SMART CONTRACT AUDIT

ZOKYO.

July 27th 2022 | v. 1.0

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

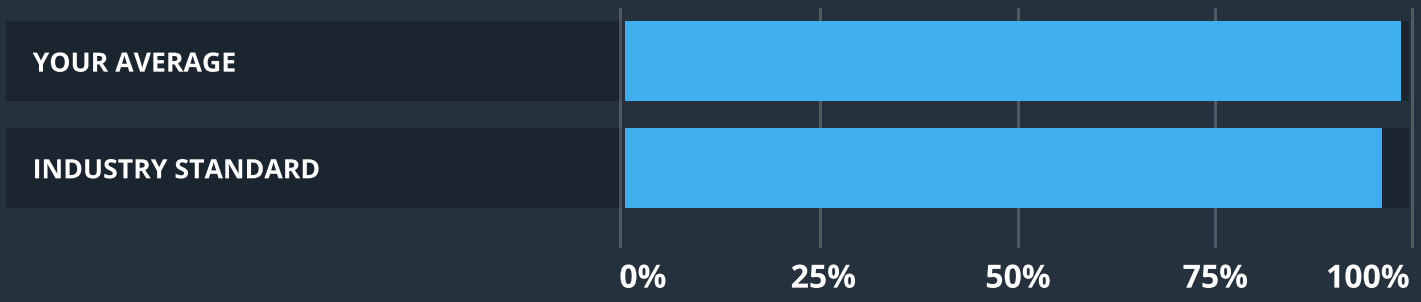
TECHNICAL SUMMARY

This document outlines the overall security of the Gumball smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the Gumball smart contract codebase for quality, security, and correctness.

Contract Status

Testable Code



The testable code is 100%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the Gumball team put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the Gumball repository.
https://github.com/GumBallProtocol/Contracts

Initial commit: 240fb50c1935ad670680197a0b48334be79497b8
Last audited commit: 22d3a460059225a51c8face497afd4a60a36802d

Within the scope of this audit Zokyo auditors have reviewed the following contract(s):

- ERC20BondingCurve.sol
- Factory.sol
- Gumball.sol
- Gumbar.sol

Throughout the review process, care was taken to ensure that the contract:

- Implements and adheres to existing standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of resources, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of Gumball smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1	Due diligence in assessing the overall code quality of the codebase.	3	Testing contract logic against common and uncommon attack vectors.
2	Cross-comparison with other, similar smart contracts by industry leaders.	4	Thorough, manual review of the codebase, line-by-line.

EXECUTIVE SUMMARY

Zokyo Security Team has carefully checked the full set of contracts for Gumball protocol. During audit, special attention was paid to the flow of users' funds to verify their safety and that only users have direct access to their funds across the whole protocol. The formulas, applied to funds during buying/selling base token was verified as well to calculate correct values.

There were two critical issues found during audit. The first of them was connected to the correctness of XGBT tokens unstaking at Gumbar contract. The issue was successfully fixed by the Gumball team. The second one was related to the correct ability of redeeming Gumball NFT tokens for GBT. The issue was resolved as well, so that the redeeming functionality cannot get blocked.

Other issues were connected to gas optimization, additional checks and unused storage variables and code. Nevertheless, all of the issues were fixed or verified by the team.

The Gumball team also verified the correctness of all the features within protocol, such as the absence of collateral liquidation. All of the concerns were successfully verified to ensure that contracts' code correspond to the business logic of the protocol.

Overall, the code is highly secure, has a good readability and performs all desired functionalities, though, it has a lack of documentation. Also, it needs to be mentioned, that the audited code did not contain native unit-tests - all unit tests were added by Zokyo Secured team in order to verify the functionality. It should be also mentioned, that all the fixes were made to duplicates of contracts in the separate folder. We recommend to move fixed code to the main contracts directory.

PROTOCOL OVERVIEW
FACTORY.SOL

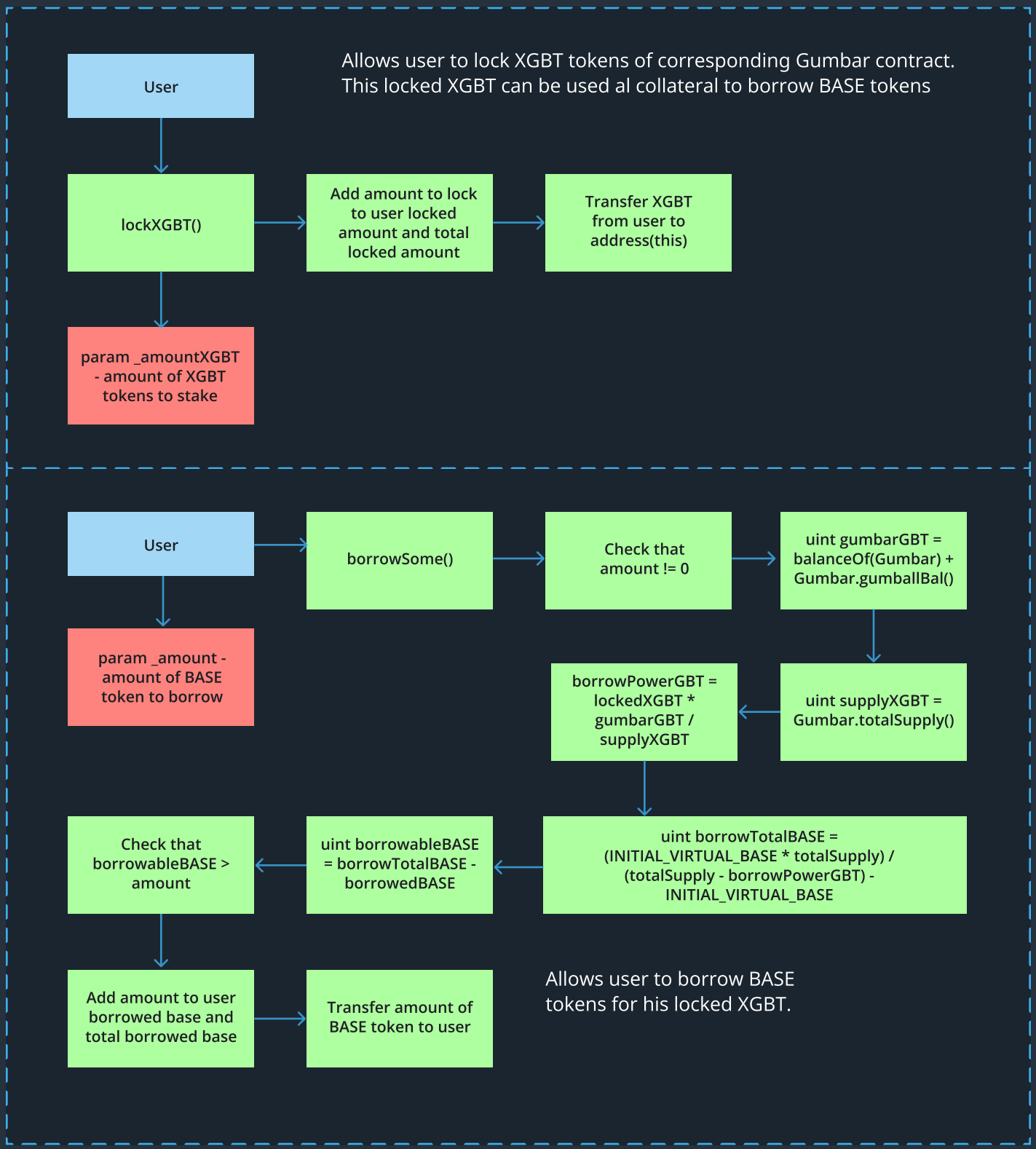
Factory creates a set of 3 contracts, connected to each other: ERC20BondingCurve, Gumbar and Gumball. ERC20BondingCurve has 2 main functionalities: selling/buying GBT tokens for BASE tokens and borrowing BASE tokens with XGBT as collateral. Distinguishing feature of loans in Gumball protocol is that user's position cannot be liquidated under any circumstances and there is no fees accrued.

