Université ABDELMALEK ESSAADI

École National des Sciences appliquée de Tanger

Département de mathématique et informatique

Master Cyber Sécurité et cybercriminalité (MCSC)

SÉCURITÉ CONTRE FORK BOMB ATTACK SOUS LINUX SYSTÈMES

Programmation système

NAJAH ISSAM

ANNEE UNIVERSITAIRE : 2020/2021

MR KHALID.AMECHNOUE

Programmation système

Sommaire

Résumé

Fork Bomb est une attaque déni de service .Un attaquant génère rapidement de nombreux processus, épuisant le ressources des systèmes informatiques cibles. Il y a plusieurs travaux antérieurs pour détecter et supprimer les processus qui causent Fork Bomb Attack , Cependant le système d'exploitation avec les méthodes précédentes présentent le risque de mettre fin processus qui ne bifurquent pas les processus de bombe . La plupart des solutions trouvées dans la littérature ont leurs propres limites comme la détection de faux positifs et l'indisponibilité des ressources. Préserver un objectif qui est la disponibilité au sein de la CIA (confidentialité, intégrité et disponibilité) des sécurités de l'information. J'ai proposé de développer une solution efficace qui gère Fork Bomb Attack dans une telle manière que ce système reste disponible pour une utilisation par l'utilisateur final. J'étudie une nouvelle méthode qui combine entre la quarantaine des ressources de processus et la détection précise des Fork Bomb par nom de processus.

Introduction

Fork Bomb est une attaque par déni de service (DoS attaque) qui épuisent les ressources du système cible en création rapide de nombreux processus. Lorsqu'un système d'exploitation crée un processus, il alloue des ressources mémoire pour le nouveau traité. Les attaques Fork Bomb exploitent l'allocation des ressources. Dans une situation d'attaque, les programmes malveillants font de nombreux processus enfants et les enfants font des enfants. Cette la création rapide de processus épuise des ressources limitées, telles que RAM, temps CPU, table de gestion des processus et ainsi de suite. Comme résultats, aucun processus légitime ne peut démarrer ses tâches car les processus malveillants épuisent les ressources du système. De plus, les pénuries de ressources perturbent les processus existants. Il est nécessaire d'arrêter le processus offensant pour battre hors des attaques. Cependant, c'est difficile car le système, les logiciels, tels que les outils de gestion des tâches, consomment également les ressources système comme le logiciel d'application général. Parfois, l'attaque force le système cible à redémarrer. Fork Bomb est une attaque perturbant les opérations de service légitimes par consommant beaucoup de ressources informatiques.

Il existe de nombreuses variantes des attaques Fork Bomb. Les attaquants peuvent toujours exécuter ce type d'attaque lorsqu'ils le peuvent exécuter des programmes arbitraires. En d'autres termes, ils n'ont accès au système cible qu'en tant qu'utilisateurs normaux pour effectuer les attaques. De plus, ils peuvent exécuter les attaques via les vulnérabilités des programmes serveur que quelqu'un peut exécuter programmes arbitraires. Dans ce cas, les attaquants peuvent nuire au système sans aucun droit d'accès. Il y a beaucoup de rapport de vulnérabilité sur l'exécution arbitraire de programme. De plus, il existe des vulnérabilités zero-day qui exploitent des vulnérabilités non divulguées. Couper la route de fork les attentats à la bombe est difficiles. Il n'y a pas que des foks intentionnelles des attentats à la bombe mais aussi des attentats à la bombe involontaires. Un exemple est le manque de programmation. La création de processus n'est pas une procédure spéciale. Les programmeurs créent souvent un programme avec création de processus. Si le programme contient un bogue qui se répète processus de création de processus à l'infini, le programme crée une situation comme une attaque Fork Bomb.

