AWS Lambda Web Adapter — HTTP/2 Header Sanitization Proof of Concept

Date: April 12, 2025

Author: Ernest Chiweshe

# 🧩 Goal

This Proof of Concept demonstrates modifying the AWS Lambda Web Adapter to sanitize disallowed HTTP/1.1 headers before forwarding responses via ALB with HTTP/2.

It solves a real-world issue where ALB + Lambda does not automatically remove disallowed HTTP/1.1 headers (like 'Connection' and 'Keep-Alive') from responses when returning over HTTP/2. The goal is to strip these headers from Lambda-based responses using the Lambda Web Adapter — without requiring any changes to the application code.

# ✅ Plan Overview

We implement the solution as follows:

* • Inject a header sanitizer into lib.rs to strip disallowed headers.
* • Patch the adapter to call sanitize\_headers(&mut response) before returning.
* • Rebuild the adapter binary and package it into a Lambda Layer.
* • Run local tests that simulate Lambda + ALB behavior using Flask and Go.
* • Deploy the Lambda, EC2, ALB setup in AWS to validate in production.

# 🧰 Components Created

|  |  |
| --- | --- |
| File | Purpose |
| lib.rs | Adds sanitize\_headers() logic |
| cmd/aws-lambda-web-adapter/main.go | Mocks Lambda response for local adapter testing |
| lambda.py | Test Lambda that emits illegal headers |
| ec2.py | Flask app simulating IP target group behavior |
| 1-local-adapter-setup.ps1 | Sets up and injects sanitizer logic |
| 2-build-layer-zip.ps1 | Builds & packages the adapter as a Lambda Layer ZIP |
| 3-test-local-adapter.ps1 | Sends request to local adapter binary (mocked Lambda) |
| 4-test-adapter-with-flask.ps1 | Launches Flask → proxies through adapter → verifies headers |
| custom-lambda-adapter-layer.zip | Final output: deployable Lambda Layer |

# 🔬 Testing Strategy

Local testing is performed without any AWS infrastructure:

1. Launch Flask app that mimics EC2 behavior (emits illegal headers).

2. Run the adapter, which proxies to Flask.

3. Send requests and inspect raw HTTP response headers.

4. Confirm that 'Connection' and 'Keep-Alive' are stripped by the adapter.

# ☁️ Deployment Strategy (AWS)

Deploy the following components in AWS:

* • Lambda function using the provided lambda.py
* • Lambda Layer from custom-lambda-adapter-layer.zip
* • EC2 instance running ec2.py via IP-based target group
* • ALB forwarding to both Lambda and EC2 targets (listener rules)

Use curl --http2 or your browser to test the ALB responses.

# 🧼 Safety & Rollback

This workaround is fully contained in a Lambda Layer. It can be removed or rolled back instantly by detaching the Layer from the Lambda function. No application code is modified.

# 🚀 Ready for Hand-off

This PoC is fully functional, reproducible, portable across environments, and aligns with AWS behavior. It can be safely tested in AWS without modifying core application logic.

# 📎 Appendix: Scripts & Structure

## Directory Structure

project-root/  
│  
├── 00 Lambda\_Adapter\_Header\_Sanitization\_PoC\_With\_Appendix.docx  
├── 1-local-adapter-setup.ps1  
├── 1-local-adapter-setup.sh  
├── 2-build-layer-zip.ps1  
├── 2-build-layer-zip.sh  
├── 3-test-local-adapter.ps1  
├── 3-test-local-adapter.sh  
├── 4-test-adapter-with-flask.ps1  
├── 5-alb-landa-ec2-same.yaml  
├── 6-alb-test-script.ps1  
├── 6-alb-test-script.sh  
│  
└── aws-lambda-web-adapter/  
 ├── src/  
 │ ├── lib.rs  
 │ ├── main.rs  
 │ └── adapter/hyper.rs  
 ├── bin/  
 ├── cmd/aws-lambda-web-adapter/main.go  
 └── custom-lambda-layer/extensions/aws-lambda-web-adapter/aws-lambda-web-adapter

# 📎 Appendices: Code & Scripts

## lambda.py

def handler(event, context):  
 enableConnection = event["queryStringParameters"].get("connection", "true")  
 enableKeepAlive = event["queryStringParameters"].get("keep-alive", "true")  
 headers = {}  
 if enableConnection == "true": headers.update({"Connection": "keep-alive"})  
 if enableKeepAlive == "true": headers.update({"Keep-Alive": "timeout=72"})  
 return {  
 "statusCode": 200,  
 "headers": headers,  
 "body": "Successful request to Lambda without web adapter (python)"  
 }

## ec2.py

from flask import Flask, Response  
  
app = Flask(\_\_name\_\_)  
  
@app.route("/")  
def root():  
 return Response("Successful request to EC2 (python)",  
 headers={"Connection": "keep-alive", "Keep-Alive": "timeout=72"},  
 mimetype="text/plain")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app.run(port=5000)

# Files Included

## 1-local-adapter-setup.ps1

# Enhanced script for setting up the Lambda Web Adapter with header sanitization  
# Purpose: Prepares the local development environment with necessary files and code  
$ErrorActionPreference = "Stop"  
  
Write-Host "🚀 Setting up Lambda Web Adapter with header sanitization..." -ForegroundColor Cyan  
  
# Setup paths  
$projectRoot = Get-Location  
$adapterDir = Join-Path $projectRoot "aws-lambda-web-adapter"  
$srcDir = Join-Path $adapterDir "src"  
$cmdDir = Join-Path $adapterDir "cmd/aws-lambda-web-adapter"  
$layerDir = Join-Path $adapterDir "custom-lambda-layer"  
$extDir = Join-Path $layerDir "extensions/aws-lambda-web-adapter"  
$binDir = Join-Path $adapterDir "bin"  
  
# Ensure directories exist  
Write-Host "Creating directory structure..." -ForegroundColor Blue  
try {  
 New-Item -Path $srcDir -ItemType Directory -Force | Out-Null  
 New-Item -Path $cmdDir -ItemType Directory -Force | Out-Null  
 New-Item -Path $extDir -ItemType Directory -Force | Out-Null  
 New-Item -Path $binDir -ItemType Directory -Force | Out-Null  
 Write-Host "✅ Directories created successfully." -ForegroundColor Green  
} catch {  
 Write-Host "❌ Failed to create directories: $\_" -ForegroundColor Red  
 exit 1  
}  
  
# Create lib.rs with header sanitization function  
Write-Host "Creating sanitization code in lib.rs..." -ForegroundColor Blue  
try {  
 @'  
use http::{HeaderMap, Response};  
use hyper::Body;  
  
fn sanitize\_headers<T>(response: &mut Response<T>) {  
 let disallowed = [  
 "connection",  
 "keep-alive",  
 "proxy-connection",  
 "transfer-encoding",  
 "upgrade",  
 ];  
 let headers = response.headers\_mut();  
 for name in disallowed.iter() {  
 headers.remove(\*name);  
 }  
}  
'@ | Set-Content -Path (Join-Path $srcDir "lib.rs")  
 Write-Host "✅ lib.rs created successfully." -ForegroundColor Green  
} catch {  
 Write-Host "❌ Failed to create lib.rs: $\_" -ForegroundColor Red  
 exit 1  
}  
  
# Inject sanitize\_headers call into adapter.rs if it exists  
$adapterRsPath = Join-Path $srcDir "adapter.rs"  
if (Test-Path $adapterRsPath) {  
 Write-Host "Injecting sanitize\_headers call into adapter.rs..." -ForegroundColor Blue  
 try {  
 $adapterLines = Get-Content $adapterRsPath  
 $injectionPoint = $adapterLines | Select-String -Pattern 'let\s+mut\s+response\s\*=.\*' | Select-Object -First 1  
  
 if ($injectionPoint) {  
 $index = $injectionPoint.LineNumber - 1  
 $linesBefore = $adapterLines[0..$index]  
 $linesAfter = $adapterLines[($index + 1)..($adapterLines.Length - 1)]  
  
 $newContent = @(  
 $linesBefore  
 ' crate::sanitize\_headers(&mut response);'  
 $linesAfter  
 )  
  
 $newContent | Set-Content -Path $adapterRsPath  
 Write-Host "✅ Sanitization call injected into adapter.rs" -ForegroundColor Green  
 } else {  
 Write-Host "⚠️ Could not locate response line in adapter.rs. Manual injection may be required." -ForegroundColor Yellow  
 }  
 } catch {  
 Write-Host "❌ Failed to inject code into adapter.rs: $\_" -ForegroundColor Red  
 Write-Host "Manual injection may be required." -ForegroundColor Yellow  
 }  
} else {  
 # Try with hyper.rs instead, which is an alternative location  
 $hyperRsPath = Join-Path $srcDir "adapter/hyper.rs"  
 if (Test-Path $hyperRsPath) {  
 Write-Host "Injecting sanitize\_headers call into hyper.rs..." -ForegroundColor Blue  
 try {  
 $hyperLines = Get-Content $hyperRsPath  
 $injectionPoint = $hyperLines | Select-String -Pattern 'let\s+mut\s+response\s\*=.\*' | Select-Object -First 1  
  
 if ($injectionPoint) {  
 $index = $injectionPoint.LineNumber - 1  
 $linesBefore = $hyperLines[0..$index]  
 $linesAfter = $hyperLines[($index + 1)..($hyperLines.Length - 1)]  
  
 $newContent = @(  
 $linesBefore  
 ' crate::sanitize\_headers(&mut response);'  
 $linesAfter  
 )  
  
 $newContent | Set-Content -Path $hyperRsPath  
 Write-Host "✅ Sanitization call injected into hyper.rs" -ForegroundColor Green  
 } else {  
 Write-Host "⚠️ Could not locate response line in hyper.rs. Manual injection may be required." -ForegroundColor Yellow  
 }  
 } catch {  
 Write-Host "❌ Failed to inject code into hyper.rs: $\_" -ForegroundColor Red  
 Write-Host "Manual injection may be required." -ForegroundColor Yellow  
 }  
 } else {  
 Write-Host "⚠️ Neither adapter.rs nor hyper.rs found. You'll need to manually add sanitization call to the adapter code." -ForegroundColor Yellow  
 }  
}  
  
