macro

Grateful A-1

Security Audit

July 4th, 2023
Version 1.0.0

Presented by 0xMacro

Table of Contents

Introduction

This document includes the results of the security audit for Grateful's smart contract code as found in the section titled ‘Source Code’. The security audit was performed by the Macro security team from June 4, 2023 to June 9, 2023.

Note: Grateful contracts are based on the Synthetix’s Router Proxy Architecture using the unstructured storage pattern. The audit was focused around the actual protocol logic, meaning that code related to the Proxy Architecture including the actual deployment of the contracts were outside of the scope of this audit.

The purpose of this audit is to review the source code of certain Grateful Solidity contracts, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Disclaimer: While Macro’s review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

Overall Assessment

The following is an aggregation of issues found by the Macro Audit team:

Severity	Count	Acknowledged	Won't Do	Addressed
High	1	-	-	1
Low	4	-	1	3
Code Quality	3	-	1	2
Informational	1	-	1	-
Gas Optimization	1	-	-	1

Grateful was quick to respond to these issues.

Specification

Our understanding of the specification was based on the following sources:

- Discussions on Telegram with the Grateful team.
- Available documentation: https://grateful.gitbook.io/docs/

Trust Model, Assumptions, and Accepted Risks (TMAAR)

Grateful contracts use an upgradable proxy architecture, allowing the owner to upgrade to a new implementation. In addition, the owner has the following privileges:

- **Configuration:** Change solvency and liquidation time.
- **Fee:** Set protocol fee.
  Note: Protocol fees are taken from the Giver who creates the subscription and are handled through subscriptions to the *Grateful Treasury*. According to Grateful team, there will be no fees charged at the beginning, but users need to be aware that fees can be increased anytime.

- **Vault management:** Add vaults, change min and max rate, deactivate/pause vaults. *Note:* Users can't withdraw their funds on pauses vaults.

Source Code

The following source code was reviewed during the audit:

- **Repository:** grateful-contracts
- **Commit Hash:** `87b9b31b1d5e7a3315d1b70ea22b409427bfed79`

Specifically, we audited the following contracts within this repository:

Contract	SHA256
contracts/modules/BalancesModule.sol	`0971a6aa93ed9794bba40fecc018c6e769a4f80a7e662ea83fe9e6d10a27ac4f`
contracts/modules/ConfigModule.sol	`0eb2425c87c63cc7b816b3bdc7f3734e49b97f7bc255fd4ee43a46624d25784b`
contracts/modules/CoreModule.sol	`3ab86d52ac1dcfc9c38a2a15407a094d9a232365e3a39c281687fedf225a2e5f`
contracts/modules/FeesModule.sol	`943fbed60efc0fbeb688ec87152ffff31fea3e2cc1e4216af9a45c470b6eeb4a`
contracts/modules/FundsModule.sol	`59f9097b841d5acc3e80180fbacf69b7aa86819e0e8392662debec776bd1f4ac`
contracts/modules/LiquidationsModule.sol	`f27dec7c9c6254dc040211d02441da359936e161c0d9c6cd8490b6be9f5e6e5d`
contracts/modules/MulticallModule.sol	`797b3e38cdcf77c5f852fe547dea309cd386d34a4829354b3c8094e38f6c3397`
contracts/modules/ProfilesModule.sol	`bb258f1519b5fa8bc690a4e6bc0916c52e27596101aafb1a229fa837af44bef6`
contracts/modules/SubscriptionsModule.sol	`4219752d73fe1a90554ec94c1625c4105e363258b73d691b03afb1846b0fd918`
contracts/modules/VaultsModule.sol	`a839706de5a4533fce703be0a135e6ae6ec91899b0908816aeb706170f64f6f5`
contracts/errors/BalanceErrors.sol	`7888ca6b2210684499a37649e4d73c9127053a9`

