Report:
Review of Frigg's Solidity Smart Contracts

Authors	**AxLabs GmbH** Hardturmstrasse 161 CH-8005 Zürich Switzerland Website: https://axlabs.com Email: contact@axlabs.com *In collaboration with:* **Axelra AG** Hagenholzstrasse 83 CH-8050 Zürich Switzerland Website: https://axelra.com Email: info@axelra.com
Date	11.08.2022 – 05.09.2022 (invested time for the review+report: ~27h)

Introduction and Scope

Frigg requested to perform an analysis of the project's smart contracts (Solidity).

The authors of this report made a thorough review of the smart contract security level against mal-functions and potential attacks, aiming to identify possible errors in the design, configuration, or programming.

Agreed with **Frigg**, external dependencies were not part of the review's scope.

The scope of this evaluation includes the following projects:

- **Original review (draft)**: August 14th, 2022
    - Repository: https://github.com/FriggGroup/smartcontracts
    - Branch: master
    - Commit: 0f2f0e05470e3f60ac18aa00d8b28f39542e5a31
- **Last review (final)**: September 5th, 2022
    - Repository: https://github.com/FriggGroup/frigg-smartcontracts
    - Branch: master
    - Commit: d0c4f23903c5c3738e40e0e5604e2ffef0250ffa

After the first report draft is sent to **Frigg** (based on the "original review" info, above)**,** the authors gave the chance for a second evaluation of the findings and re-generated this report with modified findings' states – as a final version of the review report.

For example, findings with category "Critical" and state "Not Fixed" reported in the draft review report could be turned into "Critical" and "Fixed" in the final version of the review report. It depends on how code evolved and how the team (**Frigg**) reacted/fixed each of the findings.

The final review report is based on the information found in the "Last Review (final)" above.

Disclaimer

This document should not be used in any manner as investment advice or to make investment decisions; it simply provides the findings of the code review performed by the authors.

Additionally, this report should neither be an "endorsement" nor "disapproval" of the assurance of the accurate business model of the project under analysis, nor as a guarantee on the functioning or viability of the implemented financial product.

The authors provide all their knowledge and uses every resource at their disposal, but this does not guarantee the project's functionality or safety, and it cannot be regarded as the only evaluation of the code's utility and safety, bug-free status, or any other project declarations. In addition, the authors of this report do not evaluate or pass judgment on the project's participants or the underlying business plan.

The ecosystem, platform, programming language, and other technologies that are associated to blockchain technology and cryptographic assets may have vulnerabilities that could be exploited, posing new dangers and obstacles. As a result, the review is unable to verify the projects' explicit security in relation to all its dependencies.

Findings

The contract was reviewed with the remix IDE – https://remix.ethereum.org, offering static code analysis. Most findings were found by manually reviewing the smart contracts, in a line-by-line basis.

The findings are categorized into 4 different levels: Critical, Major, Minor, and Informative. Critical issues need to be fixed immediately and poses a potential loss of funds. These are the issues that will prevent the product from working if it’s released in that state. Major issues need attention but do not necessarily pose a risk of losing funds. Minor and Informative issues are usually reserved for "nice to haves" or specific comments.

Summary of Findings			
Identifier	**Title**	**Category**	**State**
F1	Anyone able to burn tokens	Critical	Fixed
F2	Any ERC20 token owner implementing AccessControl able to add tokens to the router	Major	Fixed
F3	Lack of unit tests	Major	Fixed
F4	Inconsistent contract name and token symbol	Minor	Fixed

F5	Lack of integration tests	Minor	Fixed
F6	Terms URL not defined as interface	Minor	Fixed
F7	Router smart contract with improper contract name	Minor	Fixed
F8	Router smart contract with improper file name	Minor	Fixed
F9	Parameter of burn method with improper name	Minor	Fixed
F10	Upgradability on router	Informative	Assumed
F11	Incorrect or disguising comments	Informative	Fixed
F12	Automated checks from Remix IDE	Informative	2.b.ii: Assumed

Anyone able to burn tokens

Identifier	Category	State
F1	Critical	Fixed

Description

Line 29 of ATT.sol shows the burn() method allowing anyone to send transactions to burn an arbitrary amount of tokens. This is critical since there's no guard to this method, and attackers could keep burning tokens as soon as they are minted – turning the token smart contract unusable and ultimately causing a potential financial loss to users.

There are several ways that this could be fixed, e.g., using a require() before calling _burn() on line 30. However, we suggest using the ERC20Burnable contract from OpenZeppelin by extending it from ATT.sol. Note that ERC20Burnable uses the guard _msgSender() on line 21 and only allows the transaction originator to burn its own amount of tokens.