Gumball is ERC721 which allows users to sell/buy NFTs for GBT tokens, where 1 NFT equals 1 GBT. Gumbar is a staking contract which allows two options: to stake GBT for XGBT or to stake Gumball NFTs for XGBT. When redeeming XGBT, user receives additional GBT, collected from fees for buying/selling GBT.

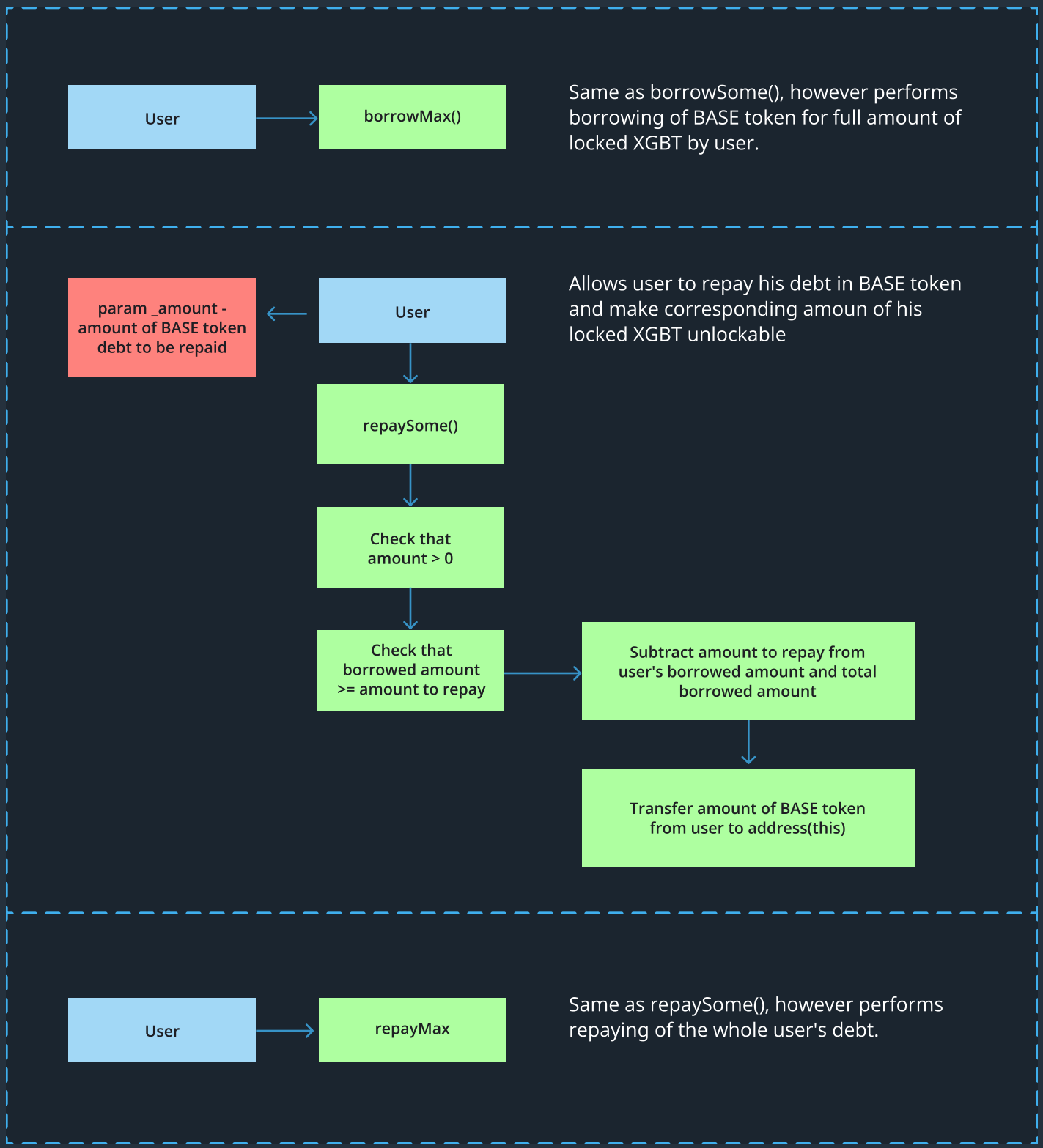
