F Y E O

Security Assessment of Bitgreen Payment Gateway

Bitgreen

September 2023
Version 1.0

Presented by:
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Strictly Confidential

Table of Contents

List of Figures

List of Tables

EXECUTIVE SUMMARY

Overview

Bitgreen engaged FYEO Inc. to perform a Security Assessment of Bitgreen Payment Gateway.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on August 14 - September 19, 2023, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following issues have been identified during the testing period. These should be prioritized for remediation to reduce the risk they pose:

- FYEO-BGGW-ID-01 – Environment Variables is logged to output
- FYEO-BGGW-ID-02 – Hardcoded Secret Key
- FYEO-BGGW-ID-03 – Insecure Storage of Sensitive Information
- FYEO-BGGW-ID-04 – Possible DoS by spamming payment requests
- FYEO-BGGW-ID-05 – Temporary data stored for payment requests can be edited / deleted
- FYEO-BGGW-ID-06 – Unhandled Errors and Promise Rejections
- FYEO-BGGW-ID-07 – Insecure Use of JavaScript's for...in Loop

Based on our review process, we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

The FYEO Review Team performed a Security Assessment of Bitgreen Payment Gateway. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a public repository at https://github.com/bitgreen/payment-gateway with the commit hash auditing.

Files included in the code review
pament-gateway ├── html/ │ └── usdstable.js ├── payment-gateway-webhook-stripe.js ├── payment-gateway-webhook-stripe.sh ├── payment-gateway.js ├── payment-gateway.sh ├── payment-validator-quicknode.sh └── payment-validator.js

Table 1: Scope

TECHNICAL ANALYSES AND FINDINGS

During the Security Assessment of Bitgreen Payment Gateway, we discovered:

- 2 findings with HIGH severity rating.
- 4 findings with MEDIUM severity rating.
- 1 finding with LOW severity rating.

The following chart displays the findings by severity.