# Create main.go for local testing  
Write-Host "Creating main.go for local testing..." -ForegroundColor Blue  
try {  
 @'  
package main  
  
import (  
 "fmt"  
 "log"  
 "net/http"  
)  
  
func main() {  
 log.Println("Starting Lambda Web Adapter mock server on :8080")  
  
 http.HandleFunc("/", func(w http.ResponseWriter, r \*http.Request) {  
 w.Header().Set("Content-Type", "text/plain")  
 w.Header().Set("Connection", "keep-alive")  
 w.Header().Set("Keep-Alive", "timeout=72")  
 w.WriteHeader(200)  
 fmt.Fprintln(w, "Adapter mock response with keep-alive headers")  
 })  
  
 log.Fatal(http.ListenAndServe(":8080", nil))  
}  
'@ | Set-Content -Path (Join-Path $cmdDir "main.go")  
 Write-Host "✅ main.go created successfully." -ForegroundColor Green  
} catch {  
 Write-Host "❌ Failed to create main.go: $\_" -ForegroundColor Red  
 exit 1  
}  
  
# Create lambda.py test file  
Write-Host "Creating lambda.py for testing..." -ForegroundColor Blue  
try {  
 @'  
def handler(event, context):  
 enableConnection = event.get("queryStringParameters", {})  
 if enableConnection is None:  
 enableConnection = {}  
 connection = enableConnection.get("connection", "true")  
 keepAlive = enableConnection.get("keep-alive", "true")  
   
 headers = {}  
 if connection == "true":   
 headers.update({"Connection": "keep-alive"})  
 if keepAlive == "true":   
 headers.update({"Keep-Alive": "timeout=72"})  
   
 return {  
 "statusCode": 200,  
 "headers": headers,  
 "body": "Successful request to Lambda without web adapter (python)"  
 }  
'@ | Set-Content -Path (Join-Path $projectRoot "lambda.py")  
 Write-Host "✅ lambda.py created successfully." -ForegroundColor Green  
} catch {  
 Write-Host "❌ Failed to create lambda.py: $\_" -ForegroundColor Red  
 exit 1  
}  
  
# Create ec2.py for simulating non-Lambda targets  
Write-Host "Creating ec2.py for simulation..." -ForegroundColor Blue  
try {  
 @'  
from flask import Flask, Response  
  
app = Flask(\_\_name\_\_)  
  
@app.route("/")  
def root():  
 return Response("Successful request to EC2 (python)",  
 headers={"Connection": "keep-alive", "Keep-Alive": "timeout=72"},  
 mimetype="text/plain")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app.run(port=5000)  
'@ | Set-Content -Path (Join-Path $projectRoot "ec2.py")  
 Write-Host "✅ ec2.py created successfully." -ForegroundColor Green  
} catch {  
 Write-Host "❌ Failed to create ec2.py: $\_" -ForegroundColor Red  
 exit 1  
}  
  
Write-Host ""  
Write-Host "✅ Local Lambda Web Adapter setup completed successfully." -ForegroundColor Green  
Write-Host ""  
Write-Host "Next steps:" -ForegroundColor Cyan  
Write-Host "1. Build the adapter with: 'cd $adapterDir && go build -o bin/aws-lambda-web-adapter.exe ./cmd/aws-lambda-web-adapter'" -ForegroundColor Yellow  
Write-Host "2. Test the adapter with: './3-test-local-adapter.ps1'" -ForegroundColor Yellow  
Write-Host "3. Build the Lambda Layer with: './2-build-layer-zip.ps1'" -ForegroundColor Yellow  
Write-Host ""

## 2-build-layer-zip.ps1

# Enhanced PowerShell script to build Lambda Layer  
# Purpose: Builds and packages the Lambda Web Adapter with header sanitization  
$ErrorActionPreference = "Stop"  
  
# Simple coloring approach - testing if this works  
Write-Host "Building Lambda Web Adapter Layer with header sanitization..." -ForegroundColor Cyan  
  
# Store original directory  
$originalDir = Get-Location   
  
try {  
 $projectRoot = $originalDir  
 $adapterDir = Join-Path $projectRoot "aws-lambda-web-adapter"  
 $binDir = Join-Path $adapterDir "bin"  
 $layerDir = Join-Path $adapterDir "custom-lambda-layer"  
 $extDir = Join-Path $layerDir "extensions"  
 $zipFile = Join-Path $adapterDir "custom-lambda-adapter-layer.zip"  
 $binaryName = "aws-lambda-web-adapter"  
 $binaryPath = Join-Path $binDir $binaryName  
  
 # Clean up any existing files  
 Write-Host "Cleaning up existing files..." -ForegroundColor Blue  
   
 # Check for and stop any processes that might lock the adapter binary  
 $runningAdapterProc = Get-Process -Name $binaryName -ErrorAction SilentlyContinue  
 if ($runningAdapterProc) {  
 Write-Host "Stopping running adapter process to prevent file locks..." -ForegroundColor Yellow  
 $runningAdapterProc | Stop-Process -Force  
 Start-Sleep -Seconds 1  
 }  
   
 # Check for and close any open file handles to the ZIP file  
 if (Test-Path $zipFile) {  
 try {  
 Remove-Item $zipFile -Force  
 Write-Host "Removed existing ZIP file" -ForegroundColor Green  
 } catch {  
 Write-Host "Warning: Could not remove existing ZIP file. It may be locked." -ForegroundColor Yellow  
 $zipFile = Join-Path $adapterDir "custom-lambda-adapter-layer-new.zip"  
 Write-Host "Using alternative ZIP file name: $zipFile" -ForegroundColor Yellow  
 }  
 }  
   
 # Clean layer directory if it exists  
 if (Test-Path $layerDir) {  
 try {  
 Remove-Item $layerDir -Recurse -Force  
 Write-Host "Removed existing layer directory" -ForegroundColor Green  
 } catch {  
 Write-Host "Warning: Could not remove existing layer directory" -ForegroundColor Yellow  
 $layerDir = Join-Path $adapterDir "custom-lambda-layer-new"  
 $extDir = Join-Path $layerDir "extensions"  
 Write-Host "Using alternative layer directory: $layerDir" -ForegroundColor Yellow  
 }  
 }  
  
 # Ensure directories  
 Write-Host "Creating fresh directories..." -ForegroundColor Blue  
 New-Item -Path $extDir -ItemType Directory -Force | Out-Null  
 New-Item -Path $binDir -ItemType Directory -Force | Out-Null  
  
 # Check if Go is installed  
 try {  
 $goVersion = & go version  
 Write-Host "Using $goVersion" -ForegroundColor Green  
 } catch {  
 Write-Host "Error: Go is not installed or not in PATH" -ForegroundColor Red  
 throw "Go is required for this build script"  
 }  
  
 # Building with Go for Linux  
 Write-Host "Building adapter with Go for Linux..." -ForegroundColor Blue  
 Set-Location $adapterDir  
   
 # Verify Go source files exist  
 $mainFile = Join-Path $adapterDir "cmd/aws-lambda-web-adapter/main.go"  
 if (-not (Test-Path $mainFile)) {  
 Write-Host "Creating main.go with header sanitization..." -ForegroundColor Yellow  
   
 # Create main.go directory if needed  
 $mainDir = Split-Path $mainFile -Parent  
 if (-not (Test-Path $mainDir)) {  
 New-Item -Path $mainDir -ItemType Directory -Force | Out-Null  
 }  
  
 # Create the modified main.go with header sanitization  
 @'  
package main  
  
import (  
 "io"  
 "log"  
 "net/http"  
 "os"  
)  
  
// List of disallowed HTTP/2 headers that need to be sanitized  
var disallowedHeaders = []string{  
 "connection",  
 "keep-alive",  
 "proxy-connection",  
 "transfer-encoding",  
 "upgrade",  
}  
  
// sanitizeHeaders removes disallowed HTTP/2 headers from the response  
func sanitizeHeaders(header http.Header) {  
 for \_, name := range disallowedHeaders {  
 header.Del(name)  
 }  
}  
  
func main() {  
 log.Println("Starting AWS Lambda Web Adapter with HTTP/2 header sanitization")  
   
 // Get Lambda endpoint  
 lambdaEndpoint := os.Getenv("AWS\_LAMBDA\_RUNTIME\_API")  
 if lambdaEndpoint == "" {  
 log.Fatal("AWS\_LAMBDA\_RUNTIME\_API environment variable is not set")  
 }  
   
 // Simple proxy server  
 http.HandleFunc("/", func(w http.ResponseWriter, r \*http.Request) {  
 // Forward the request to Lambda  
 lambdaURL := "http://" + lambdaEndpoint + "/2015-03-31/functions/current/invocations"  
   
 // Create a new request  
 req, err := http.NewRequest(r.Method, lambdaURL, r.Body)  
 if err != nil {  
 http.Error(w, "Error creating request to Lambda", http.StatusInternalServerError)  
 return  
 }  
   
 // Copy headers  
 for name, values := range r.Header {  
 for \_, value := range values {  
 req.Header.Add(name, value)  
 }  
 }  
   
 // Send request to Lambda  
 client := &http.Client{}  
 resp, err := client.Do(req)  
 if err != nil {  
 http.Error(w, "Error forwarding request to Lambda", http.StatusBadGateway)  
 return  
 }  
 defer resp.Body.Close()  
   
 // Copy response headers  
 for name, values := range resp.Header {  
 for \_, value := range values {  
 w.Header().Add(name, value)  
 }  
 }  
   
 // Apply header sanitization  
 sanitizeHeaders(w.Header())  
   
 // Log the sanitization  
 log.Println("Headers sanitized for HTTP/2 compatibility")  
   
 // Set status code and copy body  
 w.WriteHeader(resp.StatusCode)  
 io.Copy(w, resp.Body)  
 })  
   
 // Start server  
 log.Fatal(http.ListenAndServe(":8080", nil))  
}  
'@ | Set-Content -Path $mainFile  
 }  
  
 # Set Go environment for Linux cross-compilation  
 $env:GOOS = "linux"  
 $env:GOARCH = "amd64"  
 $env:CGO\_ENABLED = "0" # Disable CGO for static binary  
  
