

Report for:

PHP Secure Code Review

NCC Group

June 22, 2022

Version: 1.0

Prepared by: NCC TESTER

Email: NCCTESTER@nccgroup.com

Telephone: +6122222222



NCC Group PLC - Security Testing Audit and Compliance

Level 13,
92 Pitt Street,
Sydney NSW 2000,
Australia
<https://www.nccgroup.com>

1 Table of Contents

1	Table of Contents	2
1.1	Risk Ratings	3
1.2	Findings Overview	4
2	Technical Details	6
3	Technical Summary.....	29
3.1	Scope	29
3.2	Caveats	29
3.3	Post Assessment Cleanup	29
4	Executive Summary	30
4.1	Overview	30
4.2	Assessment Summary	30
4.3	Strategic Recommendations	31
5	Document Control	32
5.1	Client Confidentiality	32
5.2	Proprietary Information	32
6	Contact Info.....	33

1.1 Risk Ratings

The table below gives a key to the icons and symbols used throughout this report to provide a clear and concise risk scoring system. It should be stressed that quantifying the overall business risk posed by any of the issues found in any test is outside our remit. This means that some risks may be reported as high from a technical perspective but may, as a result of other controls unknown to us, be considered acceptable.

Symbol	Risk Rating	CVSS Score	Explanation
	CRITICAL	9.0 - 10	A vulnerability was discovered that has been rated as critical. This requires resolution as quickly as possible.
	HIGH	7.0 - 8.9	A vulnerability was discovered that has been rated as high. This requires resolution in the short term.
	MEDIUM	4.0 - 6.9	A vulnerability was discovered that has been rated as medium. This should be resolved as part of the ongoing security maintenance of the system.
	LOW	1.0 - 3.9	A vulnerability was discovered that has been rated as low. This should be addressed as part of routine maintenance tasks.
	INFO	0 - 0.9	A discovery was made that is reported for information. This should be addressed in order to meet leading practice.
	GOOD	N/A	Good security practices were being followed or an audit item was found to be present and correct.

1.2 Findings Overview

All the issues identified during the assessment are listed below with a risk rating for each issue.

Ref	Finding	Risk
NCCGRPHP-001	Outdated or Unsupported Third Party Software	High
		
NCCGRPHP-014	Lack of Certificate Validation	High
		
NCCGRPHP-018	Poor Software Development Lifecycle	High
		
NCCGRPHP-002	Hard-Coded Trivial Credentials	Medium
		
NCCGRPHP-003	Subscope Variable Overshadows Outer Scope Variable	Medium
		
NCCGRPHP-004	TODO Comments Indicate Incomplete Code	Medium
		
NCCGRPHP-006	Hard-coded Static Secrets	Medium
		
NCCGRPHP-012	Incorrect Creation of Static Object	Medium
		
NCCGRPHP-013	All Exceptions Caught Without Processing	Medium
		
NCCGRPHP-016	Use of MD5 and SHA1	Medium
		
NCCGRPHP-017	Inconsistent SQL Injection Prevention	Medium
		
NCCGRPHP-005	Nested Statements without Curly Braces	Low
		

Ref	Finding	Risk
NCCGRPPHP-007	PHP API Calls without Side Effects	Low 
NCCGRPPHP-008	Unreachable Code	Low 
NCCGRPPHP-009	Variable Assigned to Itself	Low 
NCCGRPPHP-010	Parameter Assigned Before Use	Low 
NCCGRPPHP-011	Hard-coded URLs and Cloud Bucket Names	Low 
NCCGRPPHP-015	HTTP Basic Authentication in Use	Low 

2 Technical Details

NCCGRPPHP-001 Outdated or Unsupported Third Party Software			
Risk Rating	Status	Open	
Description:			
A number of third party software packages in the source code repositories were outdated or unsupported and some were affected by publicly reported security vulnerabilities. This indicates there may be a gap within the security patching process.			
A full breakdown of the outdated third-party software at the time of the assessment has been provided in Affected Resources section.			
Recommendation:			
Investigate the software patching and update policy and ensure that updates are applied to all software installations, including third party applications, on a regular basis. Consideration should be given to enabling the auto-update functionality within the affected third party software, to ensure that updates are applied quickly and regularly. ¹			
Where software is no longer supported, it may be necessary to initiate a programme of work to investigate the most effective path towards the use of later, or alternative, supported software.			
Affects:			
Third-party Software	Version	Supported Version	URL
NodeJS (docker image circleci/node)	10.8	12	https://nodejs.org/en/about/releases/
Python (docker image circleci/python)	2.7	3.9	https://www.python.org/downloads/
Golang (docker image circleci/golang)	1.10	1.16.6	https://github.com/golang/go/releases
PHP	7.3	7.4.22	https://www.php.net/index.php?id=2021-07-29-2
aws/aws-sdk-php	2.8.31	3.186.3	https://github.com/aws/aws-sdk-php/releases/tag/3.186.3
beberlei/assert	v3.2.1	v3.3.1	https://github.com/beberlei/assert/releases/tag/v3.3.1
briannesbitt/Carbon	1.21.0	2.51.1	https://github.com/briannesbitt/Carbon/releases/tag/2.51.1
composer/installers	v1.10.0	v1.11.0	https://github.com/composer/installers/releases/tag/v1.11.0
dragonmantank/cron-expression	v2.2.0	v3.1.0	https://github.com/dragonmantank/cron-expression/releases/tag/v3.1.0
doctrine/cache	1.6.2		https://github.com/doctrine/cache/releases/tag/2.1.1

