Whenever you navigate to a website, your browser requests a web page, and the server responds with the content along with HTTP headers. Headers such as cache-control are used by the browser to determine how long to cache content for, others such as content-type are used to indicate the media type of a resource and therefore how to interpret such resource. In this post, you will learn how to add response headers that are specifically targeted to improve the security and privacy of both viewers and content providers. I'll also show you how you can add these headers to your website using Lambda@Edge and Amazon CloudFront.

Whenever you navigate to a website, your browser requests a web page, and the server responds with the content along with HTTP headers. Headers such as cache-control are used by the browser to determine how long to cache content for, others such as content-type are used to indicate the media type of a resource and therefore how to interpret such resource. In this post, you will learn how to add response headers that are specifically targeted to improve the security and privacy of both viewers and content providers. I'll also show you how you can add these headers to your website using Lambda@Edge and Amazon CloudFront.

Adding security response headers is often achievable by modifications to your application configuration. In this blog we will focus on how to achieve the same result when you have an application that can't be modified at the origin (e.g., a web site hosted in Amazon S3).

What are security headers?

Security headers are a group of headers in the HTTP response from a server that tell your browser how to behave when handling your site's content. For example, X-XSS-Protection is a header that Internet Explorer and Chrome respect to stop pages loading when they detect cross-site scripting (XSS) attacks. The following is a list of each header we'll be implementing with a link to more information.

Strict Transport Security

Content-Security-Policy

X-Content-Type-Options

X-Frame-Options

X-XSS-Protection

Referrer-Policy

Additional details on each of these security headers can be found in Mozilla's Web Security Guide.

Lambda@Edge Overview

Lambda@Edge provides the ability to execute a Lambda function at an Amazon CloudFront Edge Location. This capability enables intelligent processing of HTTP requests at locations that are close (for the purposes of latency) to your customers. To get started, you simply upload your code (Lambda function written in Node.js) and pick one of the CloudFront behaviors associated with your distribution.

You can run a Lambda@Edge function in response to four different CloudFront events. For the purpose of this blog post, we'll just be focusing on the Origin Response event. If you're interested in looking more broadly at Lambda@Edge, Jeff Barr does a great job of providing an overview in his blog post here.

Origin Response – This event is triggered after the origin returns a response to a request. It has access to the response from the origin.

The following diagram illustrates the available triggers for a CloudFront distribution We're focusing on number 6.:

What are security headers?

Security headers are a group of headers in the HTTP response from a server that tell your browser how to behave when handling your site's content. For example, X-XSS-Protection is a header that Internet Explorer and Chrome respect to stop pages loading when they detect cross-site scripting (XSS) attacks. The following is a list of each header we'll be implementing with a link to more information.

Strict Transport Security

Content-Security-Policy

X-Content-Type-Options

X-Frame-Options

X-XSS-Protection

Referrer-Policy

Additional details on each of these security headers can be found in Mozilla's Web Security Guide.

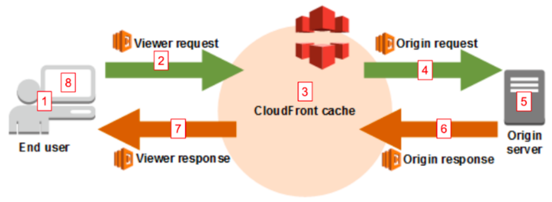
Lambda@Edge Overview

Lambda@Edge provides the ability to execute a Lambda function at an Amazon CloudFront Edge Location. This capability enables intelligent processing of HTTP requests at locations that are close (for the purposes of latency) to your customers. To get started, you simply upload your code (Lambda function written in Node.js) and pick one of the CloudFront behaviors associated with your distribution.

You can run a Lambda@Edge function in response to four different CloudFront events. For the purpose of this blog post, we'll just be focusing on the Origin Response event. If you're interested in looking more broadly at Lambda@Edge, Jeff Barr does a great job of providing an overview in his blog post here.

Origin Response – This event is triggered after the origin returns a response to a request. It has access to the response from the origin.

The following diagram illustrates the available triggers for a CloudFront distribution We're focusing on number 6.:



Here is how the process works:

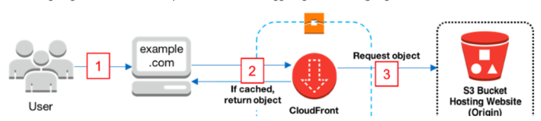
Viewer navigates to website.
Before CloudFront serves content from the cache it will trigger any Lambda function associated with the Viewer Request trigger for that behavior.
CloudFront serves content from the cache if available, otherwise it goes to step 4.
Only after CloudFront cache 'Miss', Origin Request trigger is fired for that behavior.
S3 Origin returns content.
After content is returned from S3 but before being cached in CloudFront, Origin Response trigger is fired.
After content is cached in CloudFront, Viewer Response trigger is fired and is the final step before viewer receives content.
Viewer receives content.

