BUDZ-Finance

Code Security Assessment

PREPARED BY:

THE AUDIT INSTITUTE ANALYST TEAM

PREPARED FOR:

THE BUDZ-FINANCE TEAM

PREPARED ON:

MARCH 18TH 2021

THE AUDIT INSTITUTE

Report Version 1.0

Table of Contents

Disclaimer

The Audit Institute Reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts The Audit Institute to perform a security review.

The Audit Institute Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technology’s proprietors, business, business model or legal compliance.

The Audit Institute Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

The Audit Institute Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. The Audit Institute's position is that each company and individual are responsible for their own due diligence and continuous security. The Audit Institute's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze. View our full legal terms and conditions at https://audit.institute/

What is included in a report by *The Audit Institute*?

- A document describing the detailed analysis of a particular piece(s) of source code provided to The Audit Institute by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of The Audit Institute has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name & Website	**Budz.finance -** https://budz.finance/
Project Description	The Audit Institute analyst team reviewed the contract of the Budz.finance platform. The goal of the platform is to provide investors with a high yield through an innovative concept that involves burning, staking, and farming their token. Their platform is described as a financial experiment created by a team of budding enthusiasts.
Platform	Ethereum, Solidity
Compiler Version	0.6.4
Mainnet Address	Not Yet Deployed
GitHub Commit Hash	3ab20dcb0d237defab21fa6a84f12fabb0bf7c35
Delivery Date	March 18th 2021
Method of Audit	Static Analysis, Fuzzing, Manual Review
Consultants Engaged	2

Summary of Findings

Critical	0
Medium	0
Informational	4
Total Issues	**4**

Executive Summary

This audit report exclusively covers the analysis that was conducted for the budz.finance contract written in Solidity. The goal of the platform is to provide investors with a high yield through an innovative concept that involves burning, staking, and farming their token. Budz.finance was deployed to Binance Smart Chain (BSC) at: 0x1e0A4D330b60BaBf3386125aeD73b81C6afC8526, but the initial contract had to be pulled so the team could address bugs. The scope of this audit report covers the new iteration of their platform.

Staking BUDZ advertises a minimum APY of 42.0%, which users can eventually double to 84% APY as there is a direct proportion between the number of tokens burned to the amount of BUDZ staked. To achieve this double APY, users must have burned 50% of their staked balance. Note: There is a limit to burning up to 3x of the amount of interest a user has accrued during the period of their stake. All interest and rewards distributed to users are sourced entirely from minting.

The contract uses a struct called “Farmer” which maintains all of the investors’ staking data that will be used for the platform’s calculations. After a minimum of seven days, users can utilize the claimInterest function to claim their rewards and the UnstakeTokens function to exit their staking position. Alternatively, users can opt to leverage their rewards to add to their staking position via the RollStakeInterest function.

The farming feature offers a variable APY depending on the APY that is set for the LP token and is paid in BUDZ. In order to be eligible to earn interest via farming, users must lock their tokens into a choice of various liquidity tokens (The budz.finance team mentioned there will be anywhere from 4 - 8 LP Addresses). The Farming functionality leverages APY ‘halvening’ periods, which any user can call/activate beginning after seven days, which is then increased by seven days for every subsequent period (e.g. 7,14,21,28 days). These halvening events will directly impact the APY calculation for harvesting rewards.

There are several key functions that contain the onlyAdmins modifier. These functions rely on the assumption that the "setForeverLock" will be called after all of the APYs have been set. Users should exercise caution investing in this as the admins have the capabilities to adjust the APY's as frequently as they want until the "setForeverLock" is called. At which point, the APY's are prevented from being altered (with the exception of halvenings and the benefit of individually increasing one’s APY via burning).

There is a 10% distribution to the project team in the form of minting for each “Harvest” that is called by a user. There is also a 4% distribution minted to the Dev team for claiming and rolling interest. Additional minting occurs in the form of a referral program where both the referrer and the referred user receive a 5% bonus on all token-yielding events.

