SolyzerX
BLOCKCHAIN SECURITY

BRISE

Security Assesment

MARCH 2023

SolyzerX

SolyzerX.com

Prepared for:

Gert Sanem
Bitgert Founder/CEO

Prepared by:

Justin Aaron
SolyzerX Engineer

Introduction

Auditing Firm	SolyzerX
Client Firm	Bitgert
Methodology	Automated Analysis, Manual Code Review
Language	Solidity
Contract	0x8FFf93E810a2eDaaFc326eDEE51071DA9d398E83
Blockchain	BNB Chain
Centralization	Active Ownership
Website	https://bitgert.com/
Discord	https://discord.io/bitgerbrise
Telegram	https://t.me/bitgertbrise
Twitter	https://twitter.com/bitgertbrise
Report Date	March 8, 2023

Verify the authenticity of this report on our website: https://solyzerx.com/projects/bitgert

SolyzerX

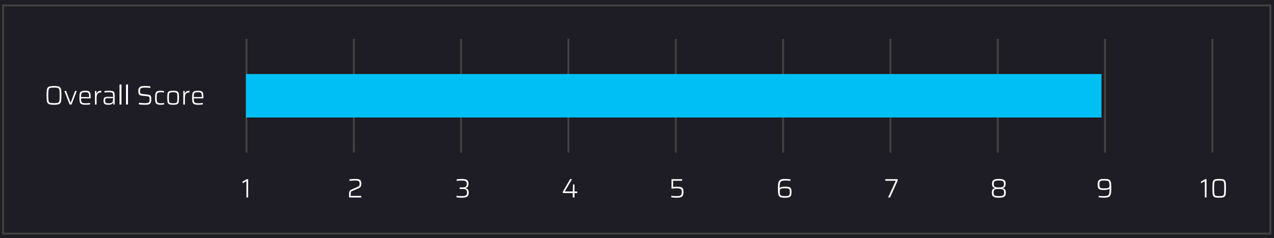
Executive Summary

SolyzerX has performed the automated and manual analysis of solidity codes. Solidity codes were reviewed for common contract vulnerabilities and centralized exploits. Here's a quick audit summary:

Severity	High	Medium	Low	Informational	Undetermined
Count	2	4	8	9	0

Category	Denial of service	Data Validation	Arithmetic	Auditing and Logging	Undefined Behavior
Count	0	1	1	10	11

BRISE smart contract source codes have achieved the following score: 9.0



Please note that smart contracts deployed on blockchains aren't resistant to exploits, vulnerabilities and/or hacks. Blockchain and cryptography assets utilize new and emerging technologies. These technologies present a high level of ongoing risks. For a detailed understanding of risk severity, source code vulnerability, and audit limitations, kindly review the audit report thoroughly.

Please note that centralization priviledges regardless of their inherited risk status - constitute an elevated impact on smart contract safety and security.

Table of Contents

Scope of Work

SolyzerX volunteered to conduct a Bitgert (BRISE) smart contract audit of their solidity source codes.

The audit scope of work is strictly limited to mentioned solidity file(s) only:

- BRISE.sol

If source codes are not deployed on the main net, they can be modified or altered before main-net deployment. Verify the contract's deployment status below:

Public Contract Link	
https://bscscan.com/token/0x8fff93e810a2edaafc326edee51071da9d398e83#code	
Contract Name	BRISE
Compiler Version	v0.6.12+commit.27d51765
License	MIT license

SolyzerX

Audit Methodology

Smart contract audits are conducted using a set standards and procedures. Mutual collaboration is essential to performing an effective smart contract audit. Here's a brief overview of SolyzerX's auditing process and methodology:

Connect

- The onboarding team gathers source codes, and specifications to make sure we understand the size, and scope of the smart contract audit.

Audit

- Automated analysis is performed to identify common contract vulnerabilities. We may use the following third-party frameworks and dependencies to perform the automated analysis:
    - Remix IDE Developer Tool
    - Open Zeppelin Code Analyzer
    - Slither-SolyzerX
    - SWC Vulnerabilities Registry

- Simulations are performed to identify centralized exploits causing contract and/or trade locks.

- A manual line-by-line analysis is performed to identify contract issues and centralized privileges. We may inspect below mentioned common contract vulnerabilities, and centralized exploits:

Centralized Exploits	• Token Supply Manipulation • Access Control and Authorization • Assets Manipulation • Ownership Control • Liquidity Access • Stop and Pause Trading • Ownable Library Verification

Common Contract Vulnerabilities	• Integer Overflow • Lack of Arbitrary limits • Incorrect Inheritance Order • Typographical Errors • Requirement Violation • Gas Optimization • Coding Style Violations • Re-entrancy • Third-Party Dependencies • Potential Sandwich Attacks • Irrelevant Codes • Divide before multiply • Conformance to Solidity Naming Guides • Compiler Specific Warnings • Language Specific Warnings

Report

- The auditing team provides a preliminary report specifying all the checks which have been performed and the findings thereof.
- The client's development team reviews the report and makes amendments to solidity codes.
- The auditing team provides the final comprehensive report with open and unresolved issues.

