Closed · Issue created 1 year ago by Demi Marie Obenour

Reference count overflow in shm leads to use-after-free

The shared memory code uses an `int` for the reference count. On 64-bit systems, a malicous client can create so many references that the `int` overflows. This is undefined behavior, but it will most likely cause the `int` to overflow, causing an exploitable use-after-free. A successful exploit will result in the execution of arbitrary code in the context of the Wayland compositor.

To fix this bug, `uintptr_t` (or `intptr_t`) should be used for all reference counts. There are only `UINT_MAX` possible distinct pointers, so a `uintptr_t` reference count can never overflow, and an `intptr_t` reference count cannot overflow if the referencing and referenced objects are at least 2 bytes. This assumes that `INTPTR_MAX == (UINTPTR_MAX >> 1) && UINTPTR_MAX == ((uintptr_t)INTPTR_MAX << 1) + 1`, which can be checked with a C11 `_Static_assert`.

To upload designs, you'll need to enable LFS and have an admin enable hashed storage. More information

Tasks 0

No tasks are currently assigned. Use tasks to break down this issue into smaller parts.

Linked items 0

Link issues together to show that they're related. Learn more.

Related merge requests 2

util: Limit size of wl_map
!231

Draft: Limit map
derekf/wayland!1

When these merge requests are accepted, this issue will be closed automatically.

Activity

Developer

Derek Foreman @derekf · 1 year ago

Ouch. Thanks for appropriately filing this as a confidential issue.

Trying to get a handle on how easy this is to trigger so we can decide what the severity is...

Are we talking about the internal_refcount in wl_shm_pool?

I haven't been down these paths in a while - isn't the only way (for a client) to increase that refcount allocating a buffer from a pool? A compositor could ref the pool intentionally quickly and without overhead, but that's not an exploitable condition.

So we'd need to allocate buffers 2^31 times to wrap the refcount into negatives? That's 40 bytes for the struct and probably 16 bytes for malloc bookkeeping. Plus we hit wl_resource_create() which allocates a struct wl_resource at 128 bytes (+ bookkeeping). We also add the object to the wl_map, which I think is just 8 bytes per object.

In total, if I'm doing my math right, 2^31 * (56 + 144 + 8), so conservatiely 416G of memory allocation to get to this point. We do touch it all, so it needs to exist as RAM or swap.

I think we can ignore all client side allocations, as the ideal malicious client would NOT(edit) use the wayland libraries, and wouldn't keep a map. I also think we can ignore the compositor's use of external references for this analysis (it would lower the buffer creation requirements, but much more uncontrollable compositor side things come into play, many of which probably increase the memory footprint as well)

Once we get into this state the next buffer deletion will cause the pool to be unmapped and freed. After that, any attempt to render one of the buffers from the tainted pool will attempt to read the unmapped space (and on compositors that use external references, trying to use the freed pool for refcounting, which is probably worse).

If the malicious code sets up a surface and attaches one of the tainted buffers, the unmapped region (or whatever is now mapped there) will be read with intent to use as pixel data. I don't think will can be used to do anything but crash, or I guess leak compositor memory to the display?

Can anyone see a path to arbitrary code execution here? I think shm_pool_finish_resize() is going to explode in the mremap() if it even gets that far. This leaves us with, I think, shm_pool_create_buffer/shm_pool_unref which can be used to manipulate a specific address on the heap (&tainted_pool->internal_refcount) to a wide family of values. We could also perhaps trigger multiple free()s on the pool.

There's an (unreleased as of yet, but in main branch) assertion in the refcounting code that will make this trickier to hit - I think the malicious code would have to allocate buffers until the refcount wrapped to 1, so I guess that would double the memory requirements. It would also limit our ability to use shm_pool_unref() to manipulate the internal_refcount down after free.

As to the long term fix, I'd personally be happier with just disconnecting the client once the internal_refcount exceeded some arbitrary but ridiculous limit (can anyone see a need for 10000 buffers from the same pool?). We don't need to let a client consume hundreds of gigabytes to know it's either buggy or evil.

Edited by Derek Foreman 1 year ago

Owner

Simon Ser @emersion · 1 year ago

I generally use `size_t` for ref'counts.

> the ideal malicious client would use the wayland libraries

I think you meant *wouldn't use*.

> As to the long term fix, I'd personally be happier with just disconnecting the client once the internal_refcount exceeded some arbitrary but ridiculous limit

That's compositor policy IMHO.

