Sécurité Logicielle

– Examen (**1**) –

1 Audit de code (Barême approximatif : 5 points)

Ce code, en langage C, contient (au moins) trois bugs dont deux sont exploitables. Trouvez les tous les trois, dites de quel type de bug il s'agit et donnez une méthode pour en exploiter deux. Puis, suggérez des correctifs pour rendre ce code plus sûr. Argumentez vos correctifs.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define MAX_LEN 256
6
7  int merge_and_test (char *one, char *two) {
8    char tmp[MAX_LEN];
9    char *ptr = tmp;
10   for (; *one != '\0'; ++one, ++ptr)
11     *ptr = *one;
12   for (; *two != '\0'; ++two, ++ptr)
13     *ptr = *two;
14   *ptr = '\0';
15   return strcmp(tmp, "file://media/usb/");
16 }
17
18 char *dup_str (char *str) {
19   char *dup = malloc(MAX_LEN);
20   snprintf(dup, MAX_LEN, str);
21   return dup;
22 }
23
24 int main (int argc, char *argv[]) {
25   if (argc < 3)
26     exit(EXIT_FAILURE);
27
28   char *path;
29   if (merge_and_test(argv[1], argv[2]) > -1)
30     path = dup_str(argv[2]);
31
32   free(path);
33   if (path != NULL)
34     fprintf(stdout, "free␣failed␣on␣path:␣'%s'\n", path);
35
36   return EXIT_SUCCESS;
37 }
```

2 Hacking Blind (Barême indicatif : 15 points)

Lisez l'article "*Hacking Blind*" par A. Bitteau, A. Belay, A. Mashtizadeh, D. Mazières, D. Boneh (IEEE Symposium on Security and Privacy, Oakland 2014). Puis rédigez des réponses aux questions.

2.1 Questions de cours

Questions

1. Rappelez comment marche la technique d'attaque par ROP (Return-Oriented-Programming).
2. Comparez la technique du "*return-into-libc*" et du ROP. Pourquoi le ROP est-il plus efficace en terme de surface d'attaque.
3. Rappelez brièvement quel rôle jouent la PLT et la GOT dans le binaire d'un programme et ce qu'ils contiennent respectivement.

2.2 L'attaque BROP

On se place dans le cas d'un serveur qui est compilé avec des canaries et la pile non-exécutable (NX) dans le contexte d'un système d'exploitation qui a l'ASLR activé.

Questions

1. Expliquez la technique utilisée pour contourner les canaries.
2. Bien que cela ne soit pas dit dans l'article, donnez le type de shellcode ROP que les auteurs cherchent à créer à la section VIII. BROP Attack, A. The pieces of the puzzle, p.5.
3. Expliquez ce qu'est un "*BROP gadget*" et ce que les auteurs veulent dire par : "*So by finding a single gadget, we find two gadgets that control the first two arguments of a call.*" (section VIII. BROP Attack, A. The pieces of the puzzle, p.5).
4. Expliquez à quoi sert la PLT et pourquoi elle est susceptible de contenir un "`call write`".
5. Expliquez comment retrouver les gadgets valides et les différencier pour savoir quel code ils contiennent.
6. En vous basant sur la technique utilisée pour reconnaître le `strcmp()`. Comment distingueriez-vous la fonction `int strncmp(char *s1, char *s2, size_t n)` et `void *memcpy(void *dest, const void *src, size_t n)` (on supposera que vous avez pu extraire quelques adresses valides du processus que vous explorez.
7. Expliquez rapidement le fonctionnement de `braille` et le code spécifique que doit produire un attaquant pour l'utiliser contre un serveur précis.

2.3 Limitations et améliorations possibles du BROP

Questions

1. Rappelez le principe du PIE (Position Independant Executable) et expliquez comment il peut gêner un attaquant qui utilise la technique du BROP.
2. Expliquez ce que les auteurs appellent la "*rerandomization*" et expliquez comment vous en programmeriez une sur un serveur existant. Enfin, expliquez en quoi elle serait gênante ici.
3. Quelle seraient les conséquences pour l'attaque BROP si le serveur possédait plusieurs autres bugs (non reliés à la faille originale) qui le faisaient planter ? Explicitez dans quels cas cela pourrait-il être gênant pour l'attaquant ?
4. Quel serait l'effet d'une protection du type RELRO (full ou partial) sur ce type d'attaque ?
5. Donnez quelques idées qui pourraient permettre de rendre `braille` encore plus efficace lors d'une attaque aveugle.