PUBLIC

Code Assessment

of the Proxy Wallet Factories Smart Contracts

11 April, 2024

Produced for

Polymarket

by

CHAINSECURITY

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help Polymarket with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Proxy Wallet Factories according to Scope to support you in forming an opinion on their security risks.

Polymarket implements two factories for creating wallets that allow Polymarket to execute transactions on behalf of users.

The most critical subjects covered in our audit are functional correctness, signature handling and correct interactions with the Gas Station Network (GSN).

Security regarding functional correctness and signature handling are high.

In summary, we find that the codebase provides a high level of security.

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	0
High-Severity Findings	0
Medium-Severity Findings	2
• Specification Changed	2
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 Proxy Wallet Factories 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	27 October 2023	b148fa19a6deee72ed33c34bf15a1bd569d9d13b	Initial Version

For the solidity smart contracts of the ProxyWallet, the compiler version `0.5.15` was chosen. For the safe factory solidity smart contracts the compiler version `0.8.4` was chosen.

2.1.1 *Included in scope*

This report covers two proxy wallet factory implementations.

- packages/proxy-factory/contracts/GSNModules/GSNLib.sol
- packages/proxy-factory/contracts/GSNModules/GNSModule01.sol
- packages/proxy-factory/contracts/GSNModules/GNSModule03.sol
- packages/proxy-factory/contracts/libraries/FactoryLib.sol
- packages/proxy-factory/contracts/libraries/MemcpyLib.sol
- packages/proxy-factory/contracts/libraries/RevertCaptureLib.sol
- packages/proxy-factory/contracts/libraries/RStoreLib.sol
- packages/proxy-factory/contracts/libraries/SliceLib.sol
- packages/proxy-factory/contracts/libraries/StringLib.sol
- packages/proxy-factory/contracts/ProxyWallet/ProxyWallet.sol
- packages/proxy-factory/contracts/ProxyWallet/ProxyWalletFactory.sol
- packages/proxy-factory/contracts/ProxyWallet/ProxyWalletLib.sol
- packages/safe-factory/contracts/SafeProxyFactory.sol
- packages/safe-factory/contracts/Deps.sol

2.1.2 *Excluded from scope*

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.

Polymarket offers two types of proxy wallet factories to help Polymarket execute transactions (and pay their gas costs) on behalf of users. The first type of proxy wallet factory creates a proxy wallet using the Gas Station Network (GSN) v1 to pay for gas costs. The second type of proxy wallet factory creates a `GnosisSafe` proxy wallet.

Both wallet types use a user's EOA address as salt so that it is later possible to verify that a wallet belongs to a certain user address.

2.2.1 *ProxyWalletLib*

`ProxyWalletLib` is a library that contains the proxy wallet creation logic and functionality to write the following data to certain storage slots:

- The address of the proxy `implementation`.
- The owner.
- The GSN module.

It also defines a salt used by the `ProxyWalletFactory` for deployment of the wallet implementation.

A `proxyCall()` function can be used to perform calls or alternatively delegatecalls by the implementing contract.

2.2.2 *FactoryLib*

`FactoryLib` is a library that contains the logic to create a new proxy wallet using the following functions:

- `computeCreationCode()` computes the creation code of the proxy wallet from hardcoded EVM bytecode and arguments such as the address of the deployer and the address of the implementation contract. This bytecode, during execution, also calls back to the deployer's `cloneConstructor()` function.
- `create2Clone()` creates a new proxy wallet by executing a CREATE2 opcode with the bytecode generated by `computeCreationCode()`.

2.2.3 *ProxyWalletFactory*

`ProxyWalletFactory` is a factory for creating proxy wallets. It accepts relayed calls from the GSN `RelayHub` that can be forwarded to any of the created proxy wallets. For this, the `RelayHub` calls `acceptRelayedCall()` on each interaction to determine if a given call will be paid for. `acceptRelayedCall()` in turn calls the GSN module which is set to `GSNModule01` by default (but can be changed).

The `proxy()` function first checks if there already exists a proxy wallet for a given `_msgSender()` using the function `maybeMakeWallet()`. If there is none, it creates a new proxy wallet by calling `makeWallet()`.

`makeWallet()` creates a proxy from the `implementation` deployed in the constructor through `deployImplemention()`.

`cloneConstructor()` allows the factory to define a function that is delegatecalled from the creation code of the proxy wallet during contract creation. In the current implementation, the function is empty.

2.2.4 ProxyWallet

`ProxyWallet` is the implementation contract to which all the proxy wallets will delegate their calls. Each wallet is owned by the factory which is the only allowed caller to a proxy wallet's `proxy()` function. It also implements `onERC1155BatchReceived()` and `onERC1155Received()` to allow receival of ERC-1155 tokens.