Author · Contributor

Demi Marie Obenour @DemiMarie · 1 year ago

I recommend using either `uint64_t`, or something like the Linux kernel's `refcount_t`. The reason is that if there is a reference leak, the use-after-free can be triggered *without* having to consume gigabytes of memory, provided that a 32-bit reference count is used. Using saturating arithmetic prevents this entirely, and using a 64-bit counter delays it until after a ridiculous amount of work.

Author · Contributor

Demi Marie Obenour @DemiMarie · 1 year ago

> In total, if I'm doing my math right, 2^31 * (56 + 144 + 8), so conservatiely 416G of memory allocation to get to this point. We do touch it all, so it needs to exist as RAM or swap.

How many of these allocations are identical? That affects exploitability on systems that use compressed memory or same-page merging.

Please register or sign in to reply

Developer

Derek Foreman @derekf · 1 year ago

Yup, thanks, I did mean "wouldn't", and have edited my post to reflect that.

Owner

Daniel Stone @daniels · 1 year ago

> So we'd need to allocate buffers 2^31 times to wrap the refcount into negatives? That's 40 bytes for the struct and probably 16 bytes for malloc bookkeeping. Plus we hit wl_resource_create() which allocates a struct wl_resource at 128 bytes (+ bookkeeping). We also add the object to the wl_map, which I think is just 8 bytes per object.
>
> In total, if I'm doing my math right, 2^31 * (56 + 144 + 8), so conservatiely 416G of memory allocation to get to this point. We do touch it all, so it needs to exist as RAM or swap.

This is pathological enough that I'd be happy to `abort()` on it to be honest. I'd hoped that we'd exhaust object ID space by here, but `WL_SERVER_ID_START` is `0xff000000` so we can in fact create nearly twice as many objects.

Maybe kill the client if its live object map grows beyond, say, 0xf0000? A million objects should be enough for everyone, right?

Owner

Daniel Stone @daniels · 1 year ago

It's worth noting that the client would have to have a handcrafted protocol implementation, else you'd eat all the `wl_proxy` allocations as well. Even then, with default values for the socket queue size, you'd need ~726,000 iterations of the server reading the maximum socket queue size to get enough wl_shm_pool.create_buffer requests through.

Please register or sign in to reply

Developer

Derek Foreman @derekf · 1 year ago

> Maybe kill the client if its live object map grows beyond, say, 0xf0000? A million objects should be enough for everyone, right?

I feel that's reasonable. Also has the added benefit of being far enough removed from refcounting that I'm not certain it would require co-ordinated disclosure to land such a patch?

Daniel Stone @daniels · 1 year ago — Owner

There are a couple of automotive environments where the compositor lives across a trust boundary from clients, but apart from that and Qubes, I don't believe there are many around for which a compositor compromise would be catastrophic.

I think it's probably still worth a CVE mind, and backporting.

[edit: sandboxed clients like Flatpak are in a different privilege domain, and there's also ChromeOS]

Edited by Daniel Stone 1 year ago

Demi Marie Obenour @DemiMarie · 1 year ago — Author, Contributor

I was going to say this does not matter for Qubes OS, but then realized that future (reasonable) enhancements to the GUI protocol could make it exploitable. That said, sandboxed Flatpak applications are certainly less trusted than the compositor, so a CVE is definitely justified.

In practice, I doubt that this is exploitable (beyond a DoS) on most systems, purely due to the amount of memory required. Nevertheless, a client should not be able to crash the compositor, so some sort of resource limit is definitely needed. An object count limit is a good place to start, but I wonder if finer-grained accounting would be worthwhile. That can certainly come later, though.

Finally, as a Qubes OS developer, thank you for considering Qubes OS when analyzing the impact of this bug! Qubes OS uses a custom GUI protocol that is currently backed by X11, but I am working on having it be backed by Wayland instead.

Edited by Demi Marie Obenour 1 year ago

Daniel Stone @daniels · 1 year ago — Owner

Righto. I'll chase a CVE number through X.Org. Do you want to do the writeup & make sure you're credited for it since you've done all the hard work and analysis here (and many thanks for that)?

Demi Marie Obenour @DemiMarie · 1 year ago — Author, Contributor

Thanks for the incredibly quick response! I recently spent several months getting some RPM vulnerabilities patched, so a next day response (on a weekend, no less!) is refreshing.

When should I send the writeup to distros@vs.openwall.com? Also, while the object map size limit fixes the immediate vulnerabilities (both DoS and LPE), I recommend backporting further hardening measures. The reason is that reference leaks are quite common in practice, and I know of at least two (now fixed) vulnerabilities in the FreeBSD kernel that resulted from them. Furthermore, Rust considers failing to call an object's destructor to be safe in most cases, and I would prefer if this did not lead to use-after-free via refcount overflows. Worst of all, reference leaks do not require gigabytes of memory to exploit. Preventing refcounts from wrapping reduces the impact to denial of service at worst.