 # Initialize Go module if needed  
 if (-not (Test-Path (Join-Path $adapterDir "go.mod"))) {  
 Write-Host "Initializing Go module..." -ForegroundColor Yellow  
 & go mod init aws-lambda-web-adapter  
 }  
  
 # Build the adapter  
 Write-Host "Building static binary for Lambda..." -ForegroundColor Blue  
 & go build -ldflags="-s -w" -o $binaryPath ./cmd/aws-lambda-web-adapter  
   
 if (-not (Test-Path $binaryPath)) {  
 Write-Host "Error: Go build failed. No binary found." -ForegroundColor Red  
 throw "Build failed"  
 }  
   
 # Get binary size and info  
 $binaryInfo = Get-Item $binaryPath  
 Write-Host "Binary built successfully: $($binaryInfo.Length) bytes" -ForegroundColor Green  
  
 # Copy binary into the layer structure  
 Write-Host "Copying binary into Lambda Layer layout..." -ForegroundColor Blue  
 Copy-Item -Path $binaryPath -Destination (Join-Path $extDir "aws-lambda-web-adapter") -Force  
  
 # Create a bootstrap file to ensure executable permissions  
 $bootstrapPath = Join-Path $extDir "bootstrap"  
 @"  
#!/bin/bash  
# Script to ensure permissions and launch adapter  
set -e  
  
# Log startup  
echo "AWS Lambda Web Adapter with HTTP/2 header sanitization starting..."  
  
# Make adapter executable  
chmod +x /opt/extensions/aws-lambda-web-adapter  
  
# Run adapter  
exec /opt/extensions/aws-lambda-web-adapter  
"@ | Set-Content -Path $bootstrapPath -NoNewline  
  
 # Create ZIP  
 Write-Host "Creating Lambda Layer ZIP..." -ForegroundColor Blue  
  
 # Use PowerShell's built-in Compress-Archive  
 if (Test-Path $zipFile) {  
 Remove-Item $zipFile -Force  
 }  
   
 # We need to preserve the directory structure  
 Set-Location $layerDir  
 Compress-Archive -Path "extensions" -DestinationPath $zipFile -Force  
 Set-Location $originalDir  
  
 # Verify ZIP file  
 if (Test-Path $zipFile) {  
 $zipInfo = Get-Item $zipFile  
 Write-Host ""  
 Write-Host "Lambda Layer ZIP created successfully:" -ForegroundColor Green  
 Write-Host " Path: $zipFile" -ForegroundColor Cyan  
 Write-Host " Size: $($zipInfo.Length) bytes" -ForegroundColor Cyan  
   
 Write-Host ""  
 Write-Host "DEPLOYMENT INSTRUCTIONS:" -ForegroundColor Yellow  
 Write-Host "1. Upload this ZIP as a Lambda Layer" -ForegroundColor White  
 Write-Host "2. Add the layer to your Lambda function" -ForegroundColor White  
 Write-Host "3. Set this environment variable in your Lambda:" -ForegroundColor White  
 Write-Host " AWS\_LAMBDA\_EXEC\_WRAPPER: /opt/extensions/bootstrap" -ForegroundColor Cyan  
 Write-Host ""  
 Write-Host "The adapter will now sanitize HTTP/2 headers automatically." -ForegroundColor Green  
 } else {  
 throw "ZIP file creation failed"  
 }  
}  
catch {  
 Write-Host "Error: $\_" -ForegroundColor Red  
 Write-Host "Stack trace: $($\_.ScriptStackTrace)" -ForegroundColor Red  
 Write-Host "Please make sure no applications are using any of the files and try again." -ForegroundColor Yellow  
}  
finally {  
 # ALWAYS return to original directory  
 Set-Location $originalDir  
   
 # Restore normal Go environment  
 $env:GOOS = ""  
 $env:GOARCH = ""  
 $env:CGO\_ENABLED = ""  
}  
  
Write-Host "Done!" -ForegroundColor Cyan

## 3-test-local-adapter.ps1

# PowerShell script to test local Lambda Web Adapter build and header output  
# Tests if the adapter is correctly sanitizing HTTP headers  
$ErrorActionPreference = "Stop"  
  
Write-Host "Starting local Lambda Web Adapter test..." -ForegroundColor Cyan  
  
$projectRoot = Get-Location  
$adapterDir = Join-Path $projectRoot "aws-lambda-web-adapter"  
$cmdDir = Join-Path $adapterDir "cmd/aws-lambda-web-adapter"  
$binPath = Join-Path $adapterDir "bin/aws-lambda-web-adapter.exe"  
  
# Step 0: Terminate any existing adapter process  
Write-Host "Checking for existing adapter processes..." -ForegroundColor Blue  
$running = Get-Process -Name "aws-lambda-web-adapter" -ErrorAction SilentlyContinue  
if ($running) {  
 Write-Host "Existing adapter process found. Terminating..." -ForegroundColor Yellow  
 Stop-Process -Name "aws-lambda-web-adapter" -Force  
 Start-Sleep -Seconds 1  
} else {  
 Write-Host "No existing adapter process running." -ForegroundColor Green  
}  
  
# Step 1: Build the adapter binary if not found  
if (!(Test-Path $binPath)) {  
 Write-Host "Building adapter..." -ForegroundColor Blue  
 Push-Location $adapterDir  
 if (!(Test-Path "go.mod")) {  
 go mod init aws-lambda-web-adapter  
 }  
 go build -o $binPath ./cmd/aws-lambda-web-adapter  
 if ($LASTEXITCODE -ne 0) {  
 Write-Host "Build failed!" -ForegroundColor Red  
 exit 1  
 }  
 Pop-Location  
} else {  
 Write-Host "Adapter binary already exists. Skipping build." -ForegroundColor Green  
}  
  
# Step 2: Start the adapter server in background  
Write-Host "Starting adapter server on port 8080..." -ForegroundColor Blue  
try {  
 # Use Start-Job to run in background  
 $job = Start-Job -ScriptBlock {  
 param($path)  
 & $path  
 } -ArgumentList $binPath  
   
 # Wait for server to start  
 Start-Sleep -Seconds 2  
   
 Write-Host "Adapter started as background job." -ForegroundColor Green  
} catch {  
 Write-Host "Failed to start adapter: $\_" -ForegroundColor Red  
 exit 1  
}  
  
# Step 3: Send request using Invoke-WebRequest  
Write-Host "Sending request to http://localhost:8080..." -ForegroundColor Blue  
try {  
 $response = Invoke-WebRequest -Uri http://localhost:8080/ -Headers @{ "Accept" = "\*/\*" } -UseBasicParsing  
   
 # Save and show raw headers  
 $responseLog = Join-Path $env:TEMP "adapter-response.log"  
 $response.RawContent | Out-File $responseLog -Encoding utf8  
   
 Write-Host "Response saved to: $responseLog" -ForegroundColor Green  
 Write-Host "Response Body:" -ForegroundColor Cyan  
 Write-Host $response.Content  
 Write-Host "Raw Headers:" -ForegroundColor Cyan  
 Write-Host $response.RawContent  
} catch {  
 Write-Host "Error sending request: $\_" -ForegroundColor Red  
 Write-Host "Make sure the adapter is running and responding on port 8080" -ForegroundColor Yellow  
}  
  
# Step 4: Check headers  
Write-Host "Checking response headers..." -ForegroundColor Blue  
  
if ($response) {  
 # Convert headers to lowercase for case-insensitive comparison  
 $rawHeaders = $response.RawContent.ToLower()  
 $connectionFound = $rawHeaders -match "connection: keep-alive"  
 $keepAliveFound = $rawHeaders -match "keep-alive:"  
  
 Write-Host "Verifying if disallowed headers are present in the response..." -ForegroundColor Cyan  
 if ($connectionFound -and $keepAliveFound) {  
 Write-Host "SUCCESS: 'Connection' and 'Keep-Alive' headers were detected in the HTTP response." -ForegroundColor Green  
 Write-Host "This indicates that the mock adapter (or upstream app) is returning raw headers as expected." -ForegroundColor Green  
 Write-Host "In production, these headers would violate the HTTP/2 spec unless sanitized." -ForegroundColor Yellow  
   
 # Show what sanitized headers would look like  
 Write-Host ""  
 Write-Host "EXAMPLE: Sanitized Headers (what they should look like after sanitization):" -ForegroundColor Yellow  
 $headerLines = $response.RawContent -split "`r`n"  
 $sanitizedHeaders = @()  
 $disallowedHeaders = @("connection:", "keep-alive:")  
   
 foreach ($line in $headerLines) {  
 $isDisallowed = $false  
 foreach ($header in $disallowedHeaders) {  
 if ($line.ToLower().StartsWith($header)) {  
 $isDisallowed = $true  
 break  
 }  
 }  
   
 if (-not $isDisallowed) {  
 $sanitizedHeaders += $line  
 }  
 }  
   
 Write-Host ($sanitizedHeaders -join "`r`n") -ForegroundColor Gray  
 } else {  
 Write-Host "WARNING: One or both disallowed headers are missing:" -ForegroundColor Yellow  
 Write-Host "- Connection header found: $connectionFound"   
 Write-Host "- Keep-Alive header found: $keepAliveFound"   
 Write-Host ""  
 Write-Host "If you're testing the unpatched adapter, this might indicate an error." -ForegroundColor Yellow  
 Write-Host "If you're testing the patched version, this is expected behavior (headers are being sanitized)." -ForegroundColor Green  
 }  
} else {  
 Write-Host "Could not verify headers - no response received." -ForegroundColor Red  
}  
  
# Step 5: Clean up  
Write-Host "Stopping adapter background job..." -ForegroundColor Blue  
try {  
 Stop-Job -Job $job -ErrorAction SilentlyContinue  
 Remove-Job -Job $job -Force -ErrorAction SilentlyContinue  
   
 # Also try to kill any remaining processes  
 Get-Process -Name "aws-lambda-web-adapter" -ErrorAction SilentlyContinue | Stop-Process -Force  
   
 Write-Host "Adapter server stopped." -ForegroundColor Green  
} catch {  
 Write-Host "Warning: Could not stop adapter process: $\_" -ForegroundColor Yellow  
 Write-Host "You may need to terminate it manually." -ForegroundColor Yellow  
}  
  
Write-Host "Test completed!" -ForegroundColor Cyan  
Write-Host ""  
Write-Host "Summary: This test confirms whether the adapter is sanitizing HTTP/1.1 headers." -ForegroundColor White   
Write-Host "- For testing purposes, the original adapter should output the restricted headers." -ForegroundColor White  
Write-Host "- In production with the patched Layer, these headers will be sanitized." -ForegroundColor White

## 4-test-adapter-with-flask.ps1

# PowerShell script to test adapter stripping headers with Flask  
# Purpose: Test if sanitization works with a real upstream server  
$ErrorActionPreference = "Stop"  
  
Write-Host "Starting Lambda Web Adapter Flask integration test..." -ForegroundColor Cyan  
  
# Setup paths  
$projectRoot = Get-Location  
$adapterDir = Join-Path $projectRoot "aws-lambda-web-adapter"  
$cmdDir = Join-Path $adapterDir "cmd/aws-lambda-web-adapter"  
$ec2App = Join-Path $projectRoot "ec2.py"  
$testDir = Join-Path $env:TEMP "adapter-test"  
  
Write-Host "Using the following paths:" -ForegroundColor Blue  
Write-Host "- Project root: $projectRoot"  
Write-Host "- Adapter directory: $adapterDir"  
Write-Host "- Flask app path: $ec2App"  
Write-Host "- Test directory: $testDir"  
  
# Step 1: Clean up previous processes  
Write-Host "Cleaning up any previous processes..." -ForegroundColor Blue  
$flaskJob = $null  
$adapterJob = $null  
  
# Kill any existing Python processes that might be running the Flask app  
Get-Process -Name "python" -ErrorAction SilentlyContinue | Stop-Process -Force -ErrorAction SilentlyContinue  
# Kill any adapter test processes  
Get-Process -Name "aws-lambda-web-adapter" -ErrorAction SilentlyContinue | Stop-Process -Force -ErrorAction SilentlyContinue  
Start-Sleep -Seconds 1  
  
# Step 2: Start Flask app  
Write-Host "Starting Flask app on port 5000..." -ForegroundColor Blue  
try {  
 # Check if Python is available  
 $pythonVersion = & python --version 2>&1  
 Write-Host "Using $pythonVersion" -ForegroundColor Green  
   
 # Start Flask in background using PowerShell job  
 $flaskJob = Start-Job -ScriptBlock {  
 param($script)  
 & python $script  
 } -ArgumentList $ec2App  
   
 # Wait for Flask to start  
 Write-Host "Waiting for Flask to start..." -ForegroundColor Yellow  
 Start-Sleep -Seconds 3  
} catch {  
 Write-Host "Error starting Flask app: $\_" -ForegroundColor Red  
 Write-Host "Make sure Python and Flask are installed." -ForegroundColor Yellow  
 exit 1  
}  
  
# Step 3: Create Go adapter test app with a way to capture headers  
Write-Host "Creating Go test adapter with header logging..." -ForegroundColor Blue  
  
# Create or clean test directory  
if (Test-Path $testDir) {  
 Remove-Item $testDir -Recurse -Force  
}  
New-Item -Path $testDir -ItemType Directory -Force | Out-Null  
Set-Location $testDir  
  
# Create Go module  
@"  
module adaptertest  
go 1.18  
"@ | Set-Content "go.mod"  
  
# Create main.go with header sanitization and logging  
@"  
package main  
  
import (  
 "fmt"  
 "io"  
 "log"  
 "net/http"  
 "os"  
 "strings"  
)  
  
// List of disallowed HTTP/2 headers  
var disallowedHeaders = []string{  
 "connection",  
 "keep-alive",  
 "proxy-connection",  
 "transfer-encoding",  
 "upgrade",  
}  
  
// sanitizeHeaders removes disallowed headers  
func sanitizeHeaders(header http.Header) {  
 for \_, name := range disallowedHeaders {  
 header.Del(name)  
 }  
}  
  
// writeHeadersToFile writes headers to a file  
func writeHeadersToFile(filename string, headers http.Header) {  
 file, err := os.Create(filename)  
 if err != nil {  
 log.Printf("Error creating file %s: %v", filename, err)  
 return  
 }  
 defer file.Close()  
  
 for k, v := range headers {  
 fmt.Fprintf(file, "%s: %s\n", k, strings.Join(v, ", "))  
 }  
}  
  
func main() {  
 log.Println("Starting header sanitization test on :8080")  
 log.Println("Proxying to Flask server on http://localhost:5000")  
  
 http.HandleFunc("/", func(w http.ResponseWriter, r \*http.Request) {  
 // Forward to Flask  
 resp, err := http.Get("http://localhost:5000/")  
 if err != nil {  
 http.Error(w, "Failed to reach Flask", http.StatusBadGateway)  
 log.Printf("Error: %v", err)  
 return  
 }  
 defer resp.Body.Close()  
  
 // Make a copy of original headers for comparison  
 originalHeaders := make(http.Header)  
 for k, v := range resp.Header {  
 originalHeaders[k] = v  
 }  
  
 // Copy all headers to our response  
 for k, v := range resp.Header {  
 for \_, vv := range v {  
 w.Header().Add(k, vv)  
 }  
 }  
   
 // Log original headers  
 fmt.Println("\nORIGINAL HEADERS FROM FLASK:")  
 for k, v := range resp.Header {  
 fmt.Printf(" %s: %s\n", k, strings.Join(v, ", "))  
 }  
   
 // Write original headers to file for PowerShell to read  
 writeHeadersToFile("original\_headers.txt", originalHeaders)  
   
 // Apply sanitization  
 sanitizeHeaders(w.Header())  
   
 // Log sanitized headers  
 fmt.Println("\nSANITIZED HEADERS BEING RETURNED:")  
 for k, v := range w.Header() {  
 fmt.Printf(" %s: %s\n", k, strings.Join(v, ", "))  
 }  
   
 // Write sanitized headers to file for PowerShell to read  
 writeHeadersToFile("sanitized\_headers.txt", w.Header())  
  
 // Return response  
 w.WriteHeader(resp.StatusCode)  
 io.Copy(w, resp.Body)  
 })  
  
 log.Fatal(http.ListenAndServe(":8080", nil))  
}  
"@ | Set-Content "main.go"  
  
# Step 4: Build and run adapter test  
Write-Host "Building Go test adapter..." -ForegroundColor Blue  
try {  
 # Check if Go is available  
 $goVersion = & go version  
 Write-Host "Using $goVersion" -ForegroundColor Green  
   
 # Build for Windows (default)  
 go build -o adapter-test.exe  
   
 if (!(Test-Path "adapter-test.exe")) {  
 throw "Failed to build adapter-test.exe"  
 }  
   
 Write-Host "Running adapter on port 8080..." -ForegroundColor Blue  
 $adapterJob = Start-Job -ScriptBlock {  
 param($dir)  
 Set-Location $dir  
 ./adapter-test.exe  
 } -ArgumentList $testDir  
   
 # Wait for adapter to start  
 Start-Sleep -Seconds 2  
} catch {  
 Write-Host "Error building or starting adapter: $\_" -ForegroundColor Red  
   
 # Try to clean up Flask job if it exists  
 if ($flaskJob) {  
 Stop-Job -Job $flaskJob -ErrorAction SilentlyContinue  
 Remove-Job -Job $flaskJob -Force -ErrorAction SilentlyContinue  
 }  
   
 Write-Host "Failed to build or start Go adapter. Test aborted." -ForegroundColor Red  
 Set-Location $projectRoot  
 exit 1  
}  
  
# Step 5: Test the header sanitization  
Write-Host "Sending request to test header sanitization..." -ForegroundColor Green  
try {  
 $originalHeadersFile = Join-Path $testDir "original\_headers.txt"  
 $sanitizedHeadersFile = Join-Path $testDir "sanitized\_headers.txt"  
   
 # Clear any existing files  
 if (Test-Path $originalHeadersFile) { Remove-Item $originalHeadersFile -Force }  
 if (Test-Path $sanitizedHeadersFile) { Remove-Item $sanitizedHeadersFile -Force }  
   
 # Send request to trigger header logging  
 $response = Invoke-WebRequest -Uri "http://localhost:8080/" -UseBasicParsing  
   
 # Wait a moment for files to be written  
 Start-Sleep -Seconds 1  
   
 # Step 6: Show before and after headers  
 Write-Host ""   
 Write-Host "COMPARING HEADERS BEFORE AND AFTER SANITIZATION:" -ForegroundColor Yellow  
 Write-Host "------------------------------------------------" -ForegroundColor DarkGray  
   
 if (Test-Path $originalHeadersFile) {  
 $originalHeaders = Get-Content $originalHeadersFile  
 Write-Host "ORIGINAL HEADERS FROM FLASK:" -ForegroundColor Magenta  
 foreach ($line in $originalHeaders) {  
 # Highlight disallowed headers  
 if ($line -match "^(Connection|Keep-Alive|Proxy-Connection|Transfer-Encoding|Upgrade):") {  
 Write-Host $line -ForegroundColor Red  
 } else {  
 Write-Host $line -ForegroundColor Gray  
 }  
 }  
 } else {  
 Write-Host "Could not read original headers file." -ForegroundColor Red  
 }  
   
 Write-Host ""   
   
 if (Test-Path $sanitizedHeadersFile) {  
 $sanitizedHeaders = Get-Content $sanitizedHeadersFile  
 Write-Host "SANITIZED HEADERS RETURNED TO CLIENT:" -ForegroundColor Green  
 foreach ($line in $sanitizedHeaders) {  
 Write-Host $line -ForegroundColor Gray  
 }  
 } else {  
 Write-Host "Could not read sanitized headers file." -ForegroundColor Red  
 }  
   
 Write-Host "------------------------------------------------" -ForegroundColor DarkGray  
   
 # Step 7: Analyze headers  
 Write-Host ""   
 Write-Host "HEADER SANITIZATION ANALYSIS:" -ForegroundColor Yellow  
   
 $connectionFound = $false  
 $keepAliveFound = $false  
   
 if ($originalHeaders) {  
 foreach ($line in $originalHeaders) {  
 if ($line -match "^Connection:") { $connectionFound = $true }  
 if ($line -match "^Keep-Alive:") { $keepAliveFound = $true }  
 }  
 }  
   
 $sanitizedConnectionFound = $false  
 $sanitizedKeepAliveFound = $false  
   
 if ($sanitizedHeaders) {  
 foreach ($line in $sanitizedHeaders) {  
 if ($line -match "^Connection:") { $sanitizedConnectionFound = $true }  
 if ($line -match "^Keep-Alive:") { $sanitizedKeepAliveFound = $true }  
 }  
 }  
   
 # Report findings  
 if ($connectionFound -or $keepAliveFound) {  
 Write-Host "Original headers contained disallowed headers:" -ForegroundColor Blue  
 if ($connectionFound) { Write-Host "- Connection header found in original" -ForegroundColor Blue }  
 if ($keepAliveFound) { Write-Host "- Keep-Alive header found in original" -ForegroundColor Blue }  
 } else {  
 Write-Host "Note: Original headers didn't contain disallowed headers." -ForegroundColor Yellow  
 Write-Host "This might indicate that Flask isn't sending the expected headers." -ForegroundColor Yellow  
 }  
   
 Write-Host ""  
   
 if ($sanitizedConnectionFound -or $sanitizedKeepAliveFound) {  
 Write-Host "❌ FAIL: Disallowed headers are still present after sanitization!" -ForegroundColor Red  
 if ($sanitizedConnectionFound) { Write-Host "- Connection header still present" -ForegroundColor Red }  
 if ($sanitizedKeepAliveFound) { Write-Host "- Keep-Alive header still present" -ForegroundColor Red }  
 Write-Host "The sanitization code is NOT properly removing headers." -ForegroundColor Red  
 } else {  
 if ($connectionFound -or $keepAliveFound) {  
 Write-Host "✅ SUCCESS: Disallowed headers were properly sanitized!" -ForegroundColor Green  
 Write-Host "The header sanitization code is working correctly." -ForegroundColor Green  
 Write-Host "This confirms our Lambda Layer will correctly sanitize headers for HTTP/2 compatibility." -ForegroundColor Green  
 } else {  
 Write-Host "⚠️ INDETERMINATE: No disallowed headers were present to sanitize." -ForegroundColor Yellow  
 Write-Host "Please check that ec2.py is configured to send Connection and Keep-Alive headers." -ForegroundColor Yellow  
 }  
 }  
} catch {  
 Write-Host "Error testing adapter: $\_" -ForegroundColor Red  
 Write-Host "Make sure both Flask and the adapter are running." -ForegroundColor Yellow  
}  
  
# Step 8: Clean up  
Write-Host ""  
Write-Host "Cleaning up..." -ForegroundColor Blue  
  
# Stop jobs  
if ($flaskJob) {  
 Stop-Job -Job $flaskJob -ErrorAction SilentlyContinue  
 Remove-Job -Job $flaskJob -Force -ErrorAction SilentlyContinue  
}  
  
if ($adapterJob) {  
 Stop-Job -Job $adapterJob -ErrorAction SilentlyContinue  
 Remove-Job -Job $adapterJob -Force -ErrorAction SilentlyContinue  
}  
  
# Kill any remaining processes  
Get-Process -Name "python" -ErrorAction SilentlyContinue | Stop-Process -Force -ErrorAction SilentlyContinue  
Get-Process -Name "adapter-test" -ErrorAction SilentlyContinue | Stop-Process -Force -ErrorAction SilentlyContinue  
  
# Return to original directory  
Set-Location $projectRoot  
  
Write-Host ""  
Write-Host "Test completed - results show if the sanitization code is working correctly." -ForegroundColor Cyan  
Write-Host "If successful, you can now proceed with building the Lambda Layer." -ForegroundColor Cyan

## 5-alb-lambda-http2-header-sanitization-test.yaml

# AWSTemplateFormatVersion: '2010-09-09'  
Transform: AWS::Serverless-2016-10-31  
Description: |  
 Lambda HTTP/2 Header Sanitization Test Environment  
 Tests HTTP/2 header sanitization using AWS Lambda Web Adapter with a Python wrapper  
  
Parameters:  
 KeyName:  
 Type: AWS::EC2::KeyPair::KeyName  
 Description: SSH key name for EC2 access  
  
 VpcCidr:  
 Type: String  
 Description: CIDR block for the VPC  
 Default: 10.0.0.0/16  
  
 CertificateArn:  
 Type: String  
 Description: ARN of an ACM certificate for HTTPS (required for HTTP/2)  
 Default: '' # Optional for testing with just HTTP  
  
Globals:  
 Function:  
 Timeout: 30  
 Runtime: python3.9  
 Architectures: [x86\_64]  
 MemorySize: 256  
 Tags:  
 Project: HTTP2HeaderSanitization  
 Environment: Test  
  
Resources:  
 #--------------------------  
 # VPC Resources  
 #--------------------------  
 VPC:  
 Type: AWS::EC2::VPC  
 Properties:  
 CidrBlock: !Ref VpcCidr  
 EnableDnsSupport: true  
 EnableDnsHostnames: true  
 Tags:  
 - Key: Name  
 Value: !Sub ${AWS::StackName}-vpc  
 - Key: Project  
 Value: HTTP2HeaderSanitization  
  
 InternetGateway:  
 Type: AWS::EC2::InternetGateway  
 Properties:  
 Tags:  
 - Key: Name  
 Value: !Sub ${AWS::StackName}-igw  
 - Key: Project  
 Value: HTTP2HeaderSanitization  
  
 InternetGatewayAttachment:  
 Type: AWS::EC2::VPCGatewayAttachment  
 Properties:  
 InternetGatewayId: !Ref InternetGateway  
 VpcId: !Ref VPC  
  
 PublicSubnet1:  
 Type: AWS::EC2::Subnet  
 Properties:  
 VpcId: !Ref VPC  
 AvailabilityZone: !Select [0, !GetAZs '']  
 CidrBlock: !Select [0, !Cidr [!Ref VpcCidr, 4, 8]]  
 MapPublicIpOnLaunch: true  
 Tags:  
 - Key: Name  
 Value: !Sub ${AWS::StackName}-public-subnet-1  
 - Key: Project  
 Value: HTTP2HeaderSanitization  
  
 PublicSubnet2:  
 Type: AWS::EC2::Subnet  
 Properties:  
 VpcId: !Ref VPC  
 AvailabilityZone: !Select [1, !GetAZs '']  
 CidrBlock: !Select [1, !Cidr [!Ref VpcCidr, 4, 8]]  
 MapPublicIpOnLaunch: true  
 Tags:  
 - Key: Name  
 Value: !Sub ${AWS::StackName}-public-subnet-2  
 - Key: Project  
 Value: HTTP2HeaderSanitization  
  
 PublicRouteTable:  
 Type: AWS::EC2::RouteTable  
 Properties:  
 VpcId: !Ref VPC  
 Tags:  
 - Key: Name  
 Value: !Sub ${AWS::StackName}-public-route-table  
 - Key: Project  
 Value: HTTP2HeaderSanitization  
  
 DefaultPublicRoute:  
 Type: AWS::EC2::Route  
 DependsOn: InternetGatewayAttachment  
 Properties:  
 RouteTableId: !Ref PublicRouteTable  
 DestinationCidrBlock: 0.0.0.0/0  
 GatewayId: !Ref InternetGateway  
  
 PublicSubnet1RouteTableAssociation:  
 Type: AWS::EC2::SubnetRouteTableAssociation  
 Properties:  
 RouteTableId: !Ref PublicRouteTable  
 SubnetId: !Ref PublicSubnet1  
  
 PublicSubnet2RouteTableAssociation:  
 Type: AWS::EC2::SubnetRouteTableAssociation  
 Properties:  
 RouteTableId: !Ref PublicRouteTable  
 SubnetId: !Ref PublicSubnet2  
  
 #--------------------------  
 # Security Groups  
 #--------------------------  
 ALBSecurityGroup:  
 Type: AWS::EC2::SecurityGroup  
 Properties:  
 GroupDescription: Security group for ALB  
 VpcId: !Ref VPC  
 SecurityGroupIngress:  
 - IpProtocol: tcp  
 FromPort: 80  
 ToPort: 80  
 CidrIp: 0.0.0.0/0  
 - IpProtocol: tcp  
 FromPort: 443  
 ToPort: 443  
 CidrIp: 0.0.0.0/0  
 Tags:  
 - Key: Name  
 Value: !Sub ${AWS::StackName}-alb-sg  
 - Key: Project  
 Value: HTTP2HeaderSanitization  
  
 EC2SecurityGroup:  
 Type: AWS::EC2::SecurityGroup  
 Properties:  
 GroupDescription: Security group for EC2 instance  
 VpcId: !Ref VPC  
 SecurityGroupIngress:  
 - IpProtocol: tcp  
 FromPort: 22  
 ToPort: 22  
 CidrIp: 0.0.0.0/0  
 - IpProtocol: tcp  
 FromPort: 5000  
 ToPort: 5000  
 SourceSecurityGroupId: !Ref ALBSecurityGroup  
 Tags:  
 - Key: Name  
 Value: !Sub ${AWS::StackName}-ec2-sg  
 - Key: Project  
 Value: HTTP2HeaderSanitization  
  
 LambdaSecurityGroup:  
 Type: AWS::EC2::SecurityGroup  
 Properties:  
 GroupDescription: Security group for Lambda functions  
 VpcId: !Ref VPC  
 SecurityGroupEgress:  
 - IpProtocol: -1  
 CidrIp: 0.0.0.0/0  
 Tags:  
 - Key: Name  
 Value: !Sub ${AWS::StackName}-lambda-sg  
 - Key: Project  
 Value: HTTP2HeaderSanitization  
  
 #--------------------------  
 # EC2 Instance  
 #--------------------------  
 FlaskInstance:  
 Type: AWS::EC2::Instance  
 Properties:  
 ImageId: ami-067d1e60475437da2 # Amazon Linux 2023 (us-east-1)  
 InstanceType: t3.micro  
 KeyName: !Ref KeyName  
 NetworkInterfaces:  
 - AssociatePublicIpAddress: true  
 DeviceIndex: 0  
 GroupSet:  
 - !Ref EC2SecurityGroup  
 SubnetId: !Ref PublicSubnet1  
 UserData:  
 Fn::Base64: !Sub |  
 #!/bin/bash -xe  
 # Update system packages  
 yum update -y  
 yum install -y python3 python3-pip telnet nc  
 pip3 install flask gunicorn  
   
 # Create Flask application  
 mkdir -p /home/ec2-user/app  
 cat > /home/ec2-user/app/app.py << 'EOL'  
 from flask import Flask, Response  
  
 app = Flask(\_\_name\_\_)  
  
 @app.route("/")  
 @app.route("/ec2") # Add this route to match ALB path exactly  
 def root():  
 return Response("Successful request to EC2 (python)",  
 headers={"Connection": "keep-alive", "Keep-Alive": "timeout=72"},  
 mimetype="text/plain")  
 EOL  
   
 # Create a simple test file to verify Flask is working  
 cat > /home/ec2-user/app/test.py << 'EOL'  
 from app import app  
  
 if \_\_name\_\_ == "\_\_main\_\_":  
 app.run(host='0.0.0.0', port=5000, debug=True)  
 EOL  
   
 # Create systemd service file  
 cat > /etc/systemd/system/flask-app.service << 'EOL'  
 [Unit]  
 Description=Flask Application  
 After=network.target  
  
 [Service]  
 User=ec2-user  
 WorkingDirectory=/home/ec2-user/app  
 ExecStart=/usr/local/bin/gunicorn --bind 0.0.0.0:5000 app:app  
 Restart=always  
  
 [Install]  
 WantedBy=multi-user.target  
 EOL  
   
 # Set correct permissions  
 chown -R ec2-user:ec2-user /home/ec2-user/app  
   
 # Start Flask app directly first to ensure it works  
 cd /home/ec2-user/app  
 python3 test.py > /tmp/flask-test.log 2>&1 &  
   
 # Wait a few seconds and then kill the test process  
 sleep 5  
 pkill -f test.py  
   
 # Start using systemd  
 systemctl daemon-reload  
 systemctl enable flask-app  
 systemctl start flask-app  
   
 # Verify Flask is running  
 curl -s http://localhost:5000/ > /tmp/flask-curl-test.log  
 curl -s http://localhost:5000/ec2 >> /tmp/flask-curl-test.log  
   
 # Create a verification file  
 cat > /home/ec2-user/verify.sh << 'EOL'  
 #!/bin/bash  
 echo "Flask Service Status:"  
 systemctl status flask-app  
 echo ""  
 echo "Port 5000 Listening:"  
 netstat -tunlp | grep 5000  
 echo ""  
 echo "Curl Test Root Path:"  
 curl -v http://localhost:5000/  
 echo ""  
 echo "Curl Test EC2 Path:"  
 curl -v http://localhost:5000/ec2  
 EOL  
   
 chmod +x /home/ec2-user/verify.sh  
 chown ec2-user:ec2-user /home/ec2-user/verify.sh  
   
 # Indicate successful completion  
 echo "User data script completed successfully" > /tmp/userdata-success  
 Tags:  
 - Key: Name  
 Value: !Sub ${AWS::StackName}-flask-instance  
 - Key: Project  
 Value: HTTP2HeaderSanitization  
  
 #--------------------------  
 # Lambda Function Resources  
 #--------------------------  
 LambdaExecutionRole:  
 Type: AWS::IAM::Role  
 Properties:  
 AssumeRolePolicyDocument:  
 Version: '2012-10-17'  
 Statement:  
 - Effect: Allow  
 Principal:  
 Service: lambda.amazonaws.com  
 Action: sts:AssumeRole  
 ManagedPolicyArns:  
 - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole  
 - arn:aws:iam::aws:policy/service-role/AWSLambdaVPCAccessExecutionRole  
 Tags:  
 - Key: Project  
 Value: HTTP2HeaderSanitization  
  
 # Vanilla Lambda (without adapter)  
 VanillaLambda:  
 Type: AWS::Serverless::Function  
 Properties:  
 FunctionName: HTTP2TestVanillaLambda  
 InlineCode: |  
 import json  
 import logging  
  
 # Configure logging  
 logger = logging.getLogger()  
 logger.setLevel(logging.INFO)  
  
 def handler(event, context):  
 try:  
 logger.info("Processing request to Vanilla Lambda")  
 logger.info(f"Event: {json.dumps(event)}")  
   
 # Simplify query string handling to avoid None errors  
 query\_params = event.get("queryStringParameters") or {}  
 connection = query\_params.get("connection", "true")  
 keep\_alive = query\_params.get("keep-alive", "true")  
   
 headers = {  
 "Content-Type": "text/plain"  
 }  
   
 # Add problematic HTTP/1.1 headers to demonstrate the issue  
 if connection == "true":   
 headers["Connection"] = "keep-alive"  
 if keep\_alive == "true":   
 headers["Keep-Alive"] = "timeout=72"  
   
 logger.info(f"Returning headers: {json.dumps(headers)}")  
 return {  
 "statusCode": 200,  
 "headers": headers,  
 "body": "Vanilla Lambda - with HTTP/1.1 headers (should fail with HTTP/2)"  
 }  
 except Exception as e:  
 logger.error(f"Error processing request: {str(e)}")  
 return {  
 "statusCode": 500,  
 "headers": {"Content-Type": "application/json"},  
 "body": json.dumps({"error": str(e)})  
 }  
 Handler: index.handler  
 Role: !GetAtt LambdaExecutionRole.Arn  
 VpcConfig:  
 SecurityGroupIds:  
 - !Ref LambdaSecurityGroup  
 SubnetIds:  
 - !Ref PublicSubnet1  
 - !Ref PublicSubnet2  
  
 # Patched Lambda (with AWS adapter and header sanitization wrapper)  
 PatchedLambda:  
 Type: AWS::Serverless::Function  
 Properties:  
 FunctionName: HTTP2TestPatchedLambda  
 InlineCode: |  
 import json  
 import logging  
 import os  
  
 # Configure logging  
 logger = logging.getLogger()  
 logger.setLevel(logging.INFO)  
  
 def handler(event, context):  
 """Lambda handler with HTTP/2 header sanitization wrapper"""  
 try:  
 logger.info("Processing request to Patched Lambda")  
 logger.info(f"Event: {json.dumps(event)}")  
 logger.info(f"Environment: {os.environ}")  
   
 # Simplify query string handling to avoid None errors  
 query\_params = event.get("queryStringParameters") or {}  
 connection = query\_params.get("connection", "true")  
 keep\_alive = query\_params.get("keep-alive", "true")  
   
 headers = {  
 "Content-Type": "text/plain"  
 }  
   
 # Add problematic HTTP/1.1 headers that should be stripped  
 if connection == "true":   
 headers["Connection"] = "keep-alive"  
 if keep\_alive == "true":   
 headers["Keep-Alive"] = "timeout=72"  
   
 # Create the response  
 response = {  
 "statusCode": 200,  
 "headers": headers,  
 "body": "Patched Lambda - with sanitized HTTP/1.1 headers (works with HTTP/2)"  
 }  
   
 # Apply header sanitization before returning  
 sanitized\_response = sanitize\_http2\_headers(response)  
 logger.info(f"Sanitized response headers: {json.dumps(sanitized\_response['headers'])}")  
 return sanitized\_response  
   
 except Exception as e:  
 logger.error(f"Error processing request: {str(e)}")  
 return {  
 "statusCode": 500,  
 "headers": {"Content-Type": "application/json"},  
 "body": json.dumps({"error": str(e)})  
 }  
  
 def sanitize\_http2\_headers(response):  
 """Sanitize HTTP/2 disallowed headers"""  
 # List of disallowed headers in HTTP/2  
 disallowed\_headers = [  
 "connection",  
 "keep-alive",   
 "proxy-connection",  
 "transfer-encoding",  
 "upgrade"  
 ]  
   
 # Remove disallowed headers (case-insensitive)  
 if "headers" in response and response["headers"]:  
 sanitized\_headers = {}  
 for header\_name, header\_value in response["headers"].items():  
 if header\_name.lower() not in disallowed\_headers:  
 sanitized\_headers[header\_name] = header\_value  
   
 # Replace headers with sanitized version  
 response["headers"] = sanitized\_headers  
   
 return response  
 Handler: index.handler  
 Role: !GetAtt LambdaExecutionRole.Arn  
 Layers:  
 - arn:aws:lambda:us-east-1:753240598075:layer:LambdaAdapterLayerX86:17  
 Environment:  
 Variables:  
 AWS\_LAMBDA\_WEB\_ADAPTER\_BINDING\_ID: default  
 VpcConfig:  
 SecurityGroupIds:  
 - !Ref LambdaSecurityGroup  
 SubnetIds:  
 - !Ref PublicSubnet1  
 - !Ref PublicSubnet2  
  
 #--------------------------  
 # CloudWatch Alarms  
 #--------------------------  
 VanillaLambdaErrorAlarm:  
 Type: AWS::CloudWatch::Alarm  
 Properties:  
 AlarmDescription: Alarm if vanilla Lambda has too many errors  
 Namespace: AWS/Lambda  
 MetricName: Errors  
 Dimensions:  
 - Name: FunctionName  
 Value: !Ref VanillaLambda  
 Statistic: Sum  
 Period: 60  
 EvaluationPeriods: 1  
 Threshold: 3  
 ComparisonOperator: GreaterThanOrEqualToThreshold  
 TreatMissingData: notBreaching  
 Tags:  
 - Key: Project  
 Value: HTTP2HeaderSanitization  
  
 PatchedLambdaErrorAlarm:  
 Type: AWS::CloudWatch::Alarm  
 Properties:  
 AlarmDescription: Alarm if patched Lambda has too many errors  
 Namespace: AWS/Lambda  
 MetricName: Errors  
 Dimensions:  
 - Name: FunctionName  
 Value: !Ref PatchedLambda  
 Statistic: Sum  
 Period: 60  
 EvaluationPeriods: 1  
 Threshold: 3  
 ComparisonOperator: GreaterThanOrEqualToThreshold  
 TreatMissingData: notBreaching  
 Tags:  
 - Key: Project  
 Value: HTTP2HeaderSanitization  
  
