
Subject: [PATCH 9/13] BC: locked pages (charge hooks)

Posted by [dev](#) on Tue, 05 Sep 2006 15:25:53 GMT

[View Forum Message](#) <> [Reply to Message](#)

Introduce calls to BC core over the kernel to charge locked memory.

Normaly new locked piece of memory may appear in insert_vm_struct, but there are places (do_mmap_pgoff, dup_mmap etc) when new vma is not inserted by insert_vm_struct(), but either link_vma-ed or merged with some other - these places call BC code explicitly.

Plus sys_mlock[all] itself has to be patched to charge/uncharge needed amount of pages.

Signed-Off-By: Pavel Emelianov <xemul@sw.ru>

Signed-Off-By: Kirill Korotaev <dev@sw.ru>

```
fs/binfmt_elf.c      |  5 ++-
include/asm-alpha/mman.h |  1
include/asm-generic/mman.h |  1
include/asm-mips/mman.h |  1
include/asm-parisc/mman.h |  1
include/linux/mm.h    |  1
mm/mlock.c            | 21 ++++++++-----
mm/mmap.c             | 59 +++++++++++++++++++++++++++++++++++++-----
mm/mremap.c           | 18 ++++++++--
mm/shmem.c            | 12 ++++++-
10 files changed, 104 insertions(+), 16 deletions(-)
```

--- ./fs/binfmt_elf.c.bclockcharge 2006-09-05 12:53:54.000000000 +0400

+++ ./fs/binfmt_elf.c 2006-09-05 13:08:26.000000000 +0400

@@ -360,7 +360,7 @@ static unsigned long load_elf_interp(str

epnt = elf_phdata;

for (i = 0; i < interp_elf_ex->e_phnum; i++, epnt++) {

if (epnt->p_type == PT_LOAD) {

- int elf_type = MAP_PRIVATE | MAP_DENYWRITE;

+ int elf_type = MAP_PRIVATE | MAP_DENYWRITE | MAP_EXECPRIO;

int elf_prot = 0;

unsigned long vaddr = 0;

unsigned long k, map_addr;

@@ -846,7 +846,8 @@ static int load_elf_binary(struct linux_

if (elf_ppnt->p_flags & PF_X)

elf_prot |= PROT_EXEC;

- elf_flags = MAP_PRIVATE | MAP_DENYWRITE | MAP_EXECUTABLE;

+ elf_flags = MAP_PRIVATE | MAP_DENYWRITE |

```

+ MAP_EXECUTABLE | MAP_EXECPRIO;

vaddr = elf_ppnt->p_vaddr;
if (loc->elf_ex.e_type == ET_EXEC || load_addr_set) {
--- ./include/asm-alpha/mman.h.mapfx 2006-04-21 11:59:35.000000000 +0400
+++ ./include/asm-alpha/mman.h 2006-09-05 18:13:12.000000000 +0400
@@ -28,6 +28,7 @@
#define MAP_NORESERVE 0x10000 /* don't check for reservations */
#define MAP_POPULATE 0x20000 /* populate (prefault) pagetables */
#define MAP_NONBLOCK 0x40000 /* do not block on IO */
+#define MAP_EXECPRIO 0x80000 /* charge against BC limit */

#define MS_ASYNC 1 /* sync memory asynchronously */
#define MS_SYNC 2 /* synchronous memory sync */
--- ./include/asm-generic/mman.h.x 2006-04-21 11:59:35.000000000 +0400
+++ ./include/asm-generic/mman.h 2006-09-05 14:02:04.000000000 +0400
@@ -19,6 +19,7 @@
#define MAP_TYPE 0x0f /* Mask for type of mapping */
#define MAP_FIXED 0x10 /* Interpret addr exactly */
#define MAP_ANONYMOUS 0x20 /* don't use a file */
+#define MAP_EXECPRIO 0x20000 /* charge againsts BC_LIMIT */

#define MS_ASYNC 1 /* sync memory asynchronously */
#define MS_INVALIDATE 2 /* invalidate the caches */
--- ./include/asm-mips/mman.h.mapfx 2006-04-21 11:59:36.000000000 +0400
+++ ./include/asm-mips/mman.h 2006-09-05 18:13:34.000000000 +0400
@@ -46,6 +46,7 @@
#define MAP_LOCKED 0x8000 /* pages are locked */
#define MAP_POPULATE 0x10000 /* populate (prefault) pagetables */
#define MAP_NONBLOCK 0x20000 /* do not block on IO */
+#define MAP_EXECPRIO 0x40000 /* charge against BC limit */

/*
 * Flags for msync
--- ./include/asm-parisc/mman.h.mapfx 2006-04-21 11:59:36.000000000 +0400
+++ ./include/asm-parisc/mman.h 2006-09-05 18:13:47.000000000 +0400
@@ -22,6 +22,7 @@
#define MAP_GROWSDOWN 0x8000 /* stack-like segment */
#define MAP_POPULATE 0x10000 /* populate (prefault) pagetables */
#define MAP_NONBLOCK 0x20000 /* do not block on IO */
+#define MAP_EXECPRIO 0x40000 /* charge against BC limit */

#define MS_SYNC 1 /* synchronous memory sync */
#define MS_ASYNC 2 /* sync memory asynchronously */
--- ./include/linux/mm.h.bclockcharge 2006-09-05 12:55:28.000000000 +0400
+++ ./include/linux/mm.h 2006-09-05 13:06:37.000000000 +0400
@@ -1103,6 +1103,7 @@ out:
extern int do_munmap(struct mm_struct *, unsigned long, size_t);