Les systèmes d'exploitation peuvent éviter une pénurie de ressources. Ils ont un mécanisme de gestion des ressources pour limiter l'utilisation de ressources. Le mécanisme est une mesure forte de la ressource. Cependant, les mécanismes de gestion des ressources existants n'ont pas une capacité suffisante pour protéger le système d'attaques Fork Bomb. Il y a plusieurs travaux antérieurs pour la détection des attaques Fork Bomb et récupération après celles-ci. Les méthodes proposées dans les travaux précédent détectent l'attaque Fork Bomb et arrêtez-la en fonction du nombre de processus en cours d'exécution dans le système ou la fréquence de création de processus. Ces méthodes sont efficaces pour l'attaque Fork Bomb. Sur d'autre part, ils ont un problème de faux positifs, c'est-à-dire qu'ils détectent parfois les processus légitimes comme les processus qui provoquent une attaque Fork Bomb.

Ainsi dans ce Projet, j'ai proposé une nouvelle approche qui gère la Fork Bomb de telle manière que ce système reste disponible. Ce document est organisé en quatre sections suivantes qui contiennent l'objectif principal de l'ensemble du travail, existant solutions avec ses limites, solution proposée avec quelques résultats expérimentaux respectivement. Enfin, la conclusion.

I. Attaques Fork Bomb

Une attaque Fork Bomb est une attaque par déni de service (attaque DoS) qui épuisent les ressources du système cible en création rapide de nombreux processus. Le prog 1 est un exemple de programme qui effectue une attaque à la Fork Bomb. Ce programme crée un processus pour chaque itération. Les processus enfants créent ses processus enfants comme son parent traité. Cela entraîne une pénurie de ressources pour la zone de mémoire, temps CPU, table de gestion des processus, etc. sur. En conséquence, le système devient instable et le service de la disponibilité sera perdu. Tout utilisateur peut déclencher une attaque à la bombe sans aucun privilège d'administrateur s'il peut exécuter des programmes arbitraires sur le système cible. La création de processus est une opération primitive non seulement pour les administrateurs, mais également pour tous les utilisateurs. Un nouveau processus sans création de processus est impossible. Les vulnérabilités des programmes serveur sont également l'attaque itinéraire. De nombreux rapports de vulnérabilité sont publiés presque tous les jours. Parmi eux, il existe des vulnérabilités réseaux qui permettent à des attaquants distants d'exécuter un code de programme arbitraire. Les vulnérabilités permettent pour se produire une attaque Fork Bomb sans aucun droit d'accès aux systèmes cibles. Il existe de nombreux rapports de vulnérabilité sur les exécutions de code de programme arbitraire illégales.

Aussi, les attaques zero-day, qui ne sont pas révélées ou n'ont pas été encore corrigées, sont disponibles. Par conséquent, il n'est pas facile de couper hors des routes d'attaque pour Fork Bomb. Il y a aussi des attaques Fork Bomb involontaires ainsi que les attaques prévues mentionnées ci-dessus. La création de processus n'est pas une procédure spéciale dans les programmes. Si un programme contient des bogues pour répéter la procédure de création de processus involontaires, le programme crée une situation comme une attaque Fork Bomb intentionnelle.

```c
#include <unistd.h>
int main(void){
    while(1){
        fork();
    }
}
```

Prog 1 : exemple d'attaque
Fork Bomb .

II. SOLUTIONS EXISTANTES

Dans cette section toutes les solutions existantes à traiter avec Fork Bomb est donnée avec leur limites.

1. Limitation du nombre de processus

Le noyau Linux implémente un mécanisme limitant le nombre de processus pour chaque utilisateur. La méthode de limitation fait qu'il est possible d'empêcher les attaques Fork Bomb. La taille de la table de gestion de processus dans un système d'exploitation est finie. Les attaques Fork Bomb épuisent les structures de données finies et font il est impossible pour le système d'exploitation cible de créer un nouveau traité. Le mécanisme de limitation empêche que du tableau de gestion des processus.

