Tails - Bug #12354

Fix shutdown and memory wipe regressions on 3.0-betaN

03/17/2017 10:17 AM - intrigeri

<table>
<thead>
<tr>
<th>Status:</th>
<th>Resolved</th>
<th>Start date:</th>
<th>03/20/2017</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>High</td>
<td>Due date:</td>
<td></td>
</tr>
<tr>
<td>Assignee:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Category:</td>
<td></td>
<td>% Done:</td>
<td>100%</td>
</tr>
<tr>
<td>Target version:</td>
<td>Tails_3.0-rc1</td>
<td>Estimated time:</td>
<td>0.00 hour</td>
</tr>
<tr>
<td>Feature Branch:</td>
<td>bugfix/12354-drop-kexec-memory-wipe</td>
<td>Spent time:</td>
<td>0.00 hour</td>
</tr>
<tr>
<td>Type of work:</td>
<td>Code</td>
<td>Starter:</td>
<td></td>
</tr>
<tr>
<td>Blueprint:</td>
<td></td>
<td>Affected tool:</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

We've been reported a number of regressions vs. 2.x on 3.0-beta1 and beta2: on shutdown, the kernel is kexec'ed but then either nothing else happens (blinking caps lock == kernel panic) or the system fails to shut down and leaves the user facing an initramfs prompt.

So:

- Do we see any cheap way to debug this? If not:
- Is it better to have an unreliable memory wiping feature, that leaves the system in a weird (and suspicious) state when it fails, or no such feature at all? In other words, do we want to optimize for the high-risk users who need this feature and got hardware where it is reliable? Or for everybody else? And is it OK to provide this feature (and then some users will rely on it) even though it doesn't work reliably (and then some users will be bitten because they rely on it and today / on other hardware) it fails?

Note that #12089 might be enough (see discussion on #12107 and tails-dev) to erase most memory without any special "memory wipe on shutdown" process.

**Subtasks:**

Feature #12397: Evaluate how kernel memory poisoning works for tmpfs

Feature #12398: Evaluate how kernel memory poisoning works for disk cache

Feature #12428: Ensure disk caches and aufs read-write branch are emptied during emerg...

**Related issues:**

Related to Tails - Feature #12089: Enable the kernel page allocator poisoning

Related to Tails - Bug #12393: Emergency shutdown leaves the Desktop on for m...

Related to Tails - Feature #5417: Improve user experience when wiping memory ...

Related to Tails - Bug #12560: Polish the new memory wiping implementation an...

Duplicated by Tails - Bug #11786: System often crashes during/after memory wi...

Blocked by Tails - Feature #12554: Bump APT snapshots for 3.0-rc1

**Associated revisions**

Revision 9b62a62b - 04/05/2017 10:51 AM - intrigeri

Test suite: drop "no memory erasure" test, that can't work with kernel memory poisoning enabled (refs: #12354).

Revision 90991d83 - 04/05/2017 10:58 AM - intrigeri

Test suite: replace memory erasure on shutdown test with a new one, that tests erasure of memory freed by a killed userspace process (refs: #12354).

Revision 9cf3f60d - 04/05/2017 12:41 PM - intrigeri

Test suite refactoring: extract parts that a different step will need into a dedicated one (refs: #12354).

Revision 405ea1b7 - 04/05/2017 12:42 PM - intrigeri

Test suite: remove now unused code (refs: #12354).
Revision 2e9a2efc - 04/05/2017 12:42 PM - intrigeri
Test suite: rename scenario and step to make it clearer what we're testing (refs: #12354).

Revision ab08b13b - 04/05/2017 01:30 PM - intrigeri
Test suite: test that memory poisoning applies to unmounted tmpfs (refs: #12354, #12397).

Revision c66df936 - 04/05/2017 03:04 PM - intrigeri
Test suite: test that memory poisoning applies to write cache for unmounted vfat and LUKS-encrypted ext4 (refs: #12354, #12398).

Revision ad1aed67 - 04/05/2017 03:40 PM - intrigeri
Test suite: test that memory poisoning applies to read cache for unmounted vfat and LUKS-encrypted ext4 (refs: #12354, #12398).

