☆ Starred by 2 users

Owner:	jannh@google.com
CC:	proje...@google.com
Status:	Fixed *(Closed)*
Components:	----
Modified:	Oct 10, 2022

Deadline-90
Vendor-Linux
CCProjectZeroMembers
Severity-High
Finder-jannh
Product-Linux
Methodology-source-review
Reported-2022-Aug-30
Fixed-2022-Sep-5
CVE-2022-42703

Participant's Hotlists:	linux-usermm-or-drivermm

Issue 2351: Linux >=3.19: anon_vma UAF through bogus merge of VMAs caused by double-reuse of leaf anon_vma because of ->degree misinterpretation

Reported by jannh@google.com on Mon, Aug 29, 2022, 6:56 PM EDT Project Member

Code
1 of 13
Back to list

(If you don't want to read the whole bug report: A suggested patch is attached as 0001-mm-rmap-Fix-anon_vma-degree-ambiguity-leading-to-dou.patch, and its commit message summarizes the issue.)

The rough concept behind struct anon_vma is:
The kernel needs rmap data structures to be able to figure out all the places where a given page might be mapped.
For file-backed pages, this is done through the inode's address_space; for anonymous pages, the equivalent is struct anon_vma.
(page->mapping can point to either one, and a low bit in the pointer signals what the type is.)
struct anon_vma provides a way to iterate through all VMAs that map it using its rb_root.

When a process is forked, it would conceptually work to keep using the same anon_vma in both the parent's VMA and the child's VMA; however, that would mean that trying to iterate all mappings of a page that's part of either one would involve scanning through both VMAs.

So instead, on fork(), the child creates a second anon_vma. The child's VMA will use the second anon_vma for new page allocations, but must be tied to both anon_vma instances since it could contain pages that were created under both anon_vmas.
From here on, I will refer to the anon_vma that a VMA uses for new page allocations as the VMA's "primary" anon_vma, and refer to other anon_vmas associated with a VMA as the VMA's "secondary" anon_vmas.

This approach comes with a problem: If a process does something like this:

```
while (1) {
  int child = fork();
  if (child == -1) [...];
  if (child > 0) exit(0);
}
```

then the anonymous VMAs will keep accumulating more secondary anon_vmas.

Commit 7a3ef208e662 ("mm: prevent endless growth of anon_vma hierarchy") tried to fix the unbounded memory usage growth in this edgecase.
The rmap code now *attempts* to implement the following approach (as explained in a comment in anon_vma_clone()):

Before allocating a new anon_vma, try to find an existing anon_vma

Before allocating a new anon_vma, try to find an existing anon_vma that:

1. is not any VMA's primary anon_vma
2. does not have more than one child anon_vma
3. is not the root anon_vma

If such an anon_vma exists, reuse it.

But these checks are actually implemented using a single field ->degree, which counts the number of VMAs using the anon_vma as their primary anon_vma PLUS the number of child anon_vmas PLUS a bias of 1 for the root anon_vma:

```
/*
 * Reuse existing anon_vma if its degree lower than two,
 * that means it has no vma and only one anon_vma child.
 *
 * Do not choose parent anon_vma, otherwise first child
 * will always reuse it. Root anon_vma is never reused:
 * it has self-parent reference and at least one child.
 */
if (!dst->anon_vma && src->anon_vma &&
    anon_vma != src->anon_vma && anon_vma->degree < 2)
        dst->anon_vma = anon_vma;
```

This check assumes that the source VMA can not have any secondary anon_vmas that are leaf nodes (zero children), and that therefore `degree < 2` implies that no VMAs use the anon_vma as their primary anon_vma.

This is incorrect because the ->degree optimization can pick a previous secondary anon_vma as the primary anon_vma in a forked child, turning the previous primary anon_vma leaf node into a secondary anon_vma leaf node.

Therefore, we can end up with two VMAs independently reusing the same anon_vma as their primary anon_vma, even though they might have different sets of secondary anon_vmas.