¹Checking for Outdated Third-party Packages : <https://github.com/FriendsOfPHP/security-advisories#checking-for-vulnerabilities>

Third-party Software	Version	Supported Version	URL
facebook/graph-sdk	5.6.1		https://github.com/facebookarchive/php-graph-sdk/releases/tag/5.7.0
firebase/php-jwt	5.0.0	5.4.0	https://github.com/firebase/php-jwt/releases/tag/v5.4.0
fluent/fluent-logger-php	v0.3.8	v1.0.1	https://github.com/fluent/fluent-logger-php/releases/tag/v1.0.1
fzaninotto/faker	1.7.1	Package fzaninotto/faker is abandoned	
gemorroj/archive7z	2.0.0		https://github.com/Gemorroj/Archive7z/releases/tag/5.1.1
protocolbuffers/protobuf-php	3.15.8	v3.17.3	https://github.com/protocolbuffers/protobuf-php/releases/tag/v3.17.3
googleapis/common-protos-php	1.3	1.3.1	https://github.com/googleapis/common-protos-php/releases/tag/1.3.1
googleapis/google-auth-library-php	v1.7.1	v1.16.0	https://github.com/googleapis/google-auth-library-php/releases/tag/v1.16.0
googleapis/google-cloud-php	v0.147.0	v0.163.1	https://github.com/googleapis/google-cloud-php/releases/tag/v0.163.1
grpc/grpc-php	1.36	1.39.0	https://github.com/grpc/grpc-php/releases/tag/v1.39.0
guzzle/guzzle	3.7.0	guzzle/guzzle is abandoned	
guzzlehttp/psr7	1.4.2	2.0.0	https://github.com/guzzle/psr7/releases/tag/2.0.0
paragonie/constant_time_encoding	v2.2.3	v2.4.0	https://github.com/paragonie/constant_time_encoding/releases/tag/v2.4.0
php-fig/cache	1.0.1	3.0.0	https://github.com/php-fig/cache/releases/tag/3.0.0
php-fig/container	1.0.0	1.1.1	https://github.com/php-fig/container/releases/tag/1.1.1
php-fig/log	1.1.3	1.1.4	https://github.com/php-fig/log/releases/tag/1.1.4
phpseclib/phpseclib	2.0.31	3.0.9	https://github.com/phpseclib/phpseclib/releases/tag/3.0.9
ramsey/uuid	3.7.1	4.1.1	https://github.com/ramsey/uuid/releases/tag/4.1.1
seldaek/monolog	1.26.0	2.3.2	https://github.com/Seldaeak/monolog/releases/tag/2.3.2



Third-party Software	Version	Supported Version	URL
silexphp/Pimple	v3.2.2	v3.4.0	https://github.com/silexphp/Pimple/releases/tag/v3.4.0
solariumphp/solarium	3.8.1	6.1.4	https://github.com/solariumphp/solarium/releases/tag/6.1.4
swiftmailer/swiftmailer	v5.4.8		https://github.com/swiftmailer/swiftmailer/releases/tag/v6.2.7
symfony/cache	4.2.12	v5.3.4	https://github.com/symfony/cache/releases/tag/v5.3.4
symfony/console	2.5.4	v5.3.6	https://github.com/symfony/console/releases/tag/v5.3.6
symfony/contracts	v1.0.2	v2.3.1	https://github.com/symfony/contracts/releases/tag/v2.3.1
symfony/event-dispatcher	v3.3.8	v5.3.4	https://github.com/symfony/event-dispatcher/releases/tag/v5.3.4
symfony/http-foundation	v3.4.36	v5.3.6	https://github.com/symfony/http-foundation/releases/tag/v5.3.6
symfony/process	v3.4.1	v5.3.4	https://github.com/symfony/process/releases/tag/v5.3.4
symfony/translation	v4.4.4	v5.3.4	https://github.com/symfony/translation/releases/tag/v5.3.4
symfony/var-exporter	v3.1.13	v5.3.4	https://github.com/symfony/var-exporter/releases/tag/v5.3.4
theophpleague/csv	9.1.4	9.7.1	https://github.com/theophpleague/csv/releases/tag/9.7.1
twilio/twilio-php	5.37.0	6.27.1	https://github.com/twilio/twilio-php/releases/tag/6.27.1

NCCGRPPHP-014 Lack of Certificate Validation

Risk Rating

High

Status

Open

Description:

An attacker can intercept and modify network communication.

The application disabled certificate validation when using TLS for secure communication. TLS ensures traffic is securely encrypted so that an attacker cannot read or modify network data. However, the security of TLS relies on a certificate, which uniquely identifies one side of the communication. By verifying the certificate against a local certificate authority, the client guarantees that communication has not been intercepted or modified.

Because the application does not correctly validate certificates, an attacker can substitute a malicious certificate in order to intercept and modify communication. Essentially, the lack of certificate validation defeats the protection offered by TLS.

The following code disabled certificate validation:

```

Line 113      $response_code = curl_exec($curl);
Line 114      if (CURLE_OK !== curl_errno($curl)) {
Line 115          Log::warning(
→   'Failed to send SMS with SSL option. Will resend with no SSL option temporarily. Error: ' .
→   curl_error($curl));
Line 116
Line 117      curl_setopt($curl, CURLOPT_SSL_VERIFYPEER, false);
Line 118      curl_setopt($curl, CURLOPT_SSL_VERIFYHOST, false);
Line 119      $response_code = curl_exec($curl);
Line 120  }
Line 121  curl_close($curl);

```

The code attempts to connect to a third-party site and if the error is received, it attempts to connect again without certificate validation. If the third-party site presents an invalid certificate or self-signed certificate the first attempt at connection fails due to security concerns. However, the code then removes the security checks and connects to the third-party site anyway.

