Online Payments Processing Platform

Owner: A development team
Reviewer: A security architect
Contributors: development engineers, product managers, security architects
Date Generated: Tue Oct 07 2025

Executive Summary

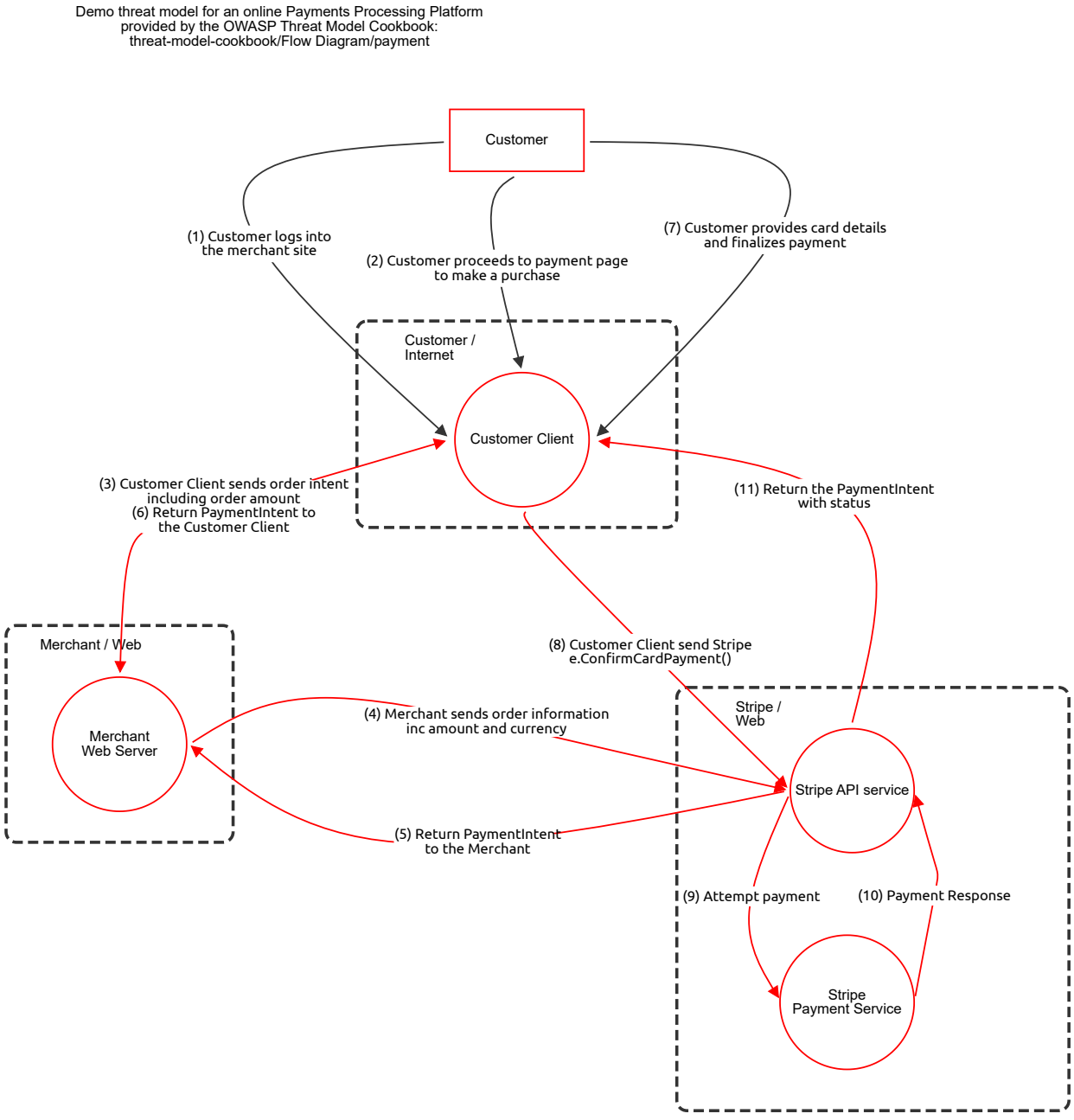
High level system description

This threat model has been provided by the OWASP Threat Model Cookbook:
threat-model-cookbook/Flow Diagram/payment

Summary

Total Threats	27
Total Mitigated	0
Total Open	27
Open / Critical Severity	0
Open / High Severity	15
Open / Medium Severity	12
Open / Low Severity	0

