|  |
| --- |
| [Catégorie ] |

[Adresse société]

[Société]

Tél : [Téléphone société]

[Télécopie société]

[Titre ]

|  |
| --- |
| Draft / Final |

Ce rapport est généré à partir de la version [État ] du projet le [Date de publication].

|  |  |  |  |
| --- | --- | --- | --- |
| **SUIVI DES EVOLUTIONS** | | | |
| **Révision** | **Date** | **Auteur** | **Modification** |
| **--** |  |  | Première version |
| **-A** |  |  |  |
| **-B** |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **APPROBATION** | | | | |
|  | **Nom** | **Rôle** | **Date** | **Signature** |
| **Créé ou modifié par** |  |  |  |  |
| **Vérifié par** |  |  |  |  |
| **Approuvé par** |  |  |  |  |
| **Approuvé par le client si nécessaire** |  |  |  |  |

**Sommaire**

[1 Activité 1 - Cadrage et socle de sécurité 6](#_Toc73618134)

[1.1 Activité 1.a - Cadrer l’étude 6](#_Toc73618135)

[1.1.1 Données principales 6](#_Toc73618136)

[1.1.2 Acteurs 7](#_Toc73618137)

[1.1.3 RACI 8](#_Toc73618138)

[1.2 Activité 1.b (Biens primordiaux/support) 9](#_Toc73618139)

[1.2.1 Valeur métier 9](#_Toc73618140)

[1.2.2 Bien support 10](#_Toc73618141)

[1.2.3 Mission 11](#_Toc73618142)

[1.3 Activité 1.c (Événement redoutés) 12](#_Toc73618143)

[1.3.1 Echelles 12](#_Toc73618144)

[1.3.1.1 Liste des échelles 12](#_Toc73618145)

[1.3.1.2 Détails des Niveaux d’échelles 13](#_Toc73618146)

[1.3.1.2.1 Détails des niveaux de l’échelle ${echelle\_X} 13](#_Toc73618147)

[1.3.1.2.2 Détails des niveaux de l’échelle ${echelle\_Y} 14](#_Toc73618148)

[1.3.2 Événements redoutés 15](#_Toc73618149)

[1.4 Activité 1.d - Les socles de sécurité 16](#_Toc73618150)

[1.4.1 Socle de sécurité 16](#_Toc73618151)

[1.4.2 Règles 17](#_Toc73618152)

[1.4.2.1 Etat de prise en compte du socle XXX 17](#_Toc73618153)

**Liste des tableaux**

[Tableau 1 : Liste des acteurs 7](#_Toc73618154)

[Tableau 2: RACI 8](#_Toc73618155)

[Tableau 3 : Tableau des valeurs métiers 9](#_Toc73618156)

[Tableau 4 : Liste des biens supports 10](#_Toc73618157)

[Tableau 5 : Tableau des missions 11](#_Toc73618158)

[Tableau 6 : Tableau des échelles de gravité 12](#_Toc73618159)

[Tableau 7 : Détails des niveaux de l'échelle ${echelle\_X} 13](#_Toc73618160)

[Tableau 8 : Détails des niveaux de l'échelle ${echelle\_Y} 14](#_Toc73618161)

[Tableau 9 : Détails des événements redoutés 15](#_Toc73618162)

[Tableau 10 : Liste des socles de sécurité 16](#_Toc73618163)

[Tableau 11 : socle de sécurité XXX 17](#_Toc73618164)

**Liste des illustrations**

# Activité 1 - Cadrage et socle de sécurité

## Activité 1.a - Cadrer l’étude

### Données principales

* Nom du projet : CarlosCoffreFort
* Objectif à atteindre : Réussir à développer un coffre-fort de sécurité visant une évalaution EAL3
* Cadre Temporel :
* Atelier 1 : 18-05-2021
* Atelier 2 : 19-05-2021
* Atelier 3 : 20-05-2021
* Atelier 4 : 21-05-2021
* Atelier 5 : 22-05-2021
* Durée des cycles stratégiques et opérationnels :
* Durée des cycles stratégiques : 2 ans
* Durée des cycles opérationnels : 1an
* Niveau de confidentialité : Confidentiel
* Personne responsable d'accepter les risques résiduels au terme de l'étude : PINTO Carlos

### Acteurs

| **Nom** | **Prénom** | **Poste** |
| --- | --- | --- |
| PINTO | Carlos | Ingénieur sécurité |
| GHAZLI | Inel | Ingénieur sécurité |
| DREUILHET | Romain | Ingénieur sécurité |
| LEBON | Michael | Ingénieur sécurité |
| BOURGAIN | Maxime | Ingénieur sécurité |

Tableau 1 : Liste des acteurs

### RACI

| # | **1.a** | **1.b** | **1.c** | **1.d** | **2.a** | **2.b** | **2.c** | **3.a** | **3.b** | **3.c** | **4.a** | **4.b** | **5.a** | **5.b** | **5.c** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **PINTO Carlos** | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |
| **GHAZLI Inel** | I | I | I | I | I | I | I | I | I | I | I | I | I | I | I |
| **DREUILHET Romain** | I | I | I | I | I | I | I | I | I | I | I | I | I | I | I |
| **LEBON Michael** | I | I | I | I | I | I | I | I | I | I | I | I | I | I | I |

Tableau 2: RACI

## Activité 1.b (Biens primordiaux/support)

### Valeur métier

| **Valeur Métier** | **Nature** | **Description** |
| --- | --- | --- |
| Mot de passe à stocker | Information | Mot de passe que le coffre fort doit stocker |
| Mot de passe du coffre fort | Information | Mot de passe permettant de déverouiller le coffre fort de mot de passe. |
| Clé de vérification de signature | Information | La mise à jour est sécuritée et impose de disposer d'une clé publique de vérification de signature |

Tableau 3 : Tableau des valeurs métiers

### Bien support

| **Bien support** | **Description** |
| --- | --- |
| Memoire Flash | Mémoire Flash permettant de stocker les données non confidentielles |
| Mémoire RAM | Mémoire RAM de travail du processeur |