ERC20BONDINGCURVE.SOL

ERC20BONDINGCURVE.SOL

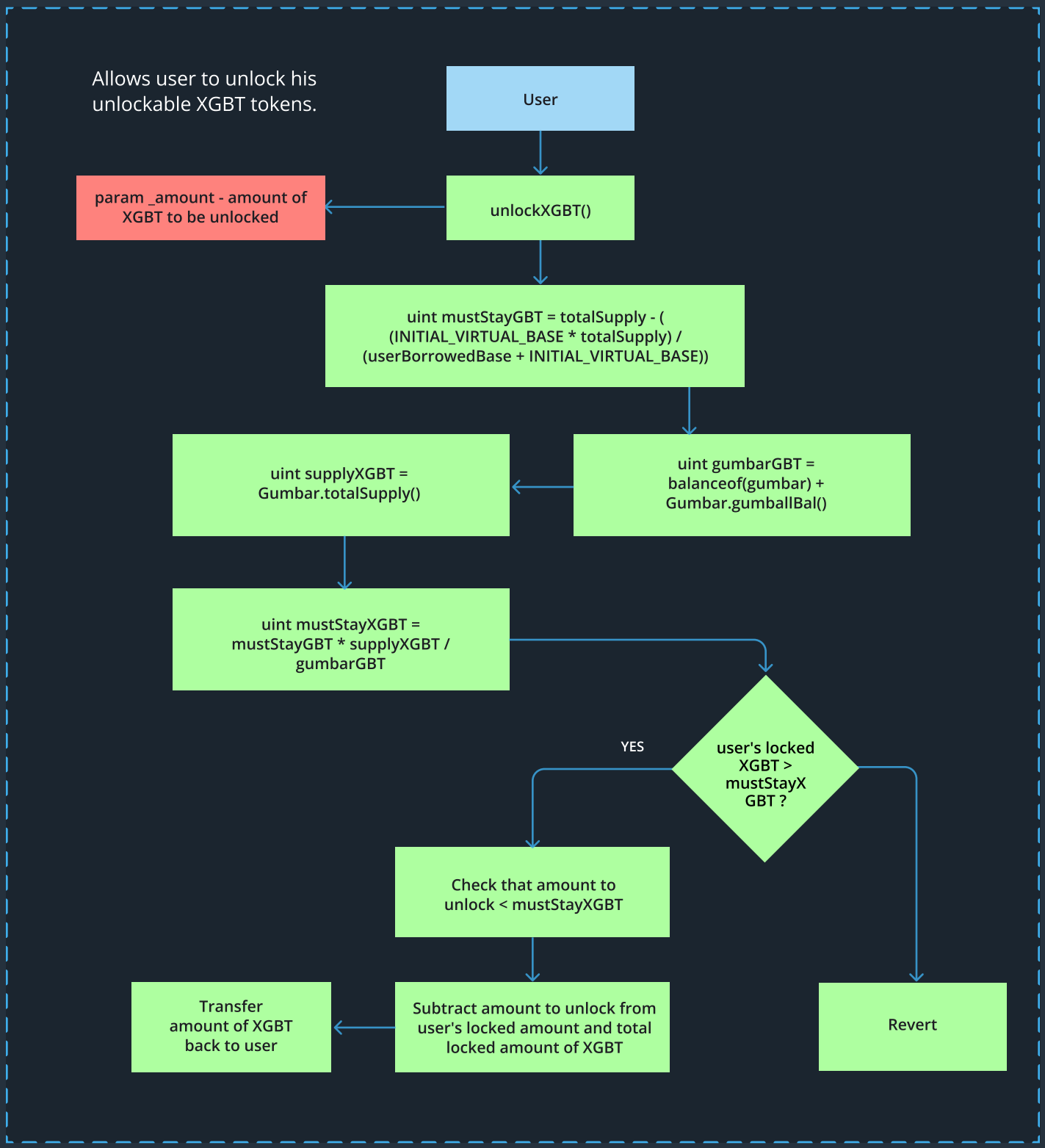
ERC20BONDINGCURVE.SOL



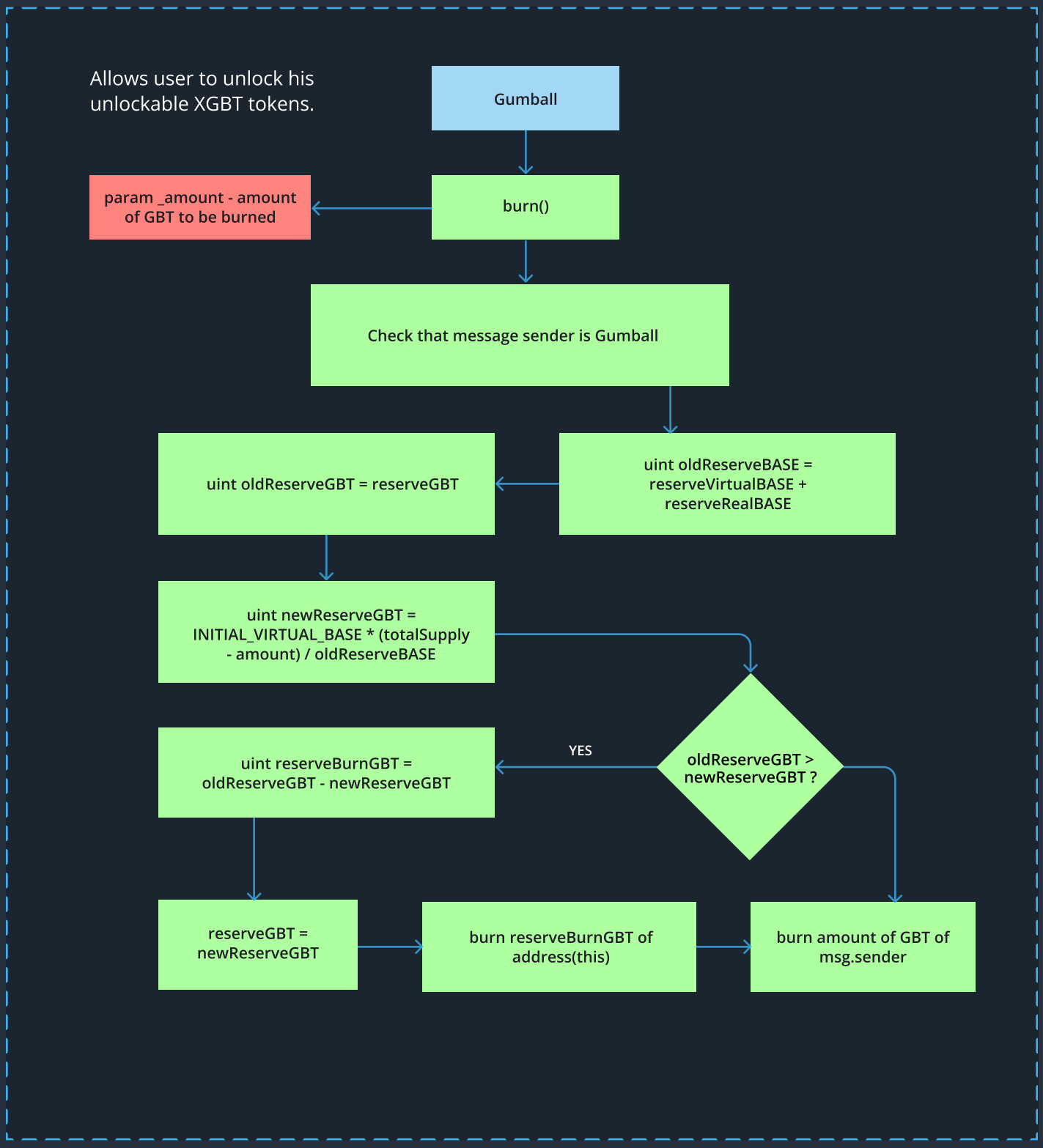
ERC20BONDINGCURVE.SOL



ERC20BONDINGCURVE.SOL



ERC20BONDINGCURVE.SOL

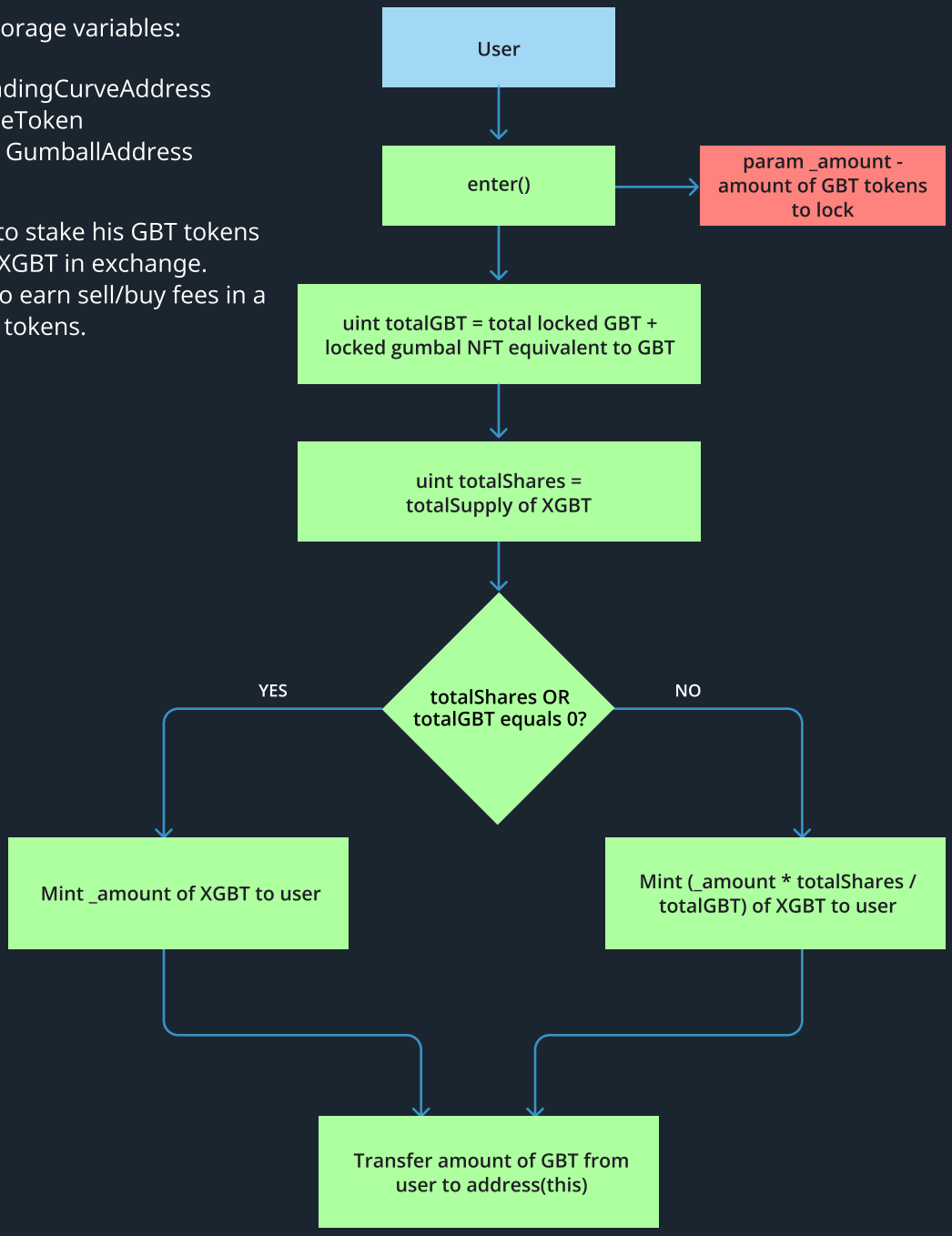


GUMBAR.SOL

Initialised storage variables:

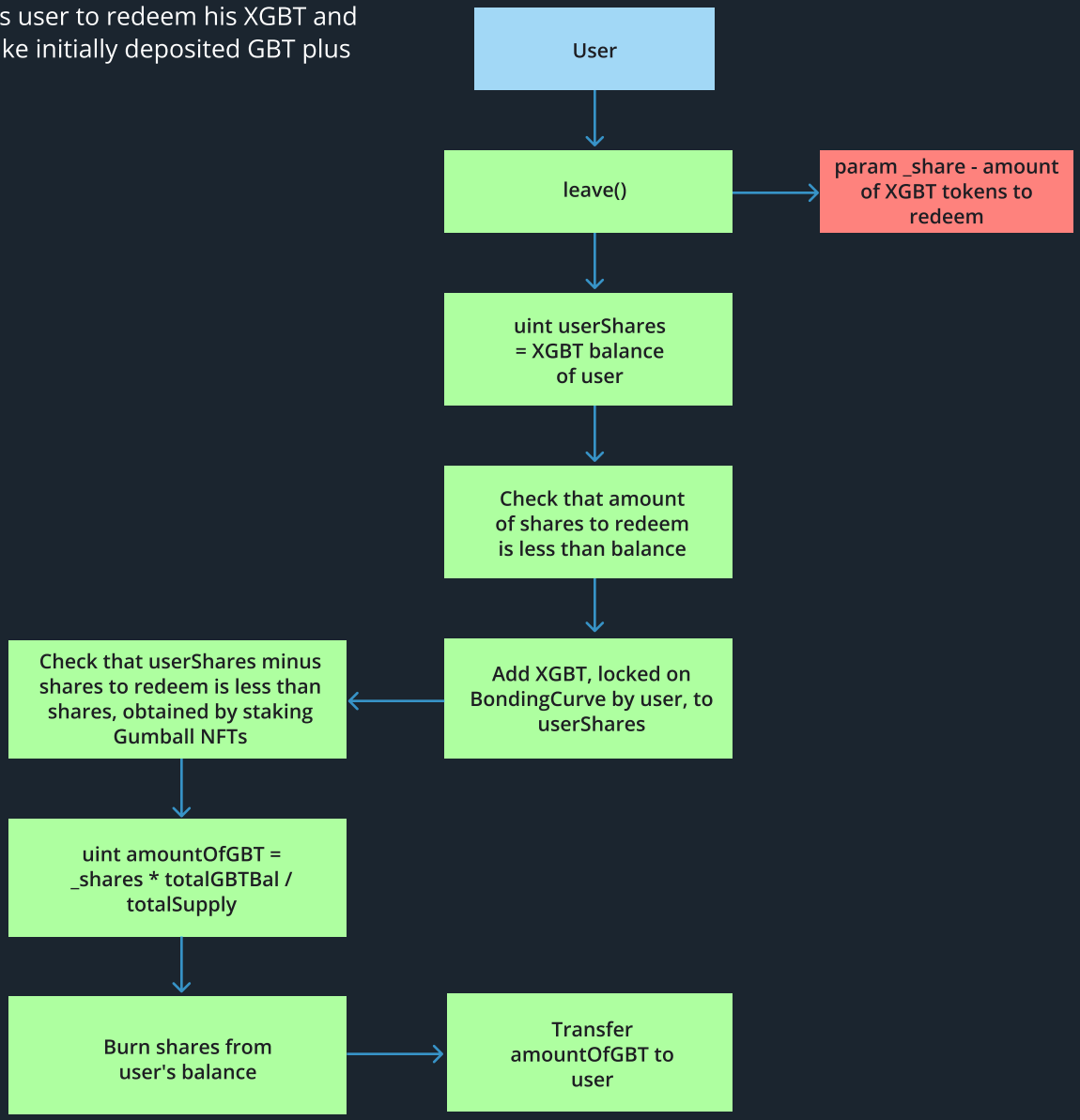
GBT = BondingCurveAddress
BASE = _baseToken
GUMBALL = GumballAddress

Allows user to stake his GBT tokens and receive XGBT in exchange. User starts to earn sell/buy fees in a form of GBT tokens.



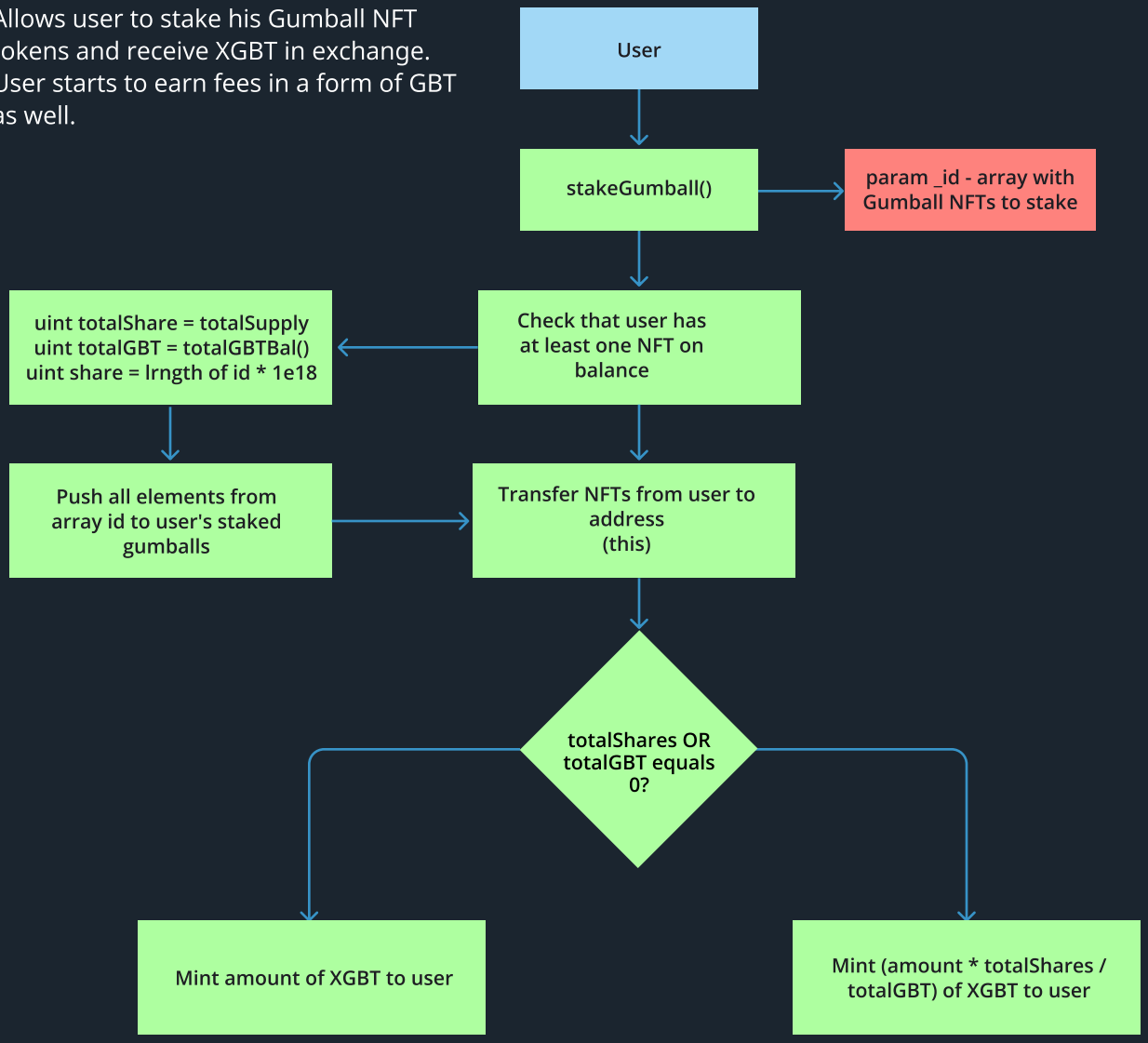
GUMBAR.SOL

Allows user to redeem his XGBT and unstake initially deposited GBT plus fees.

