PUBLIC

Code Assessment

of the GSN Integration Fix

March 31, 2023

Produced for

by

Contents

1 Executive Summary

Dear Enzyme Team,

On March 28th, 2023 an issue was reported through the Immunify platform regarding Enzyme's integration with the Gas Relay Network.

The ChainSecurity team responded immediately and supported the Enzyme team in understanding and mitigating the issue.

Furthermore, the following day a team in ChainSecurity further investigated the issue and possible relevant issues and reviewed the proposed fix.

From our investigation, no further issues were uncovered.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	1
• Code Corrected	1
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the GSN Integration Fix repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	30 March 2023	a11399942d74eb93cdd508deb2cd794e66f8ce64	GSN fix

For the solidity smart contracts, the compiler version `0.6.12` was chosen.

In scope are the following contracts:

`contracts/release/infrastructure/gas-relayer`:

- `GasRelayPaymasterFactory`
- `GasRelayPaymasterLib`
- `GasRelayRecipientMixin`
- `bases/GasRelayPaymasterLibBase1.sol`
- `bases/GasRelayPaymasterLibBase2.sol`

2.1.1 Excluded from scope

All the contracts not explicitly mentioned in scope are considered out-of-scope. More specifically, all the contracts of the GSN network with which Enzyme interacts are considered out-of-scope and are expected to work correctly. All the external libraries used such as the Openzepplin libraries are also considered out-of-scope. Finally, the authorized parties of an Enzyme fund e.g., the owner, the managers, etc are considered trusted.

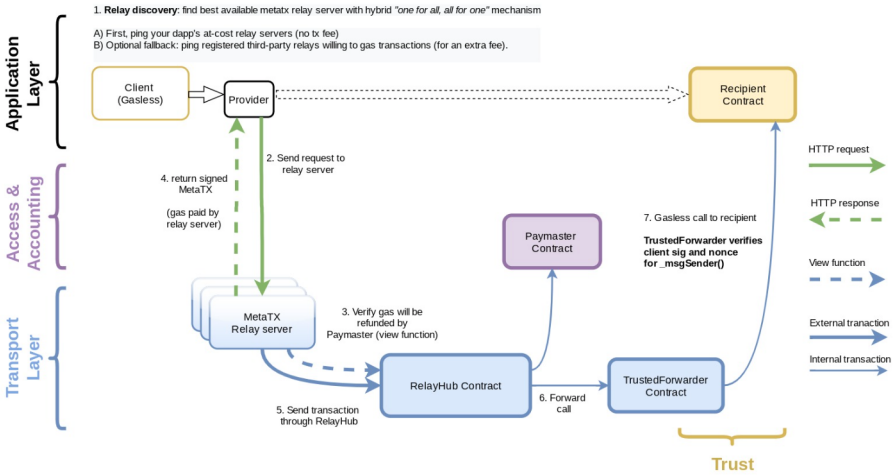
2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Avantgarde Finance integrates with Gas Station Network (GSN) in order to enable gasless transactions for the authorized users of vaults.

2.2.1 GSN Architecture



GSN allows users to submit meta transactions that are executed by another user who pays for the gas fees. A user signs the meta transaction and submits it to the relay network. A participant of the network executes the transaction by sending it to the RelayHub Contract. The executor makes sure that their transaction costs will be refunded by querying the paymaster contract. The meta transaction is then forwarded to the `TrustedForwarder` Contract which eventually will execute a call to the recipient contract. After the transaction is executed, the executor gets refunded.

2.2.2 Enzyme-GSN integration

In the Enzyme case, the paymaster contract is a proxy whose logic is implemented in the GasRelayPaymasterLib library. The paymaster contract receives and performs some checks on the `_relayRequest`. `_relayRequest` includes all the metadata required for the transaction to be relayed. Part of the `_relayRequest` is the forwarder address which is then used by the RelayHub.

In the normal case, the forwarder should be the `TrustedForwarder` used by Enzyme. As a matter of fact, Enzyme contracts will check that a call is made from the trusted forwarder. The relevant logic is implemented in `GasRelayRecipientMixin.__msgSender`.

After the call is completed, `GasRelayPaymasterLib.postRelayedCall` is called. If `_relayData` specifies it (`paymasterData == true`), it is checked if the available amount for the next call suffices. If it doesn't, an auto top-up is performed i.e., an amount is withdrawn from the vault and sent to the RelayHub Contract.

2.2.3 Roles and Trust Model

The roles and the trust model remain the same as described in ChainSecurity's security review of the system.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	1

- Calls to Arbitrary Forwarder Code Corrected

High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

5.1 Calls to Arbitrary Forwarder

Security Critical Version 1 Code Corrected

As the `GasRelayPaymasterLib.preRelayedCall` doesn't check the forwarder's address, a malicious user could submit an arbitrary address to a malicious forwarder they control. Since the malicious forwarder doesn't make any call to an Enzyme contract the call will be successful and thus the participant who executed the transaction will be refunded. Moreover, due to the auto top-up feature, repetitive execution of such transactions will gradually drain the vault from WETH.

Code corrected:

`GasRelayPaymasterLib.preRelayedCall` checks which is the specified forwarder and reverts if the forwarder specified is not Enzyme's trusted forwarder. This means that all GSN calls to Enzyme should only go through the trusted forwarder which is now checked.

In addition to this fix, throttling regarding the auto top-up was introduced as well as constraints for the RelayRequest parameters which limit the amount a GSN executor can receive after executing a transaction. More specifically, the following steps were taken:

1. a cooldown between deposits (e.g., 24 hours) was implemented
2. max total deposit is now immutable rather than constant (e.g., 1 WETH)
3. max `pctRelayFee` was introduced (`RELAY_FEE_MAX_PERCENT`)
4. max `baseRelayFee` was introduced (`RELAY_FEE_MAX_BASE`)