Recommendation:

Do not disable certificate validation. Validation of X.509 certificates is essential to create secure SSL/TLS sessions not vulnerable to man-in-the-middle attacks.²

The certificate chain validation includes these steps:

- ◆ The certificate is issued by its parent Certificate Authority or the root CA trusted by the system.
- ◆ Each CA is allowed to issue certificates.
- ◆ Each certificate in the chain is not expired.

²CWE-295: *Improper Certificate Validation* : <https://cwe.mitre.org/data/definitions/295.html>

NCCGRPPHP-018 Poor Software Development Lifecycle

Risk Rating

High

Status

Open

Description:

The code base showed insufficient adherence to a software development lifecycle (SDL). Robust SDL implementation allows development of software of consistent quality, this was not evident from the code base.

Following issues in the code indicated insufficient SDL practices:

- ◆ High volume of Static Analysis issues. During the course of the assessment the code was processed by three PHP static analyzers (Sonar QUBE, PHPStan, Psalm). All three tools generated thousands of issues. This indicated the Static Analysis was not a part of SDL process.
- ◆ High number of technical debt issues recorded as TODO in the code (finding NCCGRPPHP-004 on page 13). The usage of TODO does not match best SDL practices.
- ◆ Uneven code quality. A number of files contained trivial errors (finding NCCGRPPHP-013 on page 18), (finding NCCGRPPHP-014 on the preceding page), (finding NCCGRPPHP-009 on page 25), (finding NCCGRPPHP-003 on page 12). This indicated insufficient code review practices and absence of coding standards.
- ◆ Large number of hard-coded credentials, static keys and third-party URLs (finding NCCGRPPHP-002 on the next page), (finding NCCGRPPHP-006 on page 14), (finding NCCGRPPHP-011 on page 27). Hard-coded secrets do not match best SDL practices.
- ◆ Absence of parity between production and test environments. A number of third-party connections were found to be functional only in production environment. This practice reduces test coverage before release and forces testing to be performed in production.
- ◆ Outdated and unsupported third-party components (finding NCCGRPPHP-001 on page 6).

Recommendation:

The goal of the SDL is to minimize security-related vulnerabilities in the design, code, and documentation, and to detect and eliminate vulnerabilities as early as possible in the development lifecycle. These improvements reduce the number and severity of security vulnerabilities and improve the protection of users' privacy.

It is recommended to implement the following:

- ◆ Integrate Static Analysis and Software Composition Analysis tools in the CI pipeline.
- ◆ Migrate TODO technical issues to a issue database and perform triage of technical debt.
- ◆ Enforce code review on all code by senior members of the development team.
- ◆ Migrate secrets to the secure secret storage (this has been planned as indicated by the development team)
- ◆ Migrate configuration system to common template to enforce parity between the environments. Use mock third-party services to emulate their behavior in the test environment. Use a copy of the database with anonymized data in the test environment.
- ◆ Enforce periodic updates of third-party components including PHP and MySQL.

It is important to note that the list of proposed changes is ambitious and the implementation may take a significant period of time. However, the improvement in SDL will directly affect the quality and security of the application as well as increase the productivity of developers.

Affects:

- ◆ Project in scope

NCCGRPPHP-002 Hard-Coded Trivial Credentials

Risk Rating

Medium

Status

Open

Description:

The codebase contained hard-coded credentials. This does not conform to security best practice, and is not indicative of a secure platform. In addition, the credentials were easily guessable.

```

Line 14      environment:
Line 15      DB_HOST: 127.0.0.1
Line 16      MYSQL_ROOT_PASSWORD: *****
Line 17      MYSQL_DATABASE: test
Line 18      MYSQL_USER: user
Line 19      MYSQL_PASSWORD: *****

```

While applications may need to access credentials without user interaction, storing them in the code makes it difficult to change them - should the credentials be discovered, an attacker could gain access to the program.

Discovery of the credentials would also be easy; strings hard-coded into a binary are trivially extractable, using a variety of common tools, and so anyone with access to the binary file would be able to discover them with little effort.

Further, if multiple installations of the product share the same credentials, then an attacker with access to one installation could gain access to other instances. This can also lead in some circumstances to the possibility of the program being used as a vector to propagate worms.

A full list of the instances of credentials in code has been provided in Affected Resources section.

Recommendation:

Credentials should not be hard-coded into the application. Rather, they should be stored separately, ideally in a password vault, or in an encrypted file to which only one user has access.^{3,4,5}

Alternatively, credentials could be injected in the process environment. An example is shown below:

```

Line 184 define('MQ_PASSWORD', getenv('MYSQL_ROOT_PASSWORD'));
Line 185 define('MQ_ENQUEUE_PASSWORD', getenv('MYSQL_ROOT_PASSWORD'));

```

Affects:
File

/client-api-master/.circleci/config.yml

Line

16

³Hardcoded Credentials Example: <http://nakedsecurity.sophos.com/2014/04/03/is-amazon-hacking-our-apps-or-doing-us-all-a-security-favour/>

⁴SANS Top 25 Software Flaws: Number 11, Hardcoded Credentials: <http://software-security.sans.org/blog/2010/03/10/top-25-series-rank-11-hardcoded-credentials/>

⁵CWE-798: Use of Hard-Coded Credentials: <http://cwe.mitre.org/data/definitions/798.html>

NCCGRPHP-003 Subscope Variable Overshadows Outer Scope Variable

Risk Rating

Medium

Status

Open

Description:

Subscope variables declared with the same name as outer scope variables overshadow outer scope variables. This results in code which is difficult to read and maintain. Such code can lead to bugs that are difficult to root cause.