Tableau 4 : Liste des biens supports

### Mission

| **Nom de la mission** | **Description de la mission** | **Responsable** | **Valeur Métier** | **Responsable de la valeur métier** | **Bien Support** | **Responsable du bien support** |
| --- | --- | --- | --- | --- | --- | --- |
| Enregistrer des comptes | La mission rendue est de pouvoir stocker des mots de passe | Carlos | Mot de passe à stocker | Carlos | Memoire Flash | Carlos |
| Enregistrer des comptes | La mission rendue est de pouvoir stocker des mots de passe | Carlos | Mot de passe à stocker | Carlos | Mémoire RAM | Carlos |
| Limiter l'accès au coffre fort | L'accès au coffre-fort de mot de passe doit être limité aux personnes habilités | Carlos | Mot de passe du coffre fort | Carlos | Memoire Flash | Carlos |
| Limiter l'accès au coffre fort | L'accès au coffre-fort de mot de passe doit être limité aux personnes habilités | Carlos | Mot de passe du coffre fort | Carlos | Mémoire RAM | Carlos |
| Mise à jour sécurisé | Le coffre fort doit pouvoir être mis à jourde façon sécurisé | Carlos | Clé de vérification de signature | Carlos | Memoire Flash | Carlos |
| Démarrage sécurisé | Le coffre fort vérifie que son logiciel est authentique avant de s'executer | Carlos | Clé de vérification de signature | Carlos | Memoire Flash | Carlos |
| Mise à jour sécurisé | Le coffre fort doit pouvoir être mis à jourde façon sécurisé | Carlos | Clé de vérification de signature | Carlos | Mémoire RAM | Carlos |

Tableau 5 : Tableau des missions

## Activité 1.c (Événement redoutés)

### Echelles

#### Liste des échelles

| **Nom de l'échelle** | **Échelle de gravité** |
| --- | --- |
| EchelleDeGravité | 5 |

Tableau 6 : Tableau des échelles de gravité

#### Détails des Niveaux d’échelles

##### Détails des niveaux de l’échelle ${echelle\_X}

| **Valeur du niveau** | **Description du niveau** |
| --- | --- |
| 1 | Nulle |
| 2 | Très faible |
| 3 | Faible |
| 4 | Moyen |
| 5 | Elevée |

Tableau 7 : Détails des niveaux de l'échelle ${echelle\_X}

##### Détails des niveaux de l’échelle ${echelle\_Y}

${da\_niveau2}

Tableau 8 : Détails des niveaux de l'échelle ${echelle\_Y}

### Événements redoutés

${m\_evenement\_redoute}

Tableau 9 : Détails des événements redoutés

## Activité 1.d - Les socles de sécurité

### Socle de sécurité

| **Type de référentiel** | **Nom du référentiel** | **État d'application** | **Commentaire** |
| --- | --- | --- | --- |
| Référentiel de sécurité | Règles de codage | Non appliqué |  |

