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

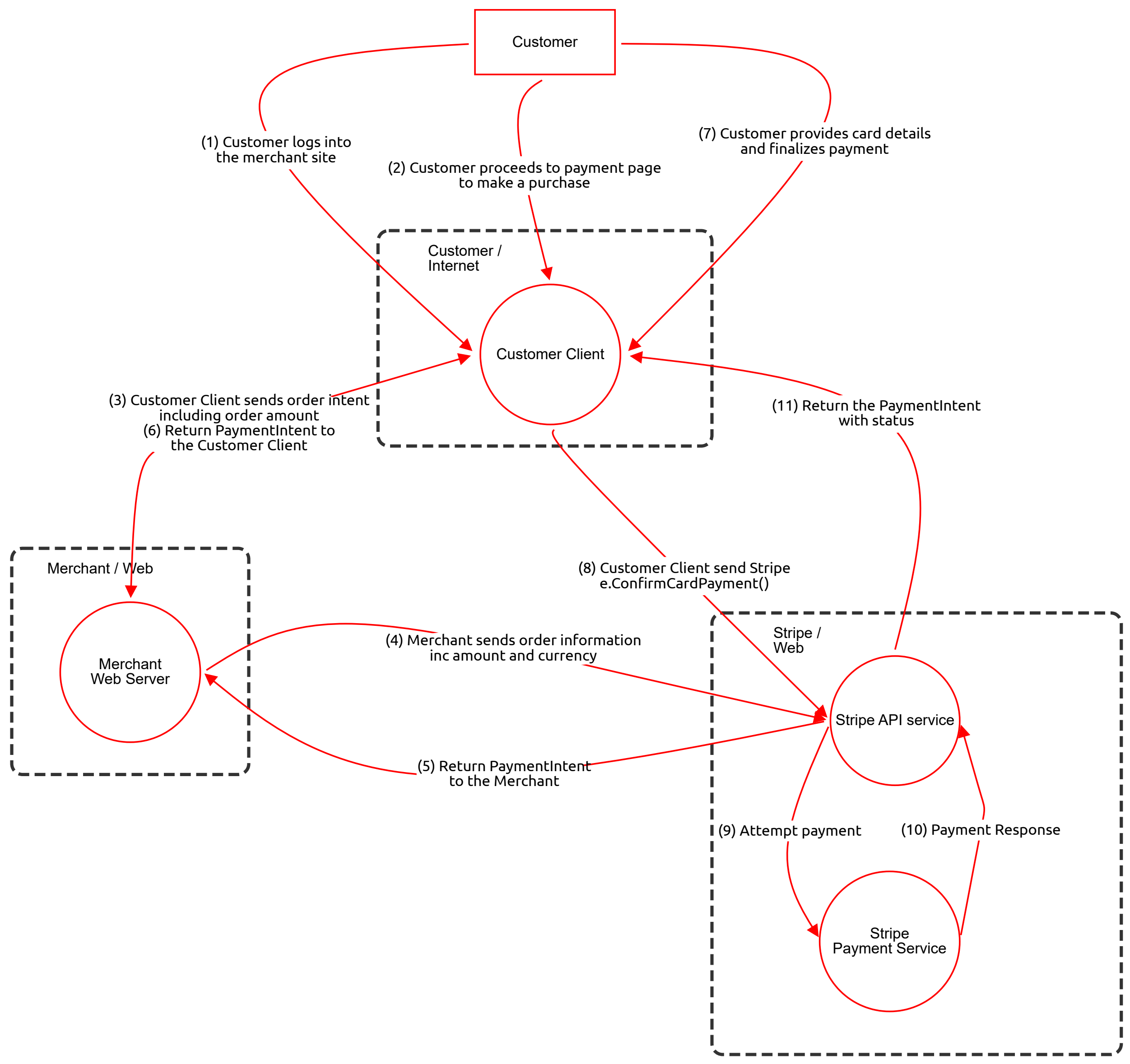
This threat model has been provided by the OWASP Threat Model Cookbook:
threat-model-cookbook/Flow Diagram/payment