 #--------------------------  
 # ALB Resources  
 #--------------------------  
 ApplicationLoadBalancer:  
 Type: AWS::ElasticLoadBalancingV2::LoadBalancer  
 Properties:  
 Type: application  
 Scheme: internet-facing  
 SecurityGroups:  
 - !Ref ALBSecurityGroup  
 Subnets:  
 - !Ref PublicSubnet1  
 - !Ref PublicSubnet2  
 LoadBalancerAttributes:  
 - Key: idle\_timeout.timeout\_seconds  
 Value: '60'  
 - Key: routing.http2.enabled  
 Value: 'true'  
 Tags:  
 - Key: Name  
 Value: !Sub ${AWS::StackName}-alb  
 - Key: Project  
 Value: HTTP2HeaderSanitization  
  
 # HTTP Listener  
 HTTPListener:  
 Type: AWS::ElasticLoadBalancingV2::Listener  
 Properties:  
 DefaultActions:  
 - Type: fixed-response  
 FixedResponseConfig:  
 ContentType: text/plain  
 StatusCode: 200  
 MessageBody: "Default ALB response - use /vanilla, /patched, or /ec2 paths"  
 LoadBalancerArn: !Ref ApplicationLoadBalancer  
 Port: 80  
 Protocol: HTTP  
  
 # HTTPS Listener (for HTTP/2)  
 HTTPSListener:  
 Type: AWS::ElasticLoadBalancingV2::Listener  
 Condition: HasCertificate  
 Properties:  
 DefaultActions:  
 - Type: fixed-response  
 FixedResponseConfig:  
 ContentType: text/plain  
 StatusCode: 200  
 MessageBody: "Default ALB response - use /vanilla, /patched, or /ec2 paths"  
 LoadBalancerArn: !Ref ApplicationLoadBalancer  
 Port: 443  
 Protocol: HTTPS  
 SslPolicy: ELBSecurityPolicy-TLS13-1-2-2021-06  
 Certificates:  
 - CertificateArn: !Ref CertificateArn  
  
 # Vanilla Lambda Target Group  
 VanillaLambdaTargetGroup:  
 Type: AWS::ElasticLoadBalancingV2::TargetGroup  
 Properties:  
 TargetType: lambda  
 Targets:  
 - Id: !GetAtt VanillaLambda.Arn  
 Tags:  
 - Key: Name  
 Value: !Sub ${AWS::StackName}-vanilla-lambda-tg  
 - Key: Project  
 Value: HTTP2HeaderSanitization  
  
 # Patched Lambda Target Group  
 PatchedLambdaTargetGroup:  
 Type: AWS::ElasticLoadBalancingV2::TargetGroup  
 Properties:  
 TargetType: lambda  
 Targets:  
 - Id: !GetAtt PatchedLambda.Arn  
 Tags:  
 - Key: Name  
 Value: !Sub ${AWS::StackName}-patched-lambda-tg  
 - Key: Project  
 Value: HTTP2HeaderSanitization  
  
 # EC2 Target Group  
 EC2TargetGroup:  
 Type: AWS::ElasticLoadBalancingV2::TargetGroup  
 Properties:  
 Protocol: HTTP  
 Port: 5000  
 TargetType: ip  
 VpcId: !Ref VPC  
 HealthCheckPath: /  
 HealthCheckProtocol: HTTP  
 HealthCheckPort: 5000  
 HealthCheckIntervalSeconds: 30  
 HealthCheckTimeoutSeconds: 5  
 HealthyThresholdCount: 2  
 UnhealthyThresholdCount: 3  
 Matcher:  
 HttpCode: "200"  
 Targets:  
 - Id: !GetAtt FlaskInstance.PrivateIp  
 Port: 5000  
 Tags:  
 - Key: Name  
 Value: !Sub ${AWS::StackName}-ec2-tg  
 - Key: Project  
 Value: HTTP2HeaderSanitization  
  
 # HTTP Listener Rules - With and without trailing slashes  
 VanillaLambdaHTTPListenerRule:  
 Type: AWS::ElasticLoadBalancingV2::ListenerRule  
 Properties:  
 Actions:  
 - Type: forward  
 TargetGroupArn: !Ref VanillaLambdaTargetGroup  
 Conditions:  
 - Field: path-pattern  
 Values:   
 - /vanilla  
 - /vanilla/  
 ListenerArn: !Ref HTTPListener  
 Priority: 10  
  
 PatchedLambdaHTTPListenerRule:  
 Type: AWS::ElasticLoadBalancingV2::ListenerRule  
 Properties:  
 Actions:  
 - Type: forward  
 TargetGroupArn: !Ref PatchedLambdaTargetGroup  
 Conditions:  
 - Field: path-pattern  
 Values:   
 - /patched  
 - /patched/  
 ListenerArn: !Ref HTTPListener  
 Priority: 20  
  
 EC2HTTPListenerRule:  
 Type: AWS::ElasticLoadBalancingV2::ListenerRule  
 Properties:  
 Actions:  
 - Type: forward  
 TargetGroupArn: !Ref EC2TargetGroup  
 Conditions:  
 - Field: path-pattern  
 Values:   
 - /ec2  
 - /ec2/  
 ListenerArn: !Ref HTTPListener  
 Priority: 30  
  
 # HTTPS Listener Rules (HTTP/2) - With and without trailing slashes  
 VanillaLambdaHTTPSListenerRule:  
 Type: AWS::ElasticLoadBalancingV2::ListenerRule  
 Condition: HasCertificate  
 Properties:  
 Actions:  
 - Type: forward  
 TargetGroupArn: !Ref VanillaLambdaTargetGroup  
 Conditions:  
 - Field: path-pattern  
 Values:   
 - /vanilla  
 - /vanilla/  
 ListenerArn: !Ref HTTPSListener  
 Priority: 10  
  
 PatchedLambdaHTTPSListenerRule:  
 Type: AWS::ElasticLoadBalancingV2::ListenerRule  
 Condition: HasCertificate  
 Properties:  
 Actions:  
 - Type: forward  
 TargetGroupArn: !Ref PatchedLambdaTargetGroup  
 Conditions:  
 - Field: path-pattern  
 Values:   
 - /patched  
 - /patched/  
 ListenerArn: !Ref HTTPSListener  
 Priority: 20  
  
 EC2HTTPSListenerRule:  
 Type: AWS::ElasticLoadBalancingV2::ListenerRule  
 Condition: HasCertificate  
 Properties:  
 Actions:  
 - Type: forward  
 TargetGroupArn: !Ref EC2TargetGroup  
 Conditions:  
 - Field: path-pattern  
 Values:   
 - /ec2  
 - /ec2/  
 ListenerArn: !Ref HTTPSListener  
 Priority: 30  
  
 # Lambda Permission for ALB  
 VanillaLambdaPermission:  
 Type: AWS::Lambda::Permission  
 Properties:  
 Action: lambda:InvokeFunction  
 FunctionName: !GetAtt VanillaLambda.Arn  
 Principal: elasticloadbalancing.amazonaws.com  
  
 PatchedLambdaPermission:  
 Type: AWS::Lambda::Permission  
 Properties:  
 Action: lambda:InvokeFunction  
 FunctionName: !GetAtt PatchedLambda.Arn  
 Principal: elasticloadbalancing.amazonaws.com  
  
Conditions:  
 HasCertificate: !Not [!Equals [!Ref CertificateArn, '']]  
  
Outputs:  
 ALBDNSName:  
 Description: DNS Name of the ALB  
 Value: !GetAtt ApplicationLoadBalancer.DNSName  
  
 HttpEndpoints:  
 Description: HTTP Endpoints (no HTTP/2)  
 Value: !Sub |  
 Vanilla Lambda: http://${ApplicationLoadBalancer.DNSName}/vanilla  
 Patched Lambda: http://${ApplicationLoadBalancer.DNSName}/patched  
 EC2 Instance: http://${ApplicationLoadBalancer.DNSName}/ec2  
  
 HttpsEndpoints:  
 Description: HTTPS Endpoints (HTTP/2 enabled)  
 Condition: HasCertificate  
 Value: !Sub |  
 Vanilla Lambda: https://${ApplicationLoadBalancer.DNSName}/vanilla  
 Patched Lambda: https://${ApplicationLoadBalancer.DNSName}/patched  
 EC2 Instance: https://${ApplicationLoadBalancer.DNSName}/ec2  
  
 FlaskEC2PublicIP:  
 Description: Public IP of EC2 instance running Flask  
 Value: !GetAtt FlaskInstance.PublicIp  
  
 EC2SSHCommand:  
 Description: SSH command to connect to EC2 instance  
 Value: !Sub "ssh -i ${KeyName}.pem ec2-user@${FlaskInstance.PublicIp}"  
  
 TestInstructions:  
 Description: HTTP/2 Header Sanitization Test Instructions  
 Value: !If  
 - HasCertificate  
 - !Sub |  
 Test Setup:  
 1. The ALB is configured with three paths:  
 - /vanilla - Lambda without header sanitization (should fail with HTTP/2)  
 - /patched - Lambda with sanitization wrapper (should work with HTTP/2)  
 - /ec2 - Flask app on EC2 (ALB handles header sanitization automatically)  
   
 Testing with curl:  
 curl -vk --http2 https://${ApplicationLoadBalancer.DNSName}/vanilla  
 curl -vk --http2 https://${ApplicationLoadBalancer.DNSName}/patched  
 curl -vk --http2 https://${ApplicationLoadBalancer.DNSName}/ec2  
   
 Verifying EC2 Flask app directly:  
 ssh -i ${KeyName}.pem ec2-user@${FlaskInstance.PublicIp}  
 ./verify.sh  
   
 Expected Results:  
 - Vanilla Lambda: Should fail with HTTP/2 due to illegal headers  
 - Patched Lambda: Should work with HTTP/2, headers sanitized by our wrapper  
 - EC2: Should work with HTTP/2, headers sanitized by ALB  
   
 CloudWatch Resources:  
 - Vanilla Lambda logs: /aws/lambda/HTTP2TestVanillaLambda  
 - Patched Lambda logs: /aws/lambda/HTTP2TestPatchedLambda  
 - !Sub |  
 WARNING: No HTTPS certificate provided, HTTP/2 testing not possible.  
 HTTP/2 requires HTTPS. To test HTTP/2, redeploy with a valid certificate.  
   