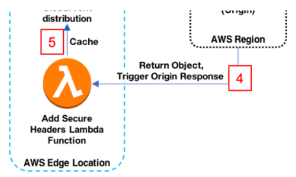
Solution overview

The solution uses a simple single page website, hosted in an Amazon S3 bucket and using Amazon CloudFront. I'll show you how to create a new Lambda@Edge function, how to associate it with your CloudFront distribution, and how to monitor its execution with Amazon CloudWatch Logs. I will be making use of the origin response trigger to execute our Lambda@Edge function. This will allow CloudFront to cache this response after the security headers are added, which means the Lambda@Edge function will only need to be triggered upon a CloudFront 'Miss' and security headers will be returned for all future 'Hits.'

I will assume that you already have an S3 bucket configured for your website, with a CloudFront distribution configured for serving your content. For the purpose of my demo, I've set up an S3 bucket, used it as an origin for my distribution, and uploaded a basic index.html file with the text "Hello World! Do I have security headers yet?"

The following diagram illustrates the sequence of events for triggering our Lambda@Edge function:

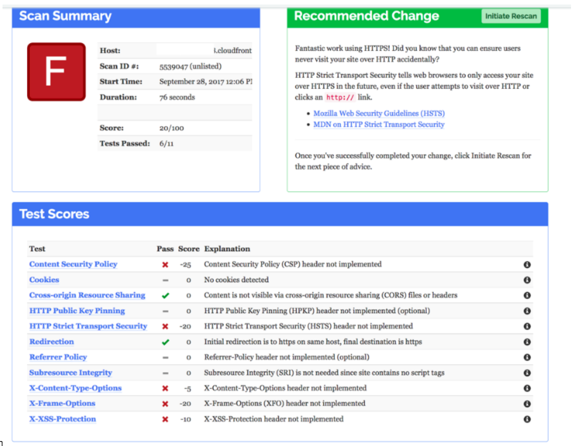



distribution
5
Cache
Add Secure Headers Lambda Function
AWS Edge Location
AWS Region
Return Object, Trigger Origin Response
4

Here is how the process works:

Viewer requests website www.example.com.
If the object is cached already, CloudFront returns the object from the cache to the viewer, otherwise it moves on to step 3.
CloudFront requests the object from the origin, in this case an S3 bucket.
S3 returns the object, which in turn causes CloudFront to trigger the origin response event.
Our Add Security Headers Lambda function triggers, and the resulting output is cached and served by CloudFront.

Before I start, I use Mozilla Observatory to rate my website, the following screenshot shows the rating without the security headers. This doesn't necessarily mean the content being served is insecure, it does however provide a needle on a gauge by which we can look to improve.



Host:	[redacted].cloudfront
Scan ID #:	5539047 (unlisted)
Start Time:	September 28, 2017 12:06 PI
Duration:	76 seconds
Score:	20/100
Tests Passed:	6/11

Fantastic work using HTTPS! Did you know that you can ensure users never visit your site over HTTP accidentally?

HTTP Strict Transport Security tells web browsers to only access your site over HTTPS in the future, even if the user attempts to visit over HTTP or clicks an `http://` link.

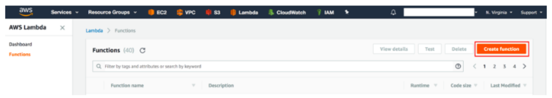
- Mozilla Web Security Guidelines (HSTS)
- MDN on HTTP Strict Transport Security

Once you've successfully completed your change, click Initiate Rescan for the next piece of advice.

