LISV Laboratoire d'ingénierie des systèmes de Versailles | UVSQ université PARIS-SACLAY

Programmation sécurisée (Secure coding)
Examen TP 1

Halim Djerroud

révision 1.0

*** Partie 1 ***

Exercice 1 : C - Dépassement arithmétique (entiers non signés)

En C, le dépassement d'entier signé est un comportement indéfini. Par conséquent, un compilateur peut supposer que les opérations signées ne débordent pas. Le code ci-dessous est censé fournir des vérifications de cohérence afin de renvoyer un code d'erreur en cas de dépassement de `offset + len` :

```
int offset, len; // entiers signés

// vérifier d'abord que offset et len sont positifs
if (offset < 0 || len <= 0)
    return -EINVAL;

// si offset + len dépasse le seuil MAXSIZE, ou en cas de dépassement,
// retourner un code d'erreur
if ((offset + len > MAXSIZE) || (offset + len < 0))
    return -EFBIG; // offset + len déborde
/* supposer à partir de maintenant que len + offset n'a pas débordé ... */
```

1. Expliquer pourquoi ce code est vulnérable (c'est-à-dire, pourquoi les vérifications peuvent échouer).
2. Proposer une solution pour le corriger.

Exercice 2 : C - Dépassement de tampon

Considérons le code C suivant :

```
void main () {
    char t;
    char t1[8];
    char t2[16];
    int i;
    t = 0;
    for (i=0; i<15; i++) t2[i]=2;
    t2[15]='\0';
    strcpy(t1, t2); // copier t2 dans t1
    printf("La valeur de t : %d \n", t);
}
```

L'organisation de la pile peut varier entre compilateurs. Dessiner un schéma correspondant aux situations suivantes :

— (a) Le programme affiche 2 comme valeur de `t`.
— (b) Le programme plante (accès mémoire invalide).
— (c) Pas de plantage, et le programme affiche 0 comme valeur de `t`.

Exercice 3 : C - Allocation dynamique

```c
typedef struct {void (*f)(void);} st;
void nothing () { printf("Rien\n"); }

int main(int argc, char * argv[]) {
    st *p1;
    char *p2;
    p1 = (st*) malloc(sizeof(st));
    p1->f = &nothing;
    free(p1);
    p2 = malloc(strlen(argv[1]));
    strcpy(p2, argv[1]);
    p1->f();
    return 0;
}
```

1. Expliquer le comportement indéfini et son exploitation possible.
2. Proposer des solutions :
   — Assigner des pointeurs libérés à NULL et appliquer cette technique.
   — Donner un exemple où cette solution est insuffisante.
   — Quel type d'analyse de code peut fournir une solution complète ?

Exercice 4 : PHP - Code vulnérable

Le script PHP suivant liste le contenu du répertoire personnel d'un utilisateur :

```php
<?php
$userName = $_POST["user"];
$command = 'ls -l /home/' . $userName;
system($command);
```

Expliquer la faille de sécurité et proposer une correction.

Exercice 5 : Python & PHP - Code vulnérable

1. Lister les vulnérabilités potentielles de `os.mkdir(path, mode)` en Python.
2. Analyser et corriger la fonction Python suivante :

```python
def makeNewUserDir(username):
    if invalidUsername(username):
        print('Les noms d'utilisateur ne doivent pas contenir de caractères invalides')
        return False
    try:
        raisePrivileges()
        os.mkdir('/home/' + username)
        lowerPrivileges()
    except OSError:
        print('Impossible de créer le répertoire utilisateur pour :', username)
        return False
    return True
```

3. Analyser et corriger la fonction PHP suivante :

```php
<?php
function createUserDir($username) {
    $path = '/home/'.$username;
    if (!mkdir($path)) return false;
    if (!chown($path, $username)) { rmdir($path); return false; }
    return true;
}
```

Exercice 6 : C - Effacement des données sensibles

```
int get_and_verify_password(char *real_password) {
    int result;
    char *user_password[64];
    get_password_from_user_somehow(user_password, sizeof(user_password));
    result = !strcmp(user_password, real_password);
    memset(user_password, 0, strlen(user_password));
    return result;
}
```

Expliquer pourquoi ce code peut ne pas garantir l'effacement du mot de passe et proposer des corrections.