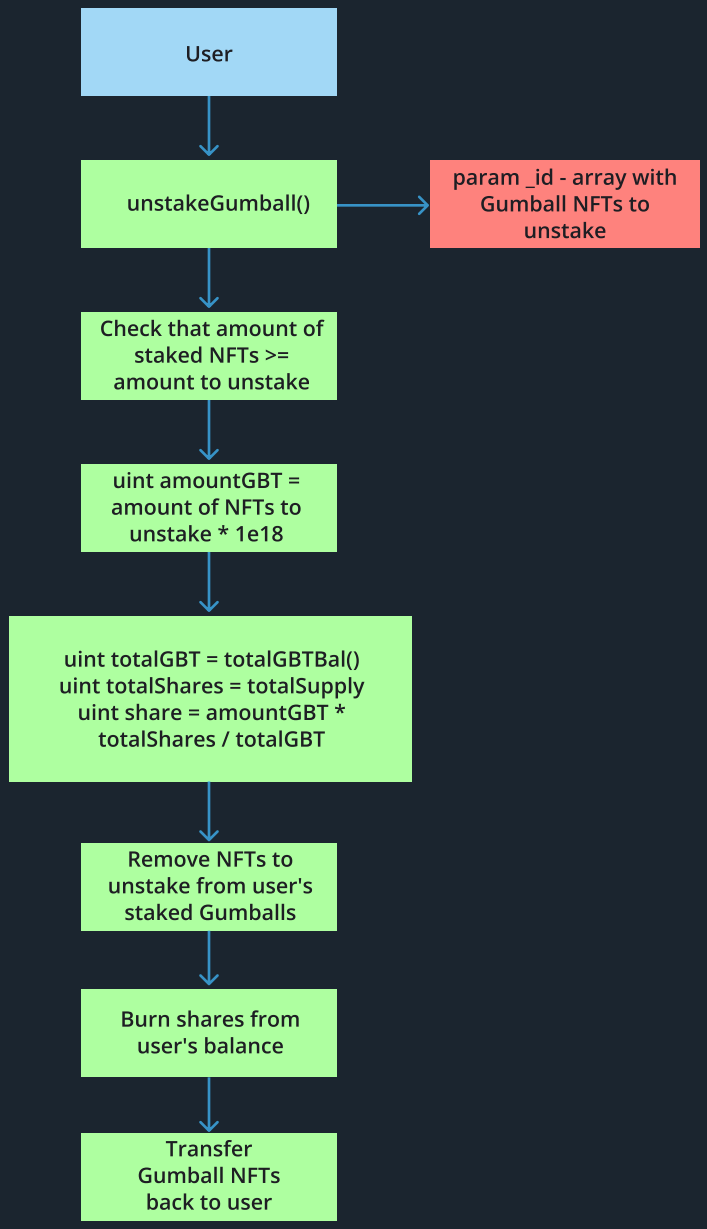


GUMBAR.SOL

Allows user to stake his Gumball NFT tokens and receive XGBT in exchange. User starts to earn fees in a form of GBT as well.



Allows user to redeem his XGBT for his Gumball NFT tokens and collect GBT rewards.

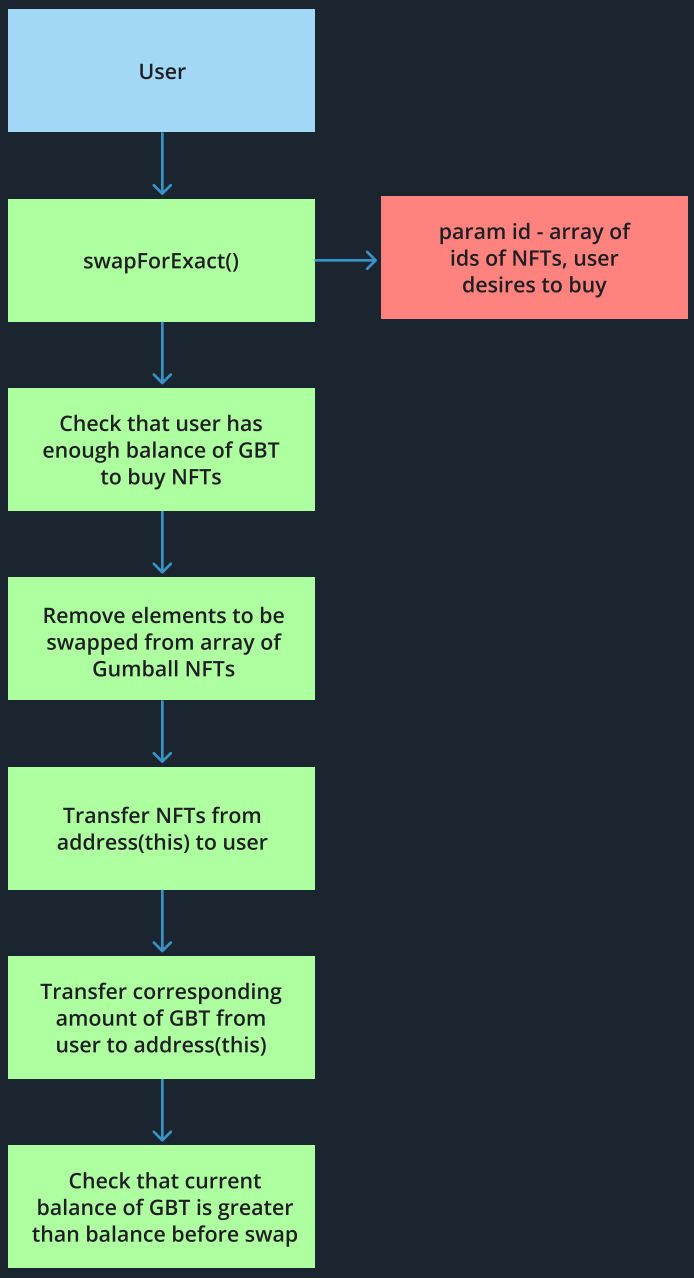


GUMBALL.SOL

Initialised storage variables:

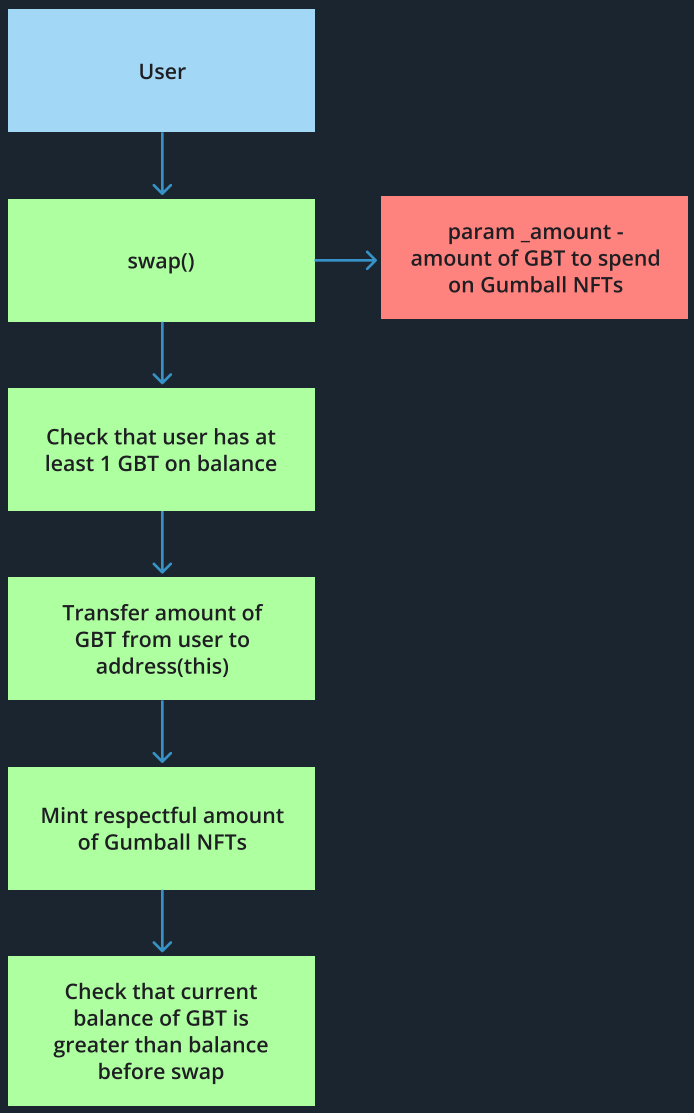
tokenContract = BondingCurveAddress

Allows user to swap his GBT tokens for an exact amount of existing Gumball NFTs, specified in the array



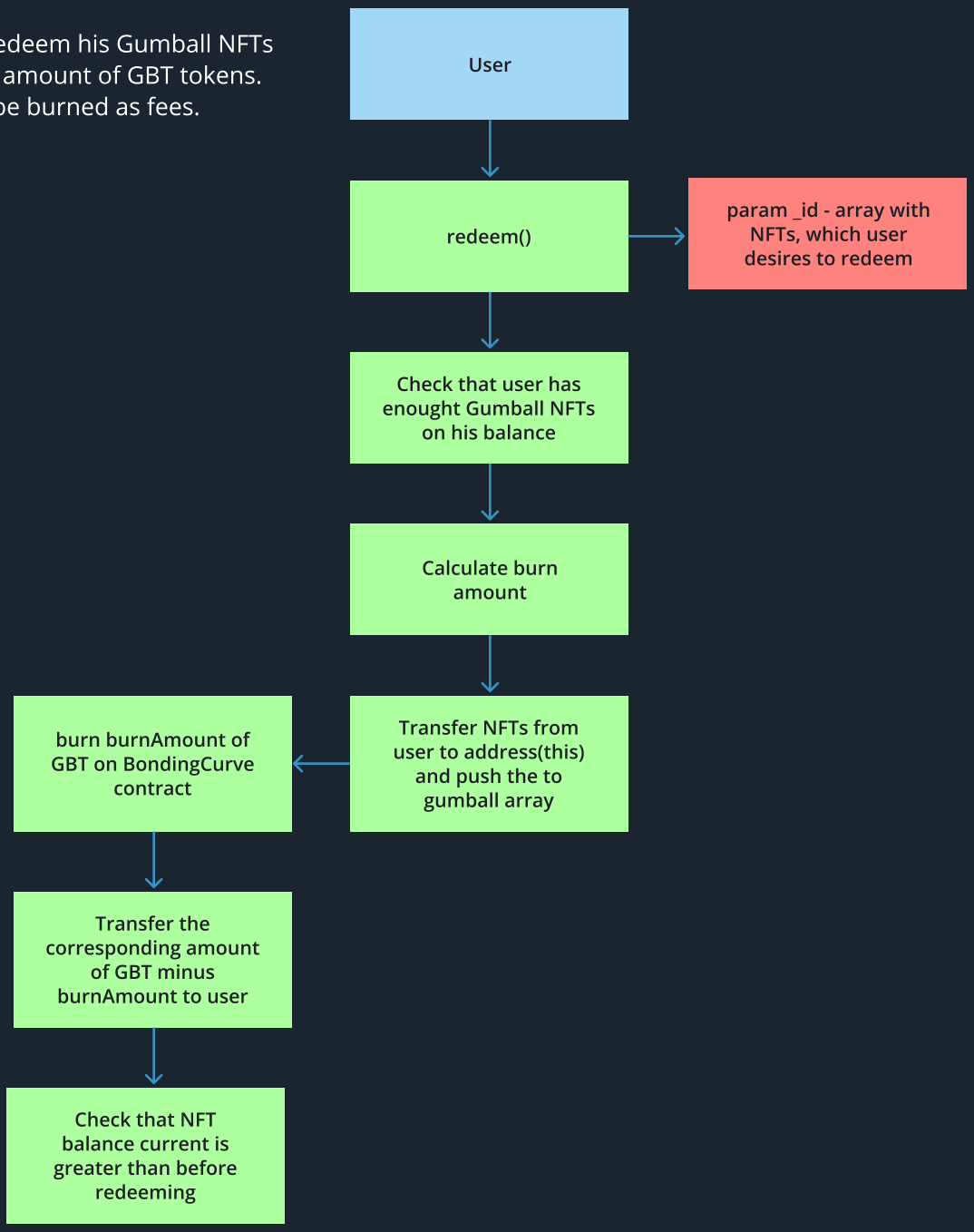
GUMBALL.SOL

Allows user to swap his GBT tokens for a respectful amount of Gumball NFTs, newly minted