Test	Pass	Score	Explanation
Content Security Policy	✗	-25	Content Security Policy (CSP) header not implemented
Cookies	–	0	No cookies detected
Cross-origin Resource Sharing	✓	0	Content is not visible via cross-origin resource sharing (CORS) files or headers
HTTP Public Key Pinning	–	0	HTTP Public Key Pinning (HPKP) header not implemented (optional)
HTTP Strict Transport Security	✗	-20	HTTP Strict Transport Security (HSTS) header not implemented
Redirection	✓	0	Initial redirection is to https on same host, final destination is https
Referrer Policy	–	0	Referrer-Policy header not implemented (optional)
Subresource Integrity	–	0	Subresource Integrity (SRI) is not needed since site contains no script tags
X-Content-Type-Options	✗	-5	X-Content-Type-Options header not implemented
X-Frame-Options	✗	-20	X-Frame-Options (XFO) header not implemented
X-XSS-Protection	✗	-10	X-XSS-Protection header not implemented

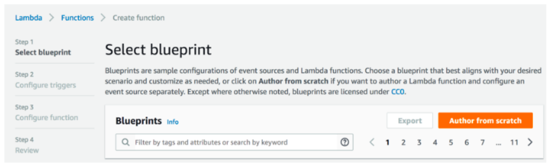
Let's get started. I'll create a new Lambda function

First, I go to the Lambda Console, and I ensure that I'm in the US-East-1 N. Virginia Region by selecting US East (N. Virginia) from the drop-down list at the top right. Then I select Create Function to create a new Lambda function.



Note: I need to select US-East-1 as the location where I create the Lambda function. Otherwise I am unable to connect to a CloudFront trigger. However after I've finished with the setup, the function will be replicated to all other Regions.

Next, I am presented with the option to select a blueprint or Author from scratch. If I type in 'CloudFront' I am presented with a range of different pre-built functions, but for this solution, I choose Author from scratch because I'll be using code provided here for this function.



Blueprints are sample configurations of event sources and Lambda functions. Choose a blueprint that best aligns with your desired scenario and customize as needed, or click on **Author from scratch** if you want to author a Lambda function and configure an event source separately. Except where otherwise noted, blueprints are licensed under CC0.

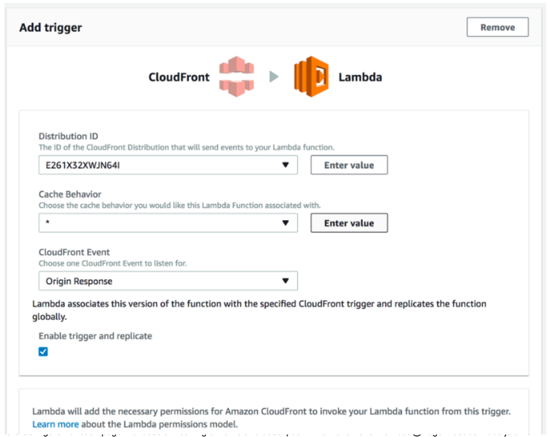
Now that I have started to create a new Lambda function, I need to configure the trigger for it. To do this, I choose the dotted grey box and then choose CloudFront. Note: If you can't see CloudFront as a trigger option, make sure you're in the US-East-1 Region, as required in step one. The options presented here are:

Distribution ID: I'm selecting the distribution I created earlier, that serves content from my S3 bucket.

Cache Behavior: I select '*' which is the default behavior. Since in this case I am not creating additional behaviors, this will apply to all requests. If I had created multiple behaviors, this would only be triggered if none of the other behaviors match.

CloudFront Event: As discussed earlier, I want this to trigger after the origin has returned the object but before the cache, so I select Origin Response.

Enable trigger and replicate: I check this box to enable CloudFront as a trigger for a Lambda function. Upon Lambda function creation, this option automatically creates a version of my function and replicates it across multiple Regions.



Add trigger

Distribution ID
The ID of the CloudFront Distribution that will send events to your Lambda function.
E261X32XWJN64I

Cache Behavior
Choose the cache behavior you would like this Lambda Function associated with.
*

CloudFront Event
Choose one CloudFront Event to listen for.
Origin Response

Lambda associates this version of the function with the specified CloudFront trigger and replicates the function globally.

Enable trigger and replicate
✓

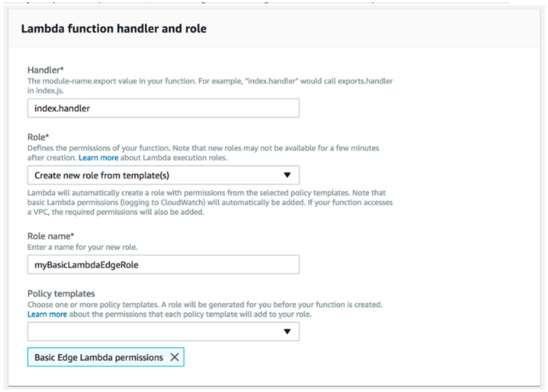
Lambda will add the necessary permissions for Amazon CloudFront to invoke your Lambda function from this trigger. Learn more about the Lambda permissions model.