Disclosed in the report below is a full analytical review of the platform after undergoing various test scenarios and code review. The findings varied in criticality as some were related to Solidity code standards and optimization.

Contracts in Scope

CONTRACT NAME	CONTRACT DESCRIPTION
budz-finance.sol	The budz.finance Platform Contract

External Vulnerability Findings

Vulnerability Category	Notes	Results
Arbitrary Storage Write	N/A	PASS
Arbitrary Jump	N/A	PASS
Delegate Call to Untrusted Contract	N/A	PASS
Dependence on Predictable Variables	N/A	PASS
Deprecated Opcodes	N/A	PASS
Ether / Token Loss	N/A	PASS
Exceptions	N/A	PASS
External Calls	N/A	PASS
External Service Providers	N/A	PASS
Flash Loans	N/A	PASS
Inconsistent Emission of Events	N/A	PASS
Integer Over/Underflow	N/A	PASS
Multiple Sends	N/A	PASS
Oracles	N/A	PASS
Reentrancy Issues	N/A	PASS
Unchecked Retval	N/A	PASS
Suicide	N/A	PASS
State Change External Calls	N/A	PASS
Unchecked Retval	N/A	PASS

Findings and Recommendations

Finding Name	Criticality	Analyst Notes
Break loop when condition is satisfied (Gas Optimization)	Informational	The `setPoolActive(…)` function loops through all of the addresses in the `lpAddresses` array to check if the `lpAddress` passed into the function already exists in the array. This operation can be very costly as the number of addresses in `lpAddresses` increases. After the loop has found a match in the array, there is no need to continue iterating through the rest of the array. We recommend inserting a `break;` after setting `_newAddress` to `false` as shown below:

```
function setPoolActive(address _lpAddress, bool _active)
    public
    onlyAdmins
{
    require(!isLocked, "cannot change pool status");
    bool _newAddress = true;
    for(uint i = 0; i < lpAddresses.length; i++){
        if(_lpAddress == lpAddresses[i]){
            _newAddress = false;
            break;
        }
    }
    if(_newAddress){
        lpAddresses.push(_lpAddress);
    }
    isPoolActive[_lpAddress] = _active;
}
```

Multiplication should be done before division	Informational	Dividing usually leads to integer truncation, which can result in calculations that are less precise. We recommend multiplying before division to reduce the risk as much as possible. **1.)** `calcStakingRewards()` (Line #885) : return (staked.div(apyAdjust .mul(stakingApyLimiter)) .div(1251) * (minsPastStakeTime(_user))); **Recommendation: The line can be re-written as follows:* return(staked.mul(minsPastStakeTime(_user))) .div(apyAdjust .mul(stakingApyLimiter) .mul(1251)); **2.)** `calcHarvestRewards()` (Line #912) : return((lpFrozenBalances[_user][_lpIndex].mul(globalApy) .div(lpApy[lpAddresses[_lpIndex]])) .mul(minsPastFreezeTime(_user, _lpIndex)) .div(halvening)); **Recommendation: The line can be re-written as follows:* return(lpFrozenBalances[_user][_lpIndex].mul(globalApy) .mul(minsPastFreezeTime(_user, _lpIndex))) .div(lpApy[lpAddresses[_lpIndex]] .mul(halvening));

