

[patch 10/14] remap_file_pages protection support: fix get_user_pages() on VM_MANYPROTS vmas

[patch 10/14] remap_file_pages protection support: fix get_user_pages() on VM_MANYPROTS vmas

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2006-04/msg06745.html>

- *From:* blaisorblade@xxxxxxxx
 - *Date:* Sun, 30 Apr 2006 19:30:03 +0200
-

From: Paolo 'Blaisorblade' Giarrusso <blaisorblade@xxxxxxxx>

Untested patch – I've not written a test case to verify functionality of ptrace on VM_MANYPROTS area.

get_user_pages may well call __handle_mm_fault() wanting to override protections... so in this case __handle_mm_fault() should still avoid checking VM access rights.

Also, get_user_pages() may give write faults on present readonly PTEs in VM_MANYPROTS areas (think of PTRACE_POKETEXT), so we must still do do_wp_page even on VM_MANYPROTS areas.

So, possibly use VM_MAYWRITE and/or VM_MAYREAD in the access_mask and check VM_MANYPROTS in maybe_mkwrite_file (new variant of maybe_mkwrite).

API Note: there are many flags parameter which can be constructed but which don't make any sense, but the code very freely interprets them too. For instance VM_MAYREAD|VM_WRITE is interpreted as VM_MAYWRITE|VM_WRITE.

XXX: Todo: add checking to reject all meaningless flag combination.

Index: linux-2.6.git/mm/memory.c

```
=====
--- linux-2.6.git.orig/mm/memory.c
+++ linux-2.6.git/mm/memory.c
@@ -1045,16 +1045,17 @@ int get_user_pages(struct task_struct *t
cond_resched());
while (!(page = follow_page(vma, start, foll_flags))) {
int ret;
- ret = __handle_mm_fault(mm, vma, start,
- foll_flags & FOLL_WRITE);
+ ret = __handle_mm_fault(mm, vma, start, vm_flags);
/*
* The VM_FAULT_WRITE bit tells us that do_wp_page has
* broken COW when necessary, even if maybe_mkwrite
* decided not to set pte_write. We can thus safely do
* subsequent page lookups as if they were reads.
*/
```

[patch 10/14] remap_file_pages protection support: fix get_user_pages() on VM_MANYPROTS vmas

[patch 10/14] remap_file_pages protection support: fix get_user_pages() on VM_MANYPROTS vmas

```
- if (ret & VM_FAULT_WRITE)
+ if (ret & VM_FAULT_WRITE) {
foll_flags &= ~FOLL_WRITE;
+ vm_flags &= ~(VM_WRITE|VM_MAYWRITE);
+ }

switch (ret & ~VM_FAULT_WRITE) {
case VM_FAULT_MINOR:
@@ -1389,7 +1390,20 @@ static inline int pte_unmap_same(struct
* servicing faults for write access. In the normal case, do always want
* pte_mkwrite. But get_user_pages can cause write faults for mappings
* that do not have writing enabled, when used by access_process_vm.
+ *
+ * Also, we must never change protections on VM_MANYPROTS pages; that's only
+ * allowed in do_no_page(), so test only VMA protections there. For other cases
+ * we *know* that VM_MANYPROTS is clear, such as anonymous/swap pages, and in
+ * that case using plain maybe_mkwrite() is an optimization.
+ * Instead, when we may be mapping a file, we must use maybe_mkwrite_file.
*/
+static inline pte_t maybe_mkwrite_file(pte_t pte, struct vm_area_struct *vma)
+{
+ if (likely((vma->vm_flags & (VM_WRITE | VM_MANYPROTS)) == VM_WRITE))
+ pte = pte_mkwrite(pte);
+ return pte;
+}
+
static inline pte_t maybe_mkwrite(pte_t pte, struct vm_area_struct *vma)
{
if (likely(vma->vm_flags & VM_WRITE))
@@ -1441,6 +1455,9 @@ static inline void cow_user_page(struct
* We enter with non-exclusive mmap_sem (to exclude vma changes,
* but allow concurrent faults), with pte both mapped and locked.
* We return with mmap_sem still held, but pte unmapped and unlocked.
+ *
+ * Note that a page here can be a shared readonly page where
+ * get_user_pages() (for instance for ptrace()) wants to write to it!
*/
static int do_wp_page(struct mm_struct *mm, struct vm_area_struct *vma,
unsigned long address, pte_t *page_table, pmd_t *pmd,
@@ -1460,7 +1477,8 @@ static int do_wp_page(struct mm_struct *
if (reuse) {
flush_cache_page(vma, address, pte_pfn(orig_pte));
entry = pte_mkyoung(orig_pte);
- entry = maybe_mkwrite(pte_mkdirty(entry), vma);
+ /* Since it can be shared, it can be VM_MANYPROTS! */
+ entry = maybe_mkwrite_file(pte_mkdirty(entry), vma);
ptep_set_access_flags(vma, address, page_table, entry, 1);
update_mmu_cache(vma, address, entry);
lazy_mmu_prot_update(entry);
@@ -1504,7 +1522,7 @@ gotten:
inc_mm_counter(mm, anon_rss);
```

[patch 10/14] remap_file_pages protection support: fix get_user_pages() on VM_MANYPROTS vmas

