Rapport sécurité

Interception d'appels systèmes

Fleury Malik malik.fleury@he-arc.ch
Wermeille Bastien bastien.wermeille@he-arc.ch

26 avril 2019

Table des matières

Auteurs : Malik Fleury & Bastien Wermeille

1 Introduction

Dans le cadre du cours de sécurité, nous devons réaliser un projet ayant un lien avec le cours. Une liste de sujets a été proposée par l'enseignant et nous avons choisi le thème suivant :

> Interception d'appels systèmes / d'appels de bibliothèques sous Linux ou Microsoft (p.ex. avec LD_PRELOAD ou SystemTap)

Nous avons eu l'idée d'effectuer une interception d'appel système sur le programme C que nous avons réalisé lors du cours de 1re année qui consistait en un client SMTP. Le but est de pouvoir ajouter une copie carbone au mail envoyé via l'interception d'appel système de ce programme. De cette façon, l'utilisateur n'a pas conscience que le mail est envoyé en copie à une autre adresse en plus du destinataire.

2 Fonctionnement

Notre logique de notre programme est relativement simple, nous interceptons les appels système `fprintf` du client SMTP et si la chaîne passée à `fprintf` contient la chaine `RCPT TO :` alors cela signifie qu'on est en train d'envoyer l'adresse email du destinataire.

Lorsque cet appel est détecté, nous envoyons notre propre ligne avec l'adresse email du pirate :
- `RCPT TO : adresse.pirate@xxx.com`

Puis nous attendons la réponse du serveur avant de finalement envoyer le contenu du `fprintf` intercepté.

3 Comment utiliser

La compilation de notre programme nécessite `gcc` et ne fonctionne que sur un système Linux.

3.1 Compilation

La ligne ci-dessous permet de compiler le programme qui intercepte l'appel `fprintf` :

```
gcc -fPIC -shared -o libpreload.so ./preload.c -ldl -DCC=\"malik.fleury@he-arc.ch\"
```

Le dernier paramètre passé permet de définir à qui sera envoyée la copie carbone lors de l'exécution de la bibliothèque partagée.

La ligne suivante permet de compiler le client SMTP pour l'envoi du mail :

```
gcc client_SMTP.c -o client_SMTP
```

3.2 Utilisation

Enfin, pour utiliser le client SMTP avec l'interception d'appel de la fonction `fprintf`:

```
LD_PRELOAD=libpreload.so ./client_SMTP
```

4 Contre-mesures potentielles

Voici quelques contre mesures qui permettraient de détecter une attaque ou qui implémentées dans notre client TCP permettraient d'empêcher une telle attaque avec notre démonstrateur.

4.1 Test avec un serveur SMTP online ou en localhost

Le client SMTP comporte un paramètre permettant de définir le serveur avec lequel communiquer. Il est ainsi possible de simuler un serveur avec netcat par exemple. La simulation d'un tel serveur pourrait permettre de détecter l'ajout d'une adresse email.

Cependant, le démonstrateur de concept que nous avons implémenté pourrait très bien se désactiver lorsque l'adresse du serveur est `localhost`.

4.2 Envoie des informations caractère par caractère

Notre bibliothèque détecte l'appel système `fprintf` et compare la chaîne de caractères envoyée avec la chaîne `RCPT TO :`. Dans le cas où le client SMTP enverrait ses caractères un à un, alors l'implémentation actuelle ne fonctionnerait pas, car elle ne prend pas en compte les anciens caractères envoyés, mais uniquement ceux présents lors de l'appel de `fprintf`.

4.3 TODO Autre contre mesure

5 Problèmes rencontrés

5.1 Interception du bon appel

Afin de trouver l'appel système à intercepter, nous avons tout d'abord essayé d'intercepter directement la commande `puts` cependant cette fonction n'était jamais appelée.