Cependant, la simple limitation de la création de processus est insuffisante pour la prévention des attaques Fork Bomb pour deux raisons. Premièrement, la simple limitation de la création de processus ne peut pas empêcher une pénurie de ressources de mémoire. Comme mentionné ci-dessus, la limitation de la création de processus peut éviter le manque du tableau de gestion des processus. Cependant, il ne peut pas éviter le manque de ressources de mémoire dans certains cas. Si l'illégal les processus consomment une certaine mémoire, les processus consomment mémoire énorme même si le nombre d'entre eux est petit. Par exemple, un processus illégal se répète pour créer un processus qui consomme 1 Mo de mémoire. Lorsque le nombre de processus est de 1000, la quantité de mémoire consommée est de près de 1 Go. Il est nécessaire de limiter le nombre de création processus inférieure à 1024 pour éviter cela. Le paramètre de limitation est irréaliste lorsque le système est en état de production. Ainsi, la simple limitation du nombre de création de processus ne peut pas empêcher le manque de mémoire par une attaque Fork Bomb même si la limitation peut éviter la pénurie de table de gestion des processus.

La deuxième raison est qu'il ne peut pas empêcher le gaspillage de la ressource CPU. La limitation de la création de processus garde le nombre de processus en dessous d'un certain niveau. Quand le nombre de processus dépasse la limite, une procédure créant un processus échoue. Cependant, si les processus continuent d'appeler une procédure de création de processus indépendamment des échecs, ils consomment d'énormes ressources CPU. La simple limitation du processus de création ne peut pas empêcher le gaspillage de ressources CPU.

2. Limitation de l'utilisation de la mémoire par processus

Limiter la quantité de mémoire pour les processus utilisateur est une méthode de contre-attaque alternative pour l'attaque fork bomb. Le limite permet d'éviter un manque de mémoire d'attaque fork bomb. Le noyau Linux peut créer une mémoire limite d'utilisation pour les groupes de processus arbitraires (par exemple, unité utilisateur et unité de programme). Nous supposons que nous limitons la quantité de mémoire pour les processus utilisateur sur un système informatique ayant 1024 Mo de RAM. Si la valeur limite est de 921 Mo (90% du total RAM), le noyau du système d'exploitation peut obtenir au moins 10% de la RAM totale. La limitation peut empêcher un manque de mémoire même si une attaque fork bomb se produit .Un problème de limitation de l'utilisation de la mémoire est qu'il ne peut pas empêcher une obstruction du service même si cela peut empêcher un manque de mémoire par des attaques fork bomb. Le mécanisme de gestion du mémoire d'un système d'exploitation ne faire la distinction entre les processus provoquant une attaque fork bomb

et d'autres processus légitimes. Par conséquent, quand une fork bomb se produit, la quantité de mémoire pour les processus légitimes diminue rapidement. En conséquence, la situation obstrue les processus légitimes.

3. Limitation du taux de création de processus de Chaque utilisateur

La solution consiste à définir l'intervalle de temps pour limiter le taux de création de processus pour chaque utilisateur. Le problème avec ça solution est de vérifier le nombre total de processus créés dans un intervalle de temps spécifié qui comprend même le processus légitime avec les processus fork bomb.

Le problème des solutions existantes ci-dessus prédisent à tort que le processus légitime est un processus de fork bomb .Il est possible de détecter un processus qui n'est pas une fork bomb peut ne pas détecter un processus de fork bomb . L'utilisateur n'a aucun contrôle sur le système et il doit redémarrer le système. Dans ce processus, il peut perdre beaucoup d'informations dans le système.

4. Atténuation des attaques Fork Bomb par processus Quarantaine des ressources

Cette méthode n'empêche pas l'attaque Fork Bomb , mais il essaie d'en atténuer l'effet. Ici, Le système d'exploitation ne met pas fin aux processus de bombe, Au lieu de cela, le système d'exploitation rendra la limitation de ressource pour les processus de bombe et ce sera vérifié périodiquement. Une fois, il commence à se comporter comme un processus normal, la limitation des ressources sera supprimée et sera autorisée à s'exécuter comme un processus normal. C'est une bonne approche pour supprimer la fausse positivité de la solution précédente. Et selon les lois sur la sécurité de l'information, le système doit rester disponible pour utilisation, soit c'est un processus de bombe ou un légitime. Ainsi, cette solution a tenté d'atténuer Fork Bomb et obéit aux lois sur la sécurité de l'information.

