User beancounter patch

Andrey V. Savochkin, saw@asp-linux.com

31 July 2000

Contents

1. Overview

This patch provides accounting and allows to configure limit user's consumption of exhaustible system resources. The m important resource controlled by this patch is unswappable memory (either `mlock`'ed or used by internal kernel structur and buffers). The main goal of this patch is to protect proce from running short of important resources because of an accidental misbehavior of processes or malicious activity ai to "kill" the system. It's worth to mention that resource lim configured by `setrlimit`(2) do not give an acceptable level protection because they cover only small fraction of resourc and work on a per-process basis. Per-process accounting do prevent malicious users from spawning a lot of resource-consuming processes.

Although the main use of this patch is accounting and limiti the amount of resources consumed by processes of each us may be used for control of resource use by any group of processes with the common "luid". "luid" is assigned to unaccounted processes (only) and is inherited over `fork`.

2. General architecture principles

User beancounter patch modifies the core parts of the kerne
(like virtual to physical address translation code) and, thus,
should be compact and efficient as much as possible. Some
functionality and system administrator convenience have be
sacrificing to achieve this compactness and efficiency.

All accounting and limiting is provided on a per-luid basis. L
assigned by `setluid` system call and is inherited over `fork`
Once being assigned to a process, it cannot be revoked or
changed in the future. When process creates new objects
consuming resources (like new processes, `struct file`, an
on) these objects also grab a reference to luid of the proces
used resources are accounted. Thus, objects do not change
luid reference and cannot get the reference at the middle o
existence. Such an architecture simplifies things a lot.

Resource use limits are just limits, and do not provide
"wait-until-available" functionality. The limits are organized
two thresholds. The exact meaning of these thresholds is
resource-specific. In general, after reaching the first thresh
creation of new resource consuming objects is denied, and t
system tries to inform applications about resource shortage
gracefully. The second threshold is the upper bound for the
resource consumption, which is maintained even by means
abrupt killing of the offending process.

To clarify this policy let's consider the limit for unswappable
memory. When the first threshold is reached, the subseque
`fork`, `mlock` and other calls start to fail. The application sh
handle these failures and correctly terminate its work. Whe
second threshold is reached, all accounted kernel memory
allocations will fail for this process. Such allocation may ha
inside, for example, page fault handler which creates memo
images of mapped files under normal circumstances. In the
of reaching the "hard" limit the kernel cannot notify the
application and does not have other choice than to kill it.

3. Current status

The initial version of this patch was developed by Alan Cox
Andrey Savochkin for early 2.2 kernels after some discussio
linux-kernel mailing list. The current maintainer is Andrey
Savochkin.

The last version accounted for the following resources:

- Unswappable kernel memory size including `struct task`
  directories, etc.
- `mlock`'ed pages.
- Address space size in pages.
- Total size of SysV IPC SHM segments created by user.

- Total size of shared anonymous memory segments create
  user.
- Number of processes.
- Resident pages (no upper limits, the number is used for
  swap-out guarantee).
- Number of sockets.
- Number of file locks.
- Number of pseudo terminals.
- Number of `siginfo` structures.

The really important resources are unswappable memory, IP
SHM segment size, and number of processes. Other resourc
rather auxiliary.

Unswappable memory is a resource consumed by applicatio
indirectly. Unswappable memory areas are created on `fork`
(different internal kernel structures like `struct task`), on
memory management calls (page directories for virtual to
physical address translation), and so on. Certain call patter
may lead to all available physical memory being occupied b
kind of data, and the inability to free enough physical mem
swapping out or any other means. The patch provides the b
protection, which needs to be extended by accounting of mo
sources of unswappable memory allocations.

IPC SHM segment size is another resource where user
beancounter patch provides the efficient protection against
abuses and denial-of-service attacks. IPC SHM API has seve
defects, one of which is the rejection of automatic garbage
collection. Automatic garbage collector keeps reference cou
for objects and release the resource when the object becom
unreferenced. Such a garbage collection exists for files, for
example. However, IPC SHM API requires explicit deletion o
segments. Such a deletion may be accidently or deliberately
omitted, which leads to memory waste. Creating a lot of SH
segments without their deletion may also work as a
denial-of-service attack.

Number of processes is limited on IA32 architecture. This li
exists because each process requires a GDT entry, number
which is limited by CPU architecture. GDT entry limit is the
reason for accounting and limiting for the number of proces
run by each user.

Other accounted quantities do not correspond to exhaustibl
resources directly. For example, the number of `mlock`'ed pa
included into accounting of number of unswappable pages.
However, administrators may wish to set the unswappable
limit to large values to allow users to spawn a lot of proces
In this case the administrator may limit the users' ability to
`mlock` pages to prevent abuses of the high unswappable me
limits.

4. Comments about individual resources

4.1 Control for virtual space and resident pages

The basics of the approach are described here. I'm drafting
more detailed description and will publish it when it's ready

4.2 Accounting for resources consumed by sockets

Current code does:

1. account the number of sockets;
2. account memory used by receive and send buffers.

Memory is charged for the socket at the moment of its crea
It would definitely be better to charge the actual used mem
but in this case I don't see a way to properly implement lim
this memory. Dropping received packets and returning error
locally originated ones isn't an acceptable variant. Sleeping
semantics of limits (wait until the quota allows queueing m
packets) can't be applied to external packets and leads to r
user-space deadlocks for local ones.

The places of the accounting hooks are:

- `struct socket` gets reference to beancounter (from `current->login_bc`) in `sock_alloc`;
- the memory is charged to the beancounter before `sk_all` call from protocol family specific creation routines;
- upon `struct sock` creation it gets beancounter reference the amount of charged memory is stored in the structure
- `sk_free` uncharges the memory and drops the reference beancounter;
- `setsockopt` calls charge the difference in the socket buff size.