That's a problem because the VMA merging code (in particular is_mergeable_anon_vma()) assumes that equal primary anon_vmas imply that all secondary anon_vmas are also equal. By violating that assumption, it becomes possible to merge two VMAs with different secondary anon_vmas such that the resulting VMA contains pages whose anon_vmas are not associated with the VMA. Since VMAs are responsible for keeping anon_vmas alive, this then leads to a situation where mapped pages have a dangling page->mapping pointer, which leads to use-after-free when the kernel tries to walk the page's rmap (for example as part of swapout).

I have written a simple reproducer (attached as forkforkfork.c):

I have written a simple reproducer (attached as forkforkfork.c);
however, it requires a small kernel patch (see attachment) to
allow the kernel to actually notice the use-after-free access,
because anon_vma is allocated from a SLAB_TYPESAFE_BY_RCU slab,
and KASAN can't detect use-after-free in such slabs unless
the whole slab page is freed.
The attached patch ensures that each anon_vma has its own
slab page by crudely increasing the allocation size to 0x8000.
It also adds some debug logging to show the anon_vma tree
structures.

You'll also have to enable CONFIG_KASAN and CONFIG_SLUB_DEBUG
in the kernel config and pass slub_debug=F on the kernel
command line to get a nice splat.

Here's the splat:

```
==================================================================
BUG: KASAN: use-after-free in folio_lock_anon_vma_read (mm/rmap.c:548)
Read of size 8 at addr ffff888034990000 by task forkforkfork/717
[ 1383.694388]
CPU: 2 PID: 717 Comm: forkforkfork Not tainted 6.0.0-rc1-00025-g274a2eebf80c-dirty #81
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.16.0-debian-1.16.0-4 04/01/2014
Call Trace:
 <TASK>
dump_stack_lvl (lib/dump_stack.c:107 (discriminator 1))
print_report.cold (mm/kasan/report.c:318 mm/kasan/report.c:433)
[...]
kasan_report (mm/kasan/report.c:162 mm/kasan/report.c:497)
[...]
folio_lock_anon_vma_read (mm/rmap.c:548)
rmap_walk_anon (mm/rmap.c:2355 mm/rmap.c:2402)
[...]
folio_referenced (mm/rmap.c:2496 mm/rmap.c:934)
[...]
shrink_page_list (./include/linux/instrumented.h:101 ./include/asm-generic/bitops/instrumented-atomic.h:85
./include/linux/page-flags.h:477 mm/vmscan.c:1441 mm/vmscan.c:1742)
[...]
reclaim_page_list (mm/vmscan.c:2620)
[...]
reclaim_pages (./include/linux/sched/mm.h:348 mm/vmscan.c:2658)
[...]
madvise_cold_or_pageout_pte_range (mm/madvise.c:490)
[...]
__walk_page_range (mm/pagewalk.c:128 mm/pagewalk.c:205 mm/pagewalk.c:240 mm/pagewalk.c:277
mm/pagewalk.c:379)
walk_page_range (mm/pagewalk.c:476)
[...]
madvise_pageout (./include/asm-generic/tlb.h:502 mm/madvise.c:549 mm/madvise.c:585)
[...]

do_madvise.part.0 (mm/madvise.c:1006 mm/madvise.c:1232 mm/madvise.c:1410)
[...]
__x64_sys_madvise (mm/madvise.c:1422 mm/madvise.c:1421 mm/madvise.c:1421)
```

```
__x64_sys_madvise (mm/madvise.c:1423 mm/madvise.c:1421 mm/madvise.c:1421)
[...]
do_syscall_64 (arch/x86/entry/common.c:50 arch/x86/entry/common.c:80)
entry_SYSCALL_64_after_hwframe (arch/x86/entry/entry_64.S:120)
[...]
 </TASK>
[ 1383.766200]
The buggy address belongs to the physical page:
page:ffffea0000d26400 refcount:0 mapcount:-128 mapping:0000000000000000 index:0xffff888034990000 pfn:0x34990
flags: 0x100000000000000(node=0|zone=1)
raw: 0100000000000000 ffffea0000c9a008 ffffea0000ced408 0000000000000000
raw: ffff888034990000 0000000000000004 00000000ffffff7f 0000000000000000
[...]
==================================================================
```

The reproducer prints the anon_vma hierarchies of the relevant VMAs
thanks to some extra debug logging that the reproducer-helper-patch
adds:

We start with task T1. T1 creates the victim VMA, populates its
anon_vma, then T1 forks to create T2, T2 forks to create T3,
T2 exits, T3 forks to create T4.
T3 then splits its VMA into two to bump its ->degree from 1 to 2.

