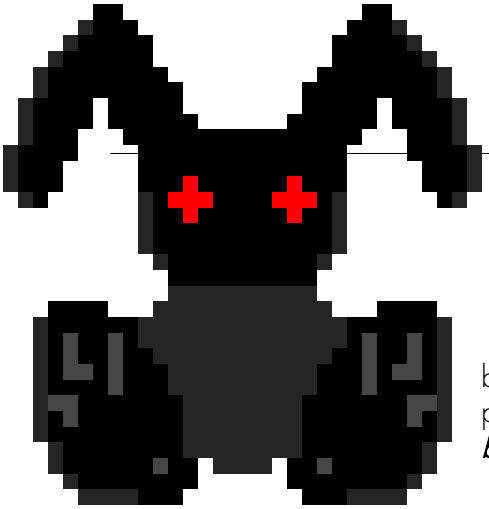

Procédure

Vous allez réaliser avec la **moitié** de votre promotion une bibliothèque, vous l'appellerez… **notre** bibliothèque. Chaque fonction sera préfixée par **our_**, symbolisant votre investissement collectif. *Cette bibliothèque sera sous licence GPL3.*

Les processus de programmation sécuritaire sont parti intégrante d'un aspect que l'on appelle… la qualité. Vous allez donc travailler cet aspect. Pour se faire, vous allez rejoindre avec la moitié de votre promotion un dépôt, tandis que l'autre moitié en rejoindra un autre. Les deux sont strictement identiques. Ils contiennent quelques dossiers et fichiers : dans **include**, vous trouverez des fichiers correspondant à des groupes de fonctions, dans **src**, les fonctions implémentées et dans **tests**, vous trouverez un dossier **src** contenant des programmes de test. Ce projet est soutenu par des **Makefile** assurant sa construction et la génération de rapport de couverture de test.

Ces dépôts contiennent une description dans un fichier **README** du travail à réaliser. Vous devrez pour chaque fonction à réaliser créer **un** ticket. Chaque fonction occupera **un** fichier pour son implémentation portant son nom, à l'exception du préfixe **our_**. Le test d'une fonction occupera **un** fichier portant son nom, de même toujours à l'exception du préfixe **our_**. Vous utiliserez les fichiers .h présent.

Chacun d'entre vous va prendre **un** ticket pour le compléter. Vous ne pouvez pas prendre un autre ticket tant que vous n'aurez pas complété ou abandonné celui-ci. Vous travaillerez dans une branche git **portant le mot « our » suivi numéro du ticket comme nom**.

Voici la procédure que vous *devriez* suivre :

1°) Vous allez rédiger un résumé du travail en anglais dans un ticket, crée pour l'occasion.
2°) Vous allez implémenter la fonction dans un fichier qui lui est propre.
- Vous programmerez en « programmation défensive » radicale. Vous provoquerez un arrêt brutal en cas de dysfonctionnement anormal (par exemple, **strlen** recevant **NULL**), cela à l'aide de la macro **efassert** implémentée dans **include/efassert.h**. Celle-ci provoque un arrêt si **NDEBUG** n'est pas établie, sinon exploite un code arbitraire passé en paramètre. Cette macro ne sera utilisée que dans les fonctions elles-même et non dans les tests où vous utiliserez **assert**.
- Vous établirez **errno** en cas de besoin.
- Vous n'aurez **ni fuite mémoire**, **ni accès incorrect** à la mémoire.

3°) Vous allez implémenter une fonction de **test** portant le nom de la fonction.
- Vous devrez avoir une **couverture de code de 100 %.**
- Vous testerez les cas **typiques**.
- Vous testerez les **extrema**.
- Vous testerez les **codes erreurs.**
- Vous testerez **errno**.

4°) Vous ferez une demande de fusion au dépôt, et attendrez la validation par trois de vos pairs suite à leur revue de code à la suite de quoi vous fusionnerez s'ils acceptent.
5°) Vous ferez vous même la revue de code d'autres tickets.
6°) Vous éliminerez le plus possible la répétition de code. Pour cela, vous commencerez par écrire des fonctions fondamentales afin de baser d'autres fonctions dessus.

7°) Vous placerez **__attribute__ ((deprecated))** à la suite des prototypes des fonctions dont vous estimez qu'elles ne peuvent être par nature totalement sécuritaire – en ce qu'elles impliquent une vigilance excessive de la part de son utilisateur ou utilisatrice.

Attaque

Vous disposez du dépôt de l'autre équipe, vous vous rappelez ?

Vous pouvez donc observer leur code et écrire des programmes de test pour eux également… Ces programmes seront dans votre propre dossier **cracks/src**. Chaque fichier devra pouvoir être compilé seul, en liant la bibliothèque de l'équipe opposée avec lui – ou la votre. Chaque fichier portera le nom de la fonction qu'elle souhaite attaquer, sans le préfixe **our_**.

Pour marquer un point, votre propre bibliothèque doit résister à votre programme de crack, tandis que celui-ci doit provoquer une erreur (fonctionnement invalide, erreur critique, ou même simplement relevée par valgrind) lorsque compilé avec celle de l'équipe adverse. *Un point est donné par fonction et par heure.*

La vérification que vos cracks provoquent des erreurs sur la bibliothèque adverse sera à la charge du corps pédagogique et s'effectuera sur la branche principale du dépôt. Si vous êtes attaqués avec succès, cela signifie que vous avez fusionné une bêtise avec la branche principale, cela malgré la couverture de test, malgré la revue de code, malgré toutes vos précautions… Celle-ci sont donc insuffisantes et **vous devez revoir votre procédure ou la respecter de manière plus efficace.**

Si vous pensez que votre adversaire a été **déloyal** (par exemple, en corrompant la mémoire avant d'exploiter vos fonctions pour les faire planter, abusant d'un détail d'implémentation de ses propres fonctions pour résister, en adjoignant des tests pour des fonctions non demandées, en ne respectant pas la politique de nommage des fichiers), n'hésitez pas à rapporter ce manque total d'honneur au corps pédagogique qui sanctionnera cet acte perfide d'un malus **important**.
N'hésitez donc pas à surveiller le dépôt de l'adversaire.

Lorsqu'une attaque est réussie, celle-ci est annoncée à l'équipe adverse afin qu'il puisse effectuer les réparations qui s'imposent.

La version de test de votre bibliothèque sera compilée avec la macro **NDEBUG**. C'est cette version qui sera également attaquée par l'équipe adverse. En somme **NDEBUG** devrait tout le temps être présente.

Pour finir, vos programmes dans **cracks/src** devront pouvoir compiler sans les fichiers .h du projet et doivent donc contenir les prototypes des fonctions qu'ils attaquent.