Revision a1ac68cc - 04/17/2017 05:02 PM - intrigeri
Merge remote-tracking branch 'origin/bugfix/12354-drop-kexec-memory-wipe' into feature/stretch (refs: #12354)
This does not replace the kexec-based memory wiping yet.

Revision e2caab51 - 05/18/2017 08:02 AM - intrigeri
Drop kexec-based memory erasure feature (refs: #12354).
It's not reliable enough and provides poor UX. Linux memory poisoning works well enough to get rid of it.

Revision e6382573 - 05/18/2017 08:02 AM - Intrigeri
Drop kernel caches before shutting down (refs: #12428, #12354).
Let's optimize how much memory the kernel memory poisoning feature acts on.

Revision 888ccc5a - 05/18/2017 08:02 AM - intrigeri
Return to the initramfs (unpacked in /run/initramfs) on shutdown (refs: #12428, #12354, Debian#778849).
… otherwise the aufs read-write (tmpfs) branch, among possibly other things, can't be properly unmounted and its content remains in memory.

Notes:
- We have to handle some unmounting ourselves in initramfs-pre-shutdown-hook: systemd-shutdown doesn't manage to unmount the aufs read-write branch (/oldroot/lib/live/mount/overlay) as it is needed by the aufs (/oldroot) filesystem, and reciprocally it cannot unmount /oldroot as it is kept busy by /oldroot/lib/live/mount/*'. So we disentangle this mess ourselves. And we have to manually empty the aufs read-write (tmpfs) branch, otherwise for some reason its content remains in memory. This code will of course need to be adapted for overlayfs some day.
- We lock /bin/kill in memory: apparently systemd-exit.service needs it.
- We remount /run on shutdown before dropping caches, just in case dropping caches removes what we've locked into memory.
- We unpack the initramfs to /run/initramfs at boot time: sadly, I was not able to have it unpacked reliably in udev-watchdog-wrapper when the boot medium is ejected, so we'll use a little bit more RAM (instead of locking the compressed initramfs into memory, we're storing the uncompressed one there) and probably slow down the boot a bit, in order to make emergency shutdown robust. Note, however, that we save some of the RAM used by the uncompressed initramfs by deleting the worst offenders (kernel modules).
For now the whole procedure is quite noisy on the screen: the pre-shutdown hook runs under "set -x", doesn't run "clear", and spits out lots of debugging information. The goal is to enable users to provide useful debugging data if they have problems with emergency shutdown. Once we have shipped this code in a few releases and trust it's robust enough, we can surely reconsider and polish the UX by making the output less noisy.

We use absolute paths in many places to avoid $PATH lookup which might fail if the root filesystem is not there anymore.

Revision eac9e419 - 05/18/2017 08:26 AM - intrigeri
Test suite: add tests for memory erasure on emergency shutdown (refs: #12354).

I want to confirm that we successfully erase memory during emergency shutdown as well, by returning to the initramfs as intended, and that several kinds of filesystems are unmounted and their caches erased.

Revision 5f588e52 - 05/18/2017 03:36 PM - intrigeri
Merge remote-tracking branch 'origin/bugfix/12354-drop-kexec-memory-wipe' into feature/stretch (Fix-committed: #12428, #12354)

History
#1 - 03/17/2017 10:20 AM - intrigeri
- Related to Feature #12089: Enable the kernel page allocator poisoning added

#2 - 03/17/2017 10:20 AM - intrigeri
- Related to Feature #12107: Can PAX_MEMORY_SANITIZE replace memory erasure on shutdown? added

#3 - 03/17/2017 10:21 AM - intrigeri
Next step is to research this a bit and answer the "Do we see any cheap way to debug this?" question.

#4 - 03/18/2017 10:09 AM - emmapeel
I am not sure if is part of this ticket, but the Emergency Shutdown now acts quickly, but the screen stayed rendered for much longer time than Tails > 3.0.

Before I would only see the screen for 2 seconds at most, before it went black.

Now it stays there and after 4 seconds start to corrupt slowly, but you can still see pretty much everything on it.

Shall I open another ticket about it, or it fits the scope of this ticket?

#5 - 03/18/2017 10:30 AM - intrigeri

I am not sure if is part of this ticket, but the Emergency Shutdown now acts quickly, but the screen stayed rendered for much longer time than Tails > 3.0.
Before I would only see the screen for 2 seconds at most, before it went black.

Now it stays there and after 4 seconds start to corrupt slowly, but you can still see pretty much everything on it.

Can you please clarify what's "the screen"?

---

#6 - 03/18/2017 11:35 AM - Anonymous
intrigeri wrote:

I am not sure if is part of this ticket, but the Emergency Shutdown now acts quickly, but the screen stayed rendered for much longer time than Tails > 3.0.

Before I would only see the screen for 2 seconds at most, before it went black.

Now it stays there and after 4 seconds start to corrupt slowly, but you can still see pretty much everything on it.

Can you please clarify what's "the screen"?

I think she meant to say that she continues to see the Tails desktop for 2 seconds.

I also had the feeling that it was slower on the 3.0 beta 2 but it's just a feeling.
Can you please clarify what's "the screen"?

I think she meant to say that she continues to see the Tails desktop for 2 seconds.

Yes, the whole Desktop of my session stays there for long... while the screen goes corrupting slowly...

I also had the feeling that it was slower on the 3.0 beta 2 but it's just a feeling.

More than 2 seconds. 2 seconds or 3 without any change, and then some 4 seconds of slow corruption. Plenty of time to discover you were reading the 'lesbian teen support' webpage or similar.

Yes, the whole Desktop of my session stays there for long... while the screen goes corrupting slowly...

OK, then please file this as a separate ticket: this one is about failure to shutdown (and possibly to actually wipe the memory).

Related to Bug #11786: System often crashes during/after memory wipe since Linux 4.6 added

- Status changed from Confirmed to In Progress
- % Done changed from 0 to 10
- Type of work changed from Research to Discuss
From the description I'm extracting the two problems we have: On shutdown:

1. the kernel panics when kexec:ing: no memory is wiped, and the system is powered so the memory is kept in a perfect state unlike if the system had shut down; the memory will quickly start to decay without power. In other words, in this failure mode our memory wipe results in something worse than a normal shutdown. :S

2. the kernel kexec:ing works, but then the system fails to shut down: the userland memory is (probably) wiped, but not the memory reserved for the kernel.

In both cases the user is faced with a system that is still running, with weird stuff on the screen, and has to manually power off the computer.

Problem 1 is due to hardware compatibility, i.e. kexec is broken on some hardware. New kernel releases may fix it for some hardware, and may break it for others. We've seen it go both ways, and it's safe to say that we neither have the expertise, time nor access to enough hardware to follow this closely and make sure each kernel version doesn't regress. So it seems impossible that we can completely (= for all hardware) solve problem 1. Conclusion: it seems kexec-based memory wiping will always remain unreliable on some of our users' hardware.

Problem 2 is more tractable for us. We have several ideas of how to proceed, e.g. someone could spend a day or two to try e.g. #11786#note-60. We can probably solve it, but it will still take quite some time.

So, given that we will not be able to solve problem 1, is it even worth spending time on problem 2? At best we'll only be able to make "guarantees" that memory wiping works on certain specific hardware that we list somewhere, that very serious "high risk" users might invest in, but users with $random hardware will be left out, and some of them will be hit by this annoying issue and the uncertainty of whether the wiping worked at all.

So we have three options:

- a) We live with problem 1 but fix problem 2.
- b) Same as a), but we also enable #12090 (memory allocated to userspace is wiped when free():d)
- c) We drop our kexec-based memory wiping, enable #12090 and rely on that as our only active memory wiping measure.

(b) is quite obviously better than (a) except that we cannot automatically test our memory wiping feature any more (#12090 makes this test impossible). So, really, (b) or (c)? More detailed:

- b): crappy UX and less safety on some hardware, but more safety when it actually works.
- c): more polished UX, and reliable though imperfect memory wiping

Another way to look at it (read "->" as "is better than"):

- Safety: (b) when it works -> (c) -> (b) when it fails
- UX: (b) when it works == (c) -> (b) when it fails

So, yeah, it's a nasty trade-off we have to decide.
7 tails contributors discussed this question while working on 3.0.

Conclusion: we would go for solution (c) after verifying that it not only wipes application memory but also tmpfs and disk cache. If (c) doesn't fill these requirements, we'll need to discuss again whether (b) is best or not.

Reason: a reliable safe solution is better than an unreliable safer solution. When (b) fails with a kernel panic (problem 1), the user ends up in a worse situation than without any memory wiping, as there is no realistic way to solve that in a reliable way. (c) wipes less kernel memory, but we will know what it wipes or not and have a reliable implementation.

Conclusion: we would go for solution (c) after verifying that it not only wipes application memory but also tmpfs and disk cache.

I'll try to write automated tests for these two things. It'll probably happen during the 6th Stretch sprint (mid-May), unless I find it an exciting enough candidate for structured procrastination earlier ;)

---

To test tmpfs:

- start Tails
- check the known pattern cannot be found
- mount a tmpfs of decent size (no need to aim for the maximum)
- fill it with a known pattern
- dump memory and check the known pattern can be found
- umount the tmpfs, verify that the known pattern cannot be found anymore.

If it still can, test shut down and/or drop caches.

To test disk cache:

- background:
  - start Tails
  - create+unlock+mount a ext4-on-LUKS filesystem
- test write buffers:
  - drop caches
  - check that the known pattern cannot be found in memory
  - fill the FS with a known pattern
  - dump memory to check the known pattern can be found
- test read cache:
  - drop caches
  - check that the known pattern cannot be found
  - read the entire filesystem
  - check that the known pattern can be found
- test umount (after each of previous tests):
  - umount the filesystem
  - check if the known pattern can be found
  - if it can, drop caches, check again, and if this works better: consider dropping caches during the shutdown process
anonym, I'm under the impression that you'll be about twice faster than me to implement the test cases designed in my previous comment. Interested? Excited? If not, reassign to me :)

---

**#15 - 03/20/2017 04:52 PM - intrigeri**

- Related to deleted (Feature #12107: Can PAX_MEMORY_SANITIZE replace memory erasure on shutdown?)

---

**#16 - 03/20/2017 06:07 PM - alant**

Procedures looks fine, but

- create+unlock+mount a ext4-on-LUKS filesystem

I think it would be good to make the same tests with vfat. This would test the common use case of reading/saving a file from/to an USB stick, e.g. to share it. Then I expect that once the stick is unplugged and Tails is shut down, the file is not accessible anymore.

---

**#17 - 03/21/2017 08:26 AM - intrigeri**

Procedures looks fine,

Great! Thanks :) but

- create+unlock+mount a ext4-on-LUKS filesystem

I think it would be good to make the same tests with vfat. This would test the common use case of reading/saving a file from/to an USB stick, e.g. to share it. Then I expect that once the stick is unplugged and Tails is shut down, the file is not accessible anymore.

I would expect that in most situations when traces of activity in RAM matter, the file will still be (in cleartext) on the USB stick that's still pretty close, so wiping memory doesn't change much. But whatever: it's trivial to also test this once we have the other tests, so let's do it just to be on the safe side!
I have written automated tests (that pass successfully and reliably on my system) that, I believe, answer the few questions that were raised when we discussed this. Next steps:

1. anonym reviews the branch, confirms that kernel poisoning is good enough for us, and merges it up to 0c8e076a5dac72cc6898ad9d3ce4156699ed2d06 into feature/stretch if happy;
2. I'm going to file a ticket about dropping the kexec-based implementation.

intrigeri wrote:
I'm going to file a ticket about dropping the kexec-based implementation.
Actually not, this topic is already spread over too many tickets. I'll just reuse this ticket to actually fix the issue it's about, i.e. drop the kexec-based implementation. So please reassign to me once you've reviewed'n'merged the branch (that only adds automated tests and enables poisoning, but that's a good start and useful in itself).

intrigeri wrote:
I have written automated tests (that pass successfully and reliably on my system) that, I believe, answer the few questions that were raised when we discussed this. Next steps:

1. anonym reviews the branch, confirms that kernel poisoning is good enough for us, and merges it up to 0c8e076a5dac72cc6898ad9d3ce4156699ed2d06 into feature/stretch if happy;
2. I'm going to file a ticket about dropping the kexec-based implementation.