Payment


Demo threat model for an online Payments Processing Platform
provided by the OWASP Threat Model Cookbook:
threat-model-cookbook/Flow Diagram/payment
Customer
(1) Customer logs into
the merchant site
(2) Customer proceeds to payment page
to make a purchase
(7) Customer provides card details
and finalizes payment
Customer /
Internet
Customer Client
(3) Customer Client sends order intent
including order amount
(6) Return PaymentIntent to
the Customer Client
(11) Return the PaymentIntent
with status
Merchant / Web
Merchant
Web Server
(8) Customer Client send Stripe
e.ConfirmCardPayment()
(4) Merchant sends order information
inc amount and currency
(5) Return PaymentIntent
to the Merchant
Stripe /
Web
Stripe API service
(9) Attempt payment
(10) Payment Response
Stripe
Payment Service

Payment

Customer (Actor)

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Customer account takeover (credential stuffing/phishing)	Spoofing	High	Open		The Customer interacts with the Customer Client within the public Customer / Internet zone, and step (1) indicates login to the merchant site. If strong authentication is not enforced, an attacker can spoof the Customer using stolen or phished credentials, leading to fraudulent payments that propagate through downstream boundary-crossing flows.	Enforce MFA/WebAuthn for customer login, implement credential stuffing defenses (rate limiting, IP/device reputation), monitor for anomalous logins, and support phishing-resistant flows (FIDO2, risk-based authentication).

Customer Client (Process)

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Client-side code and DOM tampering affecting payment confirmation	Tampering	High	Open		Customer Client resides inside the Customer / Internet boundary and exchanges data across trust boundaries with Merchant Web Server and Stripe API service (flows (3)/(6), (8), (11)). Malicious scripts (XSS/3rd-party compromise) or man-in-the-browser can tamper with payment details or redirect e.ConfirmCardPayment() calls, altering amounts or recipients before they cross boundaries.	Harden the client: enforce strict CSP, Subresource Integrity for third-party libraries (e.g., stripe.js), avoid inline scripts, sanitize/encode all untrusted data, use integrity-checked dependencies, and validate critical values server-side (never trust client-calculated amounts).
	Leakage of sensitive card data from the client	Information Disclosure	High	Open		Step (7) shows the Customer provides card details through the Customer Client in the Customer / Internet zone while also communicating with external zones (Merchant / Web and Stripe / Web). Without strict controls, PAN or tokens could leak via logs, browser storage, referrers, or mixed-content requests before crossing trust boundaries.	Use Stripe Elements/tokenization so raw PAN never touches the merchant origin, disable logging of sensitive fields, prevent storage of card data (no localStorage/sessionStorage), set Referrer-Policy to no-referrer, enforce HTTPS with HSTS, and mask or avoid rendering full PAN in the UI.

(1) Customer logs into the merchant site (Data Flow)

Description: OAuth

Number	Title	Type	Severity	Status	Score	Description	Mitigations

(2) Customer proceeds to payment page to make a purchase (Data Flow)

Number	Title	Type	Severity	Status	Score	Description	Mitigations

(7) Customer provides card details and finalizes payment (Data Flow)

Number	Title	Type	Severity	Status	Score	Description	Mitigations

(3) Customer Client sends order intent including order amount (6) Return PaymentIntent to the Customer Client (Data Flow)

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Client↔Merchant data exposure over boundary-crossing channel	Information Disclosure	High	Open		The flow “(3) Customer Client sends order intent including order amount (6) Return PaymentIntent to the Customer Client” crosses from Customer / Internet (Customer Client) to Merchant / Web (Merchant Web Server) and back. The model sets isEncrypted=false and protocol is unspecified, implying potential Internet traversal without encryption, exposing order and PaymentIntent data.	Enforce TLS 1.2+ with HSTS and modern ciphers end-to-end; set Secure/SameSite cookies; disable mixed content; and ensure no sensitive data is sent over unencrypted channels.
	Order/PaymentIntent manipulation in transit	Tampering	High	Open		Because the bidirectional flow between Customer Client and Merchant Web Server crosses trust boundaries with isEncrypted=false, a network attacker could alter the order amount or tamper with PaymentIntent details returned to the client.	Require HTTPS for all requests, recompute and validate order totals server-side, bind PaymentIntent to server-derived values, and verify PaymentIntent status on the server after client actions (do not trust client data).

(9) Attempt payment (Data Flow)

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Internal call exposes payment attempt details	Information Disclosure	Medium	Open		The flow “(9) Attempt payment” runs within Stripe / Web from Stripe API service to Stripe Payment Service. With isEncrypted=false and protocol unspecified, sensitive payment metadata could be observed by an internal adversary.	Encrypt all service-to-service traffic via mTLS/service mesh; restrict network paths and enforce per-service authorization.
	Manipulation of internal payment attempt	Tampering	Medium	Open		Lack of integrity on the intrazone flow enables alteration of payment parameters or results before they reach Stripe Payment Service.	Use mTLS with strong identities, sign requests (JWT with audience/exp), and validate message integrity and freshness at the receiver.