Now T4 contains a VMA that looks like this - the "#" marks the
primary anon_vma, indentation shows the tree hierarchy:

```
00100000-0010f000 rwxp 00000000 00:00 0
   [  0] ffff88800f670000 refcount=3 anon_vmas=4 degree=3 parent=ffff88800f670000
 #   [  1] ffff88800f8e0000 refcount=1 anon_vmas=3 degree=2 parent=ffff88800f670000
       [  2] ffff88802e230000 refcount=1 anon_vmas=3 degree=2 parent=ffff88800f8e0000
```

Now the VMA is split into two adjacent VMAs that still have
identical anon_vma information:

```
00100000-00108000 rwxp 00000000 00:00 0
   [  0] ffff88800f670000 refcount=3 anon_vmas=5 degree=3 parent=ffff88800f670000
 #   [  1] ffff88800f8e0000 refcount=1 anon_vmas=4 degree=3 parent=ffff88800f670000
       [  2] ffff88802e230000 refcount=1 anon_vmas=4 degree=2 parent=ffff88800f8e0000
00108000-0010f000 rw-p 00000000 00:00 0
   [  0] ffff88800f670000 refcount=3 anon_vmas=5 degree=3 parent=ffff88800f670000
 #   [  1] ffff88800f8e0000 refcount=1 anon_vmas=4 degree=3 parent=ffff88800f670000
       [  2] ffff88802e230000 refcount=1 anon_vmas=4 degree=2 parent=ffff88800f8e0000
```

Then T4 forks to create T5, which allocates separate new primary anon_vmas for both VMAs - note that the
layout of the VMA trees still looks the same, but the pointers for the anon_vmas with index 3 are different:

```
00100000-00108000 rwxp 00000000 00:00 0
AnonVMAs:
   [  0] ffff88800f670000 refcount=5 anon_vmas=7 degree=3 parent=ffff88800f670000

     [  1] ffff88800f8e0000 refcount=1 anon_vmas=6 degree=5 parent=ffff88800f670000
       [  2] ffff88802e230000 refcount=1 anon_vmas=6 degree=2 parent=ffff88800f8e0000
 #     [  3] ffff8880159e0000 refcount=1 anon_vmas=1 degree=1 parent=ffff88800f8e0000
 #     [  3] ffff8880159e0000 refcount=1 anon_vmas=1 degree=1 parent=ffff88800f8e0000
00108000-0010f000 rw-p 00000000 00:00 0
   [  0] ffff88800f670000 refcount=5 anon_vmas=7 degree=3 parent=ffff88800f670000
     [  1] ffff88800f8e0000 refcount=1 anon_vmas=6 degree=5 parent=ffff88800f670000
       [  2] ffff88802e230000 refcount=1 anon_vmas=6 degree=2 parent=ffff88800f8e0000
 #     [  3] ffff888034990000 refcount=1 anon_vmas=1 degree=1 parent=ffff88800f8e0000
```

Then T3 exits, which reduces the degree of the anon_vma ffff88802e230000 by 2 - the dump of the VMAs in T5 now looks like this:

```
00100000-00108000 rwxp 00000000 00:00 0
   [  0] ffff88800f670000 refcount=5 anon_vmas=3 degree=3 parent=ffff88800f670000
     [  1] ffff88800f8e0000 refcount=1 anon_vmas=2 degree=3 parent=ffff88800f670000
       [  2] ffff88802e230000 refcount=1 anon_vmas=2 degree=0 parent=ffff88800f8e0000
 #     [  3] ffff8880159e0000 refcount=1 anon_vmas=1 degree=1 parent=ffff88800f8e0000
00108000-0010f000 rw-p 00000000 00:00 0
   [  0] ffff88800f670000 refcount=5 anon_vmas=3 degree=3 parent=ffff88800f670000
     [  1] ffff88800f8e0000 refcount=1 anon_vmas=2 degree=3 parent=ffff88800f670000
       [  2] ffff88802e230000 refcount=1 anon_vmas=2 degree=0 parent=ffff88800f8e0000
 #     [  3] ffff888034990000 refcount=1 anon_vmas=1 degree=1 parent=ffff88800f8e0000
```