Tableau 10 : Liste des socles de sécurité

### Règles

#### Etat de prise en compte du socle XXX

| **ID de la règle** | **Titre de la règle** | **Description de la règle** | **État de la règle** | **Justification des écarts** | **Responsable** | **Date limite de la mise en application** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | RÈGLE — Application de conventions de codage claires et explicites | Des conventions de codage doivent être définies et documentées pour le logiciel à produire. Ces conventions doivent définir au minimum les points suivants : l’encodage des fichiers sources, la mise en page du code et l’indentation, les types standards à utiliser, le nommage (bibliothèques, fichiers, fonctions, types, variables, . . .), le format de la documentation. Ces conventions doivent être appliquées par chaque développeur. |  |  |  |  |
| 2 | RÈGLE — Seul le codage C conforme au standard est autorisé | Aucune violation des contraintes et de la syntaxe C telles que définies dans les standards C90 ou C99 n’est autorisée. |  |  |  |  |
| 3 | RECOMMANDATION — Maîtrise des actions opérées à la compilation. | Le développeur doit connaître et documenter les actions associées aux options activées du compilateur y compris en terme d’optimisation de code |  |  |  |  |
| 4 | RÈGLE — Définir précisément les options de compilation | Les options utilisées pour la compilation doivent être précisément définies pour l’ensemble des sources d’un logiciel. Ces options doivent notamment fixer précisément : - la version du standard C utilisée (par exemple C99 ou encore C90) ; - le nom et la version du compilateur utilisé ; |  |  |  |  |
| 5 | RÈGLE — Utiliser des options de durcissement | L’utilisation d’options de durcissement est obligatoire que ce soit pour imposer la génération d’exécutables relocalisables, une randomization d’adresses efficace ou la protection contre le dépassement de pile entre autres. |  |  |  |  |
| 6 | BONNE PRATIQUE — Utiliser des générateurs de projets pour la compilation. | Vide |  |  |  |  |
| 7 | RÈGLE — Compiler le code sans erreur ni avertissement en activant des options de compilation exigent | Activer le niveau d’avertissement et d’erreur le plus élevé du compilateur et de l’éditeur de liens afin de s’assurer de l’absence de problèmes potentiels liés à l’utilisation incorrecte du langage de programmation et traiter tous les avertissements et toutes les erreurs signalés par le compilateur et l’éditeur de liens pour les éliminer. |  |  |  |  |
| 8 | RECOMMANDATION — Utiliser les options des compilations les plus exigentes | Si une option élevée d’un compilateur n’apparaît pas pertinente pour un développement donné, une justification sera fournie pour expliquer ce choix. |  |  |  |  |
| 9 | RÈGLE — Tout code mis en production doit être compilé en mode release | La compilation en mode release est obligatoire pour une mise en production. |  |  |  |  |
| 10 | RECOMMANDATION — Prêter une attention particulière aux modes debug et release lors de la compilation | L’utilisation des modes debug et release à la compilation doit se faire en toute connaissance des modifications induites en terme de gestion de mémoire et d’optimisation de code. Les différences entre ces deux modes doivent être documentées de manière exhaustive. |  |  |  |  |
| 11 | RECOMMANDATION — Limiter et justifier les inclusions de fichier d'en-tête dans un autre fichier d'en | Vide |  |  |  |  |
| 12 | RECOMMANDATION — Limiter et justifier les inclusions de fichier d'en-tête dans un autre fichier d'en | Vide |  |  |  |  |
| 13 | RÈGLE — Utiliser des macros de garde d'inclusion multiple d'un fichier | Une macro de garde contre l’inclusion multiple d’un fichier doit être utilisée afin d’empêcher que le contenu d’un fichier d’en-tête soit inclus plus d’une fois : // début du fichier d'en -tête #ifndef HEADER\_H #define HEADER\_H /\* contenu du fichier \*/ #endif //fin du fichier d'en -tête |  |  |  |  |
| 14 | RÈGLE — Les inclusions de fichiers d'en-tête sont groupées en début de fichier | Toutes les inclusions de fichiers d’en-tête doivent être regroupées au début du fichier ou juste après des commentaires ou les directives de preprocessing, mais systématiquement avant la définition de variables globales ou de fonctions. |  |  |  |  |
| 15 | RECOMMANDATION — Les inclusions de fichiers d'en-tête systèmes sont effectuées avant les inclusions | Vide |  |  |  |  |
| 16 | BONNE PRATIQUE — Utiliser l'ordre alphabétique dans l'inclusion de chaque type de fichiers d'en-tête | Pour éviter les redondances dans les inclusions de fichiers d’en-tête systèmes ou utilisateur, le développeur peut les ordonner par ordre alphabétique ce qui permet d’avoir un ordre d’inclusion déterministe et de faciliter la revue de code. |  |  |  |  |
| 17 | RÈGLE — Ne pas inclure un fichier source dans un autre fichier source | Seule l’inclusion de fichiers d’en-tête est autorisée dans un fichier source. |  |  |  |  |
| 18 | RÈGLE — Les chemins des fichiers doivent être portables et la casse doit être respectée | Les chemins de fichiers, que ce soit pour une directive d’inclusion #include ou non, doivent être portables tout en respectant la casse des répertoires. |  |  |  |  |
| 19 | RÈGLE — Le nom d'un fichier d'en-tête ne doit pas contenir certains caractères ou séquences de carac | Le nom d’un fichier d’en-tête doit être exempt des caractères et des séquences de caractères suivants : « ', ", \, /\* et // ». |  |  |  |  |
| 20 | RECOMMANDATION — Les blocs préprocesseurs doivent être commentés | Les directives des blocs préprocesseurs doivent être commentées afin d’expliciter le cas traité et pour les directives « intermédiaires » et « fermantes », celles-ci doivent aussi être associées à la directive « ouvrante » correspondante par le biais d’un commentaire. |  |  |  |  |
| 21 | BONNE PRATIQUE — La double négation dans l'expression des conditions des blocs préprocesseurs doit ê | Vide |  |  |  |  |
| 22 | RÈGLE — Définition d'un bloc préprocesseur dans un seul et même fichier | Pour un bloc préprocesseur, toutes les directives associées doivent se trouver dans le même fichier. |  |  |  |  |
| 23 | RECOMMANDATION — Les expressions de contrôle des directives de preprocessing doivent être bien formé | Les expressions de contrôle doivent être évaluées uniquement à 0 ou 1 et doivent utiliser uniquement des identifiants définis (via #define). |  |  |  |  |