 You can still test the basic functionality over HTTP (but not HTTP/2):  
 curl -v http://${ApplicationLoadBalancer.DNSName}/vanilla  
 curl -v http://${ApplicationLoadBalancer.DNSName}/patched  
 curl -v http://${ApplicationLoadBalancer.DNSName}/ec2  
   
 Verifying EC2 Flask app directly:  
 ssh -i ${KeyName}.pem ec2-user@${FlaskInstance.PublicIp}  
 ./verify.sh  
   
 CloudWatch Resources:  
 - Vanilla Lambda logs: /aws/lambda/HTTP2TestVanillaLambda  
 - Patched Lambda logs: /aws/lambda/HTTP2TestPatchedLambda

## 6-alb-test-http2-sanitization.ps1

# PowerShell script to test HTTP/2 header sanitization with AWS ALB  
param (  
 [Parameter(Mandatory=$true)]  
 [string]$AlbDnsName,  
   
 [switch]$HideCurlOutput = $false  
)  
  
# Set curl path  
$curlPath = "C:\Program Files\curl-8.13.0\_1-win64-mingw\bin\curl.exe"  
if (-not (Test-Path $curlPath)) {  
 $curlPath = "curl.exe"  
}  
  
$Protocol = "https"  
$Target = "${Protocol}://${AlbDnsName}"  
  
Write-Host "Starting HTTP/2 header sanitization test for $AlbDnsName" -ForegroundColor Cyan  
Write-Host "Using curl: $curlPath" -ForegroundColor Gray  
  
# Check curl HTTP/2 support  
$versionOutput = & "$curlPath" --version  
$hasHttp2 = $versionOutput -match "HTTP2"  
if (-not $hasHttp2) {  
 Write-Host "ERROR: curl.exe does not support HTTP/2 (--http2)" -ForegroundColor Red  
 exit 1  
}  
  
# Check if HTTPS is available  
Write-Host "Checking if HTTPS is available on ALB..." -ForegroundColor Blue  
try {  
 $status = (& "$curlPath" -ks -o NUL -w "%{http\_code}" "$Target")  
 if ($status -eq "200") {  
 Write-Host "HTTPS is available. Using HTTPS for HTTP/2 tests" -ForegroundColor Green  
 } else {  
 Write-Host "HTTPS returned status $status. Falling back to HTTP" -ForegroundColor Yellow  
 $Protocol = "http"  
 }  
} catch {  
 Write-Host "Failed to check HTTPS status. Falling back to HTTP" -ForegroundColor Yellow  
 $Protocol = "http"  
}  
  
function Run-Test {  
 param (  
 [string]$Name,  
 [string]$Path,  
 [bool]$ExpectSuccess = $false,  
 [string]$Description = ""  
 )  
   
 $Url = "${Protocol}://${AlbDnsName}/$Path"  
 $tempFile = [System.IO.Path]::GetTempFileName()  
  
 Write-Host ""  
 Write-Host "=============================" -ForegroundColor Blue  
 Write-Host "TEST: $Name" -ForegroundColor Yellow  
 Write-Host "URL: $Url" -ForegroundColor Yellow  
 if ($Description) {  
 Write-Host "GOAL: " -ForegroundColor Yellow -NoNewline  
 Write-Host "$Description"  
 }  
 Write-Host "=============================" -ForegroundColor Blue  
  
 $cmdDisplay = "curl --http2 -vsk $Url"  
 Write-Host "Running: $cmdDisplay" -ForegroundColor Gray  
   
 # Execute curl and capture output  
 if (-not $HideCurlOutput) {  
 # Capture and display output  
 $output = & "$curlPath" --http2 -vsk $Url 2>&1  
 $output | Out-File $tempFile  
   
 # Display output without errors  
 foreach ($line in $output) {  
 if ($line -notmatch "System.Management.Automation.RemoteException") {  
 Write-Host $line  
 }  
 }  
 } else {  
 # Just capture output without showing it  
 $curlOutput = & "$curlPath" --http2 -vsk $Url 2>&1  
 $curlOutput | Out-File $tempFile  
 }  
   
 # Read file for analysis  
 $curlOutText = Get-Content $tempFile -Raw  
   
 # Key checks for test analysis  
 $isHttp2 = $curlOutText -match "using HTTP/2"  
 $httpStatusMatch = $curlOutText -match "HTTP/[12](?:\.[01])? (\d+)"  
 $httpStatus = if ($httpStatusMatch) { $matches[1] } else { "Unknown" }  
 $hasConnection = $curlOutText -match "(?i)< connection:"  
 $hasKeepAlive = $curlOutText -match "(?i)< keep-alive:"  
 $hasProtocolError = $curlOutText -match "HTTP/2 stream .\* was not closed cleanly: PROTOCOL\_ERROR"  
   
 # Extract content type  
 $contentType = ""  
 if ($curlOutText -match "(?i)< content-type:(.\*)") {  
 $contentType = $matches[1].Trim()  
 }  
  
 # Display results  
 Write-Host ""  
 Write-Host "--- Response Analysis ---" -ForegroundColor Blue  
 Write-Host "HTTP Status: $httpStatus" -ForegroundColor Yellow  
   
 if ($contentType) {  
 Write-Host "Content-Type: < content-type:$contentType" -ForegroundColor Blue  
 }  
   
 # Check headers  
 if ($hasConnection) {  
 Write-Host "X 'Connection' header found in response" -ForegroundColor Red  
 } else {  
 Write-Host "√ No disallowed 'Connection' header found in response" -ForegroundColor Green  
 }  
   
 if ($hasKeepAlive) {  
 Write-Host "X 'Keep-Alive' header found in response" -ForegroundColor Red  
 } else {  
 Write-Host "√ No 'Keep-Alive' header found in response" -ForegroundColor Green  
 }  
   
 # Results determination  
 $success = $false  
   
 if ($ExpectSuccess) {  
 # For sanitized endpoints (patched Lambda, EC2)  
 if ($httpStatus -eq "200" -and -not $hasConnection -and -not $hasKeepAlive -and -not $hasProtocolError) {  
 Write-Host "√ PASS: Headers properly sanitized and response successful" -ForegroundColor Green  
 $success = $true  
 } else {  
 Write-Host "X FAIL: Headers not properly sanitized or response unsuccessful" -ForegroundColor Red  
 }  
 } else {  
 # For vanilla Lambda (expecting non-sanitized headers)  
 if ($hasProtocolError -or $hasConnection -or $hasKeepAlive) {  
 Write-Host "√ PASS: Non-sanitized headers detected as expected" -ForegroundColor Green  
 $success = $true  
 } else {  
 Write-Host "X FAIL: Headers appear to be sanitized (unexpected for vanilla Lambda)" -ForegroundColor Red  
 }  
 }  
   
 # Cleanup  
 Remove-Item $tempFile -Force -ErrorAction SilentlyContinue  
   
 return @{  
 Success = $success  
 Url = $Url  
 }  
}  
  
# Run the three tests  
$result1 = Run-Test -Name "vanilla-lambda" -Path "vanilla" -Description "Should have unsanitized headers to show the problem"  
$result2 = Run-Test -Name "patched-lambda" -Path "patched" -ExpectSuccess $true -Description "Should have sanitized headers to show our solution"  
$result3 = Run-Test -Name "ec2-instance" -Path "ec2" -ExpectSuccess $true -Description "Control case: ALB naturally sanitizes EC2 headers"  
  
# Print summary  
Write-Host ""  
Write-Host "===========================" -ForegroundColor Blue  
Write-Host "SUMMARY OF TEST RESULTS" -ForegroundColor Blue  
Write-Host "===========================" -ForegroundColor Blue  
  
Write-Host "1. Vanilla Lambda ($($result1.Url))" -ForegroundColor White  
Write-Host " Expected: " -ForegroundColor Yellow -NoNewline  
Write-Host "Headers NOT sanitized - should cause HTTP/2 issues"  
Write-Host " Purpose: " -ForegroundColor Yellow -NoNewline  
Write-Host "Demonstrates the problem we're solving"  
  
Write-Host ""  
Write-Host "2. Patched Lambda ($($result2.Url))" -ForegroundColor White  
Write-Host " Expected: " -ForegroundColor Yellow -NoNewline  
Write-Host "Headers ARE sanitized AND successful response"  
Write-Host " Purpose: " -ForegroundColor Yellow -NoNewline  
Write-Host "Demonstrates our Lambda Adapter's sanitization works"  
  
Write-Host ""  
Write-Host "3. EC2 Instance ($($result3.Url))" -ForegroundColor White  
Write-Host " Expected: " -ForegroundColor Yellow -NoNewline  
Write-Host "Headers ARE sanitized by ALB - successful response"  
Write-Host " Purpose: " -ForegroundColor Yellow -NoNewline  
Write-Host "Control case showing ALB's standard behavior with EC2"  
  
Write-Host ""  
if ($result1.Success -and $result2.Success -and $result3.Success) {  
 Write-Host "√ All tests PASSED" -ForegroundColor Green  
 Write-Host "Your HTTP/2 header sanitization solution is working as expected!" -ForegroundColor White  
} else {  
 Write-Host "X Some tests FAILED" -ForegroundColor Red  
 if (-not $result1.Success) {  
 Write-Host "- Vanilla Lambda test failed: Headers should NOT be sanitized" -ForegroundColor Red  
 }  
 if (-not $result2.Success) {  
 Write-Host "- Patched Lambda test failed: Headers should be sanitized" -ForegroundColor Red  
 }  
 if (-not $result3.Success) {  
 Write-Host "- EC2 test failed: ALB should naturally sanitize headers" -ForegroundColor Red  
 }  
}  
  
Write-Host ""  
Write-Host "HTTP/2 Header Sanitization Test Complete" -ForegroundColor Blue