Finding Name	Criticality	Analyst Notes
Functions should be `external`	Informational	totalSupply() (Line #407-409) transfer(address,uint256) (Line #426-429) allowance(address,address) (Line #434-436) approve(address,uint256) (Line #445-448) transferFrom(address,address,uint256) (Line #463-467) increaseAllowance(address,uint256) (Line #481-484) decreaseAllowance(address,uint256) (Line #500-503) FreezeLP(uint256,uint256,address) (Line #624-653) UnfreezeLP(uint256) (Line #656-673) HarvestBudz(uint256) (Line #677-688) StakeTokens(uint256,address) (Line #726-745) UnstakeTokens() (Line #748-763) ClaimStakeInterest() (Line #766-772) RollStakeInterest() (Line #775-781) NewHalvening() (Line #828-836) BurnBudz(uint256) (Line #838-853) totalFrozenLpBalance(uint256) (Line #948-954) setBUDZBNBpool(address) (Line #993-999) setBurnAdjust(uint256) (Line #1002-1007) stakingApyDecrease() (Line #1010-1017) setGlobalApy(uint32) (Line #1019-1025) setApy(uint32,address) (Line #1027-1033) setPoolActive(address,bool) (Line #1035-1050) setForeverLock() (Line #1052-1057) distributeTokens(address) (Line #1060-1078) *Recommendation: set these Functions as external to slightly reduce gas cost.

Variables should be `constant`	Informational	The following variables should be set to constant:
		`BUDZFINANCE._p1`
		`BUDZFINANCE._p2`
		`BUDZFINANCE._p3`
		
		**Recommendation: set these variables as constant to slightly reduce gas cost.*

Functions Overview

($) = payable function
 # = non-constant function
Int = Internal
Ext = External
Pub = Public

+ [Int] IERC20
- [Ext] totalSupply
- [Ext] balanceOf
- [Ext] transfer #
- [Ext] allowance
- [Ext] approve #
- [Ext] transferFrom #

+ [Lib] SafeMath
- [Int] add
- [Int] sub
- [Int] sub
- [Int] mul
- [Int] div
- [Int] div
- [Int] mod
- [Int] mod

+ [Lib] Address
- [Int] isContract
- [Int] sendValue #
- [Int] functionCall #
- [Int] functionCall #
- [Int] functionCallWithValue #
- [Int] functionCallWithValue #
- [Prv] _functionCallWithValue #

+ [Lib] SafeERC20
- [Int] safeApprove #
- [Prv] _callOptionalReturn #

+ [Int] IUniswapV2Router01
- [Ext] factory
- [Ext] WETH
- [Ext] addLiquidity #
- [Ext] addLiquidityETH ($)
- [Ext] removeLiquidity #
- [Ext] removeLiquidityETH #
- [Ext] removeLiquidityWithPermit #
- [Ext] removeLiquidityETHWithPermit #
- [Ext] swapExactTokensForTokens #
- [Ext] swapTokensForExactTokens #
- [Ext] swapExactETHForTokens ($)
- [Ext] swapTokensForExactETH #
- [Ext] swapExactTokensForETH #
- [Ext] swapETHForExactTokens ($)
- [Ext] quote
- [Ext] getAmountOut
- [Ext] getAmountIn
- [Ext] getAmountsOut
- [Ext] getAmountsIn

+ [Int] IUniswapV2Router02 (IUniswapV2Router01)
- [Ext] removeLiquidityETHSupportingFeeOnTransferTokens #
- [Ext] removeLiquidityETHWithPermitSupportingFeeOnTransferTokens #
- [Ext] swapExactTokensForTokensSupportingFeeOnTransferTokens #
- [Ext] swapExactETHForTokensSupportingFeeOnTransferTokens ($)
- [Ext] swapExactTokensForETHSupportingFeeOnTransferTokens #

+ [Int] IUniswapV2Pair
- [Ext] name
- [Ext] symbol
- [Ext] decimals
- [Ext] totalSupply
- [Ext] balanceOf
- [Ext] allowance
- [Ext] approve #
- [Ext] transfer #
- [Ext] transferFrom #
- [Ext] DOMAIN_SEPARATOR
- [Ext] PERMIT_TYPEHASH
- [Ext] nonces
- [Ext] permit #
- [Ext] MINIMUM_LIQUIDITY
- [Ext] factory
- [Ext] token0
- [Ext] token1
- [Ext] getReserves
- [Ext] price0CumulativeLast
- [Ext] price1CumulativeLast
- [Ext] kLast
- [Ext] mint #
- [Ext] burn #
- [Ext] swap #
- [Ext] skim #
- [Ext] sync #