| 24 | RÈGLE — Ne pas utiliser dans une même expression plus d'un des opérateurs de preprocessing # et ## | Vide |  |  |  |  |
| 25 | RÈGLE — Utiliser les opérateurs de preprocessing # et ## en maîtrisant leur expansion | Vide |  |  |  |  |
| 26 | RÈGLE — Les macros doivent être nommées de façon spécifique | Pour différencier aisément les macros des fonctions et ne pas utiliser un nom réservé d’une autre macro C, les macros préprocesseurs doivent être en capitales et les mots composant le nom séparés par le caractère souligné « \_ » mais sans les faire débuter par le caractère souligné car il s’agit d’une convention pour les noms réservés du langage C. |  |  |  |  |
| 27 | RÈGLE — Ne pas terminer une macro par un point-virgule | Le point-virgule final doit être omis à la fin de la définition d’une macro. |  |  |  |  |
| 28 | Le point-virgule final doit être omis à la fin de la définition d’une macro. | Vide |  |  |  |  |
| 29 | RÈGLE — L'expansion d'une macro définie par le développeur ne doit pas créer de fonction | Vide |  |  |  |  |
| 30 | RÈGLE — Les macros contenant plusieurs instructions doivent utiliser une boucle do { ... } while(0) | Vide |  |  |  |  |
| 31 | RÈGLE — Parenthèses obligatoires autour des paramètres utilisés dans le corps d'une macro | Les paramètres d’une macro doivent systématiquement être entourés de parenthèses lors de leur utilisation, afin de préserver l’ordre souhaité d’évaluation des expressions. |  |  |  |  |
| 32 | RECOMMANDATION — Il faut éviter les arguments d'une macro réalisant une opération | Vide |  |  |  |  |
| 33 | RÈGLE — Les arguments d'une macro ne doivent pas contenir d'effets de bord. | Des arguments de macro avec des effets de bord peuvent entraîner des évaluations multiples non désirées. |  |  |  |  |
| 34 | RÈGLE — Ne pas utiliser de directives de preprocessing en arguments d'une macro | Vide |  |  |  |  |
| 35 | RÈGLE — La directive #undef ne doit pas être utilisée | Vide |  |  |  |  |
| 36 | RÈGLE — Ne pas utiliser de trigraphes | Vide |  |  |  |  |
| 37 | RECOMMANDATION — Les points d'interrogation ne doivent pas être utilisés de façon successive | Cette règle s’applique pour toute partie du code mais aussi pour les commentaires. |  |  |  |  |
| 38 | RECOMMANDATION — Seules les déclarations multiples de variables simples de même type sont autorisées | Vide |  |  |  |  |
| 39 | RÈGLE — Ne pas faire de déclaration multiple de variables associée à une initialisation. | Les initialisations associées (i.e. consécutives et dans une même instruction) à une déclaration multiple sont interdites. |  |  |  |  |
| 40 | RECOMMANDATION — Regrouper les déclarations de variables en début du bloc dans lequel elles sont uti | Pour des questions de lisibilité et pour éviter les redéfinitions, les déclarations de variables sont regroupées en début de fichier, fonction ou bloc d’instructions selon leur portée. |  |  |  |  |
| 41 | RÈGLE — Ne pas utiliser des valeurs en dur | Les valeurs utilisées dans le code doivent être déclarées comme des constantes. |  |  |  |  |
| 42 | BONNE PRATIQUE — Centraliser la déclaration des constantes en début de fichier | Pour faciliter la lecture, les constantes sont déclarées ensemble en début du fichier. |  |  |  |  |
| 43 | RÈGLE — Déclarer les constantes en capitales | Vide |  |  |  |  |
| 44 | RÈGLE — Les constantes sans contrôle de type sont déclarées avec la macro préprocesseur de définitio | Vide |  |  |  |  |
| 45 | RÈGLE — Les constantes avec un contrôle de type explicite doivent être déclarées avec le mot clé con | Vide |  |  |  |  |
| 46 | RÈGLE — Les valeurs constantes doivent être associées à un suffixe dépendant du type | Pour éviter toute mauvaise interprétation, les valeurs constantes doivent utiliser un suffixe selon leurs types : il faut utiliser le suffixe U pour toutes les constantes de type entier non signé ; pour indiquer une constante de type long (ou long long pour C99), il faut utiliser le suffixe L (resp. LL) et non l (resp. ll) afin d’éviter toute ambiguïté avec le chiffre 1 ; les valeurs flottantes sont par défaut considérées comme double, il faut utiliser le suffixe f pour le type float (resp. d pour le type double). |  |  |  |  |
| 47 | RÈGLE — La taille du type associé à une expression constante doit être suffisante pour la contenir | Il faut s’assurer que les valeurs ou expressions constantes utilisées ne dépassent pas du type qui leur est associé. |  |  |  |  |
| 48 | RECOMMANDATION — Proscrire les constantes en octal | Ne pas utiliser de constante ni de séquence d’échappement en octal. |  |  |  |  |
| 49 | RÈGLE — Limiter les variables globales au strict nécessaire | Limiter l’usage des variables globales et préférer les paramètres de fonctions pour propager une structure de données au travers d’une application. |  |  |  |  |
| 50 | RÈGLE — Utilisation systématique du modificateur de déclaration static | Utiliser le mot clef static pour toutes les fonctions et variables globales qui ne sont pas utilisées à l’extérieur du fichier source dans lequel elles sont définies. |  |  |  |  |
| 51 | RÈGLE — Seules les variables modifiables en dehors de l'implémentation doivent être déclarées volati | Seules les variables associées à des ports entrée/sortie ou des fonctions d’interruption asynchrone doivent être déclarées comme volatile pour empêcher toute optimisation ou réorganisation à la compilation. |  |  |  |  |
| 52 | RÈGLE — Seuls des pointeurs spécifiés volatile peuvent accéder à des variables volatile | Vide |  |  |  |  |
| 53 | RÈGLE — Aucune omission de type n'est acceptée lors de la déclaration d'une variable | Toutes les variables utilisées doivent avoir été préalablement déclarées de façon explicite. |  |  |  |  |
| 54 | RECOMMANDATION — Limiter l'utilisation des compound literals | Du fait du risque de mauvaise manipulation des compound literals, leur utilisation doit être limitée, documentée et une attention particulière doit être donnée à leur portée. |  |  |  |  |
| 55 | RÈGLE — Ne pas mélanger des constantes explicites et implicites dans une énumération | Il faut soit expliciter toutes les constantes d’une énumération avec une valeur unique soit n’en expliciter aucune. |  |  |  |  |
| 56 | RÈGLE — Ne pas utiliser des énumérations anonymes | Vide |  |  |  |  |