Contract	SHA256
	31523f6416c3087c69577f8fa
contracts/errors/InputErrors.sol	88bcedbf34825368bda4afda02418702c9a0dcbaac659c3c21576c70370087f8
contracts/errors/ProfileErrors.sol	08b9d1a15e1ffe9d91add316e2dc665469e6103f99b3fdc2f83f2133317ade86
contracts/errors/SubscriptionErrors.sol	f4cf09a5ebb8777ad3698f43589237a90e415ebaad756589f924fe41d135feab
contracts/errors/VaultErrors.sol	ca15fe502ae648629314feb677ab497997895f34c5574010ff161b46158b5c59
contracts/storage/Balance.sol	356a0cbd572a74ea55beb722e00b8aa5897bb318c480c7764b2576ad67482072
contracts/storage/Config.sol	726b1323d6ff5385bb2b85ea67f61ee85ee35dc3e8c07fe73d721cca6e9a5a3c
contracts/storage/Fee.sol	5813735dfb27ce8fea2f1adacd3fac9b9da32f5276b6cd35c00e8d3cb1750f05
contracts/storage/Profile.sol	b9de91d57cd5b5ec4951b371764e799ed41288b6e38ad7c9cb9f1d8ff99ee3f3
contracts/storage/ProfileNft.sol	7097f33d16ad550470e11c43c855c1540c0aaf60eff2edc162c9217884097aba
contracts/storage/ProfileRBAC.sol	585a581c0f3ccaaf888d17e8b93a0aa82126e9b0d2c8f1cc126543b204471917
contracts/storage/Subscription.sol	839c0585a3e073de6f5f788fec00d358dcf07bd71475563d603958598958136e
contracts/storage/SubscriptionNft.sol	7e89d835e3f5219dfa8f7cea0beac244dacf711f5c159e6d7f9ed73ff395d593
contracts/storage/SubscriptionRegistry.sol	d03be4bbbce80ee0b83ffb53ad618c81be7743093f2e701b2b608a9901bbaa79
contracts/storage/Vault.sol	62cfb785203af65bc681d5d3ce3d5330fa79af700a3c7ea01d53d0c2cd8ac88a

Contract	SHA256
contracts/utils/ProfileRenderer.sol	`5d7f0650c563bcb685354c5f1333797dc2ea0a36c21e855acddde8c3a0104c7`
contracts/utils/RendererUtils.sol	`8a030c4a20b31d754d4de9878d2d0e1d83af86d73356cd449a06cb595c16ee52`
contracts/utils/SubscriptionRenderer.sol	`269e40f208b35b677b3225488ff50ca3ef997f5138177c5fd68369e1adf3ff44`
contracts/utils/SubscriptionUtil.sol	`ca952837e4e6afcb93cbe7afd1ffe50aba3361f4daf3b37a5a97434187ebbc30`
contracts/utils/VaultUtil.sol	`c98e00d7efe7212fea19b7911acf4b6445888105beb3d05340b16640e0ba2e17`

Note: This document contains an audit solely of the Solidity contracts listed above. Specifically, the audit pertains only to the contracts themselves, and does not pertain to any other programs or scripts, including deployment scripts.

Issue Descriptions and Recommendations

Click on an issue to jump to it, or scroll down to see them all.

- ~~H-1~~ Malicious user can prevent getting liquidated
- ~~L-1~~ Subscription owner is not transferred when profile is transferred
- ~~L-2~~ Call to liquidate doesn’t validate `liquidatorId`
- ~~L-3~~ Duration property for subscriptions returns wrong value when unsubscribed
- L-4 Adding a new vault can fail when vault implementation is used in another vault already
- ~~Q-1~~ `isVaultPaused` always returns wrong value
- ~~Q-2~~ `tokenURI` returns value for invalid token ids
- Q-3 `_EDIT_PERMISSIONS` not used
- ~~G-1~~ `_hasEnoughBalance` makes `_isSolvent` redundant
- I-1 Enabling burning of `GratefulProfile` NFTs would break the `createProfile` operation

Security Level Reference

We quantify issues in three parts:

1. The high/medium/low/spec-breaking **impact** of the issue:
    - How bad things can get (for a vulnerability)
    - The significance of an improvement (for a code quality issue)
    - The amount of gas saved (for a gas optimization)
2. The high/medium/low **likelihood** of the issue:
    - How likely is the issue to occur (for a vulnerability)
3. The overall critical/high/medium/low **severity** of the issue.