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(void){
    void *mem = malloc((1 << 10)*512);
    int i, pid;
    while(1){
        pid = fork();
        if(pid == 0){
            memset(mem, 'd', (1 << 10)*512);
        }
        usleep(1 * 1000);
    }
}
```

Prog 2 : programme de test

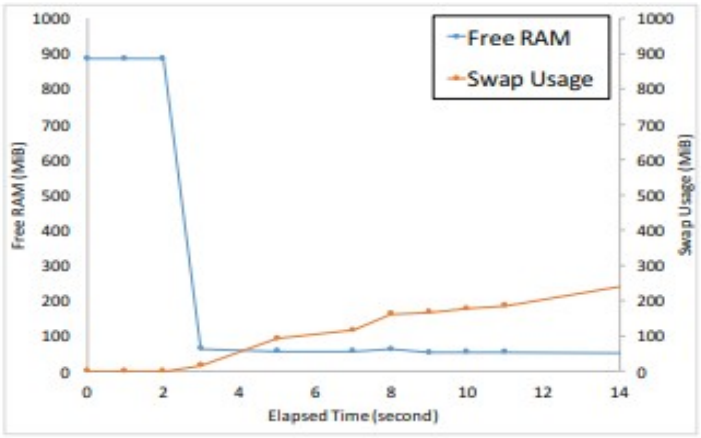

Figure 2 : changement d'usage da RAM (sans La methode

Atténuation des attaques Fork Bomb par processus Quarantaine des ressources)

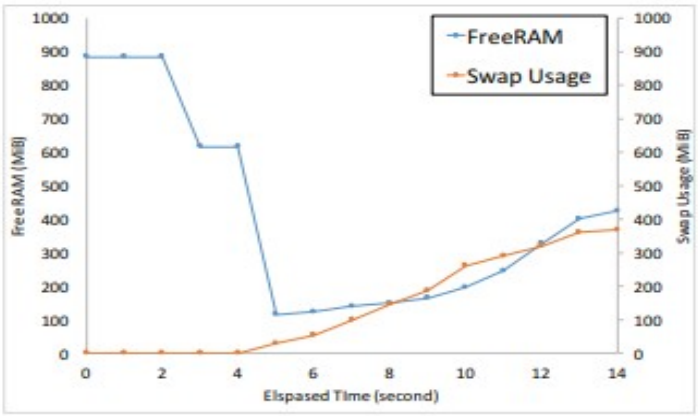

Figure 2 : changement d'usage da RAM (avec La methode

Atténuation des attaques Fork Bomb par processus Quarantaine des ressources)

Dans Le prog 2 en a un programme qui répète une création de processus toutes les millisecondes. Le processus enfant créé consomme au moins 512 Ko de mémoire car chaque processus écrit 512 Ko de données factices dans sa zone de mémoire.

La figure 1 décrit le résultat sans méthode. Ce graphique décrit le changement de la main libre mémoire et l'utilisation de la zone de swap. L'intervalle d'enregistrement est de 1 seconde. Le programme démarre le test 3 secondes après l'heure de départ. Le graphique montre que la taille libre de la mémoire a diminué rapidement 3 secondes après l'heure de début. La valeur minimale de la mémoire principale libre est de 52,52 Mo. Le graphique décrit également que la taille de la zone d'échange a augmenté. En règle générale, la taille de la mémoire libre peut augmenter par swap. Le

résultat, cependant, ne montre pas une telle tendance. La raison semble être que le programme de test continu à nécessiter une zone de mémoire.

La figure 2 décrit le résultat avec la méthode. La moyenne du graphique est la même que celle de la figure 1. Le graphique a également montré que la taille de la mémoire principale libre a diminué rapidement à 3 secondes après l'heure de début ainsi que le résultat sans méthode. La valeur minimale de la main mémoire libre est de 115,02 Mo. Cependant, la taille de la mémoire principale augmente 5 secondes après l'heure de début. La taille de la main la mémoire est d'environ 425 Mo À 14 secondes après le démarrage temps. Le résultat montre que cette méthode détecte et empêche une attaque fork bomb.

5. Détection précise des Forks Bombs par Nom processus