Publish

- The client may use the audit report internally or disclose it publicly.

It is important to note that there is no pass or fail in the audit, it is recommended to view the audit as an unbiased assessment of the safety of solidity codes.

SolyzerX

Risk Categories

Smart contracts are generally designed to hold, approve, and transfer tokens. This makes them very tempting attack targets. A successful external attack may allow the external attacker to directly exploit. A successful centralization-related exploit may allow the privileged role to directly exploit. All risks which are identified in the audit report are categorized here for the reader to view:

Risk Type	Definition
High	These risks could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
Medium	These risks are hard to exploit but very important to fix, they carry an elevated risk of smart contract manipulation, which can lead to high-risk severity.
Low	These risks should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution. Low-risk re-entrancy-related vulnerabilities should be fixed to deter exploits.
Informational	These risks do not pose a considerable risk to the contract or those who interact with it. They are code-style violations and deviations from standard practices. They should be highlighted and fixed nonetheless.
Undetermined	These risks pose uncertain severity to the contract or those who interact with it. They should be fixed to mitigate the risk uncertainty.

All category breakdown which are identified in the audit report are categorized here for the reader to review:

Category Breakdown				
Denial of service	Data Validation	Arithmetic	Auditing and Logging	Undefined Behavior

SolyzerX

Centralized Privileges

Centralization risk is the most common cause of cryptography asset loss. When a smart contract has a privileged role, the risk related to centralization is elevated.

There are some well-intended reasons have privileged roles, such as:

- Privileged roles can be granted the power to `pause()` the contract in case of an external attack.
- Privileged roles can use functions like, `include()` , and `exclude()` to add or remove wallets from fees, swap checks, and transaction limits. This is useful to run a presale and to list on an exchange.

Authorizing privileged roles to externally-owned-account (EOA) is dangerous. Lately, centralization-related losses are increasing in frequency and magnitude.

- The client can lower centralization-related risks by implementing below mentioned practices:
- Privileged role's private key must be carefully secured to avoid any potential hack.
- Privileged role should be shared by multi-signature (multi-sig) wallets.
- Authorized privilege can be locked in a contract, user voting, or community DAO can be introduced to unlock the privilege.
- Renouncing the contract ownership, and privileged roles.
- Remove functions with elevated centralization risk.

Understand the project's initial asset distribution. Assets in the liquidity pair should be locked. Assets outside the liquidity pair should be locked with a release schedule.

SolyzerX

Automated Analysis

Contract	Function	Visibility	Modifiers
BRISE	constructor	Public	
	owner	Public	
	renounceOwnership	Public	onlyOwner
	transferOwnership	Public	onlyOwner
	_msgSender	Internal	
	_msgData	Internal	
	constructor	Public	
	name	Public	
	symbol	Public	
	decimals	Public	
	totalSupply	Public	
	balanceOf	Public	
	transfer	Public	
	allowance	Public	
	approve	Public	
	transferFrom	Public	
	increaseAllowance	Public	

	decreaseAllowance	Public	
	_transfer	Internal	
	_mint	Internal	
	_burn	Internal	
	_approve	Internal	
	_beforeTokenTransfer	Internal	
	name	External	
	symbol	External	
	decimals	External	
	totalSupply	External	
	balanceOf	External	
	transfer	External	
	allowance	External	
	approve	External	
	transferFrom	External	
	constructor	Public	
	receive	External	
	decimals	Public	
	updateDividendTracker	Public	onlyOwner

	updateUniswapV2Router	Public	onlyOwner
	excludeFromFees	Public	onlyOwner
	excludeMultipleAccountsFromFees	Public	onlyOwner
	setAutomatedMarketMakerPair	Public	onlyOwner
	_setAutomatedMarketMakerPair	Private	
	updateLiquidityWallet	Public	onlyOwner
	updateGasForProcessing	Public	onlyOwner
	updateClaimWait	External	onlyOwner
	getClaimWait	External	
	getTotalDividendsDistributed	External	
	isExcludedFromFees	Public	
	withdrawableDividendOf	Public	
	dividendTokenBalanceOf	Public	
	getAccountDividendsInfo	External	
	getAccountDividendsInfoAtIndex	External	
	withdraw	External	onlyOwner