This third part – the severity level – is a summary of how much consideration the client should give to fixing the issue. We assign severity according to the table of guidelines below:

Severity	Description
(C-x) Critical	We recommend the client **must** fix the issue, no matter what, because not fixing would mean **significant funds/assets WILL be lost.**
(H-x) High	We recommend the client **must** address the issue, no matter what, because not fixing would be very bad, *or* some funds/assets will be lost, *or* the code's behavior is against the provided spec.
(M-x) Medium	We recommend the client to **seriously consider** fixing the issue, as the implications of not fixing the issue are severe enough to impact the project significantly, albiet not in an existential manner.
(L-x) Low	The risk is small, unlikely, or may not relevant to the project in a meaningful way. Whether or not the project wants to develop a fix is up to the goals and needs of the project.
(Q-x) Code Quality	The issue identified does not pose any obvious risk, but fixing could improve overall code quality, on-chain composability, developer ergonomics, or even certain aspects of protocol design.
(I-x) Informational	Warnings and things to keep in mind when operating the protocol. No immediate action required.
(G-x) Gas Optimizations	The presented optimization suggestion would save an amount of gas significant enough, in our opinion, to be worth the development cost of implementing it.

Issue Details

~~H-1~~ Malicious user can prevent getting liquidated

TOPIC	STATUS	IMPACT	LIKELIHOOD
Incentive Design	Fixed	High	Medium

A malicious user can perform a sandwich attack on the liquidate call which results in the liquidate call to be reverted. Consider the following scenario:

1. Attacker creates \`giver\` profile \`p1\`
2. Attacker creates \`creator\` profile \`p2\`
3. Attacker subscribes \`p1\` to \`p2\` with the maximum allowed \`rate\` value (to make the attack more profitable)
4. At this state, funds are flowing with a high rate from \`p1\` to \`p2\`.
5. After some time, \`p1\` is running out of funds and \`canBeLiquidated\` returns true.
6. Another user calls \`liquidate\` on p1, but attacker front-runs the \`liquidate\` tx:
    - Attacker uses another profile \`p3\` (or could use p2) and creates subscription to \`p1\`, thereby setting \`inflow > outflow\` for p1.
    - Since \`inflow > outflow\`, the liquidate call initiated by another user will revert.
    - Attacker can back-run the \`liquidate\` tx and unsubscribes \`p3\` from \`p1\`.
7. Funds are still flowing from \`p1\` to \`p2\`, thereby increasing the negative balance of \`p1\` and increasing balance of \`p2\`.
8. Attacker can withdraw the balance from \`p2\`; the negative value in p1 remains and must be compensated with funds from the treasury.

Because step 6 happens in the same block, there are no funds flowing from `p3` to `p1`, but the setup is enough to get the `liquidate` call reverted. Thus, the costs of the attack are only the tx costs for `subscribe` and `unsubscribe`.

The attacker can repeatedly perform the attack, to keep the funds flowing from `p1` to `p2` and thereby draining the funds of the protocol.

Note that this attack only pays off under the following circumstances:

- when the attacker manages to sandwich the `liquidate` call for the entire liquidation period and beyond.
- when the balance of `p2` increased due to preventing liquidation is higher than the cost of the attack (= tx costs for sandwiching `liquidate` call).

Remediations to Consider

Consider adding a delay between `subscribe` and `unsubscribe` to make the above attack vector unprofitable.

L-1 Subscription owner is not transferred when profile is transferred

TOPIC	STATUS	IMPACT	LIKELIHOOD
Protocol Design	Addressed	Low	High

When a profile NFT is transferred to a new owner, `ProfilesModule.notifyProfileTransfer` is called to revoke all existing permissions and to set new owner.

However, for subscription NFTs associated to a profile, the new owner is not set and thus still pointing to the original owner when checking `GratefulSubscriptions.ownerOf`.

Consider also transferring the profile’s associated subscription NFTs to the new owner when the the profile is transferred to a new owner.

RESPONSE BY GRATEFUL

> There is no subscriptions list in the contracts to know which NFTs to transfer. So we agreed to add it to the documentation, and delegate the responsibility to the user (also with a batch call from the frontend)

~~L-2~~ Call to liquidate doesn’t validate `liquidatorId`

TOPIC	STATUS	IMPACT	LIKELIHOOD
Protocol Design	Fixed	Low	High

`LiquidationModule.liquidate` takes `liquidatorId` as an argument. However, there is no validation done whether the `liquidatorId` is associated to a profile.

If this is not a requirement, consider including `msg.sender` when emitting the event.

Also the `SubscriptionLiquidated` event is emitted with `0` rewards. Since there are no incentives for liquidators in this version of the contract, consider removing the `reward` argument in the emitted event.