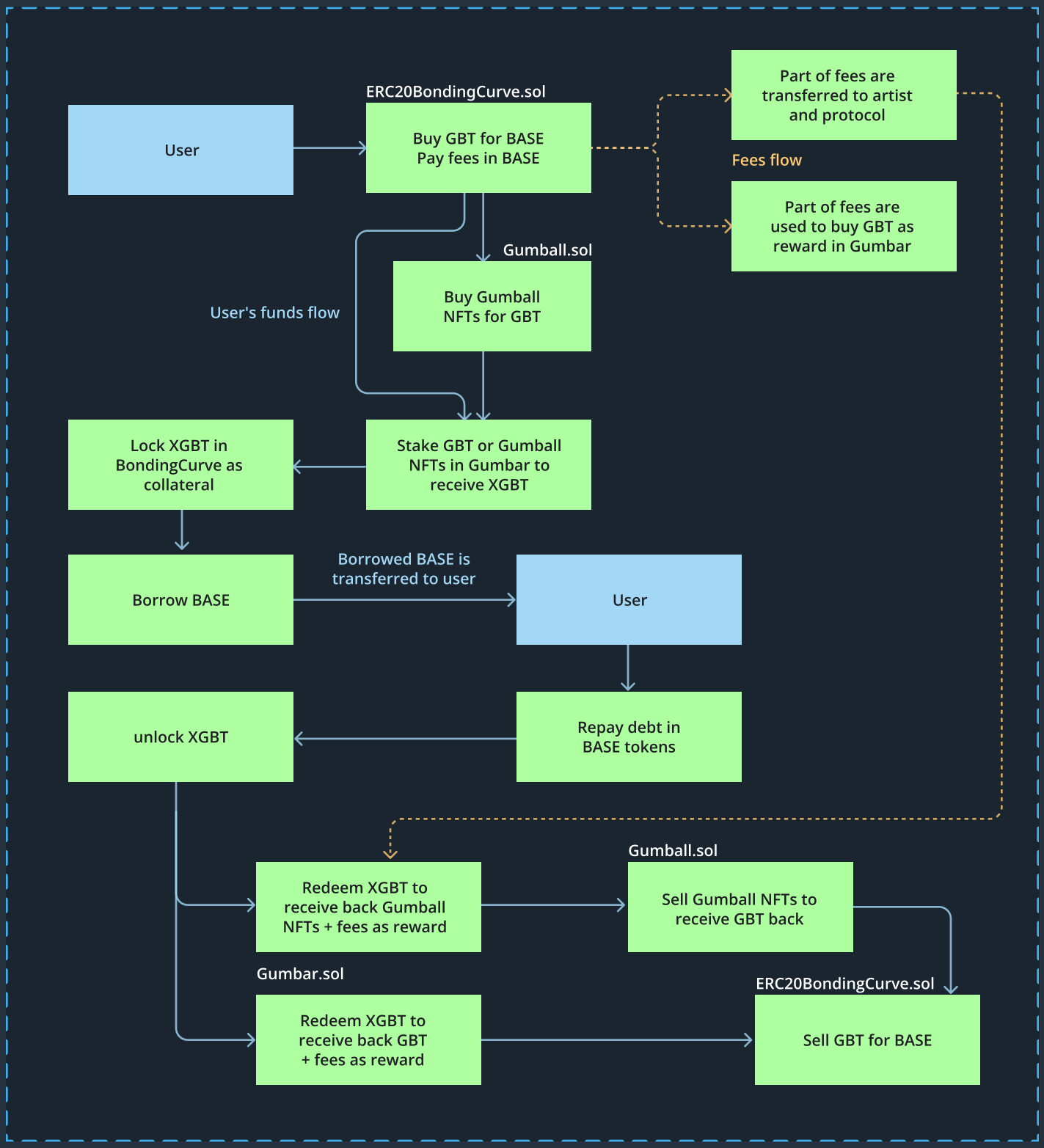


GUMBALL.SOL

Allows user toredeem his Gumball NFTs for a respectful amount of GBT tokens. Some GBT will be burned as fees.



PROTOCOL SCHEME



STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Issues tagged “Verified” contain unclear or suspicious functionality that either needs explanation from the Customer’s side or it is an issue that the Customer disregards as an issue. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

CRITICAL | RESOLVED

XGBT, pegged to Gumball NFTs, can be withdrawn directly for GBT.
Gumbar.sol.

Gumbar allows users to receive and redeem XGBT it two ways: by staking GBT and by staking Gumball NFTs. It is checked against an amount of NFTs locked by the user, that XGBT, pegged to Gumballs, cannot be redeemed directly for GBT tokens.
However, in case XGBT, received for staking Gumball NFTs, are transferred with standard ERC20 transfer functions to another address, locked NFTs are not transferred to the other address'es locked NFTs as well. Due to this, XGBT, which were minted for Gumball NFTs, can now be redeemed for GBT tokens. This can lead to scenarios, where there won't be enough GBT for other users, preventing them from receiving their funds and scenarios, where Gumball NFTs are locked forever.

Recommendation:
Transfer locked NFTs to other user's locks in case transferred XGBT are pegged to Gumball NFTs.

Post-audit:
XGBT can be transferred to limited number of addresses, such as the Gumbar itself, BondingCurve, zero address .
The commit with fix is 22d3a460059225a51c8face497afd4a60a36802d.

CRITICAL | RESOLVED

Redeeming requires additional approval and Gumball to inherit the IERC721ReceiverUpgradeable interface.

Gumball.sol
Due to external call of safeTransferFrom on address(this), Redeeming required additional approval of Gumball NFTs by the user. It also requires Gumball to inherit IERC721ReceiverUpgradeable and implement onERC721Received() function. Although issue is marked as critical, since currently redeem() reverts due to absence of onERC721Received() implemented, it should be verified, in case another version of ERC721 is used, where this check is missed.

Recommendation:
Either use internal _transfer() function instead of external safeTransferFrom() or implement onERC721Received(). Verify the necessity of additional approval tokens for Gumball in order to perform redeeming.

Post-audit:
transferFrom() is used, which doesn't call onERC721Received(). Approval to Gumball for redeeming NFTs remained.

HIGH | RESOLVED

Accuracy loss during swapping GBT for Gumball NFTs.
Gumball.sol: function swap().

During swap, the quantity of NFTs, which can be minted for provided amount of GBT, is calculated. Since one NFT is worth 1 GBT (1e18 in wei amount), in case the user provides an amount,which is not divisible by 1e18, some amount might be lost by the user, since the whole provided amount of GBT is transferred from the user.
Example:

1. User passes 2.5e18 wei of GBT tokens to function swap().
2. 2 Gumball NFT, worth 2e18 wei of GBT, is minted for user, however 2.5e18 wei of GBT is transferred.
3. When a user redeems his NFTs, he only receives 2 GBT, losing 0.5 GBT tokens.

Recommendation:
Take only the amount of GBT from user, necessary for the purchasing of Gumball NFTs.

MEDIUM | RESOLVED

SafeERC20 should be used.

Transferring of BASE and XGBT token is performed with regular transfer() and transferFrom() methods from the OpenZeppeling IERC20 interface. Though, in general, the ERC20 token may have not return value for these methods (see USDT implementation) and lead to the failure of calls and contract blocking. Thus the SafeERC20 library should be used or the token should be verified to implement to implement return values for transfer() and transferFrom() methods

Recommendation:
Use SafeERC20 library for transferring BASE token.

LOW | RESOLVED

SafeMath usage can be omitted.

Contracts set utilizes Solidity 0.8.x, which has built in support of overflow and underflow warnings, thus SafeMath library can be omitted for gas savings and code simplification.

Recommendation:
Remove SafeMath library.

LOW | RESOLVED

Parameters lack validation.

Factory.sol: function deployProxies().
Since function can be called by anyone and input parameters are vital for deployed contracts, parameters should be verified to be valid.

Recommendation:

1. Verify that strings “name”, “symbol” and values from array “_URIs” are not empty.
2. Verify that the length of array _URIs equals 2 (Containing base token URI and contract URI).
3. Verify that “_supplyCAP” and “_init_vBase” are not equal 0. Consider also adding a minimum value for these variables.
4. Verify that “_baseToken” and “artist” are zero addresses.
5. Verify that “_delay” parameter is not extremely small or great by adding minimum and maximum boundaries for this parameter.

INFORMATIONAL | RESOLVED

Unnecessary writing to storage.

Gumball.sol: function _pop(), line 215.
Gumbar.sol: function _pop(), line 205.
On line 215, an element to be removed is being written to the last element of the array. Since, this element is removed in the next line of code, the writing of the element to the last position of array is unnecessary.

Recommendation:
Remove unnecessary writing to storage to decrease gas spendings.

Post-audit:
Function was removed in Gumball.sol. Iteration in Gumbar still remains, however it is iterated through user's data.

INFORMATIONAL | RESOLVED

Iteration through the whole storage array.

Gumball.sol: function findGumball(), line 89.
Gumbar.sol: function findGumball(), line 182.
Iteration through a storage array increases user's gas spendings and in case there are a lot of elements in the array, the function will consume more gas than allowed per transaction. Issue is marked as info, since the supply of NFTs is limited and has to be large to consume more gas than allowed during iteration, however, users' gas spendings can be decreased.

Recommendation:
Consider using mapping which will store the position of NFT in an array to avoid iteration through a storage array.

Post-audit:
Issue was resolved in Gumball contract. Issue wasn't resolved in Gumbar, since the iteration is performed for the user's data.

INFORMATIONAL | RESOLVED

Unused storage variables.

1. Gumball.sol: variable supplyCap. Variable is supposed to limitate the supply of NFTs, however it is never used. Since the possible amount of NFTs to be minted is limited by the supply of GBT, the absence of using the supplyCap is not limited. Also, during initialization, supplyCap is assigned to the same variable, that initial_totalSupply and reserveGBT, which means that this cap for Gumball should be divided by 1e18.
2. ERC20BondingCurve.sol: variables initial_totalSupply, floorPrice. Variables are never used within the contract.

