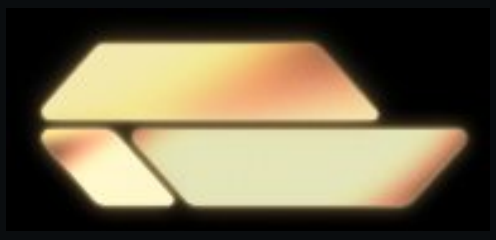
Manish Bhattacharya

https://manishbhattacharya.com/



Security Assessment Report

SECURITY REVIEW

Met Extension (met-extension-release)

February 2026

Prepared for MetEngine

From February 6, 2026 to February 22, 2026

Table of Contents

Project Summary

Project Scope

Project Name	Repository	Commit	Platform
Met Extension (met-extension-release)	met-extension-release (private)	`Latest as of 2026-02-21`	TypeScript, React, HTML, CSS (Browser Extension - Manifest V3)

Engagement Period: From February 6, 2026 to February 22, 2026

Project Overview

This document presents the findings of a manual offensive security code review of the Met Extension - a browser extension (Chrome/Firefox) built on the wxt framework and React. The extension is a UI overlay and intelligence enhancer for decentralized finance (DeFi) and prediction market platforms, specifically Meteora, Polymarket, and Hyperliquid. The review covered the full extension attack surface: background service worker message routing, content script DOM injection pipelines, inter-process communication (IPC) via chrome.runtime.onMessage, OAuth/PKCE authentication flows, bearer token storage and lifecycle management, WebSocket/SSE streaming data handlers, and npm dependency security. The engagement ran from February 6, 2026 to February 22, 2026, consisting entirely of static code review with source-to-sink tracing. No dynamic or runtime testing was performed. Every finding included in this report has survived a four-stage validation pipeline: automated detection, source-to-sink evidence gathering, real-world impact assessment, and adversarial self-review with active attempts to disprove the finding.

The following files are included in scope:

```
entrypoints/background.ts
entrypoints/polymarket.content/tooltip-enhancer.ts
entrypoints/polymarket.content/smart-money.ts
entrypoints/hyperliquid.content/smart-trades.tsx
entrypoints/meteora.content/
entrypoints/sidepanel/
lib/helpers/polymarket-messages.ts
lib/helpers/auth.ts
lib/polymarket-api.ts
components/ (all)
utils/ (all)
wxt.config.ts
package.json
```

MetEngine Browser Extension Overview

Met Extension is a browser extension that enhances three major DeFi and prediction market platforms - Meteora, Polymarket, and Hyperliquid - with advanced wallet intelligence, smart money tracking, and real-time trading signals.

The extension operates via a background service worker (background.ts) acting as an API gateway. It authenticates users through a custom OAuth/PKCE flow powered by Privy (@privy-io/react-auth), stores bearer tokens in browser.storage.local, and proxies authenticated API requests to the MetEngine backend (api.metengine.xyz, perpetuals.metengine.xyz). Content scripts injected into target sites use MutationObservers and DOM event listeners to detect UI contexts and trigger enrichment flows, injecting wallet scores, metrics badges, and profile tooltips into the host page DOM.

The extension's side panel provides a React-based UI for managing wallet connections, viewing real-time trading notifications delivered via Server-Sent Events (SSE), and managing subscription tiers. Subscription access control is enforced both client-side (UI gating) and server-side (API-level data filtering).

The security model relies heavily on the Manifest V3 browser extension isolation model: the background script validates message senders against a hardcoded ALLOWED_ORIGINS list, and content scripts are assumed to operate inside pages with functioning Content Security Policies.

Threat and Security Overview

The review focused on the unique threat surface of a privileged browser extension that straddles the boundary between web page context and elevated extension context. Primary concerns included message handler validation, inter-process communication security, DOM injection safety, authentication token lifecycle management, and session handling across browser contexts.

Two High severity findings were identified related to insufficient sender validation in the background message handler and unsafe MAIN world script injection. These findings represented critical authorization gaps that could allow malicious extensions or compromised pages to invoke privileged operations and manipulate host page state. Both have been fixed by the MetEngine team.

Three Medium severity findings were also identified and fixed: missing HTML encoding in innerHTML operations, reverse tabnabbing vulnerabilities in external wallet links, and insufficient logout validation allowing continued backend streaming after client logout. All findings represent real security gaps that were practically exploitable but have been successfully remediated. No Critical severity findings were identified.