(10) Payment Response (Data Flow)

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Exposure of internal payment results	Information Disclosure	Medium	Open		The flow “(10) Payment Response” runs from Stripe Payment Service to Stripe API service within Stripe / Web. With isEncrypted=false, internal observers could access sensitive payment outcomes.	Mandate mTLS/encrypted service mesh for all internal service communications; minimize response contents.
	Tampering with payment results upstream	Tampering	Medium	Open		Unprotected intrazone traffic allows modification of the payment response before it returns to Stripe API service, potentially causing inconsistent or fraudulent states.	Apply end-to-end integrity (mTLS + signed responses), replay protection, and strict authorization between services.

(11) Return the PaymentIntent with status (Data Flow)

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	PaymentIntent status disclosure to client over Internet	Information Disclosure	High	Open		The flow “(11) Return the PaymentIntent with status” crosses from Stripe / Web (Stripe API service) to Customer / Internet (Customer Client) with isEncrypted=false and no protocol defined. This boundary-crossing response can reveal sensitive payment metadata over the Internet.	Serve over HTTPS with HSTS; minimize status payloads; avoid exposing sensitive fields; apply strict CORS for browser delivery and set cache controls.
	Client response manipulation leading to false states	Tampering	High	Open		Without transport integrity on the cross-boundary response, an attacker could alter the PaymentIntent status received by the Customer Client (e.g., showing success when failed), driving user actions or fraud.	Enforce TLS 1.2+; always confirm final status on the Merchant Web Server via server-to-server calls to Stripe; treat client-visible status as informational only.

(8) Customer Client send Stripe e.ConfirmCardPayment() (Data Flow)

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Exposure of card confirmation data from client to Stripe	Information Disclosure	High	Open		The flow “(8) Customer Client send Stripe e.ConfirmCardPayment()” crosses from Customer / Internet (Customer Client) to Stripe / Web (Stripe API service). With isEncrypted=false and protocol unspecified, sensitive payment confirmation details may traverse the Internet without confidentiality.	Use HTTPS with HSTS and modern TLS; ensure stripe.js is loaded with Subresource Integrity; for mobile apps use certificate pinning; never include PAN in custom requests (use Stripe Elements/tokenization).
	Manipulation of confirmCardPayment request/response	Tampering	High	Open		Without transport integrity on the boundary-crossing client→Stripe flow, an attacker could alter request parameters or the response, leading to false success/failure states on the client.	Enforce TLS 1.2+; validate all critical outcomes on the merchant server by retrieving PaymentIntent status directly from Stripe; ignore client-reported results for financial decisions.

(5) Return PaymentIntent to the Merchant (Data Flow)

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Stripe→Merchant response exposure	Information Disclosure	High	Open		The flow “(5) Return PaymentIntent to the Merchant” crosses from Stripe / Web (Stripe API service) to Merchant / Web (Merchant Web Server). With isEncrypted=false and no protocol, PaymentIntent data may be exposed over an Internet path between trust boundaries.	Use TLS 1.2+ with PFS and hostname verification; prefer mTLS between servers; restrict cipher suites; and apply IP allowlisting where possible.
	Acceptance of spoofed Stripe responses	Spoofing	High	Open		Without mutual authentication on the boundary-crossing Stripe API service→Merchant Web Server channel, an attacker could masquerade as Stripe and send forged PaymentIntent responses.	Implement mTLS (client/server certs), verify certificate chains and hostnames, and require signed responses/webhooks (HTTP signatures or shared-secret verification) before updating order state.

(4) Merchant sends order information inc amount and currency (Data Flow)

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Order information tampering en route to Stripe	Tampering	High	Open		The flow “(4) Merchant sends order information inc amount and currency” crosses from Merchant / Web (Merchant Web Server) to Stripe / Web (Stripe API service). The model shows isEncrypted=false and no protocol, allowing modification of amount/currency in transit.	Use TLS 1.2+ with strong ciphers and mTLS for server-to-server calls; include integrity protections (idempotency keys, request signing) and validate amounts against server-side price data at Stripe.
	Merchant identity spoofing to Stripe	Spoofing	High	Open		If the Merchant Web Server authenticates weakly to Stripe API service on this cross-boundary flow (no mTLS/protocol specified), attackers with leaked keys or on-path access can impersonate the merchant when creating PaymentIntents.	Require mTLS with per-merchant client certificates, scope and rotate API keys, or use OAuth client credentials with JWTs; monitor for anomalous usage and enforce IP allowlisting where feasible.

Merchant Web Server (Process)

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Trusting client-supplied order amounts	Tampering	High	Open		Merchant Web Server in the Merchant / Web boundary receives order intent including amount from Customer Client across a trust boundary via flow (3). An attacker can tamper with client-supplied values (amount/currency) before they reach the server.	Recompute price/amount server-side from authoritative product and pricing data; sign and verify cart contents; ignore client-supplied totals; validate currency and amount against server-side business rules; bind PaymentIntent to server-derived values only.
	Insufficient payment lifecycle auditability	Repudiation	Medium	Open		Merchant Web Server participates in creating and handling PaymentIntent data via boundary-crossing flows (3)/(5)/(6)/(11). Without comprehensive, tamper-evident logs correlating customer identity, request IDs, idempotency keys, and responses, disputes and repudiation are difficult to resolve.	Implement append-only, time-synchronized audit logging for all payment events with request/response IDs, PaymentIntent IDs, user/session identifiers, and idempotency keys; protect logs with integrity controls and restricted access.

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Exposure of API credentials and payment data in server logs/configs	Information Disclosure	Medium	Open		Merchant Web Server communicates with Stripe API service across trust boundaries (flows (4) and (5)). Mismanaged secrets or verbose logging can leak API keys, tokens, or PaymentIntent details that could be used by attackers to impersonate the merchant or harvest data.	Use centralized secrets management (HSM/KMS), short-lived scoped API keys, strict environment separation, and log redaction; enforce least privilege for config access and rotate credentials regularly.
	Internet-facing payment endpoints susceptible to volumetric and application DoS	Denial of Service	Medium	Open		Merchant Web Server receives ingress from the Customer / Internet zone via flow (3). Attackers can overwhelm login/checkout/payment endpoints to disrupt availability.	Apply WAF/CDN protection, per-IP/user rate limiting, request queuing with circuit breakers, autoscaling with backpressure, and CAPTCHA/challenge for abuse patterns.

Stripe API service (Process)

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Merchant impersonation to Stripe API	Spoofing	High	Open		Stripe API service in the Stripe / Web boundary accepts requests from Merchant Web Server across a trust boundary (flow (4)). With protocol unspecified and isEncrypted=false in the model, weak client authentication (e.g., leaked API keys) could allow an attacker to masquerade as a merchant.	Require strong merchant authentication (mTLS with per-merchant client certs, scoped/rotated API keys or OAuth client credentials), enforce hostname verification, and monitor for anomalous API usage.
	Overexposed payment data in responses to clients/merchants	Information Disclosure	Medium	Open		Stripe API service returns PaymentIntent details across boundaries to Merchant Web Server (flow (5)) and Customer Client (flow (11)). If responses include excessive PII or secrets, exposure risk increases over Internet paths.	Minimize response fields, avoid returning secrets, apply data classification, set appropriate Cache-Control headers, and enforce strict CORS for browser clients.
	API saturation and abuse	Denial of Service	Medium	Open		Being a public-facing service inside Stripe / Web with multiple ingress/egress flows ((4), (5), (8), (9), (10), (11)), Stripe API service can be overwhelmed by bursty or malicious traffic.	Implement global rate limiting/quotas per merchant, token bucket throttling, autoscaling with circuit breakers, retries with backoff, and request validation early in the pipeline.

Stripe Payment Service (Process)

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Unauthenticated/unencrypted internal service calls	Tampering	Medium	Open		Stripe Payment Service communicates with Stripe API service within the Stripe / Web boundary via flows (9) and (10). The model shows isEncrypted=false and protocol unspecified, allowing internal attackers to alter payment requests/responses in transit.	Use a service mesh or mTLS for all intra-zone traffic with strict identity (SPIFFE/SVID), sign requests (JWT with audience/expiry), and verify integrity end-to-end.
	Internal payment data exposure	Information Disclosure	Medium	Open		Sensitive payment details may traverse flows (9) and (10) between Stripe Payment Service and Stripe API service without encryption as per the model, risking interception within the environment.	Encrypt service-to-service traffic (TLS 1.2+ with PFS), minimize payload contents, and restrict lateral movement with network segmentation and per-service authorization.
	Payment processing exhaustion	Denial of Service	Medium	Open		Flow (9) Attempt payment can be abused to generate costly downstream operations (network calls/3DS/acquirer interactions), exhausting resources of Stripe Payment Service.	Apply idempotency keys, per-PaymentIntent rate limits, circuit breakers/timeouts to downstreams, and priority queues with backpressure.