Tails - Feature #11840

Increase mmap randomization to the maximum supported value

09/25/2016 02:34 AM - cypherpunks

<table>
<thead>
<tr>
<th>Status:</th>
<th>Resolved</th>
<th>Start date:</th>
<th>09/25/2016</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</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_3.2</td>
<td>Estimated time:</td>
<td>0.00 hour</td>
</tr>
<tr>
<td>Feature Branch:</td>
<td>feature/11840-improve-aslr-for-mmap</td>
<td>Spent time:</td>
<td>0.00 hour</td>
</tr>
<tr>
<td>Type of work:</td>
<td>Code</td>
<td>Starter:</td>
<td>No</td>
</tr>
<tr>
<td>Blueprint:</td>
<td></td>
<td>Affected tool:</td>
<td></td>
</tr>
</tbody>
</table>

Description

There are now two sysctls which can be used to tweak the amount of randomization for mmap calls. The defaults are 28 bits for 64 bit binaries, and a mere 8 bits for 32 bit binaries. These can be increased to 32 bits and 16 bits, respectively, via the vm.mmap_rnd_bits and vm.mmap_rnd_compat_bits sysctls.

This change won't cause any incompatibility issues. The only reason the default is lower than the maximum is to be very conservative to reduce address space fragmentation, which isn't going to be an issue for Tails users.

More information about the sysctls:
https://www.mail-archive.com/linux-kernel@vger.kernel.org/msg1040995.html

Related issues:

- Related to Tails - Feature #11886: Upgrade Linux to 4.8 and adjust our kernel... Resolved 10/25/2016
- Blocks Tails - Feature #13234: Core work 2017Q3: Foundations Team Resolved 06/29/2017

Associated revisions

Revision a747091c - 07/24/2017 06:46 AM - intrigeri

Increase mmap randomization to the maximum supported value (refs: #11840).

This improves ASLR effectiveness, and makes address-space fragmentation a bit worse.

Revision de8cb01d - 08/15/2017 12:06 PM - anonym

Merge remote-tracking branch 'origin/feature/11840-improve-aslr-for-mmap' into devel

Fix-committed: #11840

History

#1 - 09/25/2016 03:17 AM - intrigeri
  - Assignee set to cypherpunks
  - QA Check set to Info Needed

Because Tails has a 32 bit userland, it's especially important to increase the randomization from 8 to 16,

Do you have any examples of other operating systems changing the default values? E.g. what do Chrome OS or Android do?
but it's still a good idea even after Tails switches to a pure 64 bit userland.

FYI this will be the case in Tails 3.0.

The only reason the default is lower than the maximum is to be very conservative to reduce address space fragmentation, which isn't going to be an issue for Tails users.

Why isn't it going to be an issue for Tails users? (Thanks in advance for educating me :)

---

#2 - 09/28/2016 02:06 AM - cypherpunks

intrigeri wrote:

Do you have any examples of other operating systems changing the default values? E.g. what do Chrome OS or Android do?

Android is the reason it came into existence. They specifically requested this feature because of just how broken ASLR is on 32 bit userlands, causing a very nasty vulnerability in android to be exploitable (see [https://lwn.net/Articles/667790/](https://lwn.net/Articles/667790/)).

Both Android and Chromium OS have them set to the maximum values:

https://android-review.googlesource.com/#/c/209519/
https://chromium-review.googlesource.com/#/c/322712/

Why isn't it going to be an issue for Tails users? (Thanks in advance for educating me :)

Because memory fragmentation can cause a small performance hit, especially on embedded systems. Embedded systems might have so little memory that fragmentation becomes a game stopper. Tails requires around 1 GiB of memory to run, which is plenty. From [https://pax.grsecurity.net/docs/aslr.txt](https://pax.grsecurity.net/docs/aslr.txt):

- The last set of side effects of ASLR is address space fragmentation and entropy pool exhaustion. Since randomization shifts entire ranges of memory, it will also randomly change the gaps between them (which were constant before). This in turn will change the maximum size of memory mappings that will fit in there and applications expecting to be able to create them will fail.

Note that the entropy pool exhaustion risk is no longer an issue, as ASLR uses get_random_int() and get_random_long() instead of the entropy (count) depleting get_random_bytes() function. It\'d in extreme cases, mmap() and friends will fail.
Thanks! (Anyone who wants to speed this up: prepare a Git branch, build an ISO, run our full test suite on it, report.)

#4 - 09/30/2016 01:53 AM - intrigeri

https://outflux.net/blog/archives/2016/09/28/security-things-in-linux-v4-5/ has a one-liner to "crank your entropy to the max, without regard to what architecture you're on".

#5 - 11/09/2016 04:45 AM - cypherpunks

intrigeri wrote:

https://outflux.net/blog/archives/2016/09/28/security-things-in-linux-v4-5/ has a one-liner to "crank your entropy to the max, without regard to what architecture you're on".