+ TokenEvents

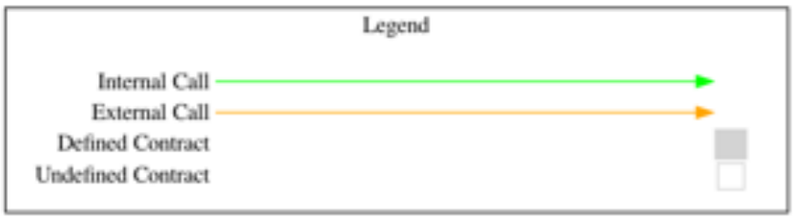
+ BUDZFINANCE (IERC20, TokenEvents)

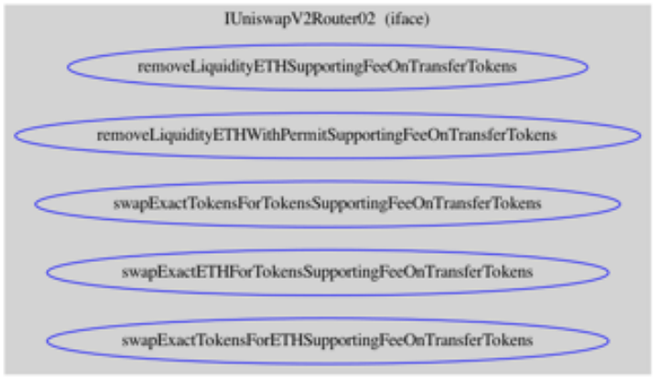
- [Pub] <Constructor> #
- [Ext] <Fallback> ($)
- [Pub] totalSupply
- [Pub] balanceOf
- [Pub] transfer #
- [Pub] allowance
- [Pub] approve #
- [Pub] transferFrom #
- [Pub] increaseAllowance #
- [Pub] decreaseAllowance #
- [Int] _transfer #
- [Int] _mint #
- [Int] _burn #
- [Int] _approve #
- [Int] _burnFrom #
- [Int] mintInitialTokens #
  - modifiers: synchronized
- [Pub] FreezeLP #
  - modifiers: synchronized
- [Pub] UnfreezeLP #
  - modifiers: synchronized
- [Pub] HarvestBudz #
  - modifiers: synchronized
- [Int] harvest #
- [Int] scopeCheck #
- [Pub] StakeTokens #
  - modifiers: synchronized
- [Pub] UnstakeTokens #
  - modifiers: synchronized
- [Pub] ClaimStakeInterest #
  - modifiers: synchronized
- [Pub] RollStakeInterest #
  - modifiers: synchronized
- [Int] rollInterest #
- [Int] claimInterest #
- [Pub] NewHalvening #
  - modifiers: synchronized

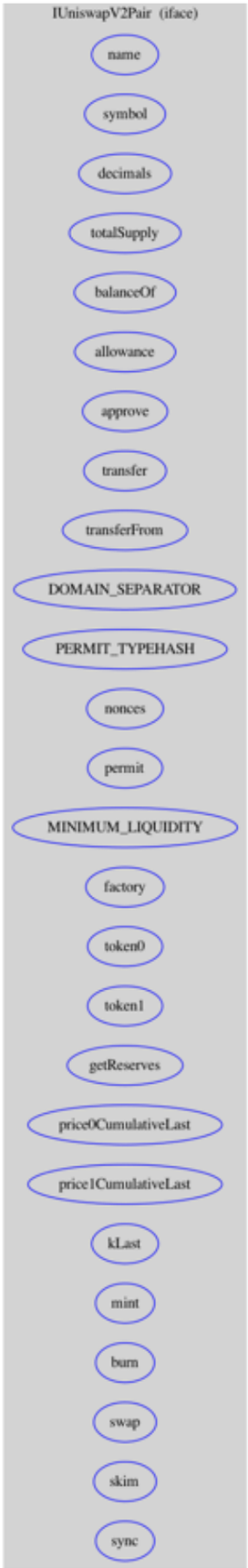
+ BUDZFINANCE (IERC20, TokenEvents) (Continued)