Findings Summary

The table below summarises the findings of this review by severity.

Severity	Discovered	Confirmed	Fixed
Critical	–	–	–
High	2	2	2
Medium	3	3	3
Low	–	–	–
Informational	–	–	–
Total	**5**	**5**	**5**

Severity Matrix

Impact (rows) × Likelihood (columns)

	Low	Medium	High
High	Medium	High	Critical
Medium	Low	Medium	High
Low	Low	Low	Medium

Detailed Findings

ID	Title	Severity	Status
H-01	No Sender Validation on Background Message Handler	High	Fixed
H-02	MAIN World Script Injection Enables State Manipulation	High	Fixed
M-01	Missing HTML Encoding in DOM Injection Operations	Medium	Fixed
M-02	Reverse Tabnabbing Vulnerability in External Wallet Links	Medium	Fixed
M-03	Client-Only Logout Allows Continued Backend Streaming	Medium	Fixed

High Severity Issues

H-01 No Sender Validation on Background Message Handler

Fixed

Severity: High **Impact:** High **Likelihood:** Medium

Files: entrypoints/background.ts (line 360)

Description

The background worker's chrome.runtime.onMessage listener accepts messages from any sender without validating sender.id, sender.origin, or sender.tab. This allows malicious extensions or compromised scripts on target pages to invoke privileged operations using the victim's authentication token.

In Chrome's extension architecture, any installed extension can send messages if it knows the extension ID (public for published extensions), and content scripts on matched domains can freely send messages. A malicious actor could call api:request endpoints to exfiltrate paid subscription data, trigger purchase:start flows, manipulate wallet watchlists, or control SSE streaming connections - all using the victim's stored bearer token.

Unvalidated message handler (background.ts:360)

```
chrome.runtime.onMessage.addListener((message, _sender, sendResponse) => {
  if (message.type === 'api:request') {
    handleApiRequest(message.endpoint, message.body)
      .then((data) => sendResponse({ success: true, data }))
      .catch((error) => sendResponse({ success: false, error: error.message }));
    return true;
  }
  // No sender validation - accepts from any source
});
```

Attack Scenario

• Malicious extension or XSS on target site sends message to Met Extension
• Message includes type: api:request with target wallet address
• Background handler accepts message without validating sender
• Extension uses stored bearer token to fetch subscription-gated data
• Attacker receives response and exfiltrates paid intelligence data

Recommendations

Implement sender validation to accept messages only from the extension itself and verify tab origins for sensitive operations. Check sender.id matches chrome.runtime.id and validate sender.tab.url against an allowlist of expected origins before processing privileged message types.

```
chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {
  // Only accept messages from our own extension
  if (sender.id !== chrome.runtime.id) {
    sendResponse({ success: false, error: 'Unauthorized sender' });
    return true;
  }

  // For sensitive operations, verify sender origin
  const allowedOrigins = [
    'https://meteora.ag', 'https://polymarket.com',
    'https://hyperliquid.xyz'
  ];

  if (message.type === 'api:request' && sender.tab) {
    const senderOrigin = new URL(sender.tab.url || '').origin;
    if (!allowedOrigins.includes(senderOrigin)) {
      sendResponse({ success: false, error: 'Unauthorized origin' });
      return true;
    }
  }
  // ... rest of handlers
});
```

Customer's Response

> *Fixed. Added comprehensive sender validation checking both extension ID and tab origin for all privileged message handlers.*

Fix Review: Verified. Sender validation has been implemented with proper checks for sender.id and origin validation for sensitive operations.

H-02

MAIN World Script Injection Enables State Manipulation

Fixed

Severity: High **Impact:** High **Likelihood:** Medium

Files: entrypoints/background.ts (lines 503-624)

Description

The fill:price-range message handler injects code into the MAIN world context of the Meteora page, giving it full access to the page's JavaScript environment and React internals. Combined with the lack of sender validation, any source can trigger this handler with arbitrary price range values.

The injected function traverses React's internal fiber tree and directly invokes onChange and onValueChange handlers to manipulate price range sliders. This can silently change LP position parameters before transaction submission, potentially causing users to provide liquidity at unfavorable price ranges. The weightedBins, minPrice, and maxPrice parameters are passed without validation.