Summary

Total Threats	42
Total Mitigated	0
Total Open	42
Open / Critical Severity	0
Open / High Severity	33
Open / Medium Severity	9
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
	Spoofing of Customer Identity	Spoofing	High	Open		The Customer actor initiates authentication flows with the Merchant Web Server. Without strong authentication mechanisms, an attacker could impersonate a legitimate customer to gain unauthorized access to accounts and perform fraudulent transactions. This actor resides outside all trust boundaries and connects across the public internet to the Customer Client within the Customer/Internet zone.	- Implement multi-factor authentication (MFA) for all customer logins - Use OAuth 2.0 with PKCE for secure authentication flows - Enforce strong password policies and account lockout mechanisms - Implement device fingerprinting and behavioral analytics
	Repudiation of Payment Actions	Repudiation	Medium	Open		The Customer actor performs payment operations through multiple flows (steps 1, 2, 7). Without proper logging and non-repudiation controls, customers could deny initiating payment transactions, leading to disputes and potential fraud. The customer interacts across trust boundaries from an external position to internal payment processing systems.	- Implement comprehensive audit logging for all customer actions with timestamps and IP addresses - Use digital signatures for payment confirmations - Store transaction receipts with cryptographic proofs - Implement session recording for critical payment flows - Maintain immutable audit trails in a separate logging system

Customer Client (Process)

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Tampering with Payment Intent Data	Tampering	High	Open		The Customer Client process handles sensitive payment data including order amounts and payment intents (flows 3, 6, 8). If this client-side process is compromised or manipulated, attackers could tamper with payment amounts, currency values, or payment confirmation data before submission to backend systems. This process resides within the Customer/Internet boundary and communicates with both Merchant and Stripe systems across trust boundaries.	- Implement client-side integrity checks and code signing - Use Content Security Policy (CSP) headers to prevent XSS attacks - Validate all payment data server-side and never trust client-supplied amounts - Implement subresource integrity (SRI) for all external JavaScript libraries - Use anti-tampering and obfuscation techniques for client code
	Information Disclosure of Payment Credentials	Information Disclosure	High	Open		The Customer Client receives and handles sensitive card details from the customer (flow 7) and communicates with Stripe API (flow 8). This data flows through a client-side process in the Customer/Internet zone. If the client is compromised through XSS, malware, or man-in-the-browser attacks, card details and payment tokens could be intercepted and exfiltrated.	- Use Stripe Elements or hosted payment forms to avoid handling raw card data in client code - Implement strong CSP to prevent unauthorized script execution - Ensure all communications use TLS 1.3 with certificate pinning - Use PCI DSS compliant tokenization for card data - Implement runtime application self-protection (RASP) mechanisms

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Denial of Service Against Customer Client	Denial of Service	Medium	Open		The Customer Client process in the Customer/Internet zone handles multiple critical flows (1, 2, 3, 6, 7, 8, 11). An attacker could overwhelm this client-side process with malicious requests, resource exhaustion attacks, or exploit client-side vulnerabilities to crash the application, preventing legitimate customers from completing payments.	- Implement client-side rate limiting and request throttling - Use service workers for resilient client-side operations - Implement graceful degradation and error handling - Use CDN with DDoS protection for client asset delivery - Implement client-side request queuing and retry logic with exponential backoff
	Elevation of Privilege via Client Manipulation	Elevation of Privilege	High	Open		The Customer Client operates within the Customer/Internet boundary with limited privileges. If an attacker can manipulate client-side code or exploit vulnerabilities, they could attempt to elevate privileges by modifying API requests, bypassing client-side authorization checks, or manipulating session tokens to access unauthorized payment operations or other customer accounts.	- Implement robust server-side authorization checks for all operations - Never rely on client-side security controls for access decisions - Use short-lived, signed JWT tokens with proper claims validation - Implement principle of least privilege for API access - Use Content Security Policy and Subresource Integrity - Monitor for anomalous API usage patterns

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

Description: OAuth

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Spoofing of Authentication Endpoint	Spoofing	High	Open		Flow (1) 'Customer logs into the merchant site' uses OAuth over HTTPS from the Customer actor to the Customer Client. This flow crosses from outside all boundaries into the Customer/Internet zone. An attacker could create a phishing site spoofing the merchant's login page to capture customer credentials. Without proper endpoint verification, customers may be tricked into authenticating with a malicious service.	- Implement Extended Validation (EV) SSL certificates with visible indicators - Use HSTS (HTTP Strict Transport Security) with preloading - Implement certificate pinning for critical authentication endpoints - Educate users about verifying domain names and SSL indicators - Use OAuth with proper redirect URI validation - Implement anti-phishing measures like password managers that verify domains
	Information Disclosure via Credential Interception	Information Disclosure	High	Open		Flow (1) carries OAuth authentication credentials over HTTPS from the external Customer actor to the Customer Client. Despite HTTPS protocol, if TLS is misconfigured, uses weak ciphers, or is subject to man-in-the-middle attacks, credentials could be intercepted. This is an ingress flow crossing into a more trusted zone from an untrusted external position.	- Enforce TLS 1.3 with strong cipher suites only - Implement certificate transparency monitoring - Use mutual TLS (mTLS) for enhanced authentication - Implement certificate pinning to prevent MITM attacks - Monitor for SSL/TLS downgrade attacks - Use OAuth 2.0 with PKCE to protect authorization codes