[patch 10/14] remap_file_pages protection support: fix get_user_pages() on VM_MANYPROTS vmas

```
flush_cache_page(vma, address, pte_pfn(orig_pte));
entry = mk_pte(new_page, vma->vm_page_prot);
- entry = maybe_mkwrite(pte_mkdirty(entry), vma);
+ entry = maybe_mkwrite_file(pte_mkdirty(entry), vma);
ptep_establish(vma, address, page_table, entry);
update_mmu_cache(vma, address, entry);
lazy_mmu_prot_update(entry);
@@ -1930,7 +1948,7 @@ again:
inc_mm_counter(mm, anon_rss);
pte = mk_pte(page, vma->vm_page_prot);
if (write_access && can_share_swap_page(page)) {
- pte = maybe_mkwrite(pte_mkdirty(pte), vma);
+ pte = maybe_mkwrite_file(pte_mkdirty(pte), vma);
write_access = 0;
}

@@ -2231,15 +2249,15 @@ static inline int handle_pte_fault(struc
pte_t entry;
pte_t old_entry;
spinlock_t *ptl;
- int write_access = access_mask & VM_WRITE;
+ int write_access = access_mask & (VM_WRITE|VM_MAYWRITE);

old_entry = entry = *pte;
if (!pte_present(entry)) {
/* when pte_file(), the VMA protections are useless. Otherwise,
* we need to check VM_MANYPROTS, because in that case the arch
* fault handler skips the VMA protection check. */
- if (!pte_file(entry) && check_perms(vma, access_mask))
- goto out_segv;
+ if (!pte_file(entry) && unlikely(check_perms(vma, access_mask)))
+ goto segv;

if (pte_none(entry)) {
if (!vma->vm_ops || !vma->vm_ops->nopage)
@@ -2269,12 +2287,14 @@ static inline int handle_pte_fault(struc
pgprot_t pgprot = pte_to_pgprot(*pte);
pte_t test_entry = pfn_pte(0, pgprot);

- if (unlikely((access_mask & VM_WRITE) && !pte_write(test_entry)))
- goto out_segv;
- if (unlikely((access_mask & VM_READ) && !pte_read(test_entry)))
- goto out_segv;
- if (unlikely((access_mask & VM_EXEC) && !pte_exec(test_entry)))
- goto out_segv;
+ if (likely(!(access_mask & (VM_MAYWRITE|VM_MAYREAD)))) {
+ if (unlikely((access_mask & VM_WRITE) && !pte_write(test_entry)))
+ goto segv_unlock;
+ if (unlikely((access_mask & VM_READ) && !pte_read(test_entry)))
+ goto segv_unlock;
+ if (unlikely((access_mask & VM_EXEC) && !pte_exec(test_entry)))
```

[patch 10/14] remap_file_pages protection support: fix get_user_pages() on VM_MANYPROTS vmas

[patch 10/14] remap_file_pages protection support: fix get_user_pages() on VM_MANYPROTS vmas

```
+ goto segv_unlock;
+ }
}
```

```
if (write_access) {
@@ -2301,8 +2321,10 @@ static inline int handle_pte_fault(struct
unlock:
pte_unmap_unlock(pte, ptl);
return VM_FAULT_MINOR;
-out_segv:
+
+segv_unlock:
pte_unmap_unlock(pte, ptl);
+segv:
return VM_FAULT_SIGSEGV;
}
```

Index: linux-2.6.git/include/linux/mm.h

```
-----
--- linux-2.6.git.orig/include/linux/mm.h
+++ linux-2.6.git/include/linux/mm.h
@@ -734,7 +734,22 @@ extern int install_file_pte(struct mm_st
```

```
#ifdef CONFIG_MMU
```

```
/* We reuse VM_READ, VM_WRITE and VM_EXEC for the @access_mask. */
/* We reuse VM_READ, VM_WRITE and (optionally) VM_EXEC for the @access_mask, to
 * report the kind of access we request for permission checking, in case the VMA
 * is VM_MANYPROTS.
 *
 * get_user_pages( force == 1 ) is a special case. It's allowed to override
 * protection checks, even on VM_MANYPROTS vma.
 *
 * To express that, it must replace VM_READ / VM_WRITE with the corresponding
 * MAY flags.
 * This allows to force copying COW pages to break sharing even on read-only
 * page table entries.
 * PITFALL: you're not allowed to override only part of the checks, and in
 * general specifying strange combinations of flags may lead to unspecified
 * results.
 */
extern int __handle_mm_fault(struct mm_struct *mm, struct vm_area_struct *vma,
unsigned long address, int access_mask);
```

```
---
Inform me of my mistakes, so I can keep imitating Homer Simpson's "Doh!".
Paolo Giarrusso, aka Blaisorblade (Skype ID "PaoloGiarrusso", ICQ 215621894)
http://www.user-mode-linux.org/~blaisorblade
```

[patch 10/14] remap_file_pages protection support: fix get_user_pages() on VM_MANYPROTS vmas

[patch 10/14] remap_file_pages protection support: fix get_user_pages() on VM_MANYPROTS vmas

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
the body of a message to majordomo@xxxxxxxxxxxxxxxxx
More majordomo info at <http://vger.kernel.org/majordomo-info.html>
Please read the FAQ at <http://www.tux.org/lkml/>