MAIN world injection with React fiber traversal (background.ts:503)

```
if (message.type === 'fill:price-range') {
  chrome.scripting.executeScript({
    target: { tabId },
    world: 'MAIN', // Runs in host page context
    func: (weightedBins, minPrice, maxPrice) => {
      // Traverse React fiber tree
      const fiberKey = Object.keys(sliders[0]).find((k) =>
        k.startsWith('__reactFiber'));
      let current = sliders[0][fiberKey];
      // Call React internal handlers directly
      if (formOnChange) formOnChange(newValue);
      if (radixOnValueChange) radixOnValueChange(newValue);
    },
    args: [message.weightedBins, message.minPrice, message.maxPrice]
  });
}
```

Attack Scenario

- Attacker sends fill:price-range message with extreme values
- Background handler accepts message without sender validation
- Code injected into MAIN world with full page access
- React fiber traversal locates and invokes form handlers
- Price range silently changed to unfavorable values (e.g., minPrice: 0.001)
- User submits transaction with manipulated LP parameters

Recommendations

Add sender validation (as in H-01), validate input types and ranges before execution, verify sender is from meteora.ag specifically, and consider moving this logic to the content script's ISOLATED world using DOM events instead of React fiber traversal.

```
if (message.type === 'fill:price-range') {
  // Validate sender is from Meteora
  const senderUrl = _sender.tab?.url || '';
  if (!senderUrl.includes('meteora.ag')) {
    sendResponse({ success: false, error: 'Invalid origin' });
    return true;
  }

  // Validate input types and ranges
  if (message.weightedBins && !Array.isArray(message.weightedBins)) {
    sendResponse({ success: false, error: 'Invalid weightedBins' });
    return true;
  }

  if (typeof message.minPrice !== 'number' || message.minPrice < 0) {
    sendResponse({ success: false, error: 'Invalid minPrice' });
    return true;
  }
  // ... proceed with validated data
}
```

Customer's Response

> *Fixed. Implemented sender validation, input validation for all price parameters, and restricted execution to meteora.ag origin only.*

Fix Review: Verified. The handler now includes comprehensive validation and origin checking before executing MAIN world scripts.

Medium Severity Issues

M-01 Missing HTML Encoding in DOM Injection Operations

Fixed

Severity: Medium **Impact:** Medium **Likelihood:** Low

Files: entrypoints/polymarket.content/smart-money.ts

Description

Content scripts use innerHTML to inject wallet intelligence data into the host page DOM. While an escapeHtml() utility exists in the codebase, several innerHTML operations in smart-money.ts insert data without HTML encoding. This creates a cross-site scripting (XSS) risk if backend API responses contain malicious HTML/JavaScript, or if localhost development endpoints (whitelisted in wxt.config.ts) serve compromised data.

The vulnerability exists because the extension trusts all data from whitelisted origins, including localhost ports used during development. An attacker who compromises the backend or intercepts localhost API responses could inject script tags or event handlers that execute in the context of the target page.

Unencoded innerHTML usage (smart-money.ts:1048)

```
element.innerHTML = profileData.displayName;
// No HTML encoding applied before injection
```

Attack Scenario

• Attacker compromises backend API or localhost development server

• Malicious response contains HTML payload with script tags or event handlers

• Extension injects unencoded response via innerHTML

• Script executes in polymarket.com context with access to cookies and user data

Recommendations

Apply HTML encoding to all data before innerHTML injection. Use the existing escapeHtml() utility or textContent assignment instead of innerHTML where possible. For development environments, implement additional input validation and sanitization even for localhost origins.

```
// Apply HTML encoding before innerHTML
element.innerHTML = escapeHtml(profileData.displayName);

// Or use textContent for plain text
element.textContent = profileData.displayName;
```

Customer's Response

> *Fixed. All innerHTML operations now use the escapeHtml() utility before injection. Development environment handling has been strengthened with additional input validation.*

Fix Review: Verified. The MetEngine team has added HTML encoding to all innerHTML operations and implemented additional sanitization for development endpoints.

M-02 Reverse Tabnabbing Vulnerability in External Wallet Links

Fixed

Severity: Medium **Impact:** Medium **Likelihood:** Low

Files: entrypoints/hyperliquid.content/smart-trades.tsx, entrypoints/polymarket.content/smart-money.ts

Description

External wallet explorer links in the extension open in new tabs without rel="noopener noreferrer" attributes. This allows the newly opened page to access the opener window object via window.opener, enabling reverse tabnabbing attacks where a malicious or compromised blockchain explorer could redirect the original polymarket.com or hyperliquid.app page to a phishing site.