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

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Tampering with Order Request Data	Tampering	High	Open		Flow (2) 'Customer proceeds to payment page to make a purchase' carries order information over HTTPS from the Customer actor to the Customer Client. An attacker with MITM capabilities or client-side access could tamper with the order details, product selections, quantities, or pricing information before the purchase is processed. This flow enters the Customer/Internet boundary from an external untrusted source.	- Implement end-to-end encryption for order data - Use digital signatures to verify order integrity - Validate all order data server-side against product catalog - Implement session binding to prevent session hijacking - Use HMAC or similar mechanisms to protect order parameters - Log all order modifications with timestamps and user context
	Information Disclosure of Shopping Cart Data	Information Disclosure	Medium	Open		Flow (2) transfers customer shopping cart and purchase intent data from the external Customer actor to the Customer Client over HTTPS. If SSL/TLS is compromised or traffic is intercepted, sensitive shopping behavior, product preferences, and personal information could be exposed. This represents an ingress flow crossing into the application zone.	- Enforce TLS 1.3 with perfect forward secrecy - Implement certificate pinning for the payment domain - Use encrypted cookies for session management - Implement Content Security Policy to prevent data exfiltration - Monitor for unusual traffic patterns or SSL anomalies - Minimize PII in transit; use tokenization where possible

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

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Information Disclosure of Card Details in Transit	Information Disclosure	High	Open		Flow (7) 'Customer provides card details and finalizes payment' transmits highly sensitive payment card information from the external Customer actor directly to the Customer Client within the Customer/Internet zone. While the protocol should be HTTPS, the flow metadata shows an empty protocol field, raising concerns. Any interception or logging of this data would expose full card details including PAN, CVV, and expiry dates.	- Mandate TLS 1.3 with strong cipher suites for this flow - Implement certificate pinning to prevent MITM attacks - Use Stripe Elements or hosted payment forms to avoid handling raw card data - Ensure PCI DSS SAQ A-EP or SAQ A compliance - Never log or store card details client-side - Implement client-side encryption before transmission - Use tokenization immediately upon card data entry

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Tampering with Payment Card Data	Tampering	High	Open		Flow (7) carries payment card information from the external Customer to the Customer Client. An attacker with client-side access or MITM capabilities could tamper with card details, potentially substituting their own card information for fraudulent purposes or corrupting the data to cause payment failures and service disruption. This is a critical ingress flow of highly sensitive financial data.	- Use Stripe hosted payment forms to minimize client-side card handling - Implement strong client-side input validation and sanitization - Use Content Security Policy to prevent script injection - Implement integrity checks using HMAC or digital signatures - Validate card data format and checksum (Luhn algorithm) immediately - Use secure, tamper-resistant payment input components - Implement fraud detection for unusual card data patterns
	Repudiation of Payment Card Submission	Repudiation	Medium	Open		Flow (7) represents the customer submitting card details for payment. Without proper audit logging and non-repudiation controls, customers could claim they never provided card information or authorized the payment, leading to chargebacks and disputes. This flow crosses from an external untrusted actor into the payment processing chain.	- Log all payment submission events with timestamps, IP addresses, and device fingerprints - Implement 3D Secure authentication for additional proof of cardholder authorization - Use digital receipts sent via email immediately upon submission - Store payment confirmation acknowledgments with cryptographic signatures - Implement session recording for payment flows (without capturing sensitive data) - Use strong customer authentication (SCA) as required by PSD2

(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
	Tampering with Order Amount During Transit	Tampering	High	Open		Bidirectional flow (3, 6) 'Customer Client sends order intent including order amount / Return PaymentIntent to the Customer Client' crosses from the Customer/Internet zone to the Merchant/Web zone. The flow carries critical payment amount data. An attacker with MITM capabilities could intercept and modify the order amount, currency, or PaymentIntent parameters in either direction, potentially causing financial fraud or incorrect charges.	- Implement mutual TLS (mTLS) authentication between Customer Client and Merchant Web Server - Use message-level encryption with HMAC for order data integrity - Sign all payment intents with server-side keys - Validate order amounts against product catalog server-side - Implement rate limiting and anomaly detection for unusual amounts - Use request/response correlation IDs to detect replay attacks - Never trust client-supplied amounts; always recalculate server-side