| 57 | RECOMMANDATION — Les variables doivent être initialisées à la déclaration ou immédiatement après | Toutes les variables doivent être systématiquement initialisées à leur déclaration ou immédiatement après dans le cas de déclarations multiples. |  |  |  |  |
| 58 | RÈGLE — Ne pas mélanger les différents types d'initialisation pour les variables structurées | Pour l’initialisation d’une variable structurée, un seul et unique type d’initialisation doit être choisi et utilisé. |  |  |  |  |
| 59 | RÈGLE — Les variables structurées ne doivent pas être initialisées sans expliciter la valeur d'initi | Les variables non scalaires doivent être initialisées explicitement : chaque élément doit être initialisé en étant clairement identifié sans valeur d’initialisation superflue ou la notation ={0} ; peut être utilisée à la déclaration. Enfin les tailles des tableaux doivent être explicitées à l’initialisation. |  |  |  |  |
| 60 | RECOMMANDATION — Chaque déclaration publique (non static) doit être utilisée | Toutes les déclarations publiques (i.e. non static) doivent être utilisées, qu’il s’agissede variables, fonctions, labels ou autres. |  |  |  |  |
| 61 | RÈGLE — Utiliser des variables pour les données sensibles distinctes des variables pour les données | Vide |  |  |  |  |
| 62 | RÈGLE — Utiliser des variables pour les données sensibles et protégées en confidentialité et/ou inté | Vide |  |  |  |  |
| 63 | RÈGLE — Ne jamais coder en dur une donnée sensible. | Vide |  |  |  |  |
| 64 | RECOMMANDATION — Seuls des types d'entiers dont la taille et le signe sont explicites doivent être u | Vide |  |  |  |  |
| 65 | RÈGLE — Seuls les types signed char et unsigned char doivent être utilisés. | Vide |  |  |  |  |
| 66 | RECOMMANDATION — Ne pas redéfinir des alias de types | Vide |  |  |  |  |
| 67 | RÈGLE — Compréhension fine et précise des règles de conversions | Le développeur se doit de connaître et comprendre toutes les règles de conversionimplicites des types entiers. |  |  |  |  |
| 68 | RÈGLE — Conversions explicites entre des types signés et non signés | Proscrire les conversions implicites de types. Utiliser des conversions explicites notamment entre type signé et type non signé. |  |  |  |  |
| 69 | RECOMMANDATION — Ne pas utiliser de transtypage de pointeurs sur des types structurés différents | Vide |  |  |  |  |
| 70 | RÈGLE — L'accès aux éléments d'un tableau se fera toujours en désignant en premier attribut le table | L’accès au ième élément d’un tableau s’écrira toujours avec le nom du tableau en premier suivi de l’indice de la case à atteindre. |  |  |  |  |
| 71 | RECOMMANDATION — L'accès aux éléments d'un tableau doit se faire en utilisant les crochets | Dans le cas d’une variable de type tableau, la notation dédiée (via les crochets) doit être utilisée pour éviter toute ambiguïté. |  |  |  |  |
| 72 | RÈGLE — Ne pas utiliser de VLA | Vide |  |  |  |  |
| 73 | RECOMMANDATION — Ne pas utiliser de taille implicite pour les tableaux | Afin de s’assurer que les accès tableaux sont bien valides, la taille de ceux-ci doit être explicitée. |  |  |  |  |
| 74 | RÈGLE — Utiliser des entiers non signés pour les tailles de tableaux | Vide |  |  |  |  |
| 75 | RÈGLE — Ne pas accèder à un élément de tableau sans vérifier la validité de l'indice utilisé | La validité des indices de tableau utilisés doit être vérifié de façon systématique : un indice de tableau est valide s’il est supérieur ou égal à zéro et strictement inférieur à la taille déclarée du tableau. Dans le cas d’un tableau de caractères, le caractère de fin de chaine ’\0’ doit être pris en compte. |  |  |  |  |
| 76 | RÈGLE — Un pointeur NULL ne doit pas être déréférencé | Avant de déréférencer un pointeur, le développeur doit s’assurer que celui-ci n’est pas NULL. |  |  |  |  |
| 77 | RÈGLE — Un pointeur doit être affecté à NULL après désallocation | Un pointeur doit être systématiquement affecté à NULL suite à la désallocation de la mémoire qu’il pointe. |  |  |  |  |
| 78 | RÈGLE — Ne pas utiliser le qualificateur de pointeur restrict | Le qualificateur restrict ne doit pas être utilisé directement par le développeur. Seule l’utilisation indirecte i.e. via l’appel de fonctions de la bibliothèque standard est tolérée mais le développeur devra s’assurer qu’aucun comportement indéfini résultera de l’utilisation de telles fonctions. |  |  |  |  |
| 79 | RECOMMANDATION — Le nombre de niveau d'indirections de pointeur doit être limité à deux | Le nombre d’indirections pour un pointeur ne doit pas dépasser deux niveaux. |  |  |  |  |
| 80 | RECOMMANDATION — Préférer l'utilisation de l'opérateur d'indirection -> | L’opérateur d’indirection -> doit être utilisé pour atteindre les champs d’une structure par l’intermédiaire d’un pointeur. |  |  |  |  |
| 81 | RÈGLE — Seul l'incrément ou le décrément de pointeurs de tableaux est autorisé | L’incrément ou le décrément de pointeurs ne doit être utilisé que sur des pointeurs représentant un tableau ou un élément d’un tableau. |  |  |  |  |
| 82 | RÈGLE — Aucune arithmétique sur les pointeurs void\* n'est autorisée | Il faut proscrire l’utilisation de toute arithmétique sur des pointeurs de type void\*. |  |  |  |  |
| 83 | RECOMMANDATION — Arithmétique des pointeurs sur tableaux contrôlée | L’arithmétique sur des pointeurs représentant un tableau ou un élément d’un tableau doit être faite en s’assurant que le pointeur résultant pointera toujours sur un élément du même tableau. |  |  |  |  |
| 84 | RÈGLE — Soustraction et comparaison entre pointeurs d'un même tableau uniquement | Seules les soustractions et comparaisons de pointeurs sur un même tableau sont autorisés. |  |  |  |  |
| 85 | RECOMMANDATION — Il ne faut pas affecter directement une adresse fixe à un pointeur. | Vide |  |  |  |  |
| 86 | RÈGLE — Une structure doit être utilisée pour regrouper les données représentant une même entité | Les données liées doivent être regroupées au sein d’une structure. |  |  |  |  |
| 87 | RÈGLE — Ne pas calculer la taille d'une structure comme la somme de la taille de ses champs | Du fait du padding, la taille d’une structure ne doit pas être supposée comme la somme de la taille de ses champs. |  |  |  |  |
| 88 | RÈGLE — Tout bitfield doit obligatoirement être déclaré explicitement comme non signé | Vide |  |  |  |  |
| 89 | RÈGLE — Ne pas faire d'hypothèse sur la représentation interne de structures avec des bitfields | Vide |  |  |  |  |
| 90 | RÈGLE — Ne pas utiliser les FAM | Vide |  |  |  |  |
| 91 | RECOMMANDATION — Ne pas utiliser les unions | L’utilisation du même espace mémoire pour plusieurs données de natures différentes n’est pas autorisée. |  |  |  |  |
| 92 | RÈGLE — Supprimer tous les débordements de valeurs possibles pour des entiers signés. | Vide |  |  |  |  |
| 93 | RECOMMANDATION — Détecter tous les wraps possibles de valeurs pour les entiers non signés. | vide |  |  |  |  |
| 94 | RÈGLE — Détecter et supprimer toute potentielle division par zéro | Cette vérification doit être systématique pour tout calcul de division ou de reste de division. |  |  |  |  |
| 95 | RECOMMANDATION — Les opérations arithmétiques doivent être écrites en favorisant leur lisibilité | Il faut utiliser des opérations arithmétiques le plus explicites possibles (naturelles) et dans la logique du programme. |  |  |  |  |
| 96 | RECOMMANDATION — Les opérateurs logiques ne doivent pas être appliqués avec des opérandes signés | Vide |  |  |  |  |
| 97 | RÈGLE — Explicitation de l'ordre d'évaluation des calculs par utilisation de parenthèses | Malgré la priorité des opérateurs, pour éviter toute ambiguïté, les expressions seront entourées de parenthèses pour rendre plus explicite l’ordre d’évaluation d’un calcul. |  |  |  |  |
| 98 | RECOMMANDATION — Eviter les expressions de comparaison ou d'égalité multiple | Vide |  |  |  |  |
| 99 | RÈGLE — Toujours utiliser les parenthèses dans les expressions de comparaison ou d'égalité multiple | Les expressions booléennes de comparaison ou d’égalité contenant au moins 2 opérateurs relationnels sont interdites sans parenthèse. |  |  |  |  |
| 100 | RÈGLE — Parenthèses autour des éléments d'une expression booléenne | Il est nécessaire de toujours mettre entre parenthèses les différents éléments d’une expression booléenne, afin qu’il n’y ait aucune ambiguïté dans l’ordre d’évaluation. |  |  |  |  |
| 101 | RÈGLE — Comparaison implicite avec 0 interdite | Toutes les expressions booléennes doivent utiliser des opérateurs de comparaison. Aucun test implicite avec une valeur égale à 0 ou différente de 0 ne doit être effectué. |  |  |  |  |
| 102 | RECOMMANDATION — Utilisation du type bool en C99 | En C99, le type bool (ou \_Bool) doit être utilisé pour les variables à valeurs booléennes. |  |  |  |  |
| 103 | RÈGLE — Pas d'opérateur bit-à-bit sur un opérande de type booléen ou assimilé | Vide |  |  |  |  |
| 104 | BONNE PRATIQUE — Ne pas utiliser la valeur retournée lors d'une affectation | Vide |  |  |  |  |
| 105 | RÈGLE — Affectation interdite dans une expression booléenne | Une affectation ne doit pas être effectuée dans une expression booléenne quelle qu’elle soit. Une affectation doit être effectuée dans une instruction indépendante. |  |  |  |  |
| 106 | BONNE PRATIQUE — Comparaison avec opérande constant à gauche | Quand une comparaison fait intervenir un opérande constant celui-ci sera de préférence mis comme opérande gauche pour éviter une affectation non intentionnelle. |  |  |  |  |
| 107 | RÈGLE — Affectation multiple de variables interdite | L’affectation multiple de variables n’est pas autorisée. |  |  |  |  |
| 108 | RÈGLE — Une seule instruction par ligne de code | Vide |  |  |  |  |
| 108 | BONNE PRATIQUE — Éviter les constantes flottantes | Ne pas utiliser de constantes numériques flottantes pour éviter les pertes de précision et autres phénomènes liés aux nombres flottants. Si cela ne peut être évité, la représentativité de la valeur flottante en question doit être vérifiée. |  |  |  |  |
| 110 | RECOMMANDATION — Limiter l'utilisation des nombres flottants au strict nécessaire | Il faut limiter l’utilisation des nombres flottants. |  |  |  |  |
| 111 | RÈGLE — Pas de compteur de boucle de type flottant | Les compteurs de boucle doivent être uniquement de type entier, avec la vérification de non débordement de type des valeurs des compteurs. |  |  |  |  |
| 112 | RÈGLE — Ne pas utiliser de nombres flottants pour des comparaisons d'égalité ou d'inégalité | Vide |  |  |  |  |
| 113 | RECOMMANDATION — Non utilisation des nombres complexes | Les nombres complexes introduits depuis C99 ne doivent pas être utilisés. |  |  |  |  |
| 114 | RÈGLE — Utilisation systématique des accolades pour les conditionnelles et les boucles | Ne jamais omettre les accolades pour délimiter un bloc d’instructions. Les accolades doivent être écrites pour délimiter un bloc d’instructions après les boucles (for, while, do) et les conditionnelles (if, else). |  |  |  |  |
| 115 | RÈGLE — Définition systématique d'un cas par défaut dans les switch | Un switch-case doit toujours contenir un cas default placé en dernier. |  |  |  |  |
| 116 | RECOMMANDATION — Utilisation de break dans chaque cas des instructions switch | Un switch-case doit par défaut toujours contenir un break pour chaque cas. L’absence de break pour éviter de dupliquer du code est tolérée mais doit être explicitée dans un commentaire. |  |  |  |  |
| 117 | RECOMMANDATION — Pas d'imbrication de structure de contrôle dans un switch-case | Même si le C l’autorise, l’imbrication de structures de contrôle à l’intérieur d’un switch est à éviter. |  |  |  |  |
| 118 | RÈGLE — Ne pas introduire d'instructions avant le premier label d'un switch-case | Vide |  |  |  |  |
| 119 | RÈGLE — Bonne construction des boucles for | Chaque élément d’une boucle for doit être complété et contenir exactement une seule instruction. Ainsi une boucle for doit contenir une initialisation de son compteur, une condition d’arrêt portant sur son compteur, et une incrémentation ou décrémentation du compteur de boucle. |  |  |  |  |
| 120 | RÈGLE — Modification d'un compteur d'une boucle for interdite dans le corps de la boucle | Le compteur d’une boucle for ne doit pas être modifié à l’intérieur du corps de la boucle for. |  |  |  |  |
| 121 | RÈGLE — Non utilisation de goto arrière (backward goto) | Proscrire, au sein d’une fonction, l’utilisation d’instructions goto renvoyant vers un label qui est placé avant cette instruction goto. |  |  |  |  |