La solution dans cette méthode est de limiter le taux de création pour chaque processus, pas pour chaque utilisateur puisque la fork bomb est normalement créé par un programme. En limitant le taux de création de processus sur chaque processus, il est facile de trouver le processus de la fork bomb. Par cette méthode, nous pouvons facilement distinguer les fork bomb processus et processus légitime. La manière la plus appropriée d'implémenter une telle solution est dans le noyau space, en utilisant les modules du noyau. La façon dont un processus est répliqué dans le système est en appelant l'appel système clone. Donc, au lieu d'accès direct à l'appel système original du clone, ce nouveau module surveille les paramètres de seuil (comme le nombre de processus pouvant être créés sur une période de temps). Si les paramètres sont bien dans les limites de seuil, le flux de contrôle du programme est autorisé à aller vers l'original l'appel système cloner, sinon le processus est tué.

Par exemple, un processus de fork bomb crée 1000 processus fils toutes les millisecondes et non-fork bombe crée 100 processus. Mais certaines des applications spéciales créent plus de 1000 processus fils en une milliseconde à traiter ce type d'applications en ajoutant à la liste d'exceptions. Liste des applications spéciales qui permettent de créer en toute sécurité processus enfants sans aucun service d'arrêt pour ces applications.

Le module noyau vérifie chaque fois qu'un appel système fork est Dans le module noyau, il y a deux listes - liste d'exceptions et liste de détection. La première liste est la liste d'exceptions qui contiennent les noms de processus qui ne doivent pas être vérifiés par le mécanisme de détection. L'utilisateur aura donc le choix pour ajouter les processus légitimes à la liste d'exceptions et donc éviter le processus de vérification d'une fork bomb. La deuxième liste est la liste de détection qui contiendra tous les processus qui a été détectée comme une fork bomb. Chaque fois qu'un processus est détecté comme une fork bomb, le nom de processus de la fork bomb sera ajouté à la liste de détection. La pertinence du nom du processus est qu'il s'agit du seul identifiant constant d'un processus car l'identifiant du processus ne cesse de changer. Les processus de fork bomb ne seront pas exécutés. La liste des processus de détection est enregistrée pour le futur afin que la prochaine

fois un processus de fork bomb dans la liste de détection tente d'attaquer, il va être immédiatement refusé d'exécuter et donc d'empêcher.

Quatre processus sont en cours de création comme indiqué dans prog 3 et chaque processus génère 100 processus par seconde jusqu'au total 1000 processus sont créés.