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Information Disclosure of PaymentIntent Tokens	Information Disclosure	High	Open		Bidirectional flow (3, 6) transmits PaymentIntent objects containing sensitive payment session information between the Customer Client in the Customer/Internet zone and Merchant Web Server in the Merchant/Web zone across trust boundaries. If intercepted, these tokens could be used to gain insights into payment sessions, amounts, and potentially replay or manipulate payments.	- Enforce TLS 1.3 end-to-end with perfect forward secrecy - Use short-lived, single-use PaymentIntent tokens - Implement token binding to prevent token theft and reuse - Encrypt sensitive payload fields within the PaymentIntent - Bind PaymentIntents to specific client sessions and IP addresses - Implement comprehensive logging and monitoring for PaymentIntent usage - Use mutual authentication to verify both endpoints
	Spoofing of Merchant Web Server	Spoofing	High	Open		Flow (3, 6) represents communication between Customer Client and Merchant Web Server across different trust zones. An attacker could spoof the Merchant Web Server endpoint, intercepting PaymentIntent requests from the Customer Client and returning malicious or fraudulent PaymentIntent objects to capture payment information or redirect funds.	- Implement certificate pinning for the merchant domain - Use Extended Validation (EV) SSL certificates - Implement DNS Security Extensions (DNSSEC) - Use mutual TLS authentication - Implement strict Transport Security (HSTS) with preloading - Verify server certificates against certificate transparency logs - Use OAuth or API keys for additional endpoint verification

(9) Attempt payment (Data Flow)

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Tampering with Payment Request to Stripe	Tampering	High	Open		Flow (9) 'Attempt payment' transmits the actual payment charge request from Stripe API service to Stripe Payment Service within the Stripe/Web zone. While this is internal to Stripe's infrastructure, if an attacker compromised the API service layer or the communication channel between services, they could tamper with payment amounts, payment methods, or destination accounts before the payment is processed.	- Use mutual TLS authentication between Stripe internal services - Implement message-level encryption and HMAC for payment requests - Use service mesh with end-to-end encryption for microservices communication - Implement cryptographic signing of payment requests - Use request correlation and integrity checks - Monitor for anomalous payment patterns or amounts - Implement service-to-service authentication using certificates or tokens - Use immutable infrastructure and regular integrity checks

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Information Disclosure of Payment Processing Data	Information Disclosure	High	Open		Flow (9) carries complete payment details including card information, amounts, and merchant identifiers from Stripe API service to Stripe Payment Service. Even within the Stripe/Web trust zone, if network segmentation is inadequate or logging is excessive, sensitive payment data could be exposed to unauthorized internal systems or through compromised internal services.	- Implement network segmentation and microsegmentation within Stripe infrastructure - Use encrypted communication channels for all inter-service communication - Implement data classification and handling procedures - Minimize logging of sensitive payment data; use tokenization - Implement strict access controls for internal services - Use data loss prevention (DLP) tools to monitor for data exfiltration - Encrypt payment data at rest and in transit internally - Implement least privilege access for service accounts

(10) Payment Response (Data Flow)

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Tampering with Payment Response Data	Tampering	High	Open		Flow (10) 'Payment Response' returns the payment result from Stripe Payment Service to Stripe API service within the Stripe/Web zone. If this internal communication is compromised, an attacker could tamper with payment status, converting failed payments to successful or vice versa, leading to financial discrepancies, unauthorized access to goods/services, or fraudulent chargebacks.	- Use mutual TLS authentication for internal service communication - Implement cryptographic signing of payment responses - Use message authentication codes (MAC) for response integrity - Implement response validation and reconciliation checks - Use immutable audit logging for all payment status changes - Implement anomaly detection for inconsistent payment outcomes - Use service mesh with end-to-end encryption - Implement correlation IDs to track request-response pairs
	Spoofing of Payment Service Response	Spoofing	High	Open		Flow (10) represents the payment result being returned from Stripe Payment Service to Stripe API service. An attacker who has compromised the internal network or a service could spoof the Payment Service, returning fraudulent success responses for failed payments or failed responses for successful payments, causing financial fraud or service disruption.	- Implement strong service-to-service authentication using mutual TLS - Use service identity verification with certificates - Implement response signature verification - Use service mesh with identity-based routing and authentication - Implement health checks and service validation - Monitor for rogue services or unexpected endpoints - Use certificate pinning for internal service communication - Implement zero-trust network architecture with service verification