Daniel Stone @daniels · 1 year ago — Owner

We usually disclose through the xorg-security@ list and get CVEs through there to co-ordinate with distros etc. I'm just drafting up an email there now and I'll CC you.

I agree with you on refcount hardening, but I think I'd again prefer to impose a stupidly-high limit rather than try to push us up to the full 64-bit boundary. Is the refcount hardening something you'd be interested in working on, or?

Hope you have more exciting things planned for your weekend after this!

Daniel Stone @daniels · 1 year ago — Owner

Thinking about it more, if you can convince the compositor to hold a ton of references to the pool, you could make it happen with fewer objects; `wl_shm_buffer_ref_pool()` exists and asserts that the external + internal refcount stays positive. So I think you'd 'just' need to get it to take `(2 << 31)` external refs, then you'd only need a single live client buffer to break the addition test we do in the unref path.

Enlightenment, KWayland, and wlroots all explicitly take a ref on the pool here. But this is harder rather than easier to practically exploit: I can't see a way which requires you to do this without at least a live surface object per ref, and these all have larger internal tracking structures for their surfaces than our internal per-buffer structures.

So I think what we need to do is:

- change the `(internal_refcount + external_refcount) < 0` test in the pool-unref path to check if either of them are non-zero
- make a new static ref function symmetrical with unref and change the internal and external refs to use this
- impose an arbitrary limit on the individual refcounts to ensure neither of them climb beyond some silly number which keeps us far from overflow on either or both

Am I on the right path?

Demi Marie Obenour @DemiMarie · 1 year ago — Author, Contributor

Probably; I will need to think about it a bit more.

Please register or sign in to reply

Demi Marie Obenour @DemiMarie · 1 year ago — Author, Contributor

You are. What about adding:

```
#ifdef NDEBUG
# error wayland-shm.c must be built with assertions enabled
#endif
#include <assert.h>

// lots of code
static void
shm_pool_unref(struct wl_shm_pool *pool, bool external)
{
	if (external) {
		pool->external_refcount--;
		if (pool->external_refcount == 0)
			shm_pool_finish_resize(pool);
	} else {
		pool->internal_refcount--;
	}
	assert(pool->internal_refcount >= 0 && "internal reference count overflowed");
	assert(pool->external_refcount >= 0 && "external reference count overflowed");
	if (pool->internal_refcount != 0 || pool->external_refcount != 0)
		return;

	assert(munmap(pool->data, pool->size) == 0 && "munmap() failed — possible memory corruption");
	free(pool);
}
```

That said, this idea needs work.

Edited by Demi Marie Obenour 1 year ago

Derek Foreman @derekf · 1 year ago — Developer

I realize this code was already an assert, but maybe it should be wl_abort() instead? On the off chance someone's actually compiling out assertions.

I think the logic is inverted on the early return? Isn't that the only case in which we want to perform the munmap/free?

Technically, getting into a refcount < 0 case isn't only possible via overflow - a compositor bug could cause the external refcount to go negative. I think that's probably more likely to happen than the overflow case, so the text could be a little confusing.

I know some users of this library are very hostile towards assert/abort in general, so I don't know that adding a new one for munmap() failure is going to be popular. I don't see how we can trust anything that occurs after the point of munmap() failure, so I certainly wouldn't complain about it.

Derek Foreman @derekf · 1 year ago — Developer

oops, I missed the #ifdef NDEBUG #error. I think I'd prefer to use wl_abort() instead, but don't really have a strong opinion. :)

Pekka Paalanen @pq · 1 year ago — Maintainer

Let's use `wl_abort()`.

Forcing asserts on just makes things more annoying to whoever wants to build stuff.

Demi Marie Obenour @DemiMarie · 1 year ago — Author, Contributor

That's fine.

Please register or sign in to reply

Derek Foreman @derekf · 1 year ago Developer

I do like @daniels suggestion of having a symmetric ref function (`static void shm_pool_ref(struct wl_shm_pool *pool, bool external)` perhaps?) and limiting the combined refcount before it gets out of hand. Then we'd be asserting exclusively on refcounting bugs and not overflow bugs in the unref function...

Demi Marie Obenour @DemiMarie · 1 year ago Author Contributor

What about something like:

```
#ifdef NDEBUG
# error wayland-shm.c must be built with assertions enabled
#endif
#include <assert.h>

typedef uint32_t wl_refcount; // no _t because POSIX reserves that namespace

static void
wl_ref(wl_refcount *refcount)
{
	const wl_refcount count = *refcount;
	assert(count < (UINT32_MAX / 2) &&
		"Reference count exceeded UINT32_MAX / 2 — possible reference leak");
	*refcount = count + 1;
}

static void
wl_unref(wl_refcount *refcount)
{
	const wl_refcount count = *refcount;
	assert(count > 0 && "too many calls to wl_unref");
	*refcount = count - 1;
}

static void
shm_pool_unref(struct wl_shm_pool *pool, bool external)
{
	if (external) {
		wl_unref(&pool->external_refcount);
		if (pool->external_refcount == 0)
			shm_pool_finish_resize(pool);
	} else {
		wl_unref(&pool->internal_refcount);
	}
	if (pool->internal_refcount != 0 || pool->external_refcount != 0)
		return;

	assert(munmap(pool->data, pool->size) == 0 &&
		"munmap() failed — possible memory corruption");
	free(pool);
}

static void
shm_pool_ref(struct wl_shm_pool *pool, bool external)
{
	assert((pool->internal_refcount || pool->external_refcount) &&
		"Calling shm_pool_ref on an already freed pool");
	wl_ref(external ? &pool->external_refcount : &pool->internal_refcount);
}
```

The major advantage of this approach is that unsigned integers are defined to wrap, whereas signed integer overflow is already undefined behavior. Of course, the `assert`s can be replace with explicit calls to `write` / `fprintf` /etc and `wl_abort` .

I am no expert on reference count hardening, but this can be further improved once the issue becmes public.

Derek Foreman @derekf · 1 year ago Developer

@DemiMarie I can format this as a patch and bring it into my MR for review there if you'd like to forego making your own private fork and messing around with the gitlab confidential MR work flow.

As usual, we'll need your `Signed-off-by:` attribution before we can land it.

Demi Marie Obenour @DemiMarie · 1 year ago Author Contributor

That is fine, but be sure to review it first; this was meant as "here is an idea" and not "this can go in straight away."

Please register or sign in to reply

Derek Foreman created merge request derekf/wayland!1 (closed) to address this issue 1 year ago

Derek Foreman @derekf · 1 year ago Developer

Seems I have little understanding of how confidential merge requests work, and I'm unsure if anyone here can access my MR...

My MR does not address refcounting, which needs to be handled (if just to close the internal+external refcount overflow), but it adds a cap to the number of elements in a wl_map.

Unsure whether it belongs here due to its relevance to this issue, or if I should create a new confidential issue for that as well.

Demi Marie Obenour @DemiMarie · 1 year ago Author Contributor

I certainly cannot see the MR.

Daniel Stone @daniels · 1 year ago Owner

I can, but only because I'm an admin. The MR is only on your own repo, not on this one ... you'll need to push to a branch in this repo to make a confidential MR I think? (You're the first one to do this, congrats.)

Derek Foreman @derekf · 1 year ago Developer

> I certainly cannot see the MR. Disappointing, but unsurprising. Sorry about that!

> I can, but only because I'm an admin. The MR is only on your own repo, not on this one ... you'll need to push to a branch in this repo to make a confidential MR I think? (You're the first one to do this, congrats.)

I think since this is a public repo, anything I push to a branch in it will have immediate visibility, so I don't think that's the way forward?

https://docs.gitlab.com/ee/user/project/merge_requests/confidential.html gives me the impression that I must fork the repository, and make my fork private (which I have done).

I think I've failed by not creating my fork in the same group or subgroup as the parent public repository, which would share ownership permissions? However, that still doesn't seem like what someone would generally want, as it will frequently leave the person who reported the issue out of the loop.

The merge request apparently ends up being against my private fork. I think once that's merged I'd have to do a MR from that new branch to the public main branch, and that would immediately (and appropriately) be visible to all.

It may be that my best way forward is just giving all participants here visibility into my private fork?

Please register or sign in to reply

Derek Foreman @derekf · 1 year ago Developer

With @daniels' help in creating a secure fork, I've put together a merge request that should theoretically have reasonable permissions...

https://gitlab.freedesktop.org/wayland/security/wayland/-/merge_requests/1

Derek Foreman mentioned in merge request !231 (merged) 7 months ago

Derek Foreman closed via commit b19488c7 6 months ago

Daniel Stone made the issue visible to everyone 4 months ago

Please register or sign in to reply