Recommendation:

1. Either use supplyCap variable to limitate the amount of NFTs, which can be minted, or remove the variable.
2. Either confirm the necessity of variables(For external contracts of Dapp for example) or remove unused variables.

INFORMATIONAL | RESOLVED

Unused functions.

Factory.sol: function isClone().
Gumball.sol: function _baseURI().
Internal functions are never used within the contracts.

Recommendation:

Remove unused functions.

INFORMATIONAL | RESOLVED

Unnecessary require statement.

ERC20BondingCurve.sol: function repaySome(), line 363.
Checking that the result of function sub() of SafeMath is greater or equal to 0 is unnecessary and will never revert, since function sub() either returns a positive result or reverts due to arithmetic underflow.

Recommendation:
Remove unnecessary require statement.

INFORMATIONAL | VERIFIED

Not clear liquidation of collateral.

ERC20BondingCurve.sol
Protocol allows users to borrow Base token for XGBT, however, in lending protocols, the position of the user can be liquidated under certain circumstances (After some period of time or in case user's borrow balance collateral value). Liquidation function is missed in protocol and its necessity should be verified, since the collateral is token, which can only be bought for base token. This means that in case a user doesn't repay his debt, he doesn't prevent other users from selling GBT for Base tokens.

From client:
There is never a liquidation of collateral under any circumstances.

INFORMATIONAL | VERIFIED

Selling GBT requires additional approval

ERC20BondingCurve.sol
Performing selling GBT requires the user to approve his GBT for GBT contract (ERC20BondingCurve) before selling. This happens due to an external call to address(this) in Line 240, where ERC20BondingCurve becomes msg.sender and requires an allowance from the user to spend his tokens.

Recommendation:
Avoid the necessity of additional approvals by using ERC20 internal _transfer() function or verify that this is an intended functionality (For example, if protocol requires user's additional confirmation that he wishes to sell GBT).

Post-audit:
The neccesity of approval remains, which seems to be an intended functionality.

INFORMATIONAL

Audit issues are fixed in other folder

It should be documented, that all the fixes, performed by Gumball team, are made in separate folder with duplicates of code. The issue is informational and should be present in final report.

Recommendation:
Move all the fixed contracts to /contracts directory.

	ERC20BondingCurve.sol	Factory.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	Gumball.sol	Gumbar.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo team

As part of our work assisting Gumball team in verifying the correctness of their contract code, our team has prepared the complete set of unit tests for the Gumball protocol.

Zokyo team prepared a complete test coverage as well as tested some additional test-scenarios.

```
  Gumball protocol
Contract: Factory
    ✓ Should update token and gumball library (702ms)
    ✓ Should add address to whitelist (552ms)
    ✓ Should add existing token and nft to whitelist (63ms)
    ✓ Should get owner address
    ✓ Should return amount of tokens and nfts deployed
    ✓ Should not deploy proxies if name/symbol/URIs are empty strings (103ms)
    ✓ Should not deploy proxies if URIs length is incorrect / supply or virutal base is zero (110ms)
    ✓ Should not deploy proxies if base token or artist is address zero
    ✓ Should not deploy proxies if delay is grater than 14 days
  Buy/Sell of GBT tokens
  When Market is closed
    ✓ Should not buy GBT for unwhitelisted user (154ms)
    ✓ Should buy GBT for whitelisted user (483ms)
    ✓ Should revert buying if amount is over whitelist limit (228ms)
    ✓ Should revert buying if whitelist amount overflow (202ms)
  When Market is open
    ✓ Shoulb buy GBT tokens for user 1 (823ms)
    ✓ Shoulb buy GBT tokens for user 1 (816ms)
    ✓ Should revert buying GBT if transaction expired (145ms)
    ✓ Should revert buying GBT if actual amount GBT is less minimum amount GBT (194ms)
    ✓ Should sell GBT tokens for BASE tokens (474ms)
    ✓ Should revert selling GBT if transaction expired (87ms)
    ✓ Should revert selling GBT if actual amount of BASE is less than minimum amount BASE (273ms)
Contract: Gumball
    ✓ Should revert swap if user has insufficient balance (62ms)
    ✓ Should revert swap if not whole GBTs provided
    ✓ Should revert if swap is not sucessfull (111ms)
```

✓ Should swap GBT for Gumball NFTs (14051ms)
✓ Should redeem Gumball NFTs (1154ms)
✓ Should revert redeem if user has insufficient balance (912ms)
✓ Should revert if redeem is not successful (195ms)
✓ Should revert redeeming if there is repeatable NFT in array (223ms)
✓ Should swap GBT for existing NFTs (809ms)
✓ Should revert swap for existing NFTs if user has insufficient balance (1801ms)
✓ Should revert swap for existing NFTs if swap is not successful (160ms)
✓ Should revert swap for exisiting NFTs if NFT is not on Gumball's balance (118ms)
✓ Should revert swap for existing NFTs if there is repeatable NFT in array (200ms)
✓ Should let owner set base uri (77ms)
✓ Only owner can set new base uri (47ms)
✓ Should let owner set contract uri (65ms)
✓ Only owner can set new contract uri (42ms)
✓ Should return current price in BASE token (99ms)

Contract: Gumbar

✓ Should mint shares equal to the amount of NFTs, when Gumbar totalSupply is 0 (699ms)
✓ Should let users enter Gumbar with GBT (481ms)
✓ Should let users stake Gumball NFTs to Gumbar (1711ms)
✓ Should revert staking Gumball NFTs if user has insufficient balance (44ms)
✓ Should distribute fee rewards and buy GBT for Gumbar (774ms)
✓ Should leave Gumbar and receive initial GBT + reward (146ms)
✓ Should not let leave Gumbar if shares amount exceed balance (65ms)
✓ Should not let leave Gumbar if GBT amount exceeds unstakeable GBT (271ms)
✓ Should unstake Gumball NFTs from Gumbar and receive rewards (1143ms)
✓ Should revert unstaking Gumball NFTs if user has insufficient balance (56ms)
✓ Should revert unstaking Gumball NFTs if user passed non-existing token (124ms)
✓ Should not let unstake same NFT in one transaction more than once (211ms)

Lock / Borrow / Repay

✓ Should let users lock XGBT in BondingCurve (311ms)
✓ Should let unlock XGBT (177ms)
✓ Should not let unlock more XGBT than locked (141ms)
✓ Should borrow BASE tokens and unlock unlockable XGBT (431ms)
✓ Should not let borrow if amount to borrow is zero (43ms)
✓ Should not let borrow if amount to borrow > borrow credit (254ms)
✓ Should let borrow max (284ms)
✓ Should repay some of the debt (327ms)
✓ Should not repay zero amount of debt (49ms)
✓ Should not repay more than actually borrowed (50ms)
✓ Should not let unlock more XGBT, than used as collateral (112ms)
✓ Should not let unlock XGBT if amount exceeds withdrawal allowance (93ms)
✓ Should not unlock zero XGBT (134ms)

```
    ✓ Should repay maximum of the debt (296ms)
  Contract: ERC20BondingCurve
    ✓ Only protocol can update ownership (51ms)
    ✓ Should update ownership (114ms)
    ✓ Only Gumball can burn (47ms)
    ✓ Should return initial supply
  68 passing (5s)
```

FILE	% STMTS	% BRANCH	% FUNCS
ERC20BondingCurve.sol	100	97.22	100
Factory.sol	100	100	100
Gumball.sol	100	100	100
Gumbar.sol	100	95	100
All files	**100**	**97.62**	**100**

We are grateful to have been given the opportunity to work with the Gumball team.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Zokyo's Security Team recommends that the Gumball team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.