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

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Information Disclosure of Payment Status	Information Disclosure	Medium	Open		Flow (11) 'Return the PaymentIntent with status' transmits payment outcome information from Stripe API service in the Stripe/Web zone back to Customer Client in the Customer/Internet zone, crossing trust boundaries. This response includes payment status, which could reveal information about card validity, account balances, or fraud detection results. If intercepted, this information could be used for reconnaissance or social engineering attacks.	- Enforce TLS 1.3 with perfect forward secrecy for all responses - Implement certificate pinning for Stripe API communications - Minimize information in error messages; use generic failure messages - Implement rate limiting on payment attempts to prevent enumeration - Use standardized, non-revealing error codes - Implement monitoring for unusual patterns of failed payment attempts - Don't reveal specific reasons for payment decline in client responses - Use secure session management to protect status information
	Repudiation of Payment Attempt	Repudiation	Medium	Open		Flow (11) returns the payment result to the Customer Client. Without proper logging and audit trails on both client and server sides, customers could dispute whether they attempted a payment, whether it was successful, or claim they never received confirmation of payment status, leading to disputes and chargebacks. This crosses from the Stripe/Web zone back to the Customer/Internet zone.	- Implement comprehensive logging of all payment status responses with timestamps - Send immediate email receipts for all payment attempts (success and failure) - Store payment confirmation logs in immutable audit storage - Use Stripe's webhook events as secondary verification of payment status - Implement non-repudiation through cryptographic receipts - Use digital signatures for payment confirmation messages - Implement session recording (without capturing sensitive data) for payment flows - Maintain correlation between payment attempts and confirmations

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

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Tampering with Payment Confirmation Request	Tampering	High	Open		Flow (8) 'Customer Client send Stripe e.ConfirmCardPayment()' transmits the final payment confirmation from Customer Client in the Customer/Internet zone to Stripe API service in the Stripe/Web zone, crossing trust boundaries. An attacker with MITM capabilities could tamper with the payment confirmation parameters, payment method details, or amounts before reaching Stripe, potentially causing failed payments, incorrect charges, or fraud.	- Enforce TLS 1.3 with certificate pinning for all Stripe API communications - Use Stripe's official SDKs which implement integrity checks - Implement client-side signature generation for payment confirmation requests - Validate payment parameters server-side before client confirmation - Use PaymentIntent IDs that are bound to specific sessions and amounts - Implement request replay protection with nonces or timestamps - Monitor for unusual payment confirmation patterns
	Information Disclosure of Payment Method Details	Information Disclosure	High	Open		Flow (8) carries sensitive payment method information and card details from the Customer Client to Stripe API service across trust boundaries. If TLS is compromised or the client-side is attacked with keyloggers or memory scraping malware, payment method details could be intercepted and stolen. This is a critical egress flow of sensitive data from the customer zone to an external payment processor zone.	- Use Stripe Elements which isolate card input in secure iframes - Implement certificate pinning to prevent MITM attacks - Use tokenization to avoid transmitting raw card data - Ensure TLS 1.3 with strong cipher suites - Implement client-side encryption before transmission - Use Stripe's Payment Request API which handles sensitive data securely - Never log or cache payment method details client-side - Implement memory protection against scraping attacks
	Spoofing of Stripe API Endpoint	Spoofing	High	Open		Flow (8) connects from Customer Client to Stripe API service across trust boundaries. An attacker could spoof the Stripe API endpoint through DNS poisoning, compromised hosts file, or MITM attacks, capturing payment confirmations and card details submitted by customers. The Customer Client needs to verify it's communicating with the legitimate Stripe service.	- Implement certificate pinning for api.stripe.com - Use Stripe's official SDKs which have built-in endpoint verification - Implement DNS Security Extensions (DNSSEC) - Validate SSL certificates against certificate transparency logs - Use HSTS with preload for Stripe domains - Implement public key pinning (HPKP) backup - Monitor for DNS resolution anomalies - Use explicit domain allowlisting in Content Security Policy