```

```

extern unsigned long do_brk(unsigned long, unsigned long);
+extern unsigned long __do_brk(unsigned long, unsigned long, int);

/* filemap.c */
extern unsigned long page_unuse(struct page *);
--- ./mm/mlock.c.bclockcharge 2006-04-21 11:59:36.000000000 +0400
+++ ./mm/mlock.c 2006-09-05 13:06:37.000000000 +0400
@@ -11,6 +11,7 @@
#include <linux/mempolicy.h>
#include <linux/syscalls.h>

+#include <bc/vmpages.h>

static int mlock_fixup(struct vm_area_struct *vma, struct vm_area_struct **prev,
    unsigned long start, unsigned long end, unsigned int newflags)
@@ -25,6 +26,14 @@ static int mlock_fixup(struct vm_area_st
    goto out;
}

+ if (newflags & VM_LOCKED) {
+     ret = bc_locked_charge(mm, end - start);
+     if (ret < 0) {
+         *prev = vma;
+         goto out;
+     }
+ }
+
pgoff = vma->vm_pgoff + ((start - vma->vm_start) >> PAGE_SHIFT);
*prev = vma_merge(mm, *prev, start, end, newflags, vma->anon_vma,
    vma->vm_file, pgoff, vma_policy(vma));
@@ -38,13 +47,13 @@ static int mlock_fixup(struct vm_area_st
    if (start != vma->vm_start) {
        ret = split_vma(mm, vma, start, 1);
        if (ret)
-         goto out;
+         goto out_uncharge;
    }

    if (end != vma->vm_end) {
        ret = split_vma(mm, vma, end, 0);
        if (ret)
-         goto out;
+         goto out_uncharge;
    }

success:
@@ -63,13 +72,19 @@ success:

```

```

    pages = -pages;
    if (!(newflags & VM_IO))
        ret = make_pages_present(start, end);
- }
+ } else
+ bc_locked_uncharge(mm, end - start);

    vma->vm_mm->locked_vm -= pages;
out:
    if (ret == -ENOMEM)
        ret = -EAGAIN;
    return ret;
+
+out_uncharge:
+ if (newflags & VM_LOCKED)
+ bc_locked_uncharge(mm, end - start);
+ goto out;
}

static int do_mlock(unsigned long start, size_t len, int on)
--- ./mm/mmap.c.bclockcharge 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/mmap.c 2006-09-05 13:07:13.000000000 +0400
@@ -26,6 +26,8 @@
#include <linux/mempolicy.h>
#include <linux/rmap.h>

+#include <bc/vmpages.h>
+
#include <asm/uaccess.h>
#include <asm/cacheflush.h>
#include <asm/tlb.h>
@@ -220,6 +222,10 @@ static struct vm_area_struct *remove_vma
    struct vm_area_struct *next = vma->vm_next;

    might_sleep();
+
+ bc_memory_uncharge(vma->vm_mm, vma->vm_end - vma->vm_start,
+ vma->vm_flags, vma->vm_file);
+
    if (vma->vm_ops && vma->vm_ops->close)
        vma->vm_ops->close(vma);
    if (vma->vm_file)
@@ -267,7 +273,7 @@ asmlinkage unsigned long sys_brk(unsigned
    goto out;

    /* Ok, looks good - let it rip. */
- if (do_brk(oldbrk, newbrk-oldbrk) != oldbrk)
+ if (__do_brk(oldbrk, newbrk-oldbrk, BC_BARRIER) != oldbrk)