```
#include<unistd.h>
#include<stdlib.h>
int main(){
        int i=0;
        for(i=0;i<1000;i++){
                fork();
                usleep(10000);
        }
}
```

Prog 3 : programme de test

1. Exécution des processus ci-dessus comme indiqué dans prog 3 parallèlement et limiter les paramètres par la solution existante Il montre une fork bomb faussement positive selon la précédente mise en œuvre aucun de ceux-ci n'est une fork bomb. Ça montre vraiment que l'un d'entre eux a une fork bomb.

Une création de processus par utilisateur et non par processus. Alors, parfois prédit à tort qu'un processus légitime est une bombe à fourche.

2. Exécution des processus ci-dessus comme indiqué dans prog 3 parallèlement et limiter les paramètres par solution proposée Il permet à tous les processus ci-dessus de créer un processus enfant et il n'y a pas de projection de fork bomb.Parce qu'il vérifie la création de processus par processus et non par utilisateur.

TAB 1 : TEMPS NÉCESSAIRE POUR IDENTIFIER LA FORK BOMBRESPECTIVE AU SEUIL

SEUIL	Temps (secs)
500	0.044
1000	0.124
5000	0.668
10000	2.468

III. SOLUTION PROPOSÉE

La méthode proposée pour traiter efficacement l'attaque fork bomb est la suivante:

Notre approche face à fork bomb est combinaison d'Atténuation des attaques Fork Bomb par processus Quarantaine des ressources et Détection précise des Forks Bombs par Nom processus.

Une détection précise est effectuée. Une fois cela fait, nous l'avons pris comme base et appliqué la solution présentée dans L'Atténuation des attaques Fork Bomb par processus Quarantaine des ressources. Le Raison derrière cela, une fois que le nom du processus est ajouté dans la liste de détection, il y reste pour toujours. Quand à futur, le processus arrive et porte le même nom mais pas fork bomb, puis la solution définie dans La Détection précise des Forks Bombs par Nom processus sera directement refuser l'exécution.

De plus, si un utilisateur malveillant crée un processus ayant un nom ou un nom bien connu de ce processus dérivé de certains logiciels populaires, il sera ajouté dans la liste de détection et après cela quand le processus légitime provient duquel l'attaquant dérivé du nom, il ne peut pas s'exécuter.

La figure 3 suivante est l'organigramme de la solution proposée. Il montre également comment les différentes étapes sont exécutées afin de traiter efficacement la fork bomb. Comme la montre la figure ci-dessus, il y a deux flux à partir de laquelle le système peut passer. Dans la première étape, lorsque le processus demande une fork, alors il sera vérifié dans la liste des exceptions. S'il se trouve dans la liste d'exceptions alors sans effectuer aucune vérification, il sera autorisé pour exécution. Sinon, ce sera coché dans la liste de détection.