~~L-3~~ Duration property for subscriptions returns wrong value when unsubscribed

TOPIC	STATUS	IMPACT	LIKELIHOOD
Use Cases	Fixed	High	Medium

`Subscription.getDuration` returns the `duration + elapsedTime` where `elapsedTime` is calculated as follows:

```
uint256 elapsedTime = block.timestamp - lastUpdate;
```

While this is true for active subscriptions, the calculation is not correct for unsubscribed subscriptions, as in this case `elapsedTime` should be 0.

L-4 Adding a new vault can fail when vault implementation is used in another vault already

TOPIC	STATUS	IMPACT	LIKELIHOOD

Protocol Design	Wont Do	Medium	Low

Some “weird” ERC20 tokens (e.g. USDT) require the approval to be set to 0 before setting it to a new value (see here). For such tokens, `VaultsModule.addVault` would fail when trying to add a vault implementation that is already used within another vault.

Note that this is seen as a low-level issue, as the scenario of adding multiple vaults pointing to same implementation is considered as very unlikely according to the Grateful team.

RESPONSE BY GRATEFUL

> Each vault will have it’s own implementation even though using the same ERC20 asset.

~~Q-1~~ `isVaultPaused` always returns wrong value

TOPIC	STATUS	QUALITY IMPACT
Best Practice	Fixed	Medium

`VaultUtil.isVaultPaused` returns the value from `Vault.isPaused()`. However, `isPaused` is not implemented correctly and returns `true` for unpaused vaults and `false` for paused vaults.

The only reason why `FundsModule.withdrawFunds` correctly allows to withdraw funds from unpaused vaults is because it checks against the inverted value as seen below:

```
if (!VaultUtil.isVaultPaused(vaultId))
   revert VaultErrors.InvalidVault();
```

Consider changing `Vault.isPaused` so that it returns `true` for pause vaults and `false` for unpaused vaults.

~~Q-2~~ `tokenURI` returns value for invalid token ids

TOPIC	STATUS	QUALITY IMPACT
Best Practice	Fixed	Medium

`GratefulProfile.tokenURI` and `GratefulSubscription.tokenURI` don’t revert when the token id doesn’t exist but instead returns default values for subscription.

In order to comply with behavior of well known ERC721 implementation like the one from Openzeppelin, consider reverting for non-existent token ids.

Q-3 `_EDIT_PERMISSIONS` not used

TOPIC	STATUS	QUALITY IMPACT
Extra Code	Wont Do	Medium

Consider removing `_EDIT_PERMISSIONS` from ProfileRBAC.sol as it is not used anywhere in the code.

RESPONSE BY GRATEFUL

> The edit permission is used by the frontend to update the optionally off-chain data from the profile

~~G-1~~ `_hasEnoughBalance` makes `_isSolvent` redundant

TOPIC	STATUS	GAS SAVINGS
Extra Code	Fixed	Medium

In Balance.sol, `canWithdraw` and `canStartSubscription` check for both `_hasEnoughBalance` and `_isSolvent`:

```
return _hasEnoughBalance(self, time) && _isSolvent(self, time);
```

However, `_hasEnoughBalance` (as it only accounts for outflow) is a much more stricter check, thus making the additional check `_isSolvent` redundant.

Consider removing `_isSolvent` from the above functions to save gas costs and make the code more readable.

I-1 Enabling burning of `GratefulProfile` NFTs would break the `createProfile` operation

TOPIC	STATUS	IMPACT
Use Case	Wont Do	Informational ✳

Note that at the current state, the protocol doesn’t allow burning of GratefulProfile NFTs. If this is something being considered to allow in future versions, it is worth noting that this would break `ProfilesModule.createProfile`.

The function `createProfile` calculates the `tokenId` as follows:

```
uint256 tokenId = profile.totalSupply() + 1;
```

When a token id is burned, `totalSupply` decreases by 1, meaning that for the next call to `createProfile`, a `tokenId` will revert, as the token id is created that was already minted before. As a result, `safeMint` will revert, essentially breaking the entire `createProfile` functionality.

RESPONSE BY GRATEFUL

> The Grateful profile will not have the burning capability.

Disclaimer

Macro makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Macro specifically disclaims all implied warranties of merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Macro will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand by any other party. In no event will Macro be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Macro has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the Emergent team and only the source code Macro notes as being within the scope of Macro's review within this report. This report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than Macro. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Macro is not responsible for the content or operation of such websites, and that Macro shall have no liability to your or any other person or entity for the use of third party websites. Macro assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.