State Review

The project provided the following fixes:

- Implemented ROUTER_ROLE.
- Added modifier for burn().
- Only router can burn tokens. Therefore, this also means that even token holders themselves can't burn tokens – due to security + compliance reasons.

Even though the authors of this report suggested to use ERC20Burnable, it is understandable that the project decided not to use it. Mainly because AccessControl functions (from OpenZeppelin) overlaps with ERC20Burnable.

Therefore, the implemented fixes solve the original findings of this report.

Any ERC20 token owner implementing AccessControl able to add tokens to the router

Identifier	Category	State
F2	Major	Fixed

Description

Lines 46-50 of testRouter.sol present programmatic checks using AccessControl that make the DEFAULT_ADMIN_ROLE key value of _outputTokenAddress able to add new tokens to the router.

As implemented, any ERC20 token owner using the AccessControl's DEFAULT_ADMIN_ROLE with key "0x00" would be able to call the router's add() method (line 36 of testRouter.sol) to add a new token.

It's not clear if this is a desirable business functionality given that no documentation about the smart contract logic is available. In addition, the authors of this report didn't want to be biased by digging into Frigg's business flow. However, we believe that this is something not desirable given the comment on line 49, where states that "only admin of the Frigg-issued token can add token to router". But, the message on the require() (line 50) contradicts the comment on line 49, saying that "only token admin can add the token to this router". Therefore, it's not clear what are the real intentions behind it.

However, in any way, we suggest that:

- Make the testRouter extend the AccessControl from OpenZeppelin and have a constructor where AccessControl is initialized with the address for the DEFAULT_ADMIN_ROLE key – which might be different than ATT's DEFAULT_ADMIN_ROLE address, but not necessarily.
- Use the onlyRole() modifier on the add() method (line 36). This will make sure that only the specified role would be able to add new tokens to the Frigg ecosystem.
- Clarify the function of add() method by adding meaningful and clearer comments and examples.

State Review

The project provided the following fixes:

- Initialized the DEFAULT_ADMIN_ROLE in the constructor.
    - The DEFAULT_ADMIN_ROLE from the router (primaryRouter.sol) is different from the token smart contract (ATT.sol). This gives flexibility for the project to define two different admins if desired.
- The onlyRole() modifier was added to the method add(), allowing only the router admin (DEFAULT_ADMIN_ROLE from primaryRouter.sol) to add tokens.
- Groomed comments, removing ambiguity.

Therefore, the implemented fixes solve the original findings of this report.

Lack of unit tests

Identifier	Category	State
F3	Major	Fixed

Description

Both contracts testRouter.sol and ATT.sol do not present any unit tests.

This poses a high risk to the project, even with carried security reviews (like this report).

It would be necessary to add unit tests, for example, to check the correct functionality

of:

- isPrimaryMarketActive() on line 34 of ATT.sol.
- setBondExpiry() and seeBondExpiryStatus() of ATT.sol, on lines 42 and 47, respectively.
- mint() and burn() of ATT.sol, on lines 25 and 29, respectively.
- add(), buy(), and sell() of testRouter.sol, on lines 36, 63, and 93, respectively.

Ideally, the unit test cases could check several inputs (min and max) and cover happy path and some of the corner cases.

State Review

The project provided the following fixes:

- The project has now a 95+% unit coverage, which is a good unit test coverage in the industry.
- Unit test cases cover major methods, including the ones mentioned in the original review.
- Included documentation for the unit test cases.

Therefore, the implemented fixes solve the original findings of this report.

Inconsistent contract name and token symbol

Identifier	Category	State
F4	Minor	Fixed

Description

The contract name of ATT.sol is set to ATT on line 8. However, on line 15 the ERC20 is initialized with the token symbol “DTT”.

This could lead to confusion and ultimately could pose risks if scam projects could explore that to deceive end users.

State Review

The project provided the following fixes:

- Smart contract name is set to “ATT”, which is consistent to the file name ATT.sol.

Therefore, the implemented fixes solve the original findings of this report.

Lack of integration tests

Identifier	Category	State
F5	Minor	Fixed

Description

Both contracts testRouter.sol and ATT.sol do not present any integration tests. The difference of unit and integration tests is that instead of testing small chunks of code, the entire system is tested – in this case, ATT, testRouter, and the relation to other contracts that both interact to. For example, the hardhat fork function could be used to perform tests using “real” mainnet/testnet smart contracts.