After I choose Next, I'm presented

I'm using the following code, which gets the contents of the response, sets the new headers, then returns the updated response that includes the new security headers.

```
use strict';
exports.handler = (event, context, callback) => {

    //Get contents of response
    const response = event.Records[0].cf.response;
    const headers = response.headers;

//Set new headers
 headers['strict-transport-security'] = [{key: 'Strict-Transport-Security', value: 'max-age=63072000; includeSubdomains; preload'}];
 headers['content-security-policy'] = [{key: 'Content-Security-Policy', value: "default-src 'none'; img-src 'self'; script-src 'self'; style-src 'self'; object-src 'none'"}];
 headers['x-content-type-options'] = [{key: 'X-Content-Type-Options', value: 'nosniff'}];
 headers['x-frame-options'] = [{key: 'X-Frame-Options', value: 'DENY'}];
 headers['x-xss-protection'] = [{key: 'X-XSS-Protection', value: '1; mode=block'}];
 headers['referrer-policy'] = [{key: 'Referrer-Policy', value: 'same-origin'}];

    //Return modified response
    callback(null, response);
};
```

After pasting this code into my function, I leave my handler as the default'index.handler and choose to Create a new role from template(s). For the function to execute I need to ensure I select the Basic Edge Lambda permissions from the Policy templates drop-down list, which will go ahead and generate a role for me upon Lambda function creation:



Lambda function handler and role

Handler*
The module-name.export value in your function. For example, "index.handler" would call exports.handler in index.js.
index.handler

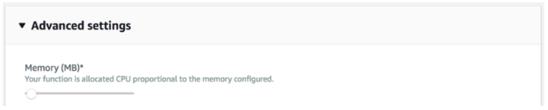
Role*
Defines the permissions of your function. Note that new roles may not be available for a few minutes after creation. Learn more about Lambda execution roles.
Create new role from template(s)
Lambda will automatically create a role with permissions from the selected policy templates. Note that basic Lambda permissions (logging to CloudWatch) will automatically be added. If your function accesses a VPC, the required permissions will also be added.

Role name*
Enter a name for your new role.
myBasicLambdaEdgeRole

Policy templates
Choose one or more policy templates. A role will be generated for you before your function is created. Learn more about the permissions that each policy template will add to your role.
Basic Edge Lambda permissions

In the Advanced Settings tab, ensure that 128 MB is the allocated memory and that 3 seconds is configured for timeout (maximum allowed for Lambda@Edge). Then choose Next. Review the details and choose Create Function.

Note: If this was a Viewer Request or Viewer Response trigger, the maximum Timeout would be 1 second.



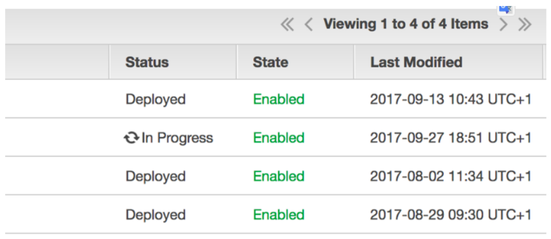
▾ Advanced settings

Memory (MB)*
Your function is allocated CPU proportional to the memory configured.

128 MB

Timeout*
0 min 3 sec

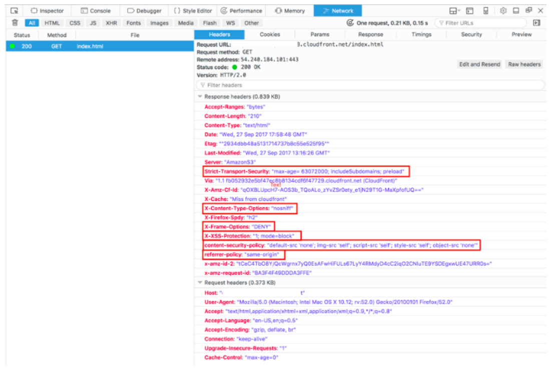
This creates the function, attaches the trigger to the distribution, and also initiates global replication of the function. The status of my distribution changes to In Progress for the duration of the replication (typically 5 to 8 minutes):


Viewing 1 to 4 of 4 Items

	Status	State	Last Modified
	Deployed	Enabled	2017-09-13 10:43 UTC+1
	In Progress	Enabled	2017-09-27 18:51 UTC+1
	Deployed	Enabled	2017-08-02 11:34 UTC+1
	Deployed	Enabled	2017-08-29 09:30 UTC+1