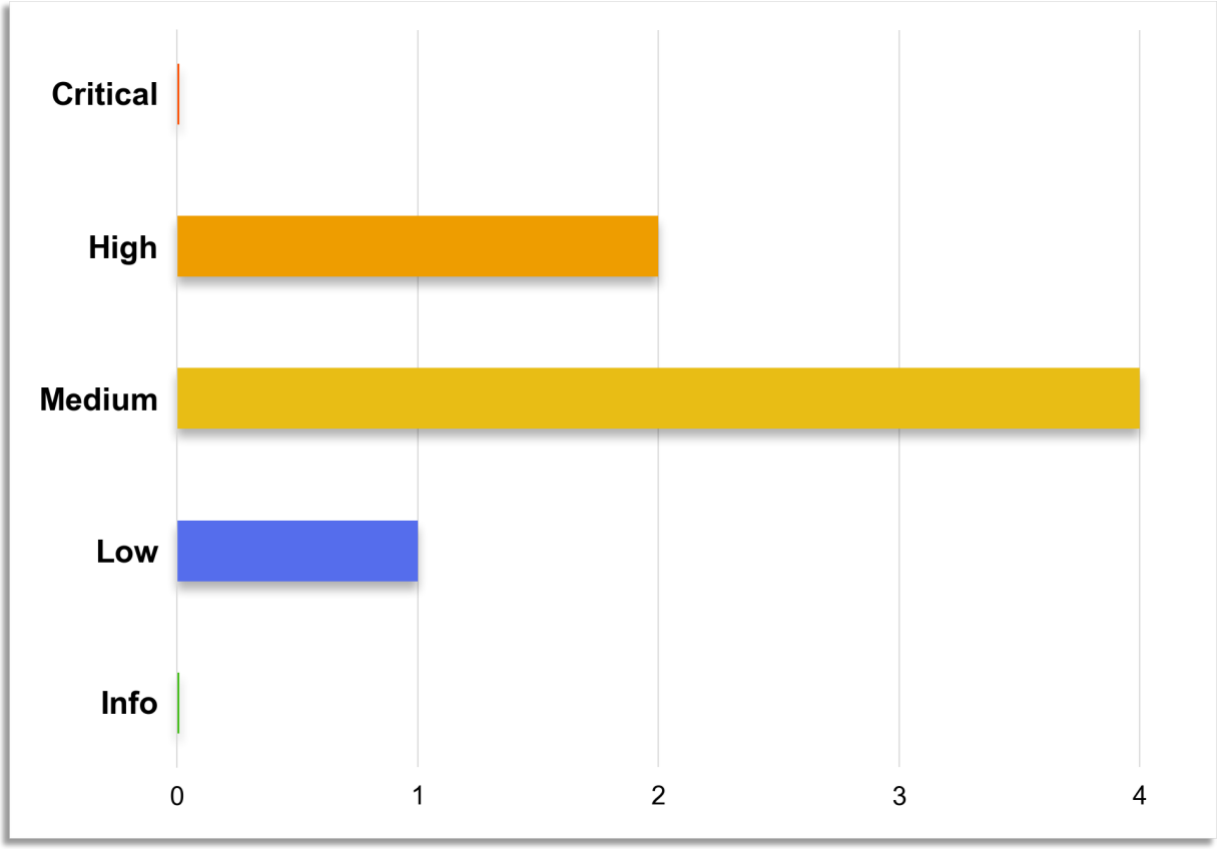


Figure 1: Findings by Severity

FINDINGS

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Severity	Description
FYEO-BGGW-ID-01	**High**	Environment Variable is logged to output
FYEO-BGGW-ID-02	**High**	Hardcoded Secret Key
FYEO-BGGW-ID-03	**Medium**	Insecure Storage of Sensitive Information
FYEO-BGGW-ID-04	**Medium**	Possible DoS by spamming payment requests
FYEO-BGGW-ID-05	**Medium**	Temporary data stored for payment requests can be edited / deleted
FYEO-BGGW-ID-06	**Medium**	Unhandled Errors and Promise Rejections
FYEO-BGGW-ID-07	**Low**	Insecure Use of JavaScript's for...in Loop

Table 2: Findings Overview

TECHNICAL ANALYSIS

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

CONCLUSION

Based on our review process, we conclude that the code implements the documented functionality to the extent of the reviewed code.

Technical Findings

General Observations

This project is in good shape even if it is fairly basic. In general from a security perspective we would like to see this rewritten in typescript since that would improve the security of the generated code.

A lot of the findings here are marked down from high to lower ratings since the code is only internal. This is because the code is managed on an internal server. However it is critical that the execution environment are properly secured. since a lot of sensitive data are stored in the environment variables.

ENVIRONMENT VARIABLES IS LOGGED TO OUTPUT

Finding ID: FYEO-BGGW-ID-01
Severity: **High**
Status: **Remediated**

Description

The code logs the value of the STRIPEAPIKEY environment variable, which is sensitive information.

Proof of Issue

File name: payment-gateway-webhook-stripe.js
Line number: 34

```
console.log("Configuring Stripe: ",STRIPEAPIKEY);
```

Severity and Impact Summary

An attacker could gain access to the Stripe API key, which could lead to unauthorized transactions.

Recommendation

Remove the log output of sensitive environment variables. If necessary, log only that the variable has been set, not its value.

HARDCODED SECRET KEY

Finding ID: FYEO-BGGW-ID-02
Severity: **High**
Status: **Remediated**

Description

The secret key for Stripe API is hardcoded in the source code. This is a security risk as anyone with access to the source code can misuse the key.

Proof of Issue

File name: payment-gateway.js
Line number: 9

```
const stripe =
require('stripe')('sk_test_51MDq0VKluWo1Xbjw9dB5xdWgGUulDA6ckewLKyz4wdQee6yrxX5QhhJ5ob
lHgWhVApt2VDOlHH0JxARlaimxnC5s00Laa66Evw');
```

Severity and Impact Summary

If exploited, an attacker can use the secret key to make unauthorized transactions, access sensitive data, and potentially compromise the payment system.

Recommendation

The secret key should be stored securely. Use secure storage mechanisms like HashiCorp Vault, AWS Secrets Manager, or Azure Key Vault to store sensitive information.

INSECURE STORAGE OF SENSITIVE INFORMATION

Finding ID: FYEO-BGGW-ID-03
Severity: **Medium**
Status: **Remediated**

Description

The scripts use environment variables to store sensitive information such as SUBSTRATE, SSL_CERT, SSL_KEY, SUBSTRATECHAIN, MNEMONIC, MNEMONIC2, STRIPEAPIKEY, STRIPESIGKEY, TOKENADDRESS, ABI, WALLETADDRESS,BLOCKCHAIN, BLOCKCHAINCODE, BLOCKSCONFIRMATION, MINVALIDATIONS. This is not a secure way to store such sensitive information.