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

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Tampering with PaymentIntent During Return	Tampering	High	Open		Flow (5) 'Return PaymentIntent to the Merchant' transmits the newly created PaymentIntent from Stripe API service in the Stripe/Web zone to Merchant Web Server in the Merchant/Web zone, crossing trust boundaries. An attacker with MITM capabilities or network access could tamper with the PaymentIntent object, modifying the amount, currency, client secret, or other parameters before it reaches the merchant server, causing payment processing errors or fraud.	- Enforce mutual TLS authentication between Stripe and Merchant servers - Use certificate pinning for Stripe API communications - Implement cryptographic signatures for PaymentIntent objects - Validate PaymentIntent amounts and parameters immediately upon receipt - Compare returned PaymentIntent against original request parameters server-side - Use HMAC to verify integrity of PaymentIntent data - Implement correlation IDs to track request-response pairs - Monitor for PaymentIntent parameter mismatches
	Information Disclosure of PaymentIntent Client Secret	Information Disclosure	High	Open		Flow (5) returns PaymentIntent objects containing client secrets that will be used to complete payments. These secrets are sensitive tokens that, if intercepted during transmission from Stripe API to Merchant Web Server across trust zones, could be used by attackers to complete payments fraudulently or gain unauthorized access to payment sessions.	- Enforce TLS 1.3 for all Stripe API communications - Implement certificate pinning for api.stripe.com - Never log PaymentIntent client secrets - Use short-lived client secrets with Stripe's default expiration - Implement secure session storage for client secrets server-side - Transmit client secrets to Customer Client over secure channels only - Implement monitoring for unusual client secret usage patterns - Use IP allowlisting for Stripe API access where feasible

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

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Tampering with Order Amount Before Stripe Processing	Tampering	High	Open		Flow (4) 'Merchant sends order information inc amount and currency' transmits payment details from Merchant Web Server in the Merchant/Web zone to Stripe API service in the Stripe/Web zone, crossing trust boundaries. An attacker with access to the merchant server or network could tamper with order amounts, currency, or other parameters before the PaymentIntent is created at Stripe, resulting in incorrect charges or financial fraud.	- Implement order amount validation against product catalog before sending to Stripe - Use cryptographic signatures for order data sent to Stripe - Implement request signing using HMAC with shared secrets - Enforce mutual TLS authentication for merchant-to-Stripe communication - Implement audit logging for all order amounts sent to Stripe - Use idempotency keys to prevent duplicate or tampered requests - Implement real-time monitoring for unusual amount patterns - Validate currency codes against ISO 4217 standards
	Information Disclosure of Order Details	Information Disclosure	Medium	Open		Flow (4) carries order information including amounts, currency, product details, and potentially customer identifiers from the Merchant Web Server to Stripe API service across trust boundaries. If this communication is intercepted through compromised TLS or network attacks, sensitive business information and customer order details could be exposed, leading to competitive intelligence loss or privacy violations.	- Enforce TLS 1.3 with strong cipher suites for all Stripe API calls - Implement certificate pinning for api.stripe.com - Minimize PII in order metadata sent to Stripe - Use tokenization or pseudonymization for customer identifiers - Encrypt sensitive order metadata at application layer before transmission - Implement data classification and handling procedures - Monitor for unusual API access patterns - Use Stripe's metadata field encryption where appropriate
	Spoofing of Stripe API Endpoint by Merchant	Spoofing	High	Open		Flow (4) connects from Merchant Web Server to Stripe API service across trust zones. An attacker who has compromised the merchant server or DNS could redirect this flow to a spoofed Stripe endpoint, capturing order information, payment amounts, and API keys. The merchant needs assurance it's communicating with the legitimate Stripe service.	- Implement certificate pinning for api.stripe.com in merchant application - Use Stripe's official server-side SDKs which have built-in endpoint verification - Implement DNS Security Extensions (DNSSEC) - Validate SSL certificates against certificate transparency logs - Use explicit domain allowlisting in application configuration - Implement monitoring for unexpected SSL certificate changes - Use IP allowlisting to restrict outbound connections to known Stripe endpoints - Implement public key pinning for Stripe certificates

Merchant Web Server (Process)

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Elevation of Privilege via API Exploitation	Elevation of Privilege	High	Open		The Merchant Web Server process in the Merchant/Web zone handles authentication, order processing, and payment orchestration (flows 3, 4, 5, 6). If an attacker exploits vulnerabilities in the web server or APIs, they could elevate privileges to access administrative functions, manipulate orders, or access other customers' payment data. This process acts as a gateway between the Customer/Internet zone and the Stripe/Web zone.	- Implement role-based access control (RBAC) with principle of least privilege - Use separate service accounts with minimal permissions for each function - Implement API authentication using OAuth 2.0 with proper scope validation - Conduct regular security code reviews and penetration testing - Use Web Application Firewall (WAF) with OWASP Top 10 protection - Implement runtime application self-protection (RASP) - Enforce strong session management with short timeouts
	Tampering with Order Data at Server	Tampering	High	Open		The Merchant Web Server receives order intents from Customer Client (flow 3) and sends order information to Stripe API (flow 4). If the server is compromised or has vulnerabilities, attackers could tamper with order amounts, product details, or payment parameters before forwarding to Stripe, resulting in incorrect charges or fraudulent transactions. This process sits at a critical trust boundary junction between customer-facing and payment processing zones.	- Implement database-level integrity constraints and triggers - Use immutable audit logs for all order modifications - Implement digital signatures for order data - Use stored procedures with parameterized queries to prevent SQL injection - Implement file integrity monitoring (FIM) for application code - Use hardware security modules (HSM) for cryptographic operations - Implement real-time monitoring and alerting for order anomalies
	Denial of Service Against Merchant Web Server	Denial of Service	High	Open		The Merchant Web Server is a critical component handling multiple flows (3, 4, 5, 6) and serving as the intermediary between customers and payment processing. An attacker could launch DDoS attacks, resource exhaustion attacks, or exploit application-level vulnerabilities to overwhelm the server, preventing legitimate customers from making purchases and disrupting business operations.	- Implement rate limiting per IP address and per user session - Deploy Web Application Firewall (WAF) with DDoS protection - Use CDN services with DDoS mitigation capabilities - Implement connection throttling and request queuing - Use auto-scaling infrastructure to handle traffic spikes - Implement circuit breakers for downstream service calls - Monitor for unusual traffic patterns and implement automated blocking - Use CAPTCHA for suspected bot traffic

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Repudiation of Merchant Actions	Repudiation	Medium	Open		The Merchant Web Server performs critical operations including order processing, PaymentIntent creation with Stripe, and transaction management. Without comprehensive audit logging, merchant administrators or compromised accounts could perform fraudulent actions (price modifications, unauthorized refunds) and deny responsibility, leading to financial losses and compliance issues.	- Implement comprehensive audit logging for all merchant actions with timestamps, user IDs, and IP addresses - Use write-once, append-only logging to prevent log tampering - Store logs in a separate, secured logging infrastructure - Implement log aggregation and SIEM integration - Use digital signatures for critical transaction logs - Implement role-based logging to track administrative actions - Regularly review and analyze audit logs for suspicious patterns
	Information Disclosure via Server Vulnerabilities	Information Disclosure	High	Open		The Merchant Web Server stores or processes sensitive customer information, order details, and payment session data. If the server has vulnerabilities (e.g., SQL injection, path traversal, insecure direct object references), attackers could extract sensitive information including customer PII, order history, payment tokens, and API keys. This server bridges multiple trust zones and has access to data from both customer and payment processor zones.	- Implement secure coding practices following OWASP guidelines - Use parameterized queries and ORM frameworks to prevent SQL injection - Implement proper input validation and output encoding - Use encryption at rest for sensitive data storage - Implement proper access controls and authentication for all APIs - Conduct regular vulnerability scanning and penetration testing - Implement Data Loss Prevention (DLP) controls - Use secrets management solutions for API keys and credentials

Stripe API service (Process)

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Elevation of Privilege via API Key Compromise	Elevation of Privilege	High	Open		The Stripe API service process in the Stripe/Web zone handles PaymentIntent creation, payment confirmations, and payment processing (flows 2, 3, 4, 5, 8, 9, 11). This service requires API keys for authentication. If these keys are compromised through exposure in client-side code, logs, or repository leaks, attackers could gain elevated access to perform unauthorized payment operations, refunds, or access customer payment data.	- Store API keys securely using secrets management services (e.g., AWS Secrets Manager, HashiCorp Vault) - Use separate publishable and secret keys with appropriate scope restrictions - Implement API key rotation policies - Never expose secret keys in client-side code or repositories - Use environment variables for key storage, not hardcoded values - Implement IP allowlisting for API access where possible - Monitor API usage for unusual patterns indicating compromised keys - Use webhook signature verification to validate Stripe callbacks