While integration tests are not strictly necessary to ATT.sol and testRouter.sol, we advise to come up with potential integration test scenarios that bring confidence on the business flow. For example:

- Test whether the require() (line 67 of testRouter) would pass given a real IERC1155-compatible token (i.e., deployed on testnet).
- Test whether the mint process outputs a correct amount of tokens (lines 82-86 of testRouter) using the real USDC smart contract (and potentially other tokens) as the issuanceTokenAddress.

State Review

The project provided the following fixes:

- Integration test cases cover major methods that are important for the business case. For example:
    - mint() and burn() methods of the ATT.sol, lines 25 and 29, respectively.
    - add() of testRouter.sol, on line 36.

    - buy() and sell() of primaryRouter.sol, on lines 63 and 93, respectively.
- Included documentation for the integration test cases.

Even though more integration test cases could be added, the authors judge that it is enough due to the smart contracts' low complexity.

Therefore, the implemented fixes solve the original findings of this report.

Terms URL not defined as interface

Identifier	Category	State
F6	Minor	Fixed

Description

Line 51 of ATT.sol shows the variable termsATT set to a constant string, potentially pointing to terms and conditions.

However, it becomes challenging for a dApp to fetch tokens' terms and conditions that doesn't have a standard way of naming where to obtain the terms URL. For example, tokens' implementation could have a variable called termsABC, or termsXYZ.

Therefore, we suggest adding a getTermsURL() method to the IFrigg.sol interface. This would allow new tokens in the Frigg ecosystem to follow the same standard (e.g., same way to get terms pointer), making it easy for the Frigg dApp to display them.

State Review

The project provided the following fixes:

- Renamed "termsATT" variable to "termsURL".
- Implemented an "external view" function in the IFrigg.sol interface, called "termsURL()"

Therefore, the implemented fixes solve the original findings of this report.

Router smart contract with improper contract name

Identifier	Category	State
F7	Minor	Fixed

Description

Line 9 of testRouter.sol shows the contract name as “testRouter”.

However, the smart contract within the file testRouter.sol does not present testing characteristics or contain any test cases.

Therefore, renaming it from “testRouter” to “FriggRouter” is advisable. Please, consider renaming all other occurrences in the code base.

State Review

The project provided the following fixes:

- Renamed “testRouter” smart contract to “primaryRouter.
- The reason for naming it “primaryRouter” and not “FriggRouter” is that it’s obvious that the router is about Frigg, and because the router will deal with primary market sales.

Therefore, the implemented fixes solve the original findings of this report.

Router smart contract with improper file name

Identifier	Category	State
F8	Minor	Fixed

Description

The file testRouter.sol does not present testing characteristics or contain any test cases.

It is a good practice to name .sol file names with the same naming as the contract. For example, if the smart contract within testRouter.sol is renamed to “FriggRouter”, it’s advisable to rename the file name to FriggRouter.sol.

State Review

The project provided the following fixes:

- Renamed “testRouter” smart contract filename from “testRouter.sol” to “primaryRouter.sol”.

Therefore, the implemented fixes solve the original findings of this report.

Parameter of burn method with improper name

Identifier	Category	State
F9	Minor	Fixed

Description

Line 29 of ATT.sol shows the burn() function with the first parameter named as “_to”.

However, we suggest to rename this parameter to “_from”, as OpenZeppelin smart contracts (e.g., ERC20Burnable, lines 24-38) also describes the burn function as “destroys amount tokens **from** account, deducting **from** the caller's allowance”.

State Review

The project provided the following fixes:

- Renamed the parameter “_to” from the burn() function to “_from”, which is compatible to OpenZeppelin smart contracts.

Therefore, the implemented fixes solve the original findings of this report.

Upgradability on router

Identifier	Category	State
F10	Informative	Assumed

Description

It’s debatable whether the testRouter contract should be upgradable or not. Making a smart contract upgradable in a correct and secure way is a difficult task that requires knowledge of the applied proxy pattern. Read OpenZeppelin’s proxy website for more info.

We suggest the project to consider this feature due to the following reasons:

- The router could potentially evolve over time in terms of new functions or how

ATT (or other tokens) are managed – for example, how tokens are added or the requirements to buy/sell them.

- Once the router is upgraded, the same smart contract address would remain the same – maintaining the ecosystem created around it, i.e., dApps or other smart contracts would not need to update testRouter's address every time.
- For example, other means of verification whether a user is able to buy/sell tokens could be added/removed in the future (in addition of the present one on line 67). Therefore, upgradability could be an important factor.