Now T5 forks again to create T6, which now has two VMAs that both reused the same anon_vma:

```
00100000-00108000 rwxp 00000000 00:00 0
   [  0] ffff88800f670000 refcount=5 anon_vmas=3 degree=3 parent=ffff88800f670000
     [  1] ffff88800f8e0000 refcount=1 anon_vmas=2 degree=3 parent=ffff88800f670000
 #     [  2] ffff88802e230000 refcount=1 anon_vmas=2 degree=2 parent=ffff88800f8e0000
       [  3] ffff8880159e0000 refcount=1 anon_vmas=1 degree=0 parent=ffff88800f8e0000
00108000-0010f000 rw-p 00000000 00:00 0
   [  0] ffff88800f670000 refcount=5 anon_vmas=3 degree=3 parent=ffff88800f670000
     [  1] ffff88800f8e0000 refcount=1 anon_vmas=2 degree=3 parent=ffff88800f670000
 #     [  2] ffff88802e230000 refcount=1 anon_vmas=2 degree=2 parent=ffff88800f8e0000
       [  3] ffff888034990000 refcount=1 anon_vmas=1 degree=0 parent=ffff88800f8e0000
```

And then T6 can use mprotect() to trigger merging of the two VMAs, which causes the secondary anon_vma ffff888034990000 of the second VMA to be lost:

```
00100000-0010f000 rwxp 00000000 00:00 0
   [  0] ffff88800f670000 refcount=4 anon_vmas=2 degree=3 parent=ffff88800f670000
     [  1] ffff88800f8e0000 refcount=1 anon_vmas=1 degree=2 parent=ffff88800f670000
 #     [  2] ffff88802e230000 refcount=1 anon_vmas=1 degree=1 parent=ffff88800f8e0000
       [  3] ffff8880159e0000 refcount=1 anon_vmas=1 degree=0 parent=ffff88800f8e0000
```

I've attached a suggested patch that fixes the issue by replacing the ->degree
field with two fields that separately track the number of child anon_vmas (not
safety-relevant) and the number of VMAs that use the anon_vma as their primary
anon_vma.
As a follow-up, we should probably later add some comments to point out the safety
implications of anon_vma reuse and merging.

**This bug is subject to a 90-day disclosure deadline. If a fix for this
issue is made available to users before the end of the 90-day deadline**

issue is made available to users before the end of the 90-day deadline, this bug report will become public 30 days after the fix was made available. Otherwise, this bug report will become public at the deadline. The scheduled deadline is 2022-11-28.

0001-mm-rmap-Fix-anon_vma-degree-ambiguity-leading-to-dou.patch
6.0 KB View Download

0001-HACK-make-anon_vma-testing-easier.patch
2.6 KB View Download

forkforkfork.c
3.6 KB View Download

Comment 1 by jannh@google.com on Thu, Sep 1, 2022, 2:38 PM EDT Project Member

Fix is in Linus' tree: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=2555283eb40d

Comment 2 by jannh@google.com on Mon, Sep 19, 2022, 1:28 PM EDT Project Member

Status: Fixed (was: New)
Labels: Fixed-2022-Sep-5

Fixed in:
v5.19.7 (2022-09-05)
v5.15.65 (2022-09-05)
v5.10.141 (2022-09-05)
v5.4.212 (2022-09-05)
v4.19.257 (2022-09-05)
v4.14.292 (2022-09-05)
v4.9.327 (2022-09-05)

Comment 3 by jannh@google.com on Wed, Oct 5, 2022, 12:56 PM EDT Project Member

Labels: -Restrict-View-Commit

Comment 4 by jannh@google.com on Mon, Oct 10, 2022, 9:26 AM EDT Project Member

Labels: CVE-2022-42703