```

```

    goto out;
set_brk:
    mm->brk = brk;
@@ -1047,6 +1053,11 @@ munmap_back:
}
}

+ error = bc_memory_charge(mm, len, vm_flags, file,
+ flags & MAP_EXECPRIO ? BC_LIMIT : BC_BARRIER);
+ if (error)
+ goto charge_fail;
+
/*
 * Can we just expand an old private anonymous mapping?
 * The VM_SHARED test is necessary because shmem_zero_setup
@@ -1160,6 +1171,8 @@ unmap_and_free_vma:
free_vma:
    kmem_cache_free(vm_area_cachep, vma);
unacct_error:
+ bc_memory_uncharge(mm, len, vm_flags, file);
+charge_fail:
    if (charged)
        vm_unacct_memory(charged);
    return error;
@@ -1489,12 +1502,16 @@ static int acct_stack_growth(struct vm_a
    return -ENOMEM;
}

+ if (bc_memory_charge(mm, grow << PAGE_SHIFT,
+ vma->vm_flags, vma->vm_file, BC_LIMIT))
+ goto err_ch;
+
/*
 * Overcommit.. This must be the final test, as it will
 * update security statistics.
 */
    if (security_vm_enough_memory(grow))
- return -ENOMEM;
+ goto err_acct;

/* Ok, everything looks good - let it rip */
    mm->total_vm += grow;
@@ -1502,6 +1519,11 @@ static int acct_stack_growth(struct vm_a
    mm->locked_vm += grow;
    vm_stat_account(mm, vma->vm_flags, vma->vm_file, grow);
    return 0;
+
+err_acct:

```

```

+ bc_memory_uncharge(mm, grow << PAGE_SHIFT, vma->vm_flags, vma->vm_file);
+err_ch:
+ return -ENOMEM;
}

#ifdef CONFIG_STACK_GROWSUP || defined(CONFIG_IA64)
@@ -1857,7 +1879,7 @@ static inline void verify_mm_writelocked
 * anonymous maps. eventually we may be able to do some
 * brk-specific accounting here.
 */
-unsigned long do_brk(unsigned long addr, unsigned long len)
+unsigned long __do_brk(unsigned long addr, unsigned long len, int bc_strict)
{
    struct mm_struct * mm = current->mm;
    struct vm_area_struct * vma, * prev;
@@ -1914,6 +1936,9 @@ unsigned long do_brk(unsigned long addr,

    flags = VM_DATA_DEFAULT_FLAGS | VM_ACCOUNT | mm->def_flags;

+ if (bc_memory_charge(mm, len, flags, NULL, bc_strict))
+ goto out_unacct;
+
    /* Can we just expand an old private anonymous mapping? */
    if (vma_merge(mm, prev, addr, addr + len, flags,
        NULL, NULL, pgoff, NULL))
@@ -1923,10 +1948,8 @@ unsigned long do_brk(unsigned long addr,
    * create a vma struct for an anonymous mapping
    */
    vma = kmem_cache_zalloc(vm_area_cachep, GFP_KERNEL);
- if (!vma) {
-     vm_unacct_memory(len >> PAGE_SHIFT);
-     return -ENOMEM;
- }
+ if (!vma)
+ goto out_uncharge;

    vma->vm_mm = mm;
    vma->vm_start = addr;
@@ -1943,6 +1966,17 @@ out:
    make_pages_present(addr, addr + len);
}
return addr;
+
+out_uncharge:
+ bc_memory_uncharge(mm, len, flags, NULL);
+out_unacct:
+ vm_unacct_memory(len >> PAGE_SHIFT);
+ return -ENOMEM;

