DM securité OS

Baptiste Rébillard & Aurélien Pouilles

28 janvier 2026

Table des matières

1 Préambule

1.1 Découverte des VMA (Virtual Memory Area)

Les régions de mémoire virtuelle (VMA) sont des structures de données utilisées par le noyau Linux pour gérer l'espace d'adressage d'un processus utilisateur. Chaque processus possède un espace mémoire virtuel divisé en plusieurs régions (segment de code, segment de données, tas, pile, bibliothèques partagées, etc.).

Rôle du noyau : Le noyau maintient une liste de ces VMA pour chaque processus (via la structure `mm_struct`).

1.2 Découverte des structures de données de type Maple Tree

Depuis le noyau 6.1, la gestion des VMA repose sur une structure de données appelée **Maple Tree**.

Le Maple Tree est un arbre de type `B-tree`[1] optimisé pour le stockage de plages de valeurs (*ranges*) qui ne se chevauchent pas.

Un nœud possèdent 16 slots, qui peuvent être des feuilles donc pointent vers des valeurs de type *vm_area_struct*, ou d'autres nœuds. Il y a 16 slots qui correspondent aux 256 octets (soit 4 lignes de caches de 64bits).

Il est particulièrement adapté pour le mécanisme RCU (*Read-Copy-Update*), permettant aux lecteurs de parcourir l'arbre sans verrouiller le système, même pendant qu'une modification (écriture) a lieu. Cela réduit l'utilisation de verrou global de la mémoire (*mmap_lock*), ce qui améliore les performances.

1.3 Découverte du mécanisme de synchronisation Read Copy Update (RCU)

Le RCU (Read-Copy-Update) permet à plusieurs lecteurs d'accéder à des données simultanément sans aucun verrou, garantissant des performances maximales. Lorsqu'une modification est requise, l'écrivain crée une copie modifiée des données et redirige le pointeur vers cette nouvelle version de manière atomique. L'ancienne version est conservée jusqu'à la fin d'une *grace periods*, garantissant qu'aucun lecteur ne l'utilise encore avant sa suppression. Ce mécanisme est idéal pour les structures de données lues très souvent et modifiées rarement.

1.4 Les vulnérabilités de type use-after-free

Une vulnérabilité *Use-After-Free* (UAF) survient lorsqu'un programme utilise un pointeur vers une zone mémoire après sa libération. Si cette mémoire est réallouée à un autre objet, le pointeur initial (devenu un *dangling pointer*) pointe désormais vers de nouvelles données. Accéder à ce pointeur permet alors de lire ou de corrompre ces données, ce qui peut mener à un crash ou à un détournement du flux d'éxecution, donc à de potentielles élévations de privilèges sur un noyau.

2 Analyse de la vulnérabilité StackRot - CVE-2023-3269

2.1 Découverte du problème

Source initiale du problème :

1. arbre trié, optimisé pour permettre une insertion/suppression/recherche en $O(\log n)$ dans le pire des cas.

Le fonctionnement initial du noyau lors de l'expansion automatique de la pile prévoit de modifier directement la structure VMA (qui n'avais pas besoin de vérifier les accès sans verrou notamment RCU).

Lors de la migration de la structure de gestion des VMA vers les Maple Trees, cette manière de fonctionner n'a pas évolué. Ce qui mène à des problématiques d'UAF à cause du mécanisme de RCU permettant d'accéder à de la donnée sans verrou. Alors qu'avant les Maple Trees, cet accès sans verrou n'était pas possible.

Description du problème :

La vulnérabilité réside dans la collision entre la modification *in-place* des limites de la VMA et la gestion structurelle par Maple Tree. Pour etendre la pile, le noyau ajuste le champ *vm_start* en ne détenant que le verrou `mmap_lock` en lecture. Le problème ici est qu'il considére l'opération comme une simple mise à jour de métadonnées. Or, dans l'architecture Maple Tree, toute modification des bornes d'une entrée (*range*) est une opération qui nécessite le remplacement du nœud via le protocole RCU. En modifiant *vm_start* sans passer par ce dernier, le noyau crée une désynchronisation : un lecteur RCU peut entamer un parcours sur un nœud dont la clé de recherche devient invalide ou dont la structure est en cours de rééquilibrage [2]. Ce défaut de barrière de synchronisation permet à un thread concurrent d'accéder à un nœud de l'arbre ayant été libéré par un callback RCU, provoquant ainsi un UAF.

En bref, `vm_start` sert de clé d'indexation dans l'arbre, le modifier revient donc à changer la valeur d'une clé de d'indexation de l'arbre sans réorganiser l'arbre, ce qui corrompt la logique de recherche pour les lecteurs RCU concurrent

2.2 La correction

La correction apportée par Linus Torvalds et les contributeurs du noyau à travers le commit `9471f1f2f50282b9e8f59198ec6bb738b4ccc009` a consisté à modifier la logique d'expansion de la pile pour s'assurer qu'elle respecte les contraintes de verrouillage du Maple Tree.

Le correctif repose sur l'abandon de l'expansion *in-place* historique au profit d'un verrouillage strict en écriture. Le noyau introduit la fonction `lock_mm_and_find_vma()` qui centralise la gestion du verrou : désormais, toute expansion de pile détectée sous un verrou de lecture provoque une montée en privilège ou une réacquisition du `mmap_lock` en mode écriture. Cette transition garantit que la modification des bornes de la VMA est traitée comme une mise à jour de la structure au complet. En forçant l'utilisation des primitives d'écriture prévu du Maple Tree, le correctif assure que les nœuds sont remplacés de manière atomique selon le protocole RCU. Cela empêche les lecteurs concurrents de se retrouver sur un nœud dont la clé (*vm_start*) a été modifiée ou dont la mémoire a été libérée, ce qui permet de supprimer la race condition menant à l'UAF.

2.3 Analyse de l'exploitation

L'exploitation de StackRot (CVE-2023-3269) repose sur la transformation de la condition de course (Race Condition) en un privilège root.

Fonctionnement de l'exploitation :

1. **Trigger (Déclencheur) :** L'attaquant crée un thread qui provoque en boucle l'expansion de la pile (via des appels récursifs ou `alloca`) et un autre thread qui lit le fichier `/proc/self/maps` (ce qui force le parcours de l'arbre des VMA).

2. Le rééquilibrage RCU est une réorganisation de la structure de l'arbre qui utilise des copies de nœuds et des mises à jour atomiques de pointeurs pour que les lecteurs puissent continuer leur parcours sans verrou pendant la restructuration.

2. **Race Condition :** Le but est de faire coïncider l'expansion de la pile (qui libère un nœud de l'arbre via RCU) avec la lecture des VMA. Si le timing est bon, le lecteur détient un pointeur vers un nœud libéré.
3. **Use-After-Free :** Le noyau utilise des délais RCU (Grace Periods). L'attaquant utilise des primitives pour "attendre" la fin de la période RCU afin que la mémoire soit réellement libérée, tout en gardant sa référence active.
4. **Spray du Tas (Heap Spraying) :** Une fois le nœud libéré, l'attaquant "spray" le tas du noyau avec des fausses structures (souvent via `userfaultfd` ou des msg_msg) pour écraser l'espace mémoire du nœud libéré avec ses propres données malveillantes.
5. **Élévation de privilèges :** En contrôlant le contenu du nœud du Maple Tree, l'attaquant peut manipuler les pointeurs internes de l'arbre pour faire pointer des écritures futures vers des structures critiques du noyau (comme les `cred` ou la table des processus), lui permettant finalement d'obtenir les droits root.