemble prédit par l'analyseur.