State Review

The project provided the following reasons on why they decided not to implement upgradability:

- Current functionality on the "primaryRouter" for primary market sales (buy and sell) is considered complete. I.e., no additional feature is expected in the router.
- Router can be upgraded by re-deploying with a new address, updating the address on front-end and grant new address admin role to the respective tokens.
- Proxy patterns lead to potential exploit due to storage collision.
- The gas consumption using the proxy pattern might be considerably high depending on ETH market conditions.

As mentioned in the original finding's description, adopting upgradability is debatable, and it is up to the project to implement it or not – given the "Informative" classification.

Therefore, the project assumed the finding description and did not change the implementation as consequence.

Incorrect or disguising comments

Identifier	Category	State
F11	Informative	Fixed

Description

There are several comments throughout the code where they are either wrong,

outdated, or not clear.

The list below shows some examples (non-exhaustive list):

1. Line 13 of ATT.sol: comment says the supply is 17k tokens, but the initialization of line 16 sets the supply capped to 170k.
2. Lines 29-35 of testRouter.sol: further elaborate what each parameter represents.
3. Line 92 of testRouter.sol: comment refers to inputAmount (which looks like a variable name), but there's no variable named as inputAmount.

We suggest to have a thorough review of comments written in the Solidity files, as well as adapting them to be fully compatible to the NatSpec Format.

State Review

The project provided the following fixes:

- Achieved a high coverage of code comments, including methods, variables, functions, etc.
- Smart contracts were annotated with comments following the NatSpec Format.

Therefore, the implemented fixes solve the original findings of this report.

Automated checks from Remix IDE

Identifier	Category	State
F12	Informative	2.b.ii: Assumed

Description

The following categories of automated checks were reported (using the static analyzer plugin from the Remix IDE), each with the warning types and specific comments:

1. Gas costs:
    a. ATT.sol: 5 warnings of Gas costs were raised, and all of them can be ignored.
    b. testRouter.sol: 3 warning of Gas costs were raised, and all of them can be ignored.

2. Miscellaneous:
    a. ATT.sol:
        i. “No return” was reported 2 times due to return values on the IFrigg.sol interface, but can be ignored.
        ii. “Constant/View/Pure functions” was reported 2 times, but this can be ignored.
    b. testRouter.sol:
        i. “Guard conditions” was reported 7 time due to the recommendation of assert(x) use. However, this can be ignored.
        ii. “Data truncated” was reported once due to the division operation on line 108. This could be a problem if the result of the division is something like 0.3. In this case, the outputTokenAmount result would be truncated to 0. We suggest detecting this case (with a modulo operation) and keep the rounding difference in the contract storage – which, over time, it’ll add up. Then, in future, decide what to do with the remaining amount of rounding difference.
3. Security:
    a. testRouter.sol:
        i. “Check-effects-interaction” was reported 3 times due to a potential re-entrancy attack. This could be a critical issue if the add() method (line 36 on testRouter.sol) allows any token to be added without previous technical diligence. However, this report assumes that Frigg’s router admin checks if tokens are compatible/safe/suitable beforehand. Therefore, this doesn’t pose a security risk for the testRouter.sol smart contract.

State Review

During some discussion with project’s software engineers about finding F12-2.b.ii, the following points were mentioned:

a. Data truncation would only happen for tokens with decimals lower than 10^6 (6 decimals).
b. For ERC-20 compliant tokens (with 18 decimals), the maximum loss is 0.9999 * 10^-6 USDC (as an example).
c. All Frigg-issued tokens are intended to be ERC-20 compliant (18 decimals) or, at least, tokens with 6 decimals or more.

Given the arguments given by (c), the project assumed the finding description and did not change the implementation as consequence – given that tokens added to the router would not cause rounding losses.

Conclusions

The authors identified critical findings and advised the project to address them before deploying the smart contracts in a productive environment (i.e., mainnet).

Besides that, the authors also recommended addressing Major findings identified in the first draft of this report. For example, Finding **F2** had the potential to open attack vectors given that malicious users could add arbitrary tokens to the router and potentially use **Frigg**'s dApp for scam activities.

The project promptly addressed and fixed all findings with Critical, Major, and Minor classification in the final report.

One of the drawbacks during the initial review was the complete lack of tests in the code base (unit and/or integration tests). Lack of tests tends to increase the chances of unexpected issues and poses overall risks to smart contracts' operations and applications interfacing with.

Based on the initial authors' feedback, the project's software engineers developed unit and integration test cases for the final version of this report, also documenting them. The average unit test coverage was brought to ~95%.

Finally, Findings **F10** and **F12-2.b.ii** were assumed by the project but they do not pose overall risks to the **Frigg** business case.