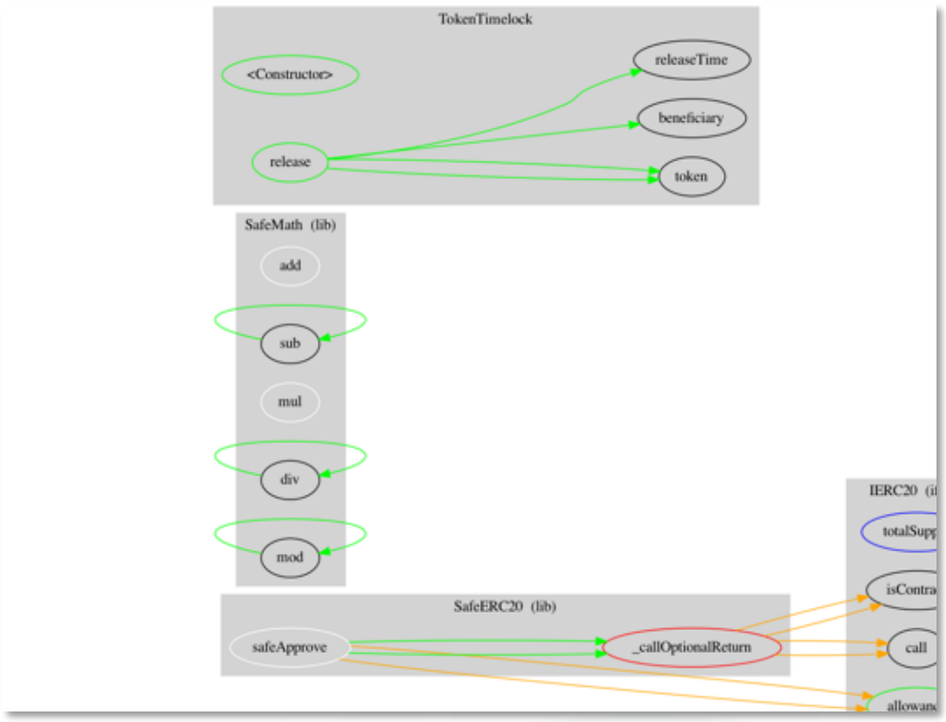
- [Pub] BurnBudz #
  - modifiers: synchronized
- [Pub] calcStakingRewards
- [Pub] minsPastStakeTime
- [Pub] calcHarvestRewards
- [Pub] minsPastFreezeTime
- [Pub] isStakeFinished
- [Pub] totalFrozenLpBalance
- [Pub] budzBalance
- [Pub] lpBalance
- [Pub] isHarvestable
- [Pub] setBUDZBNBpool #
  - modifiers: onlyAdmins
- [Pub] setBurnAdjust #
  - modifiers: onlyAdmins
- [Pub] stakingApyDecrease #
  - modifiers: onlyAdmins
- [Pub] setGlobalApy #
  - modifiers: onlyAdmins
- [Pub] setApy #
  - modifiers: onlyAdmins
- [Pub] setPoolActive #
  - modifiers: onlyAdmins
- [Pub] setForeverLock #
  - modifiers: onlyAdmins
- [Pub] distributeTokens #
  - modifiers: onlyAdmins
- [Pub] donate ($)
- [Pub] RollStakeInterest #
  - modifiers: synchronized
- [Int] rollInterest #
- [Int] claimInterest #
- [Pub] NewHalvening #
  - modifiers: synchronized