5. Development plans

First of all, the summary of control of finite resources. Ther

1. really exhaustible resources: number of processes, unswappable kernel memory associated with user's proce TCP and UDP ports (limited to 2^16 by protocol).
2. bandaids, like limit for total size of SysV IPC shared mem segments.
3. helper limits to catch process misbehavior earlier: limit o number of sockets, number of locks, virtual address spac size. Although the excessive number of locks, for exampl may do a direct harm (by slowing down lookups), the ma point of concern is the amount of occupied memory, whic

3. accounted together with the number. So, accounting of resources from the third group is considered only as a hel

At this moment almost all (except TCP and UDP ports) obvi exhaustible resources are under control. But we may not be that all possibitilities for denial-of-service attacks are close From theoretic point of view, it would be better to ensure th each non-trivial operation, each `kmalloc` is charged. In prac it's impossible. There are a lot of places where the subject resource should be charged to isn't obvious (not `current!`), where the limit can't be enforced. Socket buffer accounting Sockets section) is a clear example of such a situation. So, only possible way here is to spot suspicious places in the k and add resource control calls suitable for them. Certainly, comments and patches are welcome!

Administrators should also be given a way to implement so policy and to control memory management (i.e. how proces share the pagable memory, page cache, and how swap-out works), then, disk bandwidth, and so on. These matters are considered in the future.

6. API

This section describes user beancounter API for applications

6.1 Calls

There is a well-known conflict between kernel and libc head files. The prototypes of the system calls below are presente they may be used for making direct calls, without libc modifications.

```
long sys_getluid(void);
```

Returns the luid of the process. Returns error (`ENOENT` curre please suggest the better code) if luid hasn't been assigned this process yet.

Beware: this call (and all consequent ones) fail if the beancounter feature isn't compiled into the kernel. Do not m unreasonable assumptions that the call always succeeds or error codes you may get in return.

```
long sys_setluid(uid_t uid);
```

Set luid of the process. The call succeeds only for privileged processes (`CAP_SETUID` currently) and only if luid hasn't be assigned to this process yet. Returns `0` on success. Docume error codes are `EPERM` and `EINVAL`.

```
long sys_setublimit(uid_t uid, unsigned long reso
struct rlimit *rlim);
```

Set resource limit number `resource` for luid `uid`. Returns `0` success. Documented error codes are `EPERM` and `EINVAL`. Th operation is privileged and requires `CAP_SYS_RESOURCE` capability. Currently, if the given luid hasn't been assigned living process, the call fails with `EINVAL`.

6.2 Constants

The following constants are defined in `linux/beancounter` this moment.

```
#define UB_KMEMSIZE     0
#define UB_LOCKEDPAGES  1
#define UB_TOTVMPAGES   2
#define UB_SHMPAGES     3
#define UB_ZSHMPAGES    4
#define UB_NUMPROC      5
#define UB_RESPAGES     6
#define UB_SPCGUARPAGES 7
#define UB_OOMGUARPAGES 8
#define UB_NUMSOCK      9
#define UB_NUMFLOCK     10
#define UB_NUMPTY       11
#define UB_NUMSIGINFO   12
```

Their meaning is briefly described in section Current Status.

6.3 How to form a piece of code dealing with resource lir

A short example:

```
#include <linux/unistd.h>
#include <linux/resource.h>
#include <linux/beancounter.h>

static _syscall0(long, getpid);
static _syscall1(long, setluid, uid_t, uid);
static _syscall3(long, setublimit, uid_t, uid,
unsigned long, resource, struct rlimit *, rlim);

void f(void)
{
struct rlimit rlim;
setluid(500);
rlim.rlim_cur = 4;
rlim.rlim_max = 4;
setublimit(getpid(), UB_NUMPROC, &r);
}
```

Libc doesn't have wrappers to newly created system calls. S
the code should make system calls directly.

To do it the code should include `linux/unistd.h` header.
Unfortunately, libc and kernel headers cannot safely be incl
from the same C file. In most cases, it leads to an enormou
amount of compilation errors. But even if the code compiles
there may be more subtle problems (different data sizes, fo
example). So, the C file dealing with system calls directly s
not include any of libc headers. It's possible to use libc call
the file if you understand what you are doing, but I persona
prefer to avoid it. It's better to keep a small file which perfo
system calls and does nothing else.

7. Testing the patch

The current version of the patch is available from
http://www.asplinux.com.sg/install/user_beancounter-IV-cu
It is against `2.4.0-test1` kernel. The patch introduces two
kernel configuration options: `CONFIG_USER_RESOURCE` and
`CONFIG_USER_RESOURCE_PROC`. The first one enables user
beancounter functionality, and the second provides informa
about used resources and limits through
`/proc/user_beancounters`.

There is a small program to play with the patch:
http://www.asplinux.com.sg/install/ulim4.c. It takes the res
number and it's "soft" and "hard" limits as arguments and s
`/bin/bash` (check `include/linux/beancounter.h` for reso
numbers). All child processes of the started shell will have
same luid (i.e. belong to a single accounting group). Watch
resource use through `/proc` and try to overpass the limits!

8. Credits

Thanks to Marcelo Tosatti, Andrey Moruga, Vlad Bolkhovitin
Alexey Raschepkin for contributions to the patch.

```
$Id: UserBeancounter.sgml,v 1.7 2000/07/31 02:26:0
saw Exp $
```


E-mail: info@asp-linux.com Tel: +65 220 0306