Puis nous avons essayé de lancer notre programme avec la commande `strace` comme suit :

```
strace ./client_SMTP xxx@he-arc.ch "Salut Roger" message localhost xxx@he-arc.ch 12000
```

Qui nous indiquait qu'il fallait intercepter la fonction `write`.

```
write(14, "MAIL FROM: <xxx@he-arc.ch>\r\n", 28) = 28
read(14, "200\n", 4096)                 = 4
write(1, "200\n", 4200
)                       = 4
write(1, "The server has completed the tas"..., 48The server has completed the task successfully.
) = 48
nanosleep({tv sec=1, tv nsec=0}, 0x7ffe9248dbe0) = 0
write(1, "\n", 1
)                       = 1
write(14, "RCPT TO: <xxx@he-arc.ch>\r\n", 26) = 26
read(14, "200\n", 4096)                 = 4
write(1, "200\n", 4200
)                       = 4
write(1, "The server has completed the tas"..., 48The server has completed the task successfully.
) = 48
nanosleep({tv sec=1, tv nsec=0}, 0x7ffe9248dbe0) = 0
write(1, "\n", 1
)                       = 1
write(14, "DATA\r\n", 6)                = 6
read(14, "354\n", 4096)                 = 4
write(1, "354\n", 4354
)                       = 4
```

Fig. 1 :

Nous avons ensuite essayé d'intercepter cette fonction write mais sans succès. La technique `LD_Preload` permet d'intercepter des appels système de notre code, mais la bibliothèque standard ne fait pas appel à cette fonction.

Nous avons ensuite utilisé `ltrace` afin de voir les appels y compris de bibliothèques ce qui nous a donné le résultat suivant :

```
sprintf("MAIL FROM: <xxx@he-arc.ch>\r\n", "MAIL FROM: <%s>\r\n", "xxx@he-arc.ch") = 28
putchar(10, 0x7ffc2a518b40, 0x7ffc2a518b40, 0
)                             = 10
fprintf(0x563c68022490, "MAIL FROM: <xxx@he-arc.ch>\r\n")                = 28
fflush(0x563c68022490)                                                   = 0
fgets("200\n", 255, 0x563c68022490)                                      = 0x7ffc2a5189d0
printf("200\n"200
)                                                          = 4
puts("The server has completed the tas"...The server has completed the task successfully.
)                               = 48
sleep(1)                                                                 = 0
sprintf("RCPT TO: <xxx@he-arc.ch>\r\n", "RCPT TO: <%s>\r\n", "xxx@he-arc.ch") = 26
putchar(10, 0x7ffc2a518b40, 0x7ffc2a518b40, 0
)                             = 10
fprintf(0x563c68022490, "RCPT TO: <xxx@he-arc.ch>\r\n")                  = 26
fflush(0x563c68022490)                                                   = 0
fgets("200\n", 255, 0x563c68022490)                                      = 0x7ffc2a5189d0
printf("200\n"200
)                                                          = 4
puts("The server has completed the tas"...The server has completed the task successfully.
)                               = 48
sleep(1)                                                                 = 0
putchar(10, 0x563c66162a95, -2627, 0x7f65befef9a4
)                           = 10
fprintf(0x563c68022490, "DATA\r\n")                                      = 6
fflush(0x563c68022490)                                                   = 0
fgets("354\n", 255, 0x563c68022490)                                      = 0x7ffc2a5189d0
printf("354\n"354
)                                                          = 4
```

Fig. 2 :

Nous avons ainsi décidé d'intercepter l'appel à la fonction `fprintf`. Cette solution a été la bonne et nous l'avons implémentée.

6 Améliorations

Ce projet avait pour but la mise en place d'un démonstrateur permettant de démontrer qu'il est possible via une interception d'appels de modifier le comportement d'un programme.

Notre projet est complet, mais a été limité par le temps à disposition. Voici quelques idées d'améliorations potentielles qui seraient intéressantes d'implémenter.

6.1 SystemTap

Nous avons choisi d'utiliser le système de `LD_PRELOAD` pour ce projet, mais il serait intéressant de mettre en place le même système, mais en utilisant `SystemTap`.

6.2 Statistiques sur les appels système

Notre projet ayant comme but premier de découvrir les interceptions d'appels système, il serait intéressant de faire des statistiques sur les différents appels de programme. Et ensuite, essayer de détecter des programmes dangereux à partir de ces statistiques.

7 Conclusion

Ce projet nous a permis de voir comment fonctionnent les appels système et

8 Bibliographie

TODO Avec pandoc

Table des figures

9 References

- http ://samanbarghi.com/blog/2014/09/05/how-to-wrap-a-system-call-libc-function-in-linux/