That's very sloppy. It just increments the sysctls by one until it returns EINVAL. Is Tails ever going to be run on an architecture other than x86? If it is, there is a lot more that is going to be needed to be changed than this sysctl. It would be much more sane to put the regular x86 values in sysctl.d than it would to implement it as a startup script. And it would be very easy to set the sysctls based on the processor architecture anyway.

Also, what if the behavior of the sysctl changes? Some sysctls can be incremented infinitely, but will behave as if they were set to their maximum value. If these sysctls are changed to use that behavior, that script will not only cause the boot to lock up, but may even cause the value to overflow and wrap, potentially resulting in a lower ASLR strength if systemd decides it's taking too long and stops it while it's overflowed at a value lower than the default.

"Crank your entropy" is really just sloppy, IMO. Just set it to the max value manually and let it be.

#6 - 11/09/2016 05:02 PM - intrigeri

- Related to Feature #11886: Upgrade Linux to 4.8 and adjust our kernel tweaks accordingly added

#7 - 11/09/2016 05:11 PM - intrigeri

Hi!

https://outflux.net/blog/archives/2016/09/28/security-things-in-linux-v4-5/ has a one-liner to "crank your entropy to the max, without regard to what architecture you're on".

That's very sloppy. It just increments the sysctls by one until it returns EINVAL.
Is Tails ever going to be run on an architecture other than x86?

Possibly ARM, but not in the close future.

If it is, there is a lot more that is going to be needed to be changed than this sysctl.

Correct. However, we will think about the other things that will need to be changed, while it'll be easy to forget changing the hard-coded values you're suggesting.

It would be much more sane to put the regular x86 values in sysctl.d than it would to implement it as a startup script.

My concern with the hard-coded approach is that if the maximum allowed values ever change, we won't benefit from the new bigger possible values, until someone thinks about it (unlikely to happen). This is why I rather like the approach suggested by Kees, which is not affected by that problem.

Also, what if the behavior of the sysctl changes? Some sysctls can be incremented infinitely, but will behave as if they were set to their maximum value. If these sysctls are changed to use that behavior, that script will not only cause the boot to lock up,

Such a problem would be spotted pretty fast, easily, and way ahead of release, thanks to our automated testing infrastructure :)

but may even cause the value to overflow and wrap, potentially resulting in a lower ASLR strength if systemd decides it's taking too long and stops it while it's overflowed at a value lower than the default.

Are there any existing sysctls that overflow this way? (That's a real question, again: I'm not an expert in this area and need input from people like you.)

In passing, to protect against such an overflow, I could adjust Kees’ script to recall what's the biggest value that's ever been accepted by the sysctl, and reset it back to that value if writing to the sysctl ever decreases its resulting value. Of course, that's racy when another actor (systemd) comes into play; but I'm very confident that in practice, the "incrementally increase until overflow + reset" will simply never take enough time for systemd to be willing to kill the script.

OT: you might be interested in #11886. I really appreciate your contributions.
intrigeri wrote:

My concern with the hard-coded approach is that if the maximum allowed values ever change, we won't benefit from the new bigger possible values, until someone thinks about it (unlikely to happen). This is why I rather like the approach suggested by Kees, which is not affected by that problem.

I don't believe the maximum values even can change due to the fundamental limitations of the x86 architecture. Such a change would require very massive modifications to how ASLR works inside the kernel. With grsecurity (which does involve massive changes to ASLR of that type), vm.mmap_nod_max_bits has a maximum value of 27, rather than 32, even on x86_64. I imagine this is because grsecurity provides additional ASLR features which conflict with some of the vanilla ASLR space and limits the size. Just something to keep in mind when/if Tails includes grsecurity.

Are there any existing sysctls that overflow this way? (That's a real question, again: I'm not an expert in this area and need input from people like you.)

Some do:

```bash
# n=1; while true; do sysctl -w kernel.perf_event_paranoid=$((n++)); done
kernel.perf_event_paranoid = 1
kernel.perf_event_paranoid = 2
kernel.perf_event_paranoid = 3
...  
kernl.perf_event_paranoid = 490812
kernel.perf_event_paranoid = 490813
kernel.perf_event_paranoid = 490814
^C
# sysctl -a -r paranoid
kernel.perf_event_paranoid = 490814
```

Some don't:

```bash
# n=1; while true; do sysctl -w kernel.kptr_restrict=$((n++)); done
kernel.kptr_restrict = 1
kernel.kptr_restrict = 2
sysctl: setting key "kernel.kptr_restrict": Invalid argument
kernel.kptr_restrict = 3
...  
sysctl: setting key "kernel.kptr_restrict": Invalid argument
kernel.kptr_restrict = 155141
sysctl: setting key "kernel.kptr_restrict": Invalid argument
kernel.kptr_restrict = 155142
sysctl: setting key "kernel.kptr_restrict": Invalid argument
kernel.kptr_restrict = 155143
^C
# sysctl -a -r kptr
kernel.kptr_restrict = 2
```

It seems however I was wrong about it overflowing and wrapping to 0. The former returns EINVAL when it exceeds INT32_MAX (2^31-1, or 2147483647). It will, however increment for a long, long time. Oddly enough, on my Gentoo system, kernel.perf_event_paranoid acts more like kernel.kptr_restrict, refusing above the maximum sane value (3 on this system). The sysctl interface does not seem very consistent among different systems and kernel versions. The entirety of the sysfs (and procfs) virtual filesystems are in general a mess.

These were tested on Tails 2.6 under QEMU with KVM.

In passing, to protect against such an overflow, I could adjust Kees' script to recall what's the biggest value that's ever been accepted by the sysctl, and reset it back to that value if writing to the sysctl ever decreases its resulting value. Of course, that's racy when another actor
(systemd) comes into play; but I'm very confident that in practice, the "incrementally increase until overflow + reset" will simply never take enough time for systemd to be willing to kill the script.

Why not also use such a script for kernel.perf_event_paranoid, and kernel.kptr_restrict, in case those ever have increased maximum values in the future? A previous ticket was closed because kernel.perf_event_paranoid's default value would be set to 3 in a future Debian version, so there would be no reason to increment it automatically, as we already know that the default will be 3 for the foreseeable future.

IMO, the script still seems useless to me. It seems to be intended entirely for architecture-agnostic systems, not systems which we know will always have a maximum of 32 bits for 64 bit userspace, and 16 for 16 bit userspace, on all deployed systems.

OT: you might be interested in #11886. I really appreciate your contributions.

Thanks. The newer 4.8 kernels have a lot of great security features, thanks to the KSPP, like SLUB sanitizing (like poisoning, but initializing with zeros for security), and page freeing, like PAX_SANITIZE. I forget how it's enabled, but I'm pretty sure it's just an additional flag to slub_debug=P to turn the poisoning into zeroizing. Anyway I'll look into it. It would really be better to focus on trying to get grsecurity into Tails (whether by getting AppArmor compatible with overlays, or getting Corsac to be willing to put the PAX compatibility patch into his linux-grsec Debian package).

#10 - 11/25/2016 02:28 PM - intrigeri
- Target version changed from Tails_2.9.1 to Tails 2.10

#11 - 12/18/2016 12:42 PM - intrigeri
- Target version changed from Tails 2.10 to Tails_2.12

(I had to take over a bunch of more urgent sysadmin tasks so I'll postpone this one.)

#12 - 01/06/2017 04:04 AM - cypherpunks
If #7649 goes through and PAX_UDEREF is enabled, vm.mmap_rnd_bits will have a maximum value of 27, not 32. No change to vm.mmap_rnd_compat_bits will be needed, and its maximum value can still be increased to 16.

#13 - 01/06/2017 08:53 AM - intrigeri
If #7649 goes through and PAX_UDEREF is enabled, vm.mmap_rnd_bits will have a maximum value of 27, not 32. No change to vm.mmap_rnd_compat_bits will be needed, and its maximum value can still be increased to 16.

FYI #7649 is long term, while I'd like to address this ticket soonish.
intrigeri wrote:

If #7649 goes through and PAX_UDEREF is enabled, vm.mmap_rnd_bits will have a maximum value of 27, not 32. No change to vm.mmap_rnd_compat_bits will be needed, and its maximum value can still be increased to 16.

FYI #7649 is long term, while I'd like to address this ticket soonish.

Sure, I'm just mentioning it here so it won't come as a surprise if the tests fail in the future.

#15 - 03/08/2017 11:31 AM - intrigeri
- Target version changed from Tails_2.12 to Tails_3.0

#16 - 04/29/2017 12:59 PM - intrigeri
- Target version changed from Tails_3.0 to Tails_3.2

#17 - 06/29/2017 10:25 AM - intrigeri
- Blocks Feature #13234: Core work 2017Q3: Foundations Team added

#18 - 06/29/2017 10:26 AM - intrigeri
- Priority changed from Low to Normal

This would have mitigated Stack Clash.

#19 - 07/24/2017 06:26 AM - intrigeri
- Subject changed from Increase mmap randomization from 8 to 16 bits to Increase mmap randomization to the maximum supported value

(We now ship a 64-bit userland :) )

#20 - 07/24/2017 06:27 AM - intrigeri
- Description updated

#21 - 07/24/2017 06:33 AM - intrigeri
- Description updated

#22 - 07/24/2017 06:47 AM - intrigeri
- Status changed from Confirmed to In Progress
- % Done changed from 0 to 10
- QA Check deleted (Dev Needed)
In the end I was convinced by the "hard-code the current maximum value" approach: indeed the chances these max values change are slim, so let's KISS.

#23 - 07/24/2017 06:53 PM - intrigeri
- Assignee changed from intrigeri to anonym
- % Done changed from 10 to 50
- QA Check set to Ready for QA

#24 - 08/15/2017 02:16 PM - anonym
- Status changed from In Progress to 11
- Assignee deleted (anonym)
- % Done changed from 50 to 100
- QA Check changed from Ready for QA to Pass

#25 - 09/28/2017 06:49 PM - anonym
- Status changed from 11 to Resolved