Review + testing done! Great job, I didn't find any blockers! I've pushed some minor nitpicky stuff on top, and other than that I only have these non-blocker remarks relating to re-usability:

def mount_USB_drive(disk, fs)
[...]
if /
\bencrypted with password\b/.match(fs)
    password = /\bencrypted with password "([\^\"]+)\b/.match(fs)[1]
IMHO the fs variable shouldn't be human-readable text, but optional arguments (I recommend a Hash so a call could look like
mount_USB_drive(some_disk, filesystem: 'ext4', luks_passphrase: 'asdf')) so it fits in more caller contexts.

When(/^I plug and mount a \((\d+) \s*Mi\?B\) USB drive with an? \((.*\$)\) do \(size\_Mi\?B, fs\)\)

Here you can parameterize the size unit instead of fixing on MiB. The same applies to several other steps you created.

Agreed for both. I'll do that when I resume work on this branch.

#25 - 04/17/2017 01:16 PM - intrigeri
- Assignee changed from anonym to intrigeri
- QA Check changed from Ready for QA to Dev Needed

IMHO the fs variable shouldn't be human-readable text, but optional arguments (I recommend a Hash so a call could look like
mount_USB_drive(some_disk, filesystem: 'ext4', luks_passphrase: 'asdf')) so it fits in more caller contexts.

[...]
Here you can parameterize the size unit instead of fixing on MiB. The same applies to several other steps you created.

inintrigeri wrote:

1. anonym reviews the branch, confirms that kernel poisoning is good enough for us, and merges it up to
   0c8e076a5dac72cc6898ad9d3ce4156699ed2d06 into feature/stretch if happy;

This happened.

#27 - 18/2017 03:03 PM - intrigeri
- Target version changed from Tails_3.0 to Tails_3.0~rc1
```ruby
> def mount_USB_drive(disk, fs)
>   [...]  
>   if /encrypted with password\b/.match(fs)
>     password = /encrypted with password "([^]*)"/.match(fs)[1]
>   
IMHO the fs variable shouldn't be human-readable text, but optional arguments (I recommend a Hash so a call could look like
mount_USB_drive(some_disk, filesystem: 'ext4', luks_passphrase: 'asdf')) so it fits in more caller contexts.

Full ACK, done locally.

> When(/^I plug and mount a \(\d+\) MiB USB drive with an? \(.*\)\) do |size_MiB, fs|
>
Here you can parameterize the size unit instead of fixing on MiB. The same applies to several other steps you created.

IMO the additional code complexity it would introduce is not deserved here: the MiB unit is suitable, in these steps, for all use cases I can think of. I'd rather avoid make stuff generic "just in case", as it adds some maintenance cost.
```

#29 - 05/18/2017 08:50 AM - intrigeri
- Related to Bug #12393: Emergency shutdown leaves the Desktop on for much longer in Stretch added

#30 - 05/18/2017 09:08 AM - intrigeri
- Assignee changed from intrigeri to anonym
- QA Check changed from Dev Needed to Ready for QA

#31 - 05/18/2017 09:10 AM - intrigeri
- Related to Feature #5417: Improve user experience when wiping memory at shutdown added

#32 - 05/18/2017 09:57 AM - intrigeri
- Blocked by Feature #12554: Bump APT snapshots for 3.0~rc1 added

#33 - 05/18/2017 03:02 PM - anonym
- Assignee changed from anonym to intrigeri

02/20/2020 11/13
Awesome work! I found no blockers (so feel free to merge this yourself as long as fixing my remarks below are tracked somehow and fixed before Tails 3.0) and I was hard-pressed to find anything to comment on, but here goes:

In wiki/src/contribute/design/memory_erasure.mdwn:

In order to protect against memory recovery such as cold boot attack, most of the system RAM is overwritten when Tails is being shutdown or when the boot medium is physically removed.

