ChatGPT Account Takeover - Wildcard Web Cache Deception

ChatGPT Account Takeover - Wildcard Web Cache Deception - Author not stated, Harel Security Research.

- Published: 2024-02-04
- Original: <https://nokline.github.io/bugbounty/2024/02/04/ChatGPT-ATO.html>
- Preserved from: https://nokline.github.io/bugbounty/2024/02/04/ChatGPT-ATO.html (live) on 2026-08-09
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Intro

Here's how I was able to take over your account in ChatGPT.

Last year Nagli discovered a web cache deception vulnerability in ChatGPT. The impact of this was critical, as it lead to the leak of user's auth tokens and subsequently, an account takeover. OpenAI notified users of ChatGPT of this vulnerability, and quickly patched the bug... Or did they?

In this writeup, I will explain how I was able to abuse a path traversal URL parser confusion to achieve what I like to call a "wildcard" cache deception vulnerability, in order to steal user's auth tokens and take over their accounts. I will make the assumption that readers know the basics of the web cache deception vulnerability, as I will not go into too much depth explaining it. If you are not already familiar with this awesome vulnerability yet, or would like a refresher, I highly reccomend to check out Nagli's writeup first and come back to this one. Additionally, this bug uses a similar concept to the web cache poisoning vulnerability I found in Glassdoor last year, which allows us to cache "un-cacheable" files and endpoints. While it is not exactly the same technique, it demonstrates how much potential there is for URL parser confusions, specifically with path traversals, to open new doors for all sorts of cache vulnerabilities.

Initial Discovery

While playing around with ChatGPT's newly implemented "share" feature, which allows users to publicly share their chats with others, I noticed something weird. None of my shared chat links

would update as I continued talking with ChatGPT. After dealing with bugs like this for a while, the first thing that came to mind was a caching issue. I figured that the shared chat was cached, and therfore wouldn't update until the cache entry died. To test this out, I opened the network tab in my dev tools to check the response headers, and as I predicted, I saw the `Cf-Cache-Status: HIT` header. This was pretty interesting to me, as this was not a static file. I checked out the URL, and saw that the path did not have a static extension as expected:

`https://chat.openai.com/share/CHAT-UUID`

This meant that there was likely a cache rule that did not rely on the extension of the file, but on its location in the URL's path. To test this, I checked `https://chat.openai.com/share/random-path-that-does-not-exist` And as expected, it was also cached. It quickly became evident that the cache rule looked something like this: `/share/*` Which means that pretty much anything under the `/share/` path gets cached. This was immediatly a red flag (or green flag depending on how you look at it), as I made a note to myself during my last cache poisoning research that relaxed cache rules can be very dangerous, especially with URL parser confusions.

Path Traversal Confusion

In a website that uses caching, the request must go through the CDN before it gets to the web server. This means that the URL gets parsed twice, which makes it possible for a URL parser confusion. In ChatGPT's case, a URL parser confusion meant that the two servers parse URL encoded forward slashes differently, where Cloudflare's CDN did NOT decode and did NOT normalize a URL encoded path traversal, but the web server did. So a URL encoded path traversal allows an attacker to cache any file they wish from the server, including the highest impact API endpoints which contain authorization tokens. This sounds a bit confusing, so here is an example payload: Note that the `%2F` decodes to `/` and `/api/auth/session` is a sensitive API endpoint which contains the user's auth token `https://chat.openai.com/share/%2F..%2Fapi/auth/session?cachebuster=123` So let's break this down.

- We've already established that the CDN will cache anything under `/share/`
- We also said that the CDN will NOT decode nor normalize `%2F..%2F`, therfore, the response WILL be cached
- However, when the CDN forwards this URL, the web server WILL decode and normalize `%2F..%2F`, and will respond with `/api/auth/session`, which contains the auth token.

Putting this together, when the victim goes to `https://chat.openai.com/share/%2F..%2Fapi/auth/session?cachebuster=123`, their auth token will be cached. When the attacker later goes to visit `https://chat.openai.com/share/%2F..%2Fapi/auth/session?cachebuster=123`, they will see the victim's cached auth token. This is game over. Once the attacker has the auth token, they can now takeover the account, view chats, billing information, and more.

Here's a little sketch I drew to help you all visualize this:

[image: image]

So to sum it all up in a sentence, I was able to use a URL encoded path traversal to cache sensitive API endpoints, thanks to a path normalization inconsistency between the CDN and web server.

Surprisingly, this was probably my quickest find in bug bounty, as well as one of my more interesting ones, and my biggest bounty thus far of $6500.