An example of an overshadowed variable in the code is shown below:

```

Line 7     protected $response = '';
...
Line 9     public function render($action = null)
Line 10    {
...
Line 21      $response = self::extract($view_filename, $this->vars);

```

PHP creates a new instance of the variable when it first encounters it in the scope. The code declared variable `$response` was declared in the class scope on Line 7 and re-declared in the method `render` scope on line 21. As a result, assigning to variable `$response` on line 21 does not modify variable `$response` declared in line 7.⁶

A full list of overshadowed variables found in the code is placed in the **Affected Resources** section below.

Recommendation:

Do not reuse identifier names in subscopes. Include the requirements in the process for code review. Refer to CMU SEI rules for more information^{7,8}.

Affects:

File	Line	Variable	Overshadowed on Line
client-api-master/app/app_api_view.php	7	\$response	21

⁶PHP Variable Scope :<https://www.php.net/manual/en/language.variables.scope.php>

⁷CMU SEI DCL01-C Do not reuse variable names in subscopes. :<https://wiki.sei.cmu.edu/confluence/display/c/DCL01-C.+Do+not+reuse+variable+names+in+subscopes>

⁸CMU SEI DCL51-J Do not shadow or obscure identifiers. :<https://wiki.sei.cmu.edu/confluence/display/java/DCL51-J.+Do+not+shadow+or+obscure+identifiers+in+subscopes>

NCCGRPHP-004
TODO Comments Indicate Incomplete Code

Risk Rating

Medium

Status

Open

Description:

Source code contained a large number of comments marked TODO. This indicates incomplete code and could result in incorrect or insecure behaviour of the application.⁹

An example of a TODO comment is shown below:

```
Line 32      //TODO: meta.notification
Line 33      if($this->controller->isNeedForceLogout()==true){
```

A full list of TODO comments found in the code is placed in the **Affected Resources** section below.

Recommendation:

Review TODO comments in the code. Create issues to track the code defects associated with TODO comments. Remove TODO comments for issues fixed or no longer applicable.

Affects:

File	Line
/client-api-master/app/app_api_view.php	32

⁹CWE-546: Suspicious Comment : <http://cwe.mitre.org/data/definitions/546.html>

NCCGRPPHP-006 Hard-coded Static Secrets

Risk Rating

Medium

Status

Open

Description:

The code base contained hard-coded static secrets. Static secrets by design do not change over time and are valid until explicitly invalidated. As such static secrets represent continuous risk to the enterprise. Incorporating static secrets in the source code elevates the risk to the enterprise.

Static secrets may include:

- ◆ Cloud access keys
- ◆ JWTs with secrets and no set expiration date
- ◆ Private part of key pairs
- ◆ Service access keys and IDs that do not require second factor authentication
- ◆ Strong passwords

Static secrets are different from trivial hard-coded passwords described in finding NCCRPPHP-002 on page 11. Static secrets are designed to be resistant to brute-force attacks. An attack on static secret require gaining access to secret storage. Unlike the dedicated secret storage the source code is not meant for storage of secrets. As a result, the risk to enterprise increases.

Below are examples of static secrets found in the code.

- ◆ Configuration file contained cloud access keys (AWS). The cloud access keys were valid at the time of writing. The cloud access keys were used with `Enumerate IAM` tool¹⁰ to brute-force account privileges in AWS. The tool contained functionality to generate requests for the current version of AWS SDK. This allowed discovery of the account privileges.

```
Line 7 // AWS
Line 8 define('AWS_ACCESS_KEY', 'AKIA*****NUU7E');
Line 9 define('AWS_SECRET_KEY', '6ZD*****j');
```

```
$ aws sts get-caller-identity
{
    "UserId": "AID*****",
    "Account": "30*****",
    "Arn": "arn:aws:iam::30*****:user/*****"
}
```

The list of static secrets found in the code is placed in the `Affected Resources` section.

Recommendation:

Secrets should not be hard-coded into the application. Rather, they should be stored separately, ideally in a password vault, or in an encrypted file to which only one user has access.^{11,12,13} Alternatively, secrets could be injected in the process environment. An example is shown below:

¹⁰ *Enumerate IAM* : <https://github.com/andresriancho/enumerate-iam>

¹¹ Hardcoded Credentials Example: <http://nakedsecurity.sophos.com/2014/04/03/is-amazon-hacking-our-apps-or-doing-us-all-a-security-favour/>

¹² SANS Top 25 Software Flaws: Number 11, Hardcoded Credentials: <http://software-security.sans.org/blog/2010/03/10/top-25-series-rank-11-hardcoded-credentials/>

¹³ CWE-798: Use of Hard-Coded Credentials: <http://cwe.mitre.org/data/definitions/798.html>

```
Line 184 define('MQ_PASSWORD', getenv('MYSQL_ROOT_PASSWORD'));
Line 185 define('MQ_ENQUEUE_PASSWORD', getenv('MYSQL_ROOT_PASSWORD'));
```