| 122 | RECOMMANDATION — Utilisation limitée du saut avant (forward goto) | L’utilisation d’un forward goto est tolérée uniquement dans les cas où elle permet : de limiter significativement le nombre de points de sortie de la fonction ; de rendre le code beaucoup plus lisible. Le ou les labels référencés par les instructions goto doivent tous être situés en fin de fonction. |  |  |  |  |
| 123 | RÈGLE — Toute fonction (non static) définie doit possèder une déclaration/ prototype de fonction | Vide |  |  |  |  |
| 124 | RÈGLE — Le prototype de déclaration d'une fonction doit concorder avec sa définition | Les types des paramètres utilisés pour la définition et la déclaration d’une fonction doivent être les mêmes. |  |  |  |  |
| 125 | RÈGLE — Toute fonction doit être associée à un type de retour et à une liste de paramètres explicite | Chaque fonction est définie explicitement avec un type de retour. Les fonctions sans valeur de retour doivent être déclarées avec un paramètre du type void. De la même façon, une fonction sans paramètre devra être définie et déclarée avec void en argument. |  |  |  |  |
| 126 | RECOMMANDATION — Documentation des fonctions | Tous les fonctions doivent être documentées. Cela comprend : n une description de la fonction et du traitement effectué ; la documentation de chaque paramètre, le sens du paramètre (en entrée, en sortie, en entrée et en sortie) et les éventuelles conditions existant sur celui-ci ; les valeurs de retour possibles doivent être décrites. |  |  |  |  |
| 127 | RECOMMANDATION — Préciser les conditions d'appel pour chaque fonction | Vide |  |  |  |  |
| 128 | RÈGLE — La validité de tous les paramètres d'une fonction doit systématiquement être remise en cause | Cela inclut : la validité des adresses pour les paramètres de type pointeur doit être vérifiée (non , alignement des adresses conforme...) ; l’appartenance des paramètres à leur domaine doit être vérifiée. Cela s’applique aux fonctions définies par le développeur (cf. section 12.2) mais aussi aux fonctions de la bibliothèque standard. |  |  |  |  |
| 129 | RÈGLE — Les paramètres de fonction de type pointeur pour lesquels la zone mémoire pointée n'est pas | Marquer const tous les paramètres de type pointeur d’une fonction qui pointent vers une zone mémoire qui ne doit pas être modifiée dans le corps de celle-ci. Le qualificateur const doit s’appliquer à l’objet pointé. |  |  |  |  |
| 130 | RÈGLE — Les fonctions inline doivent être déclarées comme static | Pour éviter un comportement non défini une fonction inline est systématiquement static. |  |  |  |  |
| 131 | RÈGLE — Interdiction de redéfinir les fonctions ou macros de la bibliothèque standard ou d'une autre | Les identifiants, macros ou noms de fonctions faisant partie de la bibliothèque standard ou d’une autre bibliothèque utilisée ne doivent pas être redéfinis. |  |  |  |  |
| 132 | RÈGLE — La valeur de retour d'une fonction doit toujours être testée | Lorsqu’une fonction retourne une valeur, la valeur retournée doit être systématiquement testée. |  |  |  |  |
| 133 | RÈGLE — Retour implicite interdit pour les fonctions de type non void | Tous les chemins d’une fonction non void doivent retourner une valeur explicitement. |  |  |  |  |
| 134 | RÈGLE — Les structures doivent être passées par référence à une fonction | Il ne faut pas passer de paramètres de type structure par copie lors de l’appel d’une fonction. |  |  |  |  |
| 135 | RECOMMANDATION — Passage d'un tableau en paramètre d'une fonction | Il existe plusieurs façons de passer un tableau en paramètre d’une fonction. Lors du passage par pointeur, il faut préciser dans la documentation de la fonction que le paramètre correspond à un tableau et également utiliser la notation dédiée aux tableaux. |  |  |  |  |
| 136 | RECOMMANDATION — Utilisation obligatoire dans une fonction de tous ses paramètres | Tous les paramètres présents dans le prototype de la fonction doivent être utilisés dans son implémentation. |  |  |  |  |
| 137 | BONNE PRATIQUE — Utiliser les options de compilation -Wformat=2 et -Wformat-security dès qu'une fonc | Plus de détails sur ces options sont données en annexe B. |  |  |  |  |
| 138 | RÈGLE — Ne pas appeler de fonctions variadiques avec NULL en argument | Vide |  |  |  |  |
| 139 | RÈGLE — Usage de la virgule interdit pour le séquencement d'instructions | La virgule n’est pas autorisée dans le cadre du séquencement des instructions decode. |  |  |  |  |
| 140 | RECOMMANDATION — Les opérateurs pré-fixes ++ et -- ne doivent pas être utilisés | Les opérateurs de pré-incrémentation et pré-decrémentation ne seront pas utilisés. |  |  |  |  |
| 141 | RECOMMANDATION — Pas d'utilisation combinée des opérateurs postfixes avec d'autres opérateurs | Les opérateurs de post-incrémentation et de post-décrémentation ne doivent pas être mixés avec d’autres opérateurs. |  |  |  |  |
| 142 | RECOMMANDATION — Éviter l'utilisation d'opérateurs d'affectation combinés | Vide |  |  |  |  |
| 143 | RÈGLE — Non utilisation imbriquée de l'opérateur ternaire ?: | L’imbrication d’opérateurs ternaires ?: est interdite. |  |  |  |  |
| 144 | RÈGLE — Bonne construction des expressions avec l'opérateur ternaire ?: | Les expressions résultantes de l’opérateur ternaire ?: doivent être exactement de même type pour éviter tout transtypage. |  |  |  |  |
| 145 | RÈGLE — Allouer dynamiquement un espace mémoire dont la taille est suffisante pour l'objet alloué | Pour un pointeur ptr, on préférera utiliser ptr=malloc(sizeof(\*ptr)); quand cela est possible. |  |  |  |  |
| 146 | RÈGLE — Libérer la mémoire allouée dynamiquement au plus tôt | Tout espace mémoire alloué dynamiquement doit être libéré quand celui-ci n’est plus utile. |  |  |  |  |
| 147 | RÈGLE — Les zones mémoires sensibles doivent être mises à zéro avant d'être libérées. | Vide |  |  |  |  |
| 148 | RÈGLE — Ne pas libérer de mémoire non allouée dynamiquement | Vide |  |  |  |  |
| 149 | RÈGLE — Ne pas modifier l'allocation dynamique via realloc | Vide |  |  |  |  |
| 150 | RÈGLE — Bonne utilisation de l'opérateur sizeof | Une expression contenue dans un sizeof ne doit pas : contenir l’opérateur « = » car l’expression ne sera pas évaluée ; contenir de déréférencement de pointeur ; être appliqué sur un pointeur représentant un tableau. |  |  |  |  |
| 151 | RÈGLE — Vérification obligatoire du succès d'une allocation mémoire | Le succès d’une allocation mémoire doit toujours être vérifié. |  |  |  |  |