Dans la deuxième étape, disons que le processus est introuvable dans la liste de détection, le taux de croissance du processus est mesuré dans un intervalle de temps particulier et comparé avec la valeur seuil prédéfinie. S'il dépasse alors son nom est ajouté à la liste de détection et sera tué.

Dans la dernière étape, si une correspondance est trouvée dans la liste de détection, alors au lieu de refuser directement l'exécution, ce sera autorisé à s'exécuter dans la quarantaine des ressources, où la limitation des ressources est imposée au processus. Ici, il sera vérifié périodiquement et s'il a un comportement normal, il sera libéré de la limitation de la ressource et son nom sera supprimé de la liste de détection, sinon, il sera tué.

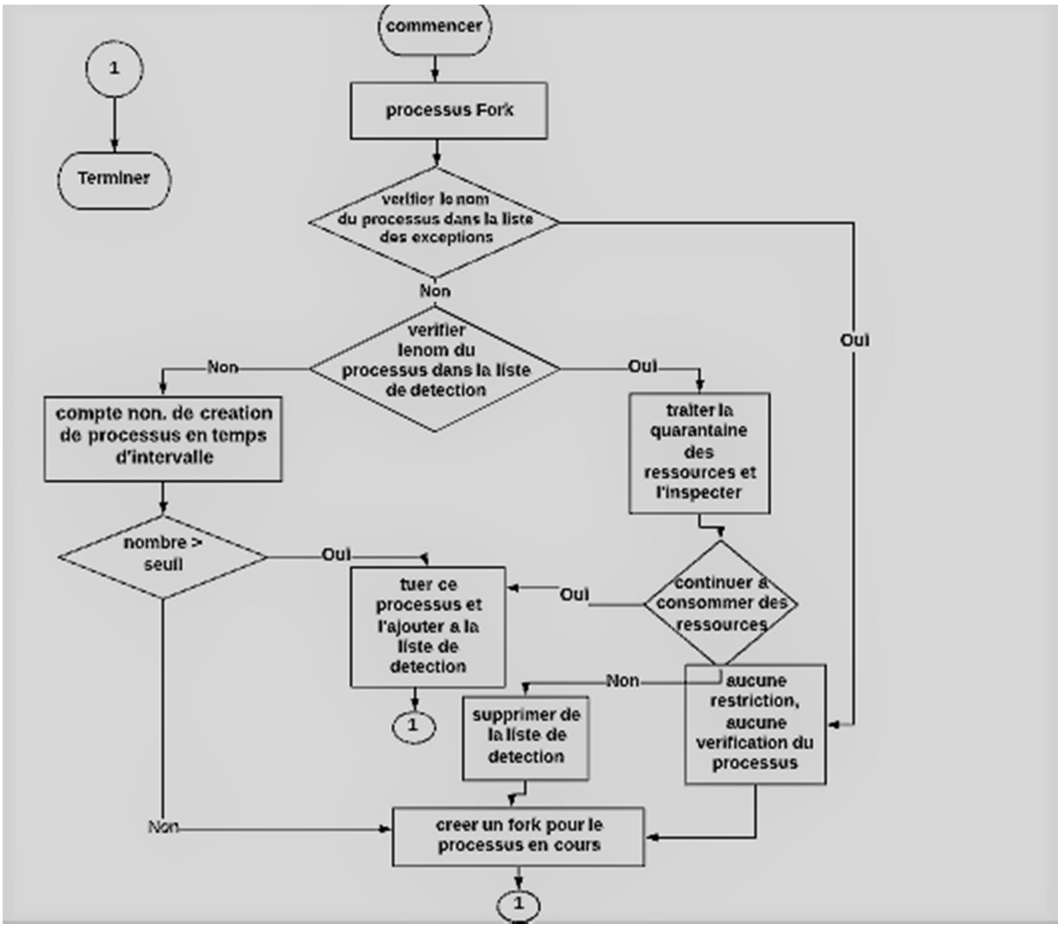

Figure 3 : Flux de la solution proposée

Lorsque vous appliquez une fork bomb dans le système que ce soit intentionnel ou non, il est clairement visible qu'avant que le système de fork bomb ne se comporte en temps normal. Une fois que la fork bomb se produit, le système commence à prendre du retard. À des fins expérimentales, nous avons utilisé fork bomb bash. Dans ce script shell, le processus de bombe se réplique deux fois. Ainsi, la croissance du processus peut être vu comme 1,2,4,8,16,32,64 ... etc. Et tout est à faire, rien ne traite. La seule tâche qu'ils accomplissent est la bifurquer d'eux-mêmes deux fois. Ici, le parent ne sera jamais mort. Et par conséquent, cela rend le système surchargé.

Dans les solutions précédentes, comme mentionné ci-dessus, le taux de croissance du processus est mesuré et s'il va au-delà d'un certain seuil prédéfini, il sera tué. Mais dans certains cas, si un utilisateur veut faire cela arrive, alors dans ce cas, notre solution proposée fournit la possibilité d'ajouter ce processus dans la liste d'exceptions.

Détails de la configuration du système et paramètres qui a été appliqué sont comme ci-dessous:

TAB 2 : PARAMETRES ET VALEURS

Paramètre	**Valeur**
FORK_RECORD_DEPTH	500 record
FORK_SPEED_LIMIT	1000 fork/sec
MEMORY_CAPTURE	750 MB
INSPECTION_INTERVAL	10 msec

TAB 3 : CONFIGURATION SYSTEME

CPU	Intel Celeron
Frequency	2.20 GHz
RAM	1GB
Linux Kernel Version	4.15.0-47-generic
ubuntu Version	18.04 LTS

Afin de trouver une fork bomb, notre système vérifiera pour l'appel système fork. Et comptera le nombre de fois l'appel système fork exécuté. Si ça va au-delà le seuil, puis il sera ajouté à la détection liste. Donc, pour les nouveaux processus, le temps de vérifier si consommé. Sinon, il ne vérifiera pas la même chose. D'où le temps est considérablement réduit.

En revanche, pour atteindre la disponibilité du système, il vérifie également les processus qui sont déjà dans la liste de détection pour sa validité. Ainsi, cette solution peut générer la charge du système de glissière, en particulier la charge sur le mémoire système.

Le tableau 4 suivant indique le temps nécessaire pour identifier la fork bomb avec respectivement au seuil comme indiqué dans le tableau 2.

TAB 4 : TEMPS NÉCESSAIRE POUR IDENTIFIER LA BOMBE DE FOURCHE RESPECTIVE AU SEUIL

Au seuil	**Temps (msec)**
500	44
700	124
1000	668
1500	2468

Pour trouver la charge sur le système, bien connaître le cockpit du serveur Web est utilisé. Le gestionnaire de tâches peut également être utilisé dans le cadre de presque tous les systèmes. Ici, dans notre cas, xfce4-taskmanager fournit de bonnes informations de diagnostic comme la mémoire utilisée, charge sur le processeur et utilisation de l'espace de swap. La figure 4 suivante montre la charge sur le système via CPU Load, Memory Load au moment de l'attaque.

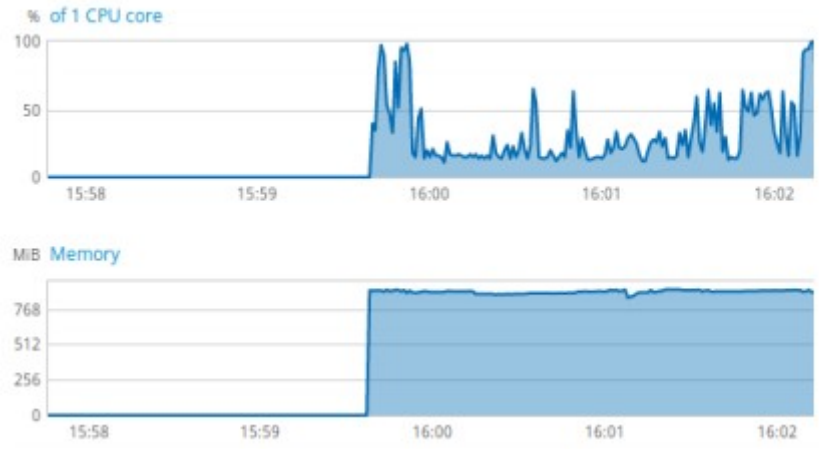

Figure 4 : Moniteur de charge

une fois la bombe exploitée sur le système, notre méthode proposée vérifiera selon le débit illustré à la figure 3. Dans la surveillance de charge ci-dessus, le nom du processus est déjà ajouté dans la liste de détection. Par conséquent, sans vérifier le système, tuer ce processus. Ainsi, cela rend le système disponible.

CONCLUSION

Dans ce projet, nous avons utilisé le mix hybride qui approche dès le stockage du nom du processus pour référence future et mettant les limites des ressources. Il existe plusieurs solutions pour manipuler la fork bomb. Ils ont leurs propres limites. Pourtant, la méthode proposée supprimera les limites de la solution existante, cela affecte légèrement la mémoire. Mais cette exigence de mémoire est faible. Ainsi, en combinant les deux solutions existantes afin de supprimer les limites les uns des autres, nous avons fourni la solution efficace dans ce projet.

Le résultat de l'expérience d'évaluation montre que, une fois le nom du processus ajouté dans la liste de détection, il ne sera pas tué, plutôt que cela, il sera examiné avec des limitations de mémoire. Par conséquent, selon la loi sur la sécurité de l'information, la disponibilité est le traitement maintenu et efficace de la fork bomb est finie.

RÉFÉRENCE :

[1] Gaku Nakagawa, Shuichi Oikawa, “Fork Bomb Attack Mitigation by Process Resource Quarantine”, Fourth International Symposium on Computing and Networking, IEEE 2016.

[2] M. Hareesh, K. Yaswanth, M. Sreeja, Saidalavi Kalady, “Accurate Fork Bomb detection by Process Name”, International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT), IEEE 2017.

[3] Mohiuddin Ali Khan, Sateesh Kumar Pradhan, Huda Fatima, “Applying Data Mining Techniques in Cyber Crimes”, IEEE 2017.

[4] Michele Berlot, Janche Sang, “Dealing with Process Overload Attacks in UNIX”, Information Security Journal: A Global Perspective, 2008.

[5] Ashvini T. Dheshmukh, Dr. Parikshit. N. Mahalle, “Survey on Linux Security and Vulnerabilities”, International Journal of Engineering And Computer Science (IJECS), 2014.

[6] Abraham Silberschatz, Peter Bear Galvin, Gerg Gagne. Operating System Principles. 7th Edition. ISBN: 9812531769.