Affects:

File	Line	Notes
/client-api-master/app/config/credentials/jp_ci.php	8-9	AWS access key

NCCGRPHP-012 Incorrect Creation of Static Object

Risk Rating

Medium

Status

Open

Description:

A PHP function declared a static object with the intention to create one instance and return it to all callers. The function was implemented incorrectly and returned a new instance of the object to every call.

```

Line 143     /**
Line 144     * @return UpdateService
Line 145     */
Line 146     private function getUpdateService()
Line 147     {
Line 148         static $service;
Line 149
Line 150         return $service ?: new UpdateService(Config::get('adapter'));
Line 151     }

```

The following test case was created to show the problem:

```

<?php

class Singleton
{
    function __construct() {
        print "In Singleton constructor\n";
    }
}

class TestSingleton
{
    public function BadCall()
    {
        static $service;
        return $service ?: new Singleton();
    }

    public function GoodCall()
    {
        static $service;
        if (!isset($service)) {
            $service = new Singleton();
        }
        return $service;
    }
}

$a = new TestSingleton();

print("=====Bad Calls=====\\n");
$singl1 = $a->BadCall();
var_dump($singl1);
$singl2 = $a->BadCall();
var_dump($singl2);
print("=====Good Calls=====\\n");

```

```
$singl3 = $a->GoodCall();
var_dump($singl3);
$singl4 = $a->GoodCall();
var_dump($singl4);

?>
```

Output of the test below shows that `BadCall()` function creates new instance of the object every call.

```
=====Bad Calls=====
In Singleton constructor
object(Singleton)#3 (0) {
}
In Singleton constructor
object(Singleton)#4 (0) {
}
=====Good Calls=====
In Singleton constructor
object(Singleton)#5 (0) {
}
object(Singleton)#5 (0) {
```

A list of incorrectly implemented static objects is placed in the `Affected Resources` section.

Recommendation:

Review list of incorrectly implemented static object. Consider if static objects are required in each case. If static objects are required, correct the code.

Affects:

File	Function	Lines
/client-api-master/src/Command/RebuildProductItemsIndexCommand.php	getUpdateService()	143-151
/client-api-master/src/Command/RebuildProductItemsIndexCommand.php	getDeleteService()	165-174

NCCGRPHP-013 All Exceptions Caught Without Processing

Risk Rating

Medium

Status

Open

Description:

In a number of places all exceptions were caught without re-throw. As a result, system exceptions were also caught and masked.

PHP class `Exception` is a base class of all exceptions in PHP.¹⁴ The hierarchy of exceptions derived from `Exception` includes such exceptions as `PharException` and `OverflowException` that are thrown by the PHP run-time. These exceptions are not meant to be processed by the application as the application has no means of fixing the underlying problem.¹⁵

The coding pattern when all exceptions are caught without reprocessing is called “swallowing exceptions” or “eating errors”. The following code shows the pattern:

```

Line 410      try {
...
    . . . <do many things> . . .

Line 428      } catch (Exception $e) {
Line 429          Log::warn(array(
Line 430              'message' => 'sendEvent failed (MessageEvent::EVENT_ID_MAIL_SEND)',
Line 431          ));
Line 432      }

```

List of “eating errors” pattern implementations is placed in `Affected Resources` section.

Recommendation:

As a general rule, the application should not catch exceptions it cannot process. This rule is applicable to a large set of languages that implement exception hierarchy (C++, Java, C#, PHP, Python, etc.). It is recommended to all review the instances of “eating errors” pattern in the code and implement specific catch clauses for application exceptions.

An example of correct exception handling is shown below. All applications level exceptions are caught and processed and all other exceptions are caught and re-thrown.

```

Line 53      try {
...
    . . . <do things> . . .

Line 59      } catch (BankAccountNotFoundException $e) {
Line 60          $db->rollback();
Line 62          Log::info([
...
Line 68      } catch (ACLEException $e) {
Line 69          $db->rollback();
Line 71          Log::info([
...
Line 122     } catch (Exception $e) {
Line 123         $db->rollback();
Line 124         throw $e;

```

¹⁴PHP Class `Exception` : <https://www.php.net/manual/en/language.exceptions.php>

¹⁵PHP Hierarchy of Exceptions : <https://www.php.net/manual/en/class.error.php>

```
Line 125      }
```

Affects:**File****Lines**

/client-api-master/app/models/bulk_distribution_service.php	400-405
/client-api-master/app/models/bulk_distribution_service.php	410-432
/client-api-master/app/models/bulk_distribution_service.php	437-458
/client-api-master/app/models/config.php	141-145

NCCGRPPHP-016 Use of MD5 and SHA1**Risk Rating** Medium**Status** Open**Description:**

The application made use of the MD5 and SHA1 algorithms. These algorithms are now considered to be cryptographically weak, in that it is vulnerable to collision attacks. This means that it is possible for an attacker to create two messages that have the same computed hash value.

Additionally MD5 is known to have a number of weaknesses, and is not allowed by the Microsoft SDL cryptographic standards.¹⁶

A full list of MD5 and SHA1 usage found in the code is included in the **Affected Resources** section.

Recommendation:

MD5 and SHA1 should be replaced with stronger algorithms, such as one of the SHA-2 family.