I guess we could add "and immediately after applications exit" or something to that effect? That is an improvement over the old kexec-based approach when treated in isolation.

#### The big picture

Can we put the story about the old design in another sub section, like "##### Obsolete kexec-based approach", and the current one in "##### Current memory poisoning-based approach"? I.e. just insert these two headers (and possibly adapt the edges of the affected paragraphs so the text still makes sense). I'd just like to make the history lesson more optional. :) Is it really needed for understanding the current design?

Different kinds of events trigger the memory erasure process. All lead to run the 'tails-kexec' shutdown script.

There's no tails-kexec any more.

[[!tails_gitweb features/erase_memory.feature desc="Automated tests"]]

ensure that the most important parts of memory are erased this way.

Can we describe "the most important parts of memory" a bit better than this (not necessarily at this place in the design page, btw)? Or do we want the .feature file to enumerate what we think is "important"? I really think this design page should list any limitations of the memory poisoning approach.

In features/step_definitions/common_steps.rb:

```ruby
-def mount_USB_drive(disk, fs)
+def mount_USB_drive(disk, fs_options = {})
+  fs_options[:encrypted] ||= false
```

Just a FYI: nil is treated as false when interpreted as a boolean, so this assignment is unnecessary. That said, I don't really mind making the default more explicit, for readability.

In features/step_definitions/erase_memory.rb:

```ruby
kernel_mem_reserved_k = 64*1024 # Duplicated in /usr/share/initramfs-tools/scripts/init-premount/memory_wipe
 [...]  
admin_mem_reserved_k = 128*1024 # Duplicated in /usr/share/initramfs-tools/scripts/init-premount/memory_wipe
```

These don't duplicate anything any longer.

In config/chroot_local-includes/usr/share/initramfs-tools/hooks/shutdown:

```bash
# systemd-shutdown itself
mkdir -p $DESTDIR/lib/systemd
copy_exec /lib/systemd/systemd-shutdown /shutdown

# Our shutdown hook (run from inside the initramfs)
mkdir -p $DESTDIR/lib/systemd/systemd-shutdown
```
To me it looks like you are following the common pattern of `mkdir -p`ing the target directories for the subsequent copy actions, but these directories are not used as the target (in fact they are not used at all). I'm just double-checking that these `mkdir` calls still are relevant, and no residues from some Git history rewrite.

In features/emergency_shutdown.feature:

```
Then I find very few patterns in the guest's memory
And Tails eventually shuts down
```

This means that we will always wait the full two minutes (the "Happy dumping!" timeout) for each of these scenarios: there's currently four, so that's almost $2^4 = 8$ minutes of needless waiting. I'm tempted to suggest that we add a scenario dedicated to make sure that the shutdown happens without any of the memory erasure checks, and that we simply remove the Tails eventually shuts down step from these four scenarios. In fact, I think you should move these four scenario to erase_memory.feature as well -- I know it can be argued both ways, but I slightly prefer what I suggest (so no strong opinion here, just ignore this move if you disagree).

---

#34 - 05/18/2017 03:35 PM - intrigeri
- Related to Bug #12560: Polish the new memory wiping implementation and design doc added

#35 - 05/18/2017 03:37 PM - intrigeri
- Status changed from In Progress to 11

Applied in changeset 5f586e526056ed638b459e7e04ef39edf0a93634.

#36 - 05/18/2017 03:39 PM - intrigeri
- Assignee deleted (intrigeri)
- QA Check changed from Dev Needed to Pass

Follow-ups are tracked on #12560. Thanks!

#37 - 05/23/2017 09:13 AM - intrigeri
- Status changed from 11 to Resolved