Proof of Issue

File name: payment-gateway.js
Line number: 13, 21

```
const SUBSTRATE = process.env.SUBSTRATE;
...
const SSL_CERT = process.env.SSL_CERT
const SSL_KEY = process.env.SSL_KEY
```

File name: payment-gateway-webhook-stripe.js
Line number: 8

```
const SUBSTRATECHAIN = process.env.SUBSTRATECHAIN;
...
const MNEMONIC = process.env.MNEMONIC;
...
const MNEMONIC2 = process.env.MNEMONIC2;
...
const STRIPEAPIKEY = process.env.STRIPEAPIKEY;
...
const STRIPESIGKEY = process.env.STRIPESIGKEY;
```

File name: payment-validator.js
Line number: 8

```
const TOKENADDRESS = process.env.TOKENADDRESS;
...
const ABI = process.env.ABI;
...
const WALLETADDRESS = process.env.WALLETADDRESS;
...
const BLOCKCHAIN = process.env.BLOCKCHAIN;
...
const BLOCKCHAINCODE = process.env.BLOCKCHAINCODE;
...
const MNEMONIC = process.env.MNEMONIC;
...
const SUBSTRATECHAIN = process.env.SUBSTRATECHAIN;
...
const BLOCKSCONFIRMATION = process.env.BLOCKSCONFIRMATION;
```

```
...
const MINVALIDATIONS = process.env.MINVALIDATIONS;
```

Severity and Impact Summary

If exploited, an attacker can gain access to these sensitive details which could lead to unauthorized actions.

Recommendation

Use secure storage mechanisms like HashiCorp Vault, AWS Secrets Manager, or Azure Key Vault to store sensitive information or other configuration services that provide encryption.

POSSIBLE DOS BY SPAMMING PAYMENT REQUESTS

Finding ID: FYEO-BGGW-ID-04
Severity: **Medium**
Status: **Remediated**

Description

The `paymentrequest` is public and stores data in the `paymentrequests` table. This table is said to be fairly small with somewhat temporary data. Therefore no index was added to facilitate lookups done by the following statements.

Proof of Issue

File name: payment-gateway.js
Line number: 289

```
delete from paymentrequests where sender=$1 and recipient=$2 and amount=$3 and
originaddress=$4 and chainid=$5
```

File name: payment-validator.js
Line number: 116

```
SELECT * from paymentrequests where sender=$1 and recipient=$2 and amount=$3 and
chainid=$4
```

Severity and Impact Summary

It is possible for users to spam this endpoint which would fill this table with lots and lots of rows. At a certain point the `DELETE` statement will struggle without a key. This will also affect the payment validators lookup query.

Recommendation

Prevent spamming of the endpoint and add search indexes if required.

TEMPORARY DATA STORED FOR PAYMENT REQUESTS CAN BE EDITED / DELETED

Finding ID: FYEO-BGGW-ID-05
Severity: **Medium**
Status: **Remediated**

Description

The `paymentrequest` route takes a number of user provided query values which are then used in a delete query run against the `paymentrequests` table. To delete items one has to know a number of variables as can be seen in the code below. Of those `sender`, `recipient`, `amount` and `chainid` appear to be public, leaving only `originaddress`. If `originaddress` can be obtained, guessed or brute-forced, someone can delete other peoples data, specifically the `referenceid`.

Proof of Issue

File name: payment-gateway.js
Line number: 276

```
const queryText="SELECT COUNT(*) AS tot from paymentrequests where referenceid=$1";
rs=await client.query(queryText, [referenceid]);

...

queryText = 'delete from paymentrequests where sender=$1 and recipient=$2 and
amount=$3 and originaddress=$4 and chainid=$5';
await client.query(queryText,
[sender,recipient,amount,originaddress,parseInt(chainid)]);

...

queryText = 'INSERT INTO
paymentrequests(referenceid,token,sender,recipient,amount,created_on,originaddress,cha
inid,blockhash) values($1,$2,$3,$4,$5,current_timestamp,$6,$7,$8)';
await client.query(queryText,
[referenceid,token,sender,recipient,amount,originaddress,parseInt(chainid),blockhash.t
oString()]);
```

The user here is also in control of the `referenceid` which is stored and could potentially be used to insert data:

```
const queryText="SELECT * from paymentrequests where sender=$1 and recipient=$2 and
amount=$3 and chainid=$4";

...

if(orderid.search(",")==-1)
    ao.push(orderid);
else
    ao=orderid.split(",");

...

for(x in ao){
```

```
    ...
    const queryText = 'INSERT INTO
validationsqueue(validatoraddress,buyorderid,txhash,chainid) values($1,$2,$3,$4)';
```

However, while the loop iterates on `ao` the query will always use `orderid` unlike the else branch which uses `ao[x]` and since there is a primary key on `orderid` the iteration would only work once before causing errors.

Severity and Impact Summary

This code appears to be not well thought out and may lead to potential abuse since part of the data is read from a public blockchain, it will be known to others.

Recommendation

Make sure this functionality is correctly implemented and does not allow people to delete or otherwise modify data of other users.