Affects:

File	Lines	Notes
/client-api-master/app/models/raw_iv_cert.php	32	SHA1

¹⁶Analysis of MD5 Weaknesses : <http://msdn.microsoft.com/en-us/magazine/ee321570.aspx>

NCCGRPH-017	Inconsistent SQL Injection Prevention	
Risk Rating	Medium	Status Open

Description:

Best practice recommendations for prevention of SQL injection were not followed in the PHP code base. As a result, the application has potential SQL injection vulnerabilities.

The application used PHP Data Objects (PDO) and MySQL PDO driver to access the database. Following vulnerabilities related to SQL injection were found:

- ◆ An attribute `PDO::ATTR_EMULATE_PREPARES` was not set in the PDO. As a result, there was a possibility of prepared statements being emulated. Emulation of prepared statements does not offer protection against SQL injection.¹⁷
- ◆ The application used `PDO::quote()` for processing of untrusted data. `PDO::quote` is not recommended for defense against SQL injection.¹⁸ List of places where `PDO::quote()` was found is placed in `Affected Resources` section.

Recommendation:

Prepared statements or equivalent APIs that always correctly escape input to protect against SQL injection should be used.¹⁹

Affects:

File	Lines	Notes
/client-api-master/app/models/admin_bulk_contact_sent_push_likes_service.php	20-26	<code>PDO::quote()</code> used

¹⁷ PDO Attribute `ATTR_EMULATE_PREPARES` : <https://www.php.net/manual/en/pdo.setattribute.php>

¹⁸ `PDO::quote()` : <https://www.php.net/manual/en/pdo.quote.php>

¹⁹ PHP Security Cheat Sheet: https://www.owasp.org/index.php/PHP_Security_Cheat_Sheet

NCCGRPPHP-005 Nested Statements without Curly Braces

Risk Rating

Low

Status

Open

Description:

A number of nested statements did not have curly braces. The omission of curly braces can be misleading, and may lead to the introduction of errors during maintenance.

An example of affected code is shown below:

```

Line 32      foreach(Param::params() as $key => $value){
Line 33          if(!is_array($value))
Line 34              $_REQUEST[$key] = convert_en_to_em($value);
Line 35      }

```

A full list of locations where nested statements were found without curly braces is placed in the **Affected Resources** section.

Recommendation:

Add curly braces around nested statements. More information on code correctness rules related to this issue can be found in the references.²⁰

Affects:

File	Lines
/client-api-master/app/app_controller.php	33-34

²⁰EXP52-J. Use braces for the body of an if, for, or while statement : <https://wiki.sei.cmu.edu/confluence/display/java/EXP52-J.+Use+braces+for+the+body+of+an+if%2C+for%2C+or+while+statement>

NCCGRPHP-007 PHP API Calls without Side Effects



Risk Rating Low

Status Open

Description:

A number of PHP functions contained PHP API calls without side effects. Such statements do not produce results and are effectively meaningless. This may indicate misunderstanding by software developer of how PHP APIs work. As a result, the program can behave incorrectly as it relies on the result of the API call that was not assigned.

An example of a PHP statements without side effects is placed below:

```
Line 68      trim($key);
Line 69      trim($value);
```

The call to API `trim()` was never assigned. The original string value was not changed and a new value created by `trim()` was lost. As a result, the expectation of the software developer that the string was trimmed was invalid.²¹

The list of PHP API calls without side effects is placed in **Affected Resources** section.

Recommendation:

A variable should be assigned the result of PHP API call. PHP API calls that do not have assignment (no side effects) should be reviewed and removed.

Affects:

File	Line
client-api-master/app/commands/standalone_command.php	68

²¹ PHP `trim()` manual : <https://www.php.net/manual/en/function.trim.php>

NCCGRPPHP-008 Unreachable Code

Risk Rating

Low

Status

Open

Description:

A number of PHP functions contained unreachable code. Such code would never execute in the program. However, the presence of the code itself may introduce logical errors as developers incorrectly rely on the functionality provided by unreachable code.

Following is the example of unreachable code in the program:

```

Line 252          // DatabaseDeprecatedQueryConfig::$deprecated_config
Line 254          return $next_proxy($dbh, $sql, $params);
Line 255
Line 257          try {
Line 258              list($ret, $message) = self::searchDeprecatedQuery($dbh, $sql);
Line 259              if ($ret) {
Line 261                  Log::error([
Line 262                      'tag'    => 'detect_DEPRECATED_query',
Line 263                      'message' => $message,
Line 264                  ]);
Line 265              }
Line 266          } catch (Exception $e) {
Line 267              Log::warning($e);
Line 268      }

```

The `return` statement on Line 254 exits the scope of the function. As a result, code on Lines 257-268 would never execute. However, a software developer that misses the `return` statement on Line 254 would incorrectly assume code on Lines 257-268 executes and would rely on the results. The presence of unreachable or dead code in the program indicates insecure Software Development Lifecycle practices.[^footnoteName1]

The list of instances of unreachable code is placed in **Affected Resources** section.

[^footnoteName1]: *MITRE CWE-561 Dead Code* :<http://cwe.mitre.org/data/definitions/561.html>

Recommendation:

Review all instances of unreachable code. Correct the flow to reach the code if the functionality is required. Remove the unreachable code if functionality is not required.

Affects:**File****Lines**

/client-api-master/app/config/database_sql_hook.php

257-268

NCCGRPPHP-009 Variable Assigned to Itself**Risk Rating**

Low

Status

Open

Description:

A number of PHP functions contained variables assigned to itself. There is not reason to re-assign a variable to itself. Either the statement is redundant and should be removed, or the re-assignment is a mistake and some other variable was intended for the assignment instead.