2.2.5 Gas reimbursement and GSN modules

`GSNModule01` is used by Polymarket to only accept relayed calls that call the `proxy()` function of the factory.

`GSNModule03` additionally only accepts relayed calls from a whitelisted relayer.

Polymarket relays calls on the GSN `RelayHub` by creating signatures with minimal gas costs and then executing them. This way, the majority of gas costs are not handled inside the `RelayHub` contract but by the account that handles the transaction. It is therefore not profitable for external relayers to handle these signatures.

2.2.6 SafeProxyFactory

`SafeProxyFactory` allows any user to create a new `GnosisSafe` proxy wallet. It is instantiated with the address of the `masterCopy` and the address of the `fallbackHandler` to be used by the proxy wallets. Polymarket can create a wallet by calling `createProxy()` with an EIP-712 compliant signature signed by the user the wallet is created for.

2.2.7 Roles and Trust Model

Proxy wallets are created on behalf of users. Each wallet's `proxy()` function can only be called by `ProxyWalletFactory.proxy()` which in turn can be called by the user or the `RelayHub` on behalf of the user (as long as a valid signature exists).

`SafeProxyFactory` also allows Polymarket to create wallets on behalf of the user by accepting a signature the user has to sign. The created Gnosis wallets are 1-of-1 multisig wallets granting the user sole rights. Calls to the wallet can also be performed by Polymarket on behalf of the user based valid on signatures. The signature does not include the owner and the `fallbackHandler`, however, this is acceptable as the contract is immutable and the `fallbackHandler` cannot be changed later.

Both wallet systems are therefore completely trustless.

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 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors

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

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

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

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

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	2

- Proxy Wallet DoS Specification Changed
- RelayHub Account Can Be Drained Specification Changed

Low-Severity Findings	0

6.1 Proxy Wallet DoS

Security Medium Version 1 Specification Changed

CS-PMPWF-002

As can be seen in RelayHub account can be drained, all the funds of the `ProxyWalletFactory` account on the GSN `RelayHub` can be completely drained. Since Polymarket currently executes signatures with a `gasPrice` set to 1, the calls can only be successfully relayed as long as there are funds available. If an attacker were to completely drain the account, no new transactions can be performed until either:

- The process is changed to create signatures with a `gasPrice` of 0.
- The `RelayHub` balance is replenished.

Specification changed:

Relayer clients will now use a gas price of 0.

6.2 RelayHub Account Can Be Drained

Security Medium Version 1 Specification Changed

CS-PMPWF-001

Relayed calls executed over the GSN `RelayHub` are calling `ProxyWalletFactory.acceptRelayedCall()` to determine, whether a call is paid for by the funds deposited to the `RelayHub` by Polymarket. Using the `GSNModule01` always accepts any call that targets the factory's `proxy()` function.

Since neither the `relayer` nor the `transactionFee` are verified, any account can perform a transaction on the `RelayHub` with a transactionFee set to an amount that completely drains the balance of the `ProxyWalletFactory`. This fee is then credited to the relayer's balance and can be withdrawn.

Specification changed:

Polymarket now publishes signatures with a gas price of `0`, allowing them to execute the transactions without any balance changes in the `RelayHub`. The balance in the `RelayHub` can therefore be withdrawn completely, mitigating any possible draining.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Ambiguous Naming

Informational **Version 1** **Risk Accepted**

CS-PMPWF-003

The name of the function `ProxyWalletFactory.deployImplementation()` does not start with an underscore although all other internal functions do so.

7.2 Code Copies

Informational **Version 1** **Risk Accepted**

CS-PMPWF-004

`ProxyWalletLib` and `FactoryLib` contain overlapping code parts (e.g., `computeCreationCode()`). The libraries could be merged into one.

7.3 Gas Optimizations

Informational **Version 1** **Risk Accepted**

CS-PMPWF-005

The following gas inefficiencies can be improved:

- The `implementation` address in `ProxyWalletFactory` could be replaced by an `immutable` variable.
- The `whitelistedRelayer` in `GSNModule03` can be `immutable`.
- The storage variables of `SafeProxyFactory` can be `immutable`.

7.4 Irregular Rejection Code

Informational **Version 1** **Risk Accepted**

CS-PMPWF-008

All GSN modules return `1` when rejecting a call. According to the GSN specification, however, the correct code for rejection is at least `11` as codes `1 - 10` are reserved.

7.5 Lost Revert Errors

Informational **Version 1** **Risk Accepted**

CS-PMPWF-006

`RevertCaptureLib` can not handle custom error messages. Instead, a default error is returned which removes information in the output.

7.6 Missing Event

Informational **Version 1** **Risk Accepted**

CS-PMPWF-007

`ProxyWalletFactory.makeWallet()` does not emit an event on successful proxy wallet creation.