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Denial of Service via API Rate Limit Exhaustion	Denial of Service	Medium	Open		The Stripe API service handles multiple critical flows (2, 3, 4, 5, 8, 9, 11) and is subject to rate limits. An attacker could intentionally exhaust API rate limits by generating excessive PaymentIntent creation requests or payment confirmations, preventing legitimate transactions from completing and disrupting business operations. This service is at the boundary of the Stripe/Web zone and critical for payment processing.	- Implement client-side rate limiting before calling Stripe APIs - Use request queuing and retry logic with exponential backoff - Implement CAPTCHA for user-initiated payment flows to prevent bot attacks - Monitor API usage and implement alerting for approaching rate limits - Use Stripe's idempotency keys to prevent duplicate requests - Implement request deduplication at application layer - Consider upgrading to higher Stripe API rate limits for high-volume scenarios - Implement circuit breakers to fail fast when rate limits are hit
	Tampering with PaymentIntent Objects	Tampering	High	Open		The Stripe API service creates and returns PaymentIntent objects (flows 4, 5, 6) that contain payment amount, currency, and status information. If an attacker intercepts these objects in transit or compromises the merchant server, they could tamper with PaymentIntent parameters before they're used for payment confirmation, potentially causing incorrect charges or payment failures. This service sits at a critical trust boundary in the Stripe/Web zone.	- Validate PaymentIntent amounts server-side before and after creation - Use HMAC or digital signatures to protect PaymentIntent integrity - Implement server-side validation that amounts haven't changed between creation and confirmation - Use TLS 1.3 for all communications with Stripe - Store PaymentIntent IDs and amounts in secure server-side session storage - Implement monitoring for PaymentIntent amount mismatches - Use Stripe's built-in security features like payment method verification
	Information Disclosure via API Response Interception	Information Disclosure	High	Open		The Stripe API service returns PaymentIntent objects and payment responses (flows 5, 10, 11) containing sensitive information including payment status, amounts, payment method details, and customer identifiers. If API responses are intercepted through MITM attacks or logged insecurely, sensitive payment information could be exposed. This service communicates across trust boundaries back to both merchant and customer zones.	- Enforce TLS 1.3 for all Stripe API communications - Implement certificate pinning for api.stripe.com - Never log full API responses containing sensitive data - Redact sensitive fields before logging (payment methods, amounts) - Use Stripe's restricted API keys with minimal required permissions - Implement secure response handling that clears sensitive data from memory - Use ephemeral storage for PaymentIntent objects - Implement Content Security Policy to prevent response exfiltration

Stripe Payment Service (Process)

Number	Title	Type	Severity	Status	Score	Description	Mitigations
	Denial of Service Against Payment Processing	Denial of Service	High	Open		The Stripe Payment Service process in the Stripe/Web zone is responsible for executing actual payment charges (flows 9, 10). An attacker who gains access to internal Stripe systems or exploits vulnerabilities could overload this service with fraudulent payment requests, resource exhaustion attacks, or exploit processing bottlenecks, preventing legitimate payments from completing and causing widespread business disruption for all Stripe merchants.	- Implement rate limiting and throttling at the payment service layer - Use message queuing with backpressure mechanisms - Implement circuit breakers to prevent cascade failures - Use auto-scaling infrastructure for payment processing capacity - Implement payment request prioritization based on merchant tier - Use distributed processing and load balancing - Implement anomaly detection for unusual payment volumes - Use resource quotas and isolation between merchant payment streams
	Tampering with Payment Authorization	Tampering	High	Open		The Stripe Payment Service executes the final payment charge and communicates with card networks and banks. If this service is compromised or has vulnerabilities, attackers could tamper with authorization requests, settlement amounts, or merchant identifiers, causing incorrect charges, fund misdirection, or payment fraud. This is the most critical component in the payment processing chain within the Stripe/Web zone.	- Implement cryptographic signing of all payment authorization requests - Use hardware security modules (HSM) for sensitive cryptographic operations - Implement dual authorization for high-value transactions - Use immutable audit logging for all payment authorizations - Implement real-time fraud detection and authorization - Use tokenization for card data to minimize sensitive data exposure - Implement strict access controls and separation of duties - Use secure enclaves or trusted execution environments for payment processing
	Information Disclosure of Payment Network Credentials	Information Disclosure	High	Open		The Stripe Payment Service maintains credentials and connections to payment networks, acquiring banks, and card schemes. If this service is compromised or credentials are exposed through misconfiguration, logs, or memory dumps, attackers could gain access to payment network connections, enabling large-scale fraud, unauthorized transaction processing, or access to sensitive financial data across many merchants.	- Store payment network credentials in hardware security modules (HSM) - Use secrets management with automatic rotation - Implement strict access controls for credential access - Use separate credentials for each payment network with minimal permissions - Implement comprehensive audit logging for credential usage - Use encrypted memory and secure enclaves for credential storage - Implement anomaly detection for unusual credential usage patterns - Use certificate-based authentication where possible instead of static credentials