The following code shows variables assigned to itself.

```
Line 36      } elseif (strpos($url, '#') === 0) {  
Line 37          $url = $url;  
Line 38      } elseif (strpos($url, '?') === 0) {  
Line 39          $url = $url;  
Line 40      }
```

The list of instances of variables assigned to themselves is placed in the **Affected Resources** section.

Recommendation:

Review the instances of variables assigned to themselves and correct the code.

- ◆ If the assignment was intended for some other variable fix the code.
- ◆ If the assignment is a placeholder no operation, refactor the code to avoid assignment altogether.

Affects:

File	Line
client-api-master/app/config/router.php	37

NCCGRPHP-010 Parameter Assigned Before Use



Risk Rating Low

Status Open

Description:

A function parameter was assigned in the body of the function before use. This indicates potentially incorrect function logic.

The following code illustrates the problem:

```

Line 410    public static function getItemBrandGroup($options = null)
Line 411    {
Line 412        $db = DB::conn();
Line 414        $rows = $db->rows('SELECT id, name FROM item_brand_groups ORDER BY id ASC');
Line 415        if (!$rows) {
Line 416            throw new NotFoundException();
Line 417        }
Line 419        $data = [];
Line 420        foreach ($rows as $row) {
Line 421            $item_brand_group = new ItemBrandGroup($row);
Line 423            $rows_brands = $db->rows('SELECT item_brand_id FROM item_brands_item_brand_groups
→ WHERE item_brand_group_id = ? ORDER BY id ASC', [$row['id']]);
Line 424            if ($rows_brands) {
Line 425                $ids = [];
Line 426                foreach ($rows_brands as $rows_brand) {
Line 427                    $ids[] = $rows_brand['item_brand_id'];
Line 428                }
Line 429                if (count($ids) > 0) {
Line 431                    $options = ['ids' => join(',', $ids),];

```

Function `getItemBrandGroup` accepted parameter `$options` and did not use the value. The parameter was assigned on Line 431. As a result the value of the parameter was lost. Potential explanations include:

- ◆ The function is correct. It did not require parameter `$options` in the first place. The reuse of the name happened accidentally.
- ◆ The function has a logical error. It should use parameter `$options` before assignment but never did.

It is impossible to evaluate the risk of this issue due to ambiguity of the code.

Recommendation:

Do not reuse parameter names in the body of the function. Treat all parameters as read-only values.

NCCGRPPH-011 Hard-coded URLs and Cloud Bucket Names

Risk Rating

Low

Status

Open

Description:

A number of PHP files contained hard-coded web service and AWS S3 bucket URLs. The implementation requires high manual effort to maintain and is prone to human errors. This is indicative of a poor Software Development Lifecycle process.

The application was deployed in a number of environments (Development, CI, Production). Corresponding PHP configuration files are listed below.

- ◆ Development deployment
 - /client-api-aster/app/config/credentials/development.php
- ◆ CI deployment
 - /client-api-master/app/config/credentials/ci.php
- ◆ Production deployment
 - /client-api-master/app/config/credentials/production.php
- ◆ Locale Specific
 - /client-api-master/app/config/deferred_payment_delegator.php

The number and format of configuration options for environments were different from each other. This indicated the deployments were also different from each other. As a result, functional tests executed in one environment (CI deployment) could not guarantee successful deployment in the other environment (Production).

The list of hard-coded URLs, S3 bucket names and AWS ARNs is placed in the **Affected Resources** section.

Recommendation:

In a mature SDL environment it is beneficial to have very similar deployments in integration test and production. This allows to quickly root cause issues without interrupting production application. New development can also be tested efficiently in the integration test environment before deployment in production.

To achieve very similar deployments, the configuration options for both environments should be generated from the same template. This ensures number of options and their format matches between environments. The configuration options should be stored in non-executable format to avoid side effects of PHP execution.

It is recommended to re-engineer the configuration system as follows:

- ◆ Store all secrets, credentials and URLs in a secure database
- ◆ Create single configuration file per environment. Use high-privilege script with access to secrets database to generate configuration file
- ◆ Inject configuration to the process environment

This process will allow to separate life cycle of configuration from source code. The changes in URLs and S3 buckets can be tracked in the configuration database.

Affects:

File	Line	Notes
/client-api-master/app/config/deferred_payment_delegator.php	6, 12, 21, 27	AWS S3 buckets
/client-api-master/app/config/locale_specific_config/discount_sales_items_banner.php	10-12, 18-20	URLs

NCCGRPPHP-015 HTTP Basic Authentication in Use

Risk Rating Low

Status Open

Description:

The application was accessible from the Internet and made use of HTTP basic authentication. In HTTP basic authentication, HTTP requests are sent by the client with the Authorization header, containing the username and password in a base64-encoded string. This is not a significant security concern by itself and this issue has been included in the report for informational and completeness purposes.

The following code used HTTP basic authentication for a cURL connection:

```
Line 103      curl_setopt($curl, CURLOPT_USERPWD, sprintf("%s:%s", $api["BASICAUTH_USER"],  
→ $api["BASICAUTH_PASSWORD"]));
```

The username and password in HTTP basic authentication can be trivially decoded by an attacker who is able to intercept network traffic. For this reason, a secure encrypted transport mechanism should be used in conjunction with HTTP basic authentication. As the application did not enforce validation of TLS certificate (details in the issue finding NCCGRPPHP-014 on page 9), a suitably positioned attacker able to intercept network traffic, would be able to derive the username and password.