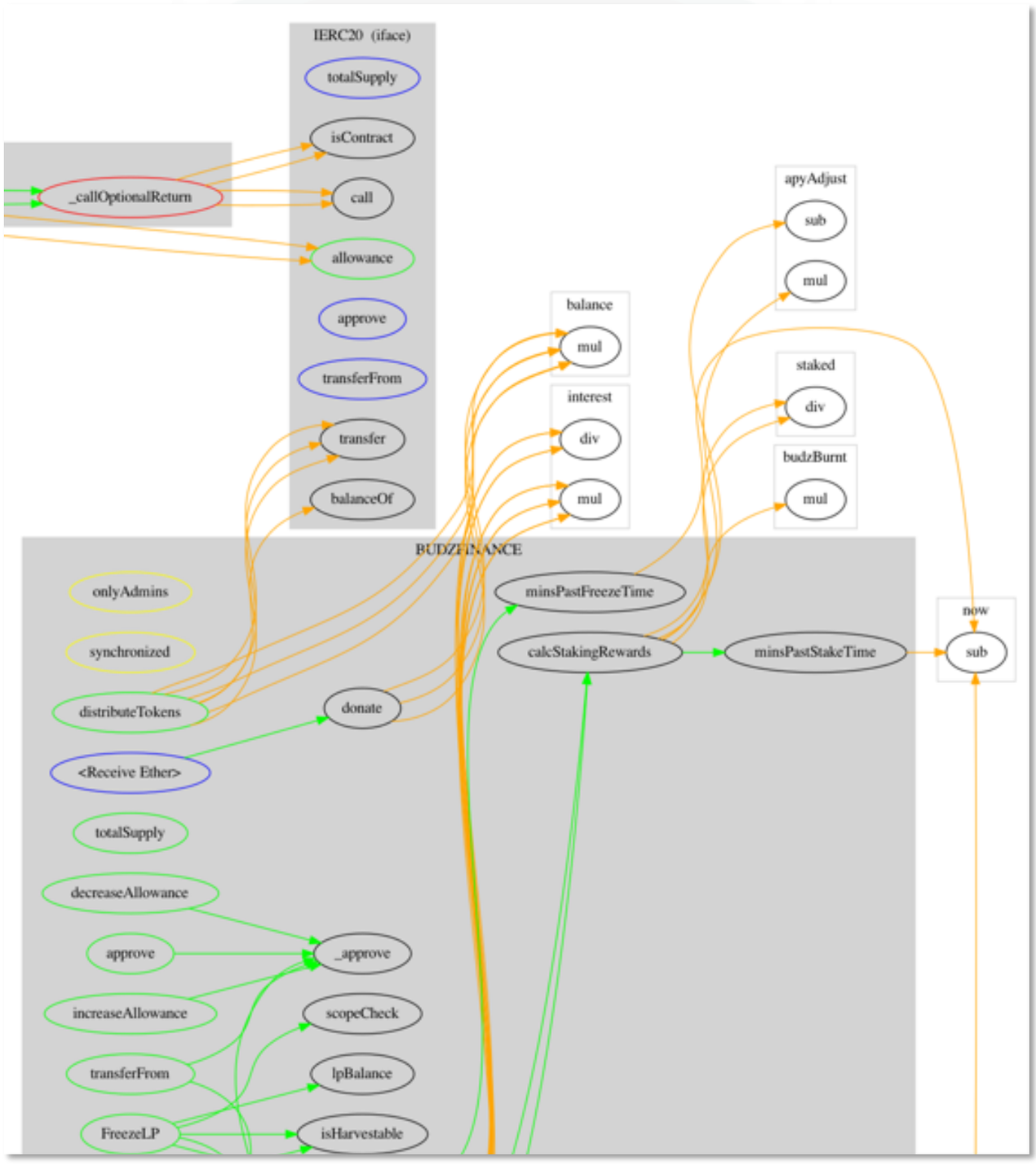
Control Flow

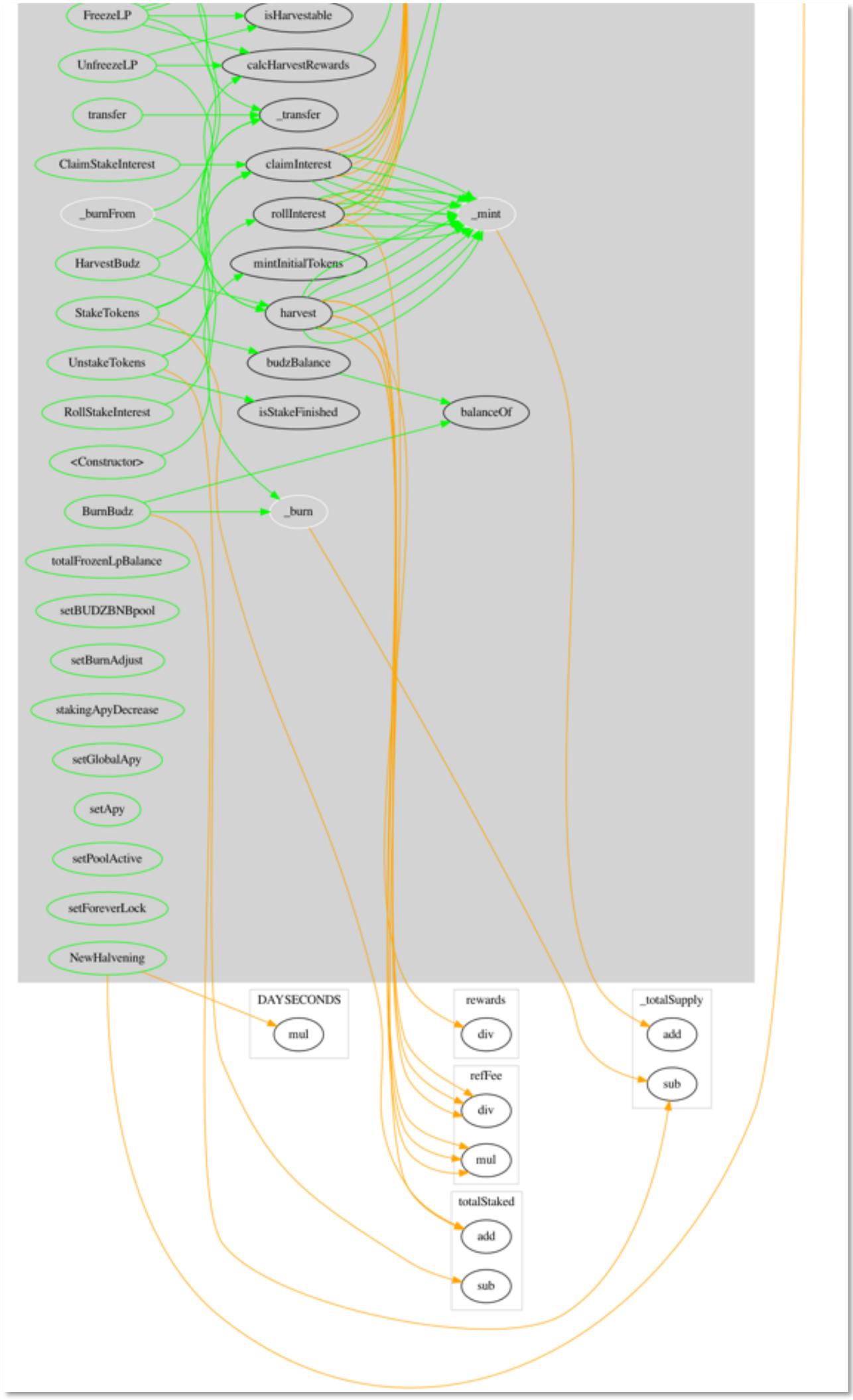




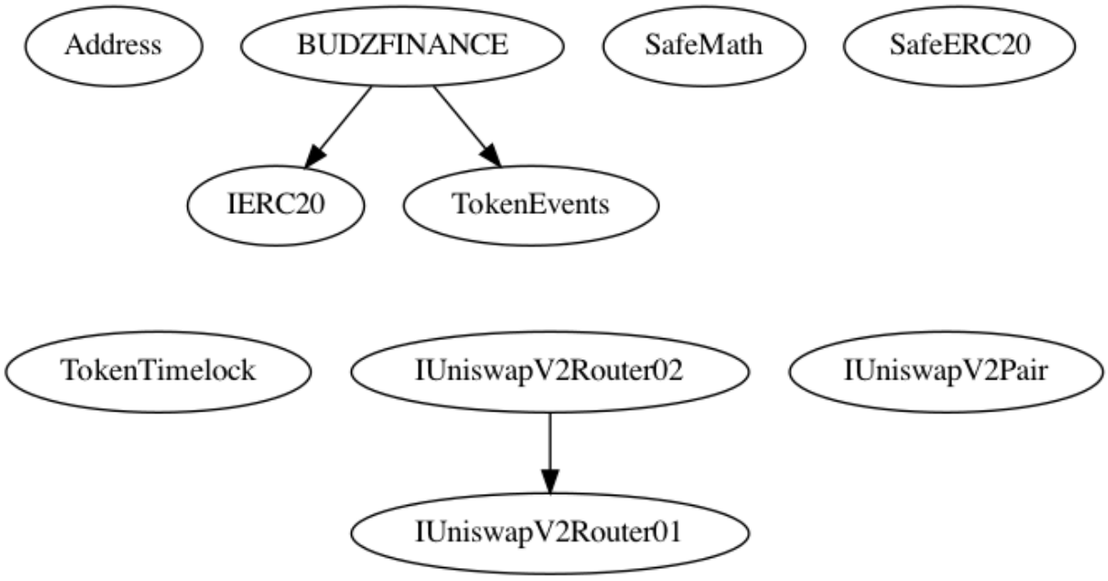



TokenTimelock
<Constructor>
releaseTime
beneficiary
release
token
SafeMath (lib)
add
sub
mul
div
mod
IERC20 (i
totalSupp
isContra
call
allowans
SafeERC20 (lib)
safeApprove
_callOptionalReturn


IERC20 (iface)
totalSupply
isContract
call
allowance
approve
transferFrom
transfer
balanceOf
_callOptionalReturn
apyAdjust
sub
mul
balance
mul
staked
div
interest
div
budzBurnt
mul
mul
BUDZFINANCE
onlyAdmins
synchronized
distributeTokens
<Receive Ether>
donate
totalSupply
decreaseAllowance
approve
_approve
increaseAllowance
scopeCheck
transferFrom
lpBalance
FreezeLP
isHarvestable
minsPastFreezeTime
calcStakingRewards
minsPastStakeTime
now
sub


FreezeLP
UnfreezeLP
transfer
ClaimStakeInterest
_burnFrom
HarvestBudz
StakeTokens
UnstakeTokens
RollStakeInterest
<Constructor>
BurnBudz
totalFrozenLpBalance
setBUDZBNBpool
setBurnAdjust
stakingApyDecrease
setGlobalApy
setApy
setPoolActive
setForeverLock
NewHalvening
isHarvestable
calcHarvestRewards
_transfer
claimInterest
rollInterest
mintInitialTokens
harvest
budzBalance
isStakeFinished
_burn
_mint
balanceOf
DAYSECONDS
mul
rewards
div
refFee
div
mul
totalStaked
add
sub
_totalSupply
add
sub

Inheritance Graph



END OF REPORT