Exercice 7 : C - Dépassements arithmétiques (entiers signés)

1. Donner des exemples critiques liés aux normes CERT (https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards) sur le débordement d'entiers non signés.
2. Analyser et corriger cette vulnérabilité

```
unsigned int i, nrep;
nrep = packet_get_int();
response = malloc(nrep * sizeof(char*));
if (response != NULL)
    for (i = 0; i < nrep; i++)
        response[i] = packet_get_string(NULL);
```

Exercice 8 : C - Dépassement de tampon

Analyser et corriger la fonction `safewrite` :

```
void safewrite(int tab[], int size, signed char ind, int val) {
    if (ind < size)
        tab[ind] = val;
    else
        printf("Hors limites\n");
}
```

Expliquer quel cas échoue et proposer une correction.

** Partie 2 ***

Exercice 1

On considère le programme C suivant :

```
int main() {
    int x;
    int tab[33];
    x = 1;
    while (x < 24)
        x = x * 2;
    tab[x] = 0;
    return 0;
}
```

1. Insérer l'assertion nécessaire pour rendre ce code "sécurisé".
2. Dessiner son CFG (Control Flow Graph).

Exercice 2

On considère le programme C suivant :

```
void compute(int x, int y) {
    int z, u;
    u = x;
    z = 0;
    while (z < y) {
        z = z + 1;
        u = u + 1;
    }
    return u;
}

int main() {
    int a;
    a = compute(5, 3);
    tab[a] = 0;
    return 0;
}
```

1. Insérer l'assertion nécessaire pour rendre ce code "sécurisé".
2. Dessiner le CFG de la fonction `compute` (en supposant que les valeurs initiales de `x` et `y` sont 5 et 3). `compute`.

*** Partie 3 ***

Exercice 1

On considère le programme C suivant :

```
int main() {
    int x;
    int tab[33];
    x = 1;
    while (x < 24)
        x = x * 2;
    tab[x] = 0;
    return 0;
}
```

1. Insérer l'assertion nécessaire pour rendre ce code "sécurisé".
2. Dessiner son CFG (Control Flow Graph).

Exercice 2

On considère le programme C suivant :

```
void compute(int x, int y) {
    int z, u;
    u = x;
    z = 0;
    while (z < y) {
        z = z + 1;
        u = u + 1;
    }
    return u;
```

```
}

int main() {
    int a;
    a = compute(5, 3);
    tab[a] = 0;
    return 0;
}
```

1. Insérer l'assertion nécessaire pour rendre ce code "sécurisé".
2. Dessiner le CFG de la fonction `compute` (en supposant que les valeurs initiales de `x` et `y` sont 5 et 3).

Exercice 3

Identifier les vulnérabilités potentielles dans le code suivant et expliquer :
— Pourquoi elles sont dangereuses ?
— Comment les corriger ?

```
int main() {
    char t1[255], *p, *q, *r;
    char n;

    scanf("%s", t1); // lire le contenu de t1
    scanf("%i", n); // lire la valeur de n

    if (n < 255) {
        p = malloc(n);
        q = p;
        strcpy(p, t1);
        printf("%s", q); // afficher le contenu de p
        free(q);
    }

    r = malloc(120);
    ...
    printf("%s", p); // afficher le contenu de p

    return 0;
}
```

Livrables

Les livrables à fournir à la fin du TP sont les suivants :
— Rapport détaillé qui répond aux questions
— Capture d'écran des résultats avec explications
— Le code corrigé si demandé.

Évaluation

Le rapport sera évalué en fonction des critères suivants :
— Exhaustivité et clarté des explications.
— Documentation précise des résultats obtenus.
— Propositions de solutions pour corriger les failles.