The vulnerability appears in two locations: wallet explorer links in smart-trades.tsx and wallet tooltip links in smart-money.ts. While blockchain explorers are generally trustworthy, any compromise or malicious redirect on those platforms could be leveraged to attack extension users.

Vulnerable link without noopener (smart-trades.tsx)

```
<a href={explorerUrl} target="_blank">
  View Wallet
</a>
// Missing rel="noopener noreferrer"
```

Attack Scenario

• User clicks wallet explorer link injected by extension

• Link opens in new tab without noopener

• Compromised or malicious explorer page accesses window.opener

• Attacker redirects original tab to phishing site mimicking polymarket.com

• User returns to tab and enters credentials on fake site

Recommendations

Add rel="noopener noreferrer" to all external links that open in new tabs. This severs the window.opener reference and prevents the opened page from navigating the original tab. This is a standard security practice for all target="_blank" links.

```
<a href={explorerUrl} target="_blank" rel="noopener noreferrer">
  View Wallet
</a>
```

Customer's Response

> *Fixed. All external links now include rel="noopener noreferrer" to prevent reverse tabnabbing attacks.*

Fix Review: Verified. Both hyperliquid.content/smart-trades.tsx and polymarket.content/smart-money.ts have been updated with proper noopener/noreferrer attributes on external links.

M-03 Client-Only Logout Allows Continued Backend Streaming

Fixed

Severity: Medium **Impact:** Medium **Likelihood:** Medium

Files: lib/helpers/auth.ts, entrypoints/background.ts

Description

The Privy logout flow only clears local extension state and does not invalidate the session on the backend. After a user logs out via the extension UI, the background script's Server-Sent Events (SSE) connection to the MetEngine backend continues to stream authenticated trading notifications using the cached bearer token.

This means a user who logs out expecting their session to end will continue receiving real-time data updates until the browser is closed or the extension is reloaded. In a shared-device scenario, another user could access the side panel and view notifications intended for the previous user.

Logout clears local state only (auth.ts)

```
export async function logout() {
  privy.logout(); // Local only
  await browser.storage.local.remove('auth_token');
  // No backend session invalidation
}
```

Attack Scenario

• User A logs into extension on shared device

• Background SSE connection established with bearer token

• User A clicks logout - local state cleared

• SSE connection remains active with cached token

• User B opens extension and views User A's notifications

• Sensitive wallet tracking data leaked across users

Recommendations

Implement proper logout validation on the backend. When the extension calls logout, send a token revocation request to the MetEngine API to invalidate the bearer token server-side. Close the SSE connection immediately on logout and require re-authentication to establish a new streaming connection.

```
export async function logout() {
  const token = await getAuthToken();
  if (token) {
    // Revoke token on backend
    await fetch('https://api.metengine.xyz/auth/revoke', {
      method: 'POST',
      headers: { 'Authorization': `Bearer ${token}` }
    });
  }
  privy.logout();
  await browser.storage.local.remove('auth_token');
  // Close SSE connection
  closeStreamingConnection();
}
```

Customer's Response

> *Fixed. Backend logout endpoint implemented. Extension now calls token revocation API on logout and immediately closes the SSE streaming connection.*

Fix Review: Verified. The authentication flow now includes proper server-side session invalidation and streaming connections are terminated on logout.

Disclaimer

This security review was performed on a point-in-time snapshot of the codebase provided by MetEngine. The findings reflect the state of the code as reviewed and should not be interpreted as a guarantee of the absence of all vulnerabilities. Security reviews are inherently limited by scope, time, and available context. New vulnerabilities may be introduced after the review period. This report is intended solely for use by MetEngine and should not be shared with third parties without prior written consent. All findings should be verified by a qualified security professional before remediation decisions are made in production systems.

About Manish Bhattacharya

Manish Bhattacharya is a Staff Security Engineer with 9+ years securing FinTech and Crypto startups. He is an 18x Google Vulnerability Research Grant recipient recognised in the Hall of Fame of 25+ organisations including Google, Meta, Apple, Microsoft, Coinbase, and Stripe. Past roles include first security hire and SOC2 lead at Crossmint, SynapseFi and consulted Web3 companies like XBTO Group, and Unstoppable Domains.

Website: https://manishbhattacharya.com/