```

```

+}
+
+unsigned long do_brk(unsigned long addr, unsigned long len)
+{
+ return __do_brk(addr, len, BC_LIMIT);
+}

EXPORT_SYMBOL(do_brk);
@@ -2005,9 +2039,18 @@ int insert_vm_struct(struct mm_struct *
    return -ENOMEM;
    if ((vma->vm_flags & VM_ACCOUNT) &&
        security_vm_enough_memory(vma_pages(vma)))
- return -ENOMEM;
+ goto err_acct;
+ if (bc_memory_charge(mm, vma->vm_end - vma->vm_start,
+   vma->vm_flags, vma->vm_file, BC_LIMIT))
+ goto err_charge;
    vma_link(mm, vma, prev, rb_link, rb_parent);
    return 0;
+
+err_charge:
+ if (vma->vm_flags & VM_ACCOUNT)
+   vm_unacct_memory(vma_pages(vma));
+err_acct:
+ return -ENOMEM;
+}

/*
--- ./mm/mremap.c.bclockcharge 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/mremap.c 2006-09-05 13:06:37.000000000 +0400
@@ -19,6 +19,8 @@
#include <linux/security.h>
#include <linux/syscalls.h>

+#include <bc/vmpages.h>
+
#include <asm/uaccess.h>
#include <asm/cacheflush.h>
#include <asm/tlbflush.h>
@@ -350,6 +352,13 @@ unsigned long do_mremap(unsigned long ad
    goto out_nc;
}

+ if (new_len > old_len) {
+   ret = bc_memory_charge(mm, new_len - old_len,
+     vma->vm_flags, vma->vm_file, BC_BARRIER);
+   if (ret)
+     goto out;

```

```

+ }
+
/* old_len exactly to the end of the area..
 * And we're not relocating the area.
 */
@@ -374,7 +383,7 @@ unsigned long do_mremap(unsigned long ad
    addr + new_len);
}
ret = addr;
- goto out;
+ goto out_ch;
}
}

@@ -393,10 +402,15 @@ unsigned long do_mremap(unsigned long ad
    vma->vm_pgoff, map_flags);
ret = new_addr;
if (new_addr & ~PAGE_MASK)
- goto out;
+ goto out_ch;
}
ret = move_vma(vma, addr, old_len, new_len, new_addr);
}
+out_ch:
+ if (ret & ~PAGE_MASK)
+ if (new_len > old_len)
+ bc_memory_uncharge(mm, new_len - old_len,
+ vma->vm_flags, vma->vm_file);
out:
if (ret & ~PAGE_MASK)
vm_unacct_memory(charged);
--- ./mm/shmem.c.bclockcharge 2006-09-05 12:59:27.000000000 +0400
+++ ./mm/shmem.c 2006-09-05 13:06:37.000000000 +0400
@@ -1309,21 +1309,31 @@ int shmem_lock(struct file *file, int lo
    struct inode *inode = file->f_dentry->d_inode;
    struct shmem_inode_info *info = SHMEM_I(inode);
    int retval = -ENOMEM;
+ unsigned long size;
+
+ size = (inode->i_size + PAGE_SIZE - 1) >> PAGE_SHIFT;

    spin_lock(&info->lock);
    if (lock && !(info->flags & VM_LOCKED)) {
- if (!user_shm_lock(inode->i_size, user))
+ if (bc_locked_shm_charge(info, size))
    goto out_nomem;
+ if (!user_shm_lock(inode->i_size, user))
+ goto out_uncharge;

```

```

    info->flags |= VM_LOCKED;
}
if (!lock && (info->flags & VM_LOCKED) && user) {
    user_shm_unlock(inode->i_size, user);
+ bc_locked_shm_uncharge(info, size);
    info->flags &= ~VM_LOCKED;
}
    retval = 0;
out_nomem:
    spin_unlock(&info->lock);
    return retval;
+
+out_uncharge:
+ bc_locked_shm_uncharge(info, size);
+ goto out_nomem;
}

int shmem_mmap(struct file *file, struct vm_area_struct *vma)

```