| 152 | RÈGLE — L'isolement des données sensibles doit être effectué | Contrôler le bon usage d’une zone mémoire stockant des données sensibles i.e. minimiser l’exposition en mémoire, minimiser la copie et effacer la/les zones ayant contenu les données sensibles au plus tôt. |  |  |  |  |
| 153 | RÈGLE — Initialiser et consulter la valeur de errno avant et après toute exécution d'une fonction de | Vide |  |  |  |  |
| 154 | RÈGLE — La gestion des erreurs retournées par une fonction de la bibliothèque standard doit être sys | Vide |  |  |  |  |
| 155 | RÈGLE — Documentation des codes d'erreur | Tous les codes d’erreur retournés par une fonction doivent être documentés. Dans le cas où plusieurs codes d’erreur peuvent être retournés en même temps par la fonction, la documentation doit définir la priorité de gestion de ces codes. |  |  |  |  |
| 156 | RECOMMANDATION — Structuration des codes de retour | Les codes de retour doivent être structurés de façon à pouvoir obtenir rapidement une information concernant le déroulement de la fonction appelée : erreur ; type d’erreur ; alarme ; type d’alarme ; ok ; |  |  |  |  |
| 157 | RÈGLE — Code de retour d'un programme C en fonction du résultat de son exécution | Le code de retour d’un programme C doit avoir une signification afin d’indiquer le bon déroulement du programme ou la survenue d’une erreur : la valeur du code de retour doit être comprise entre 0 et 127 ; la valeur 0 indique que le programme s’est exécuté sans erreur ; la valeur 2 est généralement utilisée sous Unix pour indiquer une erreur dans les arguments passés en paramètres au programme. La signification des codes de retour du programme doit être indiquée dans sa documentation. |  |  |  |  |
| 158 | RECOMMANDATION — Privilégier les retours d'erreurs via des codes de retour dans la fonction principa | Un programme C doit disposer d’une fonction main() minimale. Les retours d’erreurs se font par un retour de code dédié (et donc documenté) de cette fonction. |  |  |  |  |
| 159 | RÈGLE — Ne pas utiliser les fonctions abort() ou \_Exit() | Vide |  |  |  |  |
| 160 | RECOMMANDATION — Limiter les appels à exit() | Les appels à la fonction exit() doivent être commentés et non systématiques. Le développeur doit le plus souvent possible les remplacer par un retour de code d’erreur dans la fonction principale. |  |  |  |  |
| 161 | RÈGLE — Ne pas utiliser les fonctions setjmp() et longjump() | Vide |  |  |  |  |
| 162 | RÈGLE — Ne pas utiliser les bibliothèques standards setjmp.h et stdarg.h | Vide |  |  |  |  |
| 163 | RECOMMANDATION — Limiter l'utilisation des bibliothèques standards manipulant des nombres flottants | Les bibliothèques standards float.h, fenv.h, complex.h et math.h ne doivent être utilisées que si cela est vraiment nécessaire. |  |  |  |  |
| 164 | RÈGLE — Ne pas utiliser les fonctions atoi() atol() atof() et atoll() de la bibliothèque stdlib.h | Les fonctions équivalentes strto\*() sont à utiliser en remplacement. |  |  |  |  |
| 165 | RÈGLE — Ne pas utiliser la fonction rand() de la bibliothèque standard | Vide |  |  |  |  |
| 166 | RÈGLE — Utiliser les versions « plus sécurisées » pour les fonctions de la bibliothèque standard | Quand des fonctions de la bibliothèque standard existent en différentes versions, la version « plus sécurisée » doit être utilisée. |  |  |  |  |
| 167 | RÈGLE — Ne pas utiliser de fonctions de la bibliothèque obsolescentes ou devenues obsolètes dans des | Vide |  |  |  |  |
| 168 | RÈGLE — Ne pas utiliser de fonctions de la bibliothèque manipulant des buffers sans prendre la taill | Vide |  |  |  |  |
| 169 | BONNE PRATIQUE — Tout code doit être soumis à relecture | Vide |  |  |  |  |
| 170 | RECOMMANDATION — Indentation des expressions longues | Lorsqu’une instruction ou une expression s’étale sur plusieurs lignes, il est indispensable de l’indenter afin de faciliter la compréhension du code. |  |  |  |  |
| 171 | RÈGLE — Identifier et supprimer tout code mort | Vide |  |  |  |  |
| 172 | RÈGLE — Le code doit être exempt de code non atteignable en dehors de code défensif et de code d'int | Il ne doit jamais y avoir de code inatteignable, sauf s’il s’agit de code défensif ou s’il s’agit de code d’une interface et dans ces deux cas, il faut le préciser en commentaire. |  |  |  |  |
| 173 | RECOMMANDATION — Evaluation outillée du code source pour limiter les risques d'erreurs d'exécution | Le code source du logiciel doit être analysé via au moins un outil d’analyse de code. Les résultats produits par l’outil d’analyse doivent être étudiés par le développeur et les corrections doivent être effectuées par rapport aux problèmes découverts. |  |  |  |  |
| 174 | RECOMMANDATION — Limitation de la complexité cyclomatique | La complexité cyclomatique d’une fonction doit être limitée au maximum. |  |  |  |  |
| 175 | RECOMMANDATION — Limitation de la longueur et la complexité d'une fonction | Une fonction doit être associée idéalement à un seul et unique traitement et doit donc correspondre à un nombre de lignes de code raisonnable. |  |  |  |  |
| 176 | RÈGLE — Ne pas utiliser de mots clés du C++ | Vide |  |  |  |  |
| 177 | RÈGLE — Séquences de caractères interdites dans les commentaires | Les séquences /\* et // sont interdites dans tous les commentaires. Et un commentaire sur une ligne introduit par // ne doit pas contenir de caractère de continuation de ligne \. |  |  |  |  |
| 178 | RÈGLE — Mise en oeuvre manuelle de mécanismes « canari » si les options de durcissement ne sont pas | Vide |  |  |  |  |
| 179 | RÈGLE — Pas d'assertions de mise au point sur un code mis en production | Les assertions de mise au point ne doivent pas être présentes en production. |  |  |  |  |
| 180 | RECOMMANDATION — La gestion des assertions d'intégrité doit inclure un effacement des données d'urge | Les assertions d’intégrité doivent apparaître en production. En cas de déclenchement d’une assertion d’intégrité, le code de traitement doit aboutir à un effacement d’urgence des données sensibles. |  |  |  |  |
| 181 | RÈGLE — Tout fichier non vide doit se terminer par un retour à la ligne et les directives de preproc | Un fichier non vide ne doit pas se terminer au milieu d’un commentaire ou d’une directive de preprocessing. |  |  |  |  |

Tableau 11 : socle de sécurité XXX