UNHANDLED ERRORS AND PROMISE REJECTIONS

Finding ID: FYEO-BGGW-ID-06
Severity: **Medium**
Status: **Remediated**

Description

The code does not handle promise rejections, return values and exceptions. If any of the promises fail, the application may crash or behave unexpectedly. There are also several un-handled exceptions. Asynchronous JavaScript and throw statements are used throughout the code with little error and exception handling.

Proof of Issue

File name: payment-validator.js
Line number: 60

```
// execute a main loop for async oeprations
mainloop();

async function mainloop(){ ... }
```

File name: payment-validator.js
Line number: 115

```
try {
    ...
} catch (e) {
    throw e;
}
```

File name: payment-validator.js
Line number: 183

```
validate_payment(rs['rows'][0]['referenceid'],BLOCKCHAINCODE,event['transactionHash'],
keys,api);

async function validate_payment(orderid,blockchainid,tx,keys,api){ ... }
```

The function call does not wait on the promise nor are possibly thrown exceptions handled:

```
try {
  const queryText = 'INSERT INTO
validationsqueue(validatoraddress,buyorderid,txhash,chainid) values($1,$2,$3,$4)';
  await client.query(queryText, [keys.address,orderid,tx,blockchainid]);
} catch (e) {
    throw e;
}
```

File name: payment-gateway-webhook-stripe.js
Line number: 42

```
// we execute the main loop in an async function
mainloop();
```

```
async function mainloop(){...}
```

File name: payment-gateway-webhook-stripe.js
Line number: 112, 227

```
// store the payment received
await
store_orders_paid(rs.rows[0]['referenceid'],api,client,pi.currency,rs.rows[0]['stripei
d']);

...

try {
      const queryText = 'INSERT INTO
paymentsreceived(referenceid,sender,recipient,amount,fees,created_on,selleraddress,tok
en,chainid,paymentid,blockhash,nrvaliation,minvalidation)
values($1,$2,$3,$4,$5,current_timestamp,$6,$7,$8,$9,$10,1,1)';
      await client.query(queryText,
[v.orderId,"","",amount,fees,selleraddress,token,0,stripeid,hash.toHex()]);
} catch (e) {
        throw e;
}
```

The caller of `store_orders_paid` does not catch exceptions thrown, neither are they handled in `mainloop` nor outside of `mainloop`. The returned value is also ignored which appears to be unused and a copy and paste from another function.

File name: payment-gateway.js
Line number: 30

```
console.log("Payment Gateway 1.0 - Starting");
mainloop();
```

File name: payment-gateway.js
Line number: 185

```
try {
      const queryText = 'INSERT INTO
striperequests(stripeid,referenceid,amount,created_on,status)
values($1,$2,$3,current_timestamp,$4)';
      await client.query(queryText, [paymentIntent.id,r,(a/100),status]);
} catch (e) {
   throw e;
}
```

File name: payment-gateway.js
Line number: 275, 293, 301, 311

```
try{
...
} catch (e) {
    throw e;
}
```

Severity and Impact Summary

Application crash or unexpected behavior.

Recommendation

Always handle promise rejections. Use try-catch blocks around your async-await code.

INSECURE USE OF JAVASCRIPT'S FOR...IN LOOP

Finding ID: FYEO-BGGW-ID-07
Severity: **Low**
Status: **Remediated**

Description

The code uses for…in loops to iterate over arrays. This is not recommended because the for…in loop enumerates properties in the prototype chain, and if the array has been modified, unexpected results may occur.

Proof of Issue

File name: payment-gateway-webhook-stripe.js
Line number: 140, 165, 195

```
for(x in ao){
    ...
}
```

File name: payment-validator.js
Line number: 197, 225

```
for(x in ao){
    ...
}
```

Severity and Impact Summary

This is a maintenance concern and could lead to unexpected behavior in the application, potentially leading to logical errors or incorrect data processing.

Recommendation

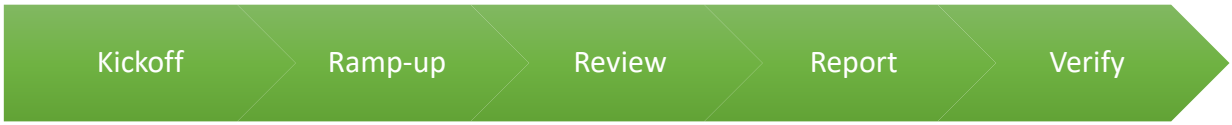
Use a for…of loop or Array.prototype.forEach() instead of a for…in loop when iterating over arrays.

OUR PROCESS

METHODOLOGY

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

Figure 2: Methodology Flow



KICKOFF

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

RAMP-UP

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

REPORTING

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

VERIFY

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

ADDITIONAL NOTE

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.

THE CLASSIFICATION OF VULNERABILITIES

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations