Muse + Stripe Card Readers — How It Works on the Museum Floor

Audience: Your IT / network engineer **Purpose:** Explain every connection that happens when a guest pays at a Stripe card reader on the museum floor, so firewalls, Wi-Fi, and content filtering can be configured to let the system work reliably.

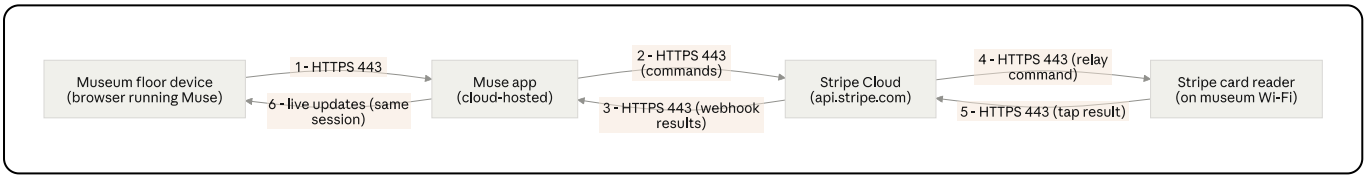
---

1. The short version

Muse uses Stripe's **cloud-/server-driven Terminal model**. This is important for your network setup:

- The museum floor device (a browser running the Muse app) **never talks directly to the card reader**.
- The card reader **does not need to be reachable from the floor device or the local network**. There is no local pairing, no local IP discovery, no inbound ports to open to the reader.
- Instead, **everything is brokered through Stripe's cloud.** Muse tells Stripe "charge $X on reader Y," Stripe relays that command to the reader over the internet, the reader takes the tap/dip, sends the result back up to Stripe, and Stripe notifies Muse.

So there are **three independent participants**, and the only thing they share is **outbound HTTPS (port 443) to Stripe's cloud**:



Every arrow is an outbound HTTPS/TLS connection on port 443. Nothing in this system requires an inbound port to be opened on the museum network, and nothing requires the floor device and the reader to see each other on the LAN.

---

2. The participants

Participant	What it is	Where it lives
Museum floor device	A tablet/PC/kiosk running a web browser pointed at the Muse app	Museum Wi-Fi / LAN
Muse app	The application itself, hosted in the cloud	Cloud (not on museum premises)

Participant	What it is	Where it lives
Stripe Cloud	Stripe's payment platform and Terminal service at `api.stripe.com`	Cloud
Stripe card reader	A Stripe internet-connected smart reader (e.g. WisePOS E / Reader S700)	Museum Wi-Fi

Two notes that matter for network planning:

- The **Muse app is not hosted on the museum's network.** When the floor device "talks to Muse," that traffic leaves the building over the internet.
- The card reader is registered to the museum's own Stripe account. This doesn't change the network picture — it's still all HTTPS to Stripe.

---

3. The connections, one by one

Connection 1 — Floor device → Muse app (the staff UI)

- **Direction:** Outbound from the floor device to the internet.
- **Destination:** The Muse app at `app.musesoftware.ai`.
- **Protocol/port:** HTTPS / TCP 443.
- **What happens:** A staff member loads the Muse floor screen, selects items/admission, and presses "charge." Standard secure web traffic.

Connection 2 — Muse → Stripe (commands)

- **Does not affect the museum network.** This call originates from the Muse app in the cloud, not from the museum building, and never traverses the museum firewall. Listed only for completeness: it's how Muse tells Stripe to create a payment and run it on a specific reader.

Connection 3 — Stripe → Muse (webhooks / results)

- **Does not affect the museum network.** This is a cloud-to-cloud notification from Stripe back to the Muse app, so the floor screen and records update after a payment. It never enters the museum network. Listed only for completeness.

Connection 4 — Stripe → Card reader (relay the command)

- **Direction:** The reader holds a persistent secure connection **out to Stripe**, and Stripe pushes the command down that connection.
- **Destination:** Stripe's Terminal cloud (Stripe-owned domains, see §4).
- **Protocol/port:** HTTPS/TLS / TCP 443.

- **What happens:** Stripe tells the reader "collect a tap/dip/swipe for $X," and the reader wakes up and prompts the guest.
- **Network impact for the museum: This is the one you must allow.** The reader must be able to reach Stripe's cloud over the internet from the museum Wi-Fi.

Connection 5 — Card reader → Stripe (send the result)

- **Direction:** Outbound from the reader to Stripe.
- **Destination:** Stripe's Terminal cloud.
- **Protocol/port:** HTTPS/TLS / TCP 443.
- **What happens:** The reader captures the card, encrypts it, and sends the result back up to Stripe. Stripe then notifies Muse so the floor screen learns the outcome.
- **Network impact for the museum:** Same as Connection 4 — the reader needs outbound internet to Stripe.

Connection 6 — Muse → Floor device (live updates)

- **Direction:** Data travels from Muse back to the floor device — but it rides **back down the same connection the device already opened in Connection 1.** It is **not** a separate, inbound connection into the museum network.
- **Destination:** The floor device, over its existing session with `app.musesoftware.ai`.
- **Protocol/port:** HTTPS / TCP 443 (the same outbound session).
- **What happens:** When a payment finishes, Muse pushes the result down to the floor screen so it updates to "Approved/Declined" automatically, with no manual refresh.
- **Network impact for the museum: None beyond Connection 1.** Because it reuses the device's own outbound session, **no inbound port needs to be opened to the floor device.**

> **Key takeaway:** From the museum's network, the only device that needs special attention is **the card reader**, and all it needs is **clear outbound internet access to Stripe on port 443** (plus the basics: DHCP, DNS, NTP). The floor device just needs normal web access.

---

4. What to allow on the firewall / Wi-Fi

A. For the card reader (the critical part)

Stripe's internet-connected readers need **outbound TCP 443** to Stripe's cloud, plus standard network services. At minimum, allow the reader's VLAN/SSID to reach:

Need	Destination	Port/Protocol
Standard Stripe domains	`api.stripe.com`, `terminal.stripe.com`, `js.stripe.com` (and other `*.stripe.com` hosts)	TCP 443 (HTTPS/TLS)
Stripe Terminal domains	`api.emms.bbpos.com`, `armada.stripe.com`, `gator.stripe.com`, `stripe-point-of-sale-us-west-2.s3.us-west-2.amazonaws.com`, `*.terminal-events.stripe.com`	TCP 443 (HTTPS/TLS)
Name resolution	Your DNS server	UDP/TCP 53
IP address	DHCP	UDP 67/68
Clock sync (TLS needs accurate time)	NTP: `time.android.com`, `time.cloudflare.com`, `2.android.pool.ntp.org`	UDP 123

> Both sets of domains must be allowlisted — the standard Stripe domains **and** the Stripe Terminal domains. A `*.stripe.com` wildcard does not cover `api.emms.bbpos.com`, the S3 bucket, or the NTP hosts, so include those explicitly.

> ⚠️ **Confirm the current domain list with Stripe.** Stripe occasionally updates the exact hostnames and IPs their Terminal hardware uses. The authoritative, always-current sources are:
>
> - **Terminal network requirements:** https://docs.stripe.com/terminal/network-requirements
> - **Stripe IPs, domains & ports:** https://docs.stripe.com/ips
>
> Treat the table above as the working set and verify against those pages before locking down the firewall.

If your firewall can't use wildcard hostnames (`*.stripe.com`)

Some firewalls only accept exact FQDNs or IP addresses. In rough order of preference:

1. **Enumerate the specific FQDNs instead of wildcards.** Pull the exact hostnames from Stripe's domains page and add each one as a literal rule (e.g. `api.stripe.com`, `terminal.stripe.com`, `js.stripe.com`, `api.emms.bbpos.com`, `armada.stripe.com`, `gator.stripe.com`, `stripe-point-of-sale-us-west-2.s3.us-west-2.amazonaws.com`, and `*.terminal-events.stripe.com`). This is the closest equivalent to wildcards. The trade-off: if Stripe adds a new subdomain, you'll need to add it too, so re-check the list periodically.
2. **Allow all outbound TCP 443 from the reader's VLAN/SSID only.** Put the readers on a dedicated, isolated network segment that has nothing else on it, and permit

unrestricted outbound 443 from just that segment. This is the most robust and lowest-maintenance option, and because the segment contains only payment readers, the security exposure is minimal.

3. **Use FQDN-aware filtering on a proxy / next-gen firewall.** Many appliances (Palo Alto, Fortinet, Meraki, etc.) support domain/SNI-based rules even when classic ACLs don't — match on the TLS SNI hostname rather than a wildcard ACL.
4. **Avoid IP-address allowlisting.** Stripe does **not** publish a fixed, stable list of IP ranges for Terminal, and the addresses change without notice. Hard-coding IPs will eventually break payments. Use domain/FQDN-based rules (options 1–3) instead.

Whichever option you choose, the captive-portal, TLS-inspection, DNS/DHCP/NTP, and inbound-port notes below still apply.

Additional reader recommendations:

- **No inbound ports** need to be opened to the reader. Do not port-forward to it.
- **Do not require a captive portal / sign-in splash page** on the reader's network. Smart readers can't click through captive portals; put them on an SSID/VLAN that has open outbound internet without interception.
- **Avoid TLS/SSL inspection (deep packet inspection)** on the reader's traffic to Stripe. The reader pins/validates Stripe's certificates; an intercepting proxy will break the secure connection. Whitelist Stripe domains to bypass inspection.
- **Wi-Fi:** WPA2-PSK (or WPA2-Enterprise if supported by your reader model) on **2.4 GHz and/or 5 GHz** per the reader's spec. Ensure strong, stable signal at each reader's physical location and that **client/AP isolation** does not block the reader's outbound path to the gateway.
- Keep the reader on a **stable IP/DHCP lease** and ensure the reader can renew DHCP and reach DNS/NTP at all times.

B. For the museum floor device (the staff screen)

This is just a web browser. Allow outbound **HTTPS 443** to:

- `app.musesoftware.ai`
- `api.stripe.com` and `js.stripe.com` (Stripe assets the app may load)

No inbound ports, no special config. If you run SSL inspection generally, allow-list `app.musesoftware.ai` and `*.stripe.com` so the app and any Stripe components load cleanly. If wildcards aren't supported, add `api.stripe.com` and `js.stripe.com` as literal FQDNs alongside `app.musesoftware.ai`.

Recommended: raise the TCP connection / idle timeout to cover the open day

Many firewalls default to a short **TCP connection (idle) timeout** — often **15 minutes** — which silently drops long-lived connections once they go quiet. Stripe requires that Terminal reader network sessions (including idle sessions) last **at least an entire workday**,

and the firewall's TCP idle timeout is typically a global setting that applies to all traffic, so set it high enough to cover the full open day.

Recommendation: set the firewall's TCP connection/idle timeout to **at least the museum's full daily open duration plus a ~1-hour buffer —** so connections that sit quiet between transactions are never torn down during opening hours.

Formula: `(daily open hours + 1 hour buffer) × 60 = timeout in minutes`

Museum open hours per day	Suggested timeout
6 hours (e.g. 10am–4pm)	**420 min** (7 h)
8 hours (e.g. 9am–5pm)	**540 min** (9 h)
10 hours (e.g. 9am–7pm)	**660 min** (11 h)
12 hours (e.g. 8am–8pm)	**780 min** (13 h)

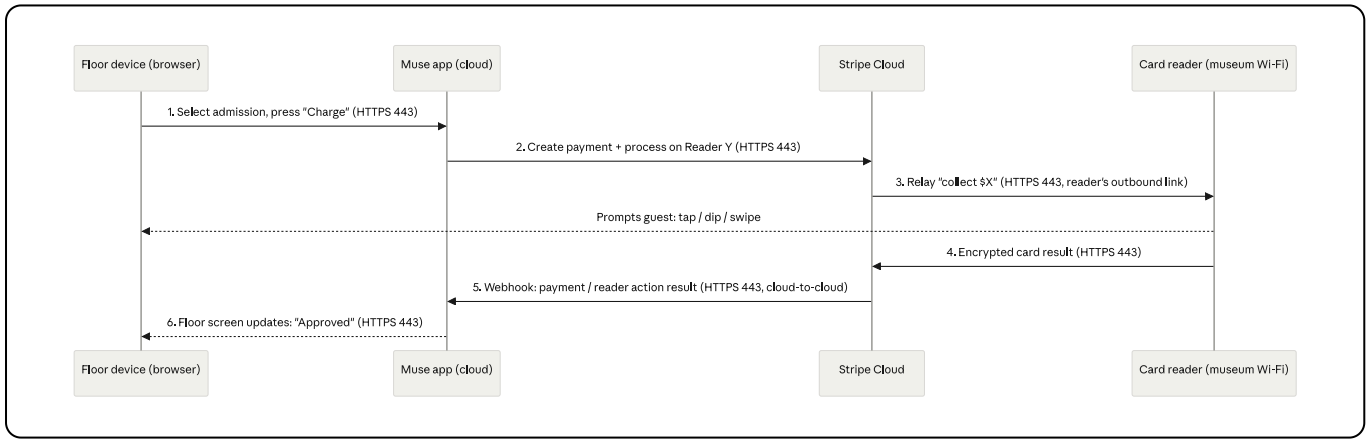
> Example from a live deployment: a museum's default was **15 minutes**; raising it to **500 minutes** (a full open day) resolved the dropped connections. Pick the row that matches your hours, or just round up to comfortably cover open-to-close.
> This only affects how long an *idle* connection is allowed to live; it does not weaken security or open any inbound access.

C. For Muse ↔ Stripe (Connections 2 & 3)

Nothing to configure on the museum network. Both of these legs are cloud-to-cloud and never touch the museum's firewall or Wi-Fi.

---

5. End-to-end example: a guest pays admission



Notice that the **floor device and the reader never communicate directly —** steps 3 and 4 go through Stripe's cloud, not across the local network. That's the defining feature of this integration and why the local network only has to worry about giving each device clean outbound internet to Stripe.

---

6. Quick checklist for the network engineer

- [ ] Put card readers on an SSID/VLAN with **open outbound TCP 443 to Stripe domains** (see §4A).
- [ ] Allow **DNS (53), DHCP (67/68), and NTP (123)** for the readers.
- [ ] **No captive portal** and **no TLS inspection** on reader traffic to Stripe.
- [ ] **No inbound ports** required to the readers — do not port-forward.
- [ ] Ensure strong, stable Wi-Fi at each reader location; disable client isolation if it blocks the gateway path.
- [ ] Allow the floor devices outbound **443 to** `app.musesoftware.ai` **and** `*.stripe.com` (or literal FQDNs if wildcards aren't supported).
- [ ] **Don't cut idle/long-lived HTTPS connections** (no aggressive proxy idle-timeout).
- [ ] **Raise the TCP connection/idle timeout to cover the full open day** (open hours + ~1 h buffer; see §4B table) — short defaults like 15 min drop long-lived connections, and Stripe requires reader sessions to last at least a full workday.
- [ ] **No inbound ports** to the floor device either — live updates ride its existing outbound session.
- [ ] If wildcards aren't supported, use **literal FQDNs or an isolated reader VLAN with open outbound 443 — not** IP allowlisting (Stripe's IPs aren't fixed).
- [ ] Nothing to open for Muse↔Stripe — that traffic is cloud-to-cloud.
- [ ] **Verify the exact Stripe domain/IP list** before finalizing rules: https://docs.stripe.com/terminal/network-requirements and https://docs.stripe.com/ips

---

If anything in the network behaves unexpectedly (readers showing offline, payments hanging), the usual culprits are captive portals, TLS/DPI inspection, or NTP/DNS being blocked on the reader's network — start there.