Recommendation:

Ensure that all traffic between the user's browser and the web server should be enforced over an encrypted channel. This can be achieved by using HTTPS, which uses SSL/TLS to protect the data. Do not disable TLS certificate validation as it effectively removes the protection of an encrypted channel.

3 Technical Summary

NCC Group was contracted by CLIENT to conduct a secure code review of the systems within scope in order to identify security issues that could negatively affect CLIENT's business or reputation if they led to the compromise or abuse of systems.

3.1 Scope

The code review was carried out remotely and included the following sections:

- ◆ PHP Source Code Review

The source code was downloaded from the following GitHub repositories master branches with the "Download ZIP" API:

Repository	Date	zip file	MD5
/CLIENT/CLIENT-api	26/07/2021	client-api-master.zip	2c6af7e43e25212b85e98f374ff42100

The source code was written in PHP. The following tools were used to assist with the review:

- ◆ Sonar QUBE (<https://www.sonarqube.org/>)
- ◆ Psalm (<https://psalm.dev/>)
- ◆ PHPStan (<https://github.com/phpstan/phpstan>)

3.2 Caveats

No caveats were recorded.

3.3 Post Assessment Cleanup

Access to private GitHub repositories for account NCCREVIEW should be disabled.

4 Executive Summary

This report presents the findings of the PHP source code review conducted on behalf of CLIENT. The assessment was conducted between START DATE and END DATE.

The system being assessed allowed users to perform logistical tasks of payment for goods, shipping of goods to customers.

4.1 Overview

The security posture of the systems within scope was found to be appropriate to the assets which required protection. Nevertheless, a small number of high risk issues were identified which should be addressed if the organisation's security model is to maintain an appropriate defense in depth basis. This illustrates the importance of ensuring that an otherwise robust security model cannot be undermined by isolated weaknesses.

The following table breaks down the issues which were identified by component and severity of risk (issues which are reported for information only are not included in the totals):

Component	Critical	High	Medium	Low	Total
	0	3	8	7	18
Total	0	3	8	7	18

4.2 Assessment Summary

The most significant issue identified in the PHP source code review was outdated and unsupported third-party software, lack of certificate validation and insufficient Software development Lifecycle (SDL).

Outdated and unsupported third-party software may contain security vulnerabilities with exploits publicly available. An attack on the third-party component may result in compromise of customer data and enterprise assets. The issue was deemed to be high risk. Lack of certificate validation exposes the enterprise to spoofing where the attacker impersonates trusted third-party vendor. This too may lead to compromise of customer data and enterprise assets. The issue was deemed to be high risk. Insufficient SDL results in low quality releases of software and increased risk of application-level security vulnerabilities. The issue was deemed to be high risk.

The prevention of SQL injection in the source code was implemented in an inconsistent manner. It was possible under specific conditions for MySQL to emulate prepared statements via dynamic SQL execution. This effectively would remove prevention of SQL injection in the project. The issue was deemed to be medium risk.

A number of issues were raised based on the Static Code Analysis report produced by the Sonar QUBE tool .These issue include usage of insecure MD5 and SHA1 hash functions, exception processing defects, incorrect creation of static objects, and variable scope defects. The code defects were deemed to be medium risk.

A number of file contained hard-coded secrets. This issue was discussed with the development team. A plan to migrate secrets to a secure service has been established and will be implemented in the coming months.

The remaining issues were all assessed to pose low risk. Nevertheless, it is recommended that these are reviewed and addressed so as to bring the web application within scope into line with security best practice. It is important to recognise that even low risk issues can be exploited in combination with other issues as part of a wider attack which seeks to compromise an environment or application. In addition, resolving lower risk issues can have the dual benefit of reducing the attractiveness of systems to opportunistic attackers as well as enhancing the overall security posture.

More detailed information on each of the issues which were identified is included in the Technical Details section of this report.

4.3 Strategic Recommendations

A proportion of the risk to which CLIENT was exposed was as a result of the use of outdated or unsupported software. It is therefore recommended that, in addition to addressing the individual issues which are set out in this report, the organisation's patching policy and procedures should also be reviewed to ensure that these issues do not recur once the individual instances documented here have been addressed.

In the long term the SDL process should be improved. The modern SDL process allows to consistently release quality software with minimal number of security issues. Therefore, it is important to invest in uplift of SDL.

In the long term it is important to achieve the environment parity. At the time of assessment the environments (production, test, development) were all different from each other. Configuration for each of the environment required manual effort and was customized. It is important to simplify the maintenance of environments by moving to common "Infrastructure as code" template. This requires a significant development effort and is suitable as a long term goal.

It is acknowledged that operational business requirements may mean that a risk has to be accepted (or partly accepted) rather than mitigated. Where this is the case, it is recommended that this is appropriately documented within the relevant Risk Register to ensure that the organisation maintains full visibility of the risk to which it is exposed.

5 Document Control

5.1 Client Confidentiality

This document contains CLIENT Client Confidential information and may not be copied without written permission.

5.2 Proprietary Information

The content of this document should be considered proprietary information and should not be disclosed outside of CLIENT.

NCC Group gives permission to copy this report for the purposes of disseminating information within your organisation or any regulatory agency.

6 Contact Info