Sécurité Système

– Examen (**1**) –

1 CVE-2019-13272 : Élévation de privilèges avec `ptrace` (5 points)

Lisez les paragraphes suivants (à propos des RCU et de `ptrace`) ainsi que le document décrivant la CVE-2019-13272. Puis répondez aux questions.

1.1 À propos des RCU (Read-Copy Update)

« Read-copy update (RCU) is a synchronization mechanism that was added to the Linux kernel in October of 2002. RCU achieves scalability improvements by allowing reads to occur concurrently with updates. In contrast with conventional locking primitives that ensure mutual exclusion among concurrent threads regardless of whether they be readers or updaters, or with reader-writer locks that allow concurrent reads but not in the presence of updates, RCU supports concurrency between a single updater and multiple readers. RCU ensures that reads are coherent by maintaining multiple versions of objects and ensuring that they are not freed up until all pre-existing read-side critical sections complete. RCU defines and uses efficient and scalable mechanisms for publishing and reading new versions of an object, and also for deferring the collection of old versions. These mechanisms distribute the work among read and update paths in such a way as to make read paths extremely fast. In some cases (non-preemptable kernels), RCU's read-side primitives have zero overhead. »

Tiré de "*What is RCU, Fundamentally ?*" sur LWN [1] **Paul McKenney**, 17 Décembre 2007.

1.2 À propos de `ptrace`

« `ptrace()` system call stands for process trace, which provides a way for debuggers such as `gdb`/`strace` to control a process (tracee). "Debuggers" can be any process that sends a `PTRACE_ATTACH`/`PTRACE_SEIZE`, or receives a `PTRACE_TRACEME` from its child. Several things to notice :

1. A tracee's ptrace relationship can be found in `tracee->ptracer_cred`, which, holds tracer's credentials. in the credentials we care about `uid`/`gid`, that decides what this process can do.
2. Under `ptrace`, `execve()` cannot set-UID when executing SUID programs in current ptraced process. and yes, it allows set-UID without `ptrace`.
3. But if the tracee had a privileged tracer, whose uid is stored in `tracee->ptracer_cred`, indicating that the tracee is being traced by a privileged process, the tracee can set-UID to the whatever UID the SUID program holds. its like, when you launch gdb with root, tracing an unprivileged process that executes an SUID program, it has to be allowed, right ?
4. As for 2, the reason is when using ptrace, we can replace the tracee's SUID program with something we control, and keeps the privileged process, meaning we get root.

Now, we know from note 3, the `tracee->ptracer_cred` decides if the tracee's parent is privileged, if yes, we can set-UID. Actually in `kernel/ptrace.c`, a child is able to grab a privileged parent, and force it to become a tracer, thus get a privileged `ptrace` relationship. »

Tiré de "*CVE-2019-13272 : Linux LPE via 'PTRACE_TRACEME'*" [2] `jm33_ng`, 23 Août 2019.

Questions

1. Expliquez le mécanisme set-UID lors d'un appel système `ptrace`.
2. Expliquez quel est le rôle des RCU dans le bug.
3. Expliquez le schéma de l'attaque suggéré par Jan Horn en détaillant chacune des trois étapes.
4. À votre avis, quelle est la portée de ce bug (est-il exploitable, dans quelles conditions, ...) ?

1. Voir : `https://lwn.net/Articles/262464/`
2. Voir : `https://jm33.me/cve-2019-13272-linux-lpe-via-ptrace_traceme.html`

2 Linux Kernel ROP : Ropping your way to # (15 points)

Lisez l'article « *Linux Kernel ROP : Ropping your way to #* » de Vitaly Nikolenko.
Puis répondez aux questions suivantes.

Questions

1. Rappelez le principe du ROP et quel type de protection il contre-carre principalement.
2. Rappelez les principes du SMEP et du SMAP et dans quel contexte ils peuvent empêcher l'exploitation de failles noyau. Expliquez en quoi, l'utilisation d'un ROP permet de contourner une partie des protections du noyau.
3. Rappelez quelle séquence de fonctions du noyau Linux permet de passer root et expliquez (rapidement) ce que font chacune d'entre elle.
4. Expliquez rapidement et dans le cadre général le principe de « *stack pivot* » pour un ROP, expliquez à quoi cela sert et pourquoi il peut s'avérer intéressant d'y avoir recours pendant une exploitation.
5. Expliquez la vulnérabilité prise comme exemple dans l'article (section 4) et comment la déclencher du côté *user-space*.
6. Expliquez plus en détail le *stack pivot* mis en place dans l'article, avec quels gadgets il est réalisé et comment il fonctionne.
7. Détaillez et expliquez la forme et le contenu du payload (section 6). Expliquez aussi comment le SMEP est-il contourné dans ce cas-là.
8. Expliquez ce qui se passe entre au début d'un `syscall` et au retour d'un `iret` (section 7). Et expliquez pourquoi il est nécessaire d'avoir recours à cette fonction `save_state()`.
9. Donnez une autre façon de contourner le SMEP (que nous avons vue en cours).
10. Sur quelles hypothèses repose cette attaque ? Comment pourriez-vous sécuriser le système pour l'éviter (ou la rendre plus difficile) ? Quel impact aurait le KASLR s'il avait été activé ?