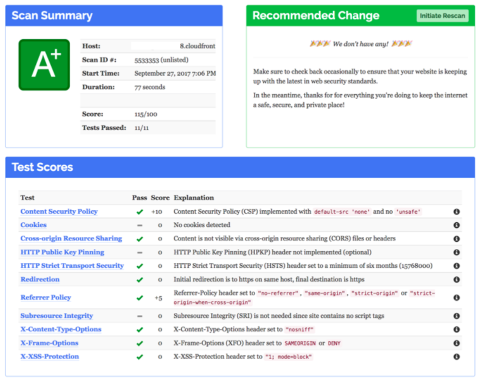
As soon as the status changes back to Deployed I access the root of my distribution, and I see the index.html file:

More importantly, to make sure the headers are being added, I open a browser's Web Developer toolbar, choose the Network tab, and Reload the page. When I choose the GET request for index.html, I'm presented with the added security headers in the response from CloudFront. I've highlighted the ones that my function added:



```
One request, 0.21 KB, 0.15 s

Status  Method  File
200     GET     index.html

Request URL: [redacted]3.cloudfront.net/index.html
Request method: GET
Remote address: 54.240.184.101:443
Status code: ● 200 OK
Version: HTTP/2.0

Response headers (0.839 KB)
  Accept-Ranges: "bytes"
  Content-Length: "210"
  Content-Type: "text/html"
  Date: "Wed, 27 Sep 2017 17:58:48 GMT"
  Etag: ""2934dbb48a5131714737b8c55e525f95""
  Last-Modified: "Wed, 27 Sep 2017 13:16:26 GMT"
  Server: "AmazonS3"
  Strict-Transport-Security: "max-age= 63072000; includeSubdomains; preload"
  Via: "1.1 fb052932e5bf47ec8b8134cdf6f47729.cloudfront.net (CloudFront)"
  X-Amz-Cf-Id: "qOXBLUpcH7-AOS3b_TQoALo_zYvZSr0ety_e1jN29T1G-MaXpfofUQ=="
  X-Cache: "Miss from cloudfront"
  X-Content-Type-Options: "nosniff"
  X-Firefox-Spdy: "h2"
  X-Frame-Options: "DENY"
  X-XSS-Protection: "1; mode=block"
  content-security-policy: "default-src 'none'; img-src 'self'; script-src 'self'; style-src 'self'; object-src 'none'"
  referrer-policy: "same-origin"
  x-amz-id-2: "tCeC4TbO8Y/QcWgrnx7yQ0EsAFwHiFULs67LyY4RMdyD4cC2iqO2CNluTE9YSDEgxwUE47URR0s="
  x-amz-request-id: "8A3F4F49DDDA3FFE"

Request headers (0.373 KB)
  Host: "[redacted]t"
  User-Agent: "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:52.0) Gecko/20100101 Firefox/52.0"
  Accept: "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
  Accept-Language: "en-US,en;q=0.5"
  Accept-Encoding: "gzip, deflate, br"
  Connection: "keep-alive"
  Upgrade-Insecure-Requests: "1"
  Cache-Control: "max-age=0"
```

I re-run my website through Mozilla Observatory. The following results show we've moved from an F rating to an A+.



Scan Summary

Host:	[redacted]8.cloudfront
Scan ID #:	5533353 (unlisted)
Start Time:	September 27, 2017 7:06 PM
Duration:	77 seconds
Score:	115/100
Tests Passed:	11/11

Recommended Change

🎉🎉🎉 *We don't have any!* 🎉🎉🎉

Make sure to check back occasionally to ensure that your website is keeping up with the latest in web security standards.

In the meantime, thanks for for everything you're doing to keep the internet a safe, secure, and private place!

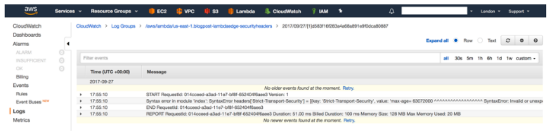
Test Scores

Test	Pass	Score	Explanation
Content Security Policy	✓	+10	Content Security Policy (CSP) implemented with `default-src 'none'` and no `'unsafe'`
Cookies	–	0	No cookies detected
Cross-origin Resource Sharing	✓	0	Content is not visible via cross-origin resource sharing (CORS) files or headers
HTTP Public Key Pinning	–	0	HTTP Public Key Pinning (HPKP) header not implemented (optional)
HTTP Strict Transport Security	✓	0	HTTP Strict Transport Security (HSTS) header set to a minimum of six months (15768000)
Redirection	✓	0	Initial redirection is to https on same host, final destination is https
Referrer Policy	✓	+5	Referrer-Policy header set to `"no-referrer"`, `"same-origin"`, `"strict-origin"` or `"strict-origin-when-cross-origin"`
Subresource Integrity	–	0	Subresource Integrity (SRI) is not needed since site contains no script tags
X-Content-Type-Options	✓	0	X-Content-Type-Options header set to `"nosniff"`
X-Frame-Options	✓	0	X-Frame-Options (XFO) header set to `SAMEORIGIN` or `DENY`
X-XSS-Protection	✓	0	X-XSS-Protection header set to `"1; mode=block"`

Monitoring and debugging

Although I didn't experience any errors in my execution here, it's important to know where to go if there are any problems. In the same way that I monitor any Lambda function, I can use Amazon CloudWatch Logs to monitor the execution of Lambda@Edge functions. The slight difference here is that the logs are stored in the Region closest to the location where the function is executed. So, for my test, I need to look at CloudWatch Logs in the London Region because I'm visiting the website from London. I'll need to change the Region to view the CloudWatch Logs for my Lambda function, according to where my viewers are located.

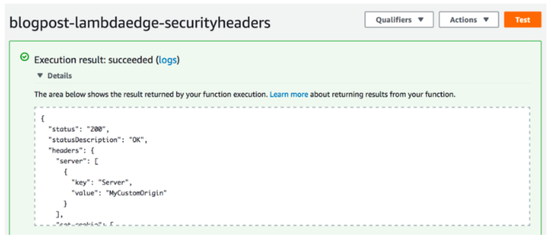
In this screenshot, I've forced an error to show you the log output:



CloudWatch > Log Groups > /aws/lambda/us-east-1.blogpost-lambdaedge-securityheaders > 2017/09/27/[1]d583f16f283a4a68a891e9f0dca80887

```
Time (UTC +00:00)  Message
2017-09-27
                   No older events found at the moment. Retry.
17:55:10           START RequestId: 014cceed-a3ad-11e7-bf8f-652404f6aee3 Version: 1
17:55:10           Syntax error in module 'index': SyntaxError headers['Strict-Transport-Security'] = [{key: 'Strict-Transport-Security', value: 'max-age= 63072000 ^^^^^^^^^^^^^^^^^ SyntaxError: Invalid or unexp
17:55:10           END RequestId: 014cceed-a3ad-11e7-bf8f-652404f6aee3
17:55:10           REPORT RequestId: 014cceed-a3ad-11e7-bf8f-652404f6aee3 Duration: 51.00 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 20 MB
                   No newer events found at the moment. Retry.
```

I find it helpful to test my Lambda function directly in the Lambda console before I enable it to be triggered and replicate. That way I save the time it takes to create a new version, assign a trigger, visit the website then view the logs. To do this, I need to configure a test event in the Lambda function in the way I normally would for Lambda, and pass it a sample request or response specific to CloudFront. After I choose Save and Test, I'm presented with the output and any errors. That way I can quickly fix them.



blogpost-lambdaedge-securityheaders

Execution result: succeeded (logs)

Details

The area below shows the result returned by your function execution. Learn more about returning results from your function.

```
{
  "status": "200",
  "statusDescription": "OK",
  "headers": {
    "server": [
      {
        "key": "Server",
        "value": "MyCustomOrigin"
      }
    ],
```

In this post, I showed you how to use Lambda@Edge to improve the security of your website by adding security headers to the origin response trigger of a CloudFront distribution behavior. I demonstrated creating a Lambda@Edge function, associating it with a trigger on a CloudFront distribution, then proving the result and monitoring the output. This is a very simple example of what can be achieved with Lambda@Edge, and I'm sure you can come up with far more creative ways to use it!

If you have any questions about this blog please post them in the Comments section below. If you have any awesome ideas of creative ways you can use Lambda@Edge, please share them in the AWS Lambda Forums.

If you're new to AWS Lambda@Edge, I encourage you to try out some of the blueprints in the Lambda console and refer to Getting Started with Amazon CloudFront and Getting Started with AWS Lambda@Edge documentation for more information.

Ref

https://aws.amazon.com/vi/blogs/networking-and-content-delivery/adding-http-security-headers-using-lambdaedge-and-amazon-cloudfront/

https://shaun.net/notes/using-lambda-at-edge-to-add-security-headers/

https://johnlouros.com/blog/setup-security-headers-s3-host-website