	processDividendTracker	External	
	claim	External	
	getLastProcessedIndex	External	
	getNumberOfDividendTokenHolders	External	
	setMaxSellTxAMount	External	onlyOwner
	setSwapTokensAmt	External	onlyOwner
	setBNBRewardsFee	External	onlyOwner
	setMarketingFee	External	onlyOwner
	setMarketingWallet	External	onlyOwner
	addToBlackList	External	onlyOwner
	removeFromBlackList	External	onlyOwner
	setSwapEnabled	External	onlyOwner
	setBuyBackFee	External	onlyOwner
	_transfer	Internal	
	swapAndSendToMarketing	Private	lockTheSwap
	swapTokensForEth	Private	
	buyBackTokens	Private	lockTheSwap
	swapETHForTokens	Private	

	swapBuyBackTokens	Private	lockTheSwap
	setBuyBackEnabled	Public	onlyOwner
	setBuybackUpperLimit	External	onlyOwner
	swapAndSendDividends	Private	lockTheSwap
BRISEDividendTracker	constructor	Public	
	owner	Public	
	renounceOwnership	Public	onlyOwner
	transferOwnership	Public	onlyOwner
	_msgSender	Internal	
	_msgData	Internal	
	constructor	Public	
	receive	External	
	distributeDividends	Public	
	withdrawDividend	Public	
	_withdrawDividendOfUser	Internal	
	dividendOf	Public	
	withdrawableDividendOf	Public	
	withdrawnDividendOf	Public	

	accumulativeDividendOf	Public	
	_transfer	Internal	
	_mint	Internal	
	_burn	Internal	
	_setBalance	Internal	
	withdrawableDividendOf	External	
	withdrawnDividendOf	External	
	accumulativeDividendOf	External	
	dividendOf	External	
	distributeDividends	External	
	withdrawDividend	External	
	constructor	Public	
	name	Public	
	symbol	Public	
	decimals	Public	
	totalSupply	Public	
	balanceOf	Public	
	transfer	Public	

	allowance	Public	
	approve	Public	
	transferFrom	Public	
	increaseAllowance	Public	
	decreaseAllowance	Public	
	_transfer	Internal	
	_mint	Internal	
	_burn	Internal	
	_approve	Internal	
	_beforeTokenTransfer	Internal	
	name	External	
	symbol	External	
	decimals	External	
	totalSupply	External	
	balanceOf	External	
	transfer	External	
	allowance	External	
	approve	External	
	transferFrom	External	

	constructor	Public	
	_transfer	Internal	
	withdrawDividend	Public	
	excludeFromDividends	External	onlyOwner
	updateClaimWait	External	onlyOwner
	getLastProcessedIndex	External	
	getNumberOfTokenHolders	External	
	getAccount	Public	
	getAccountAtIndex	Public	
	canAutoClaim	Private	
	setBalance	External	onlyOwner
	process	Public	
	processAccount	Public	onlyOwner
Context	_msgSender	Internal	
	_msgData	Internal	
DividendPayingToken	withdrawableDividendOf	External	
	withdrawnDividendOf	External	

	accumulativeDividendOf	External	
	dividendOf	External	
	distributeDividends	External	
	withdrawDividend	External	
	constructor	Public	
	name	Public	
	symbol	Public	
	decimals	Public	
	totalSupply	Public	
	balanceOf	Public	
	transfer	Public	
	allowance	Public	
	approve	Public	
	transferFrom	Public	
	increaseAllowance	Public	
	decreaseAllowance	Public	
	_transfer	Internal	
	_mint	Internal	
	_burn	Internal	

	_approve	Internal	
	_beforeTokenTransfer	Internal	
	name	External	
	symbol	External	
	decimals	External	
	totalSupply	External	
	balanceOf	External	
	transfer	External	
	allowance	External	
	approve	External	
	transferFrom	External	
	_msgSender	Internal	
	_msgData	Internal	
	constructor	Public	
	receive	External	
	distributeDividends	Public	
	withdrawDividend	Public	
	_withdrawDividendOfUser	Internal	
	dividendOf	Public	

	withdrawableDividendOf	Public	
	withdrawnDividendOf	Public	
	accumulativeDividendOf	Public	
	_transfer	Internal	
	_mint	Internal	
	_burn	Internal	
	_setBalance	Internal	
DividendPayingTokenInterface	dividendOf	External	
	distributeDividends	External	
	withdrawDividend	External	
DividendPayingTokenOptionalInterface	withdrawableDividendOf	External	
	withdrawnDividendOf	External	
	accumulativeDividendOf	External	
ERC20	name	External	
	symbol	External	

	decimals	External	
	totalSupply	External	
	balanceOf	External	
	transfer	External	
	allowance	External	
	approve	External	
	transferFrom	External	
	_msgSender	Internal	
	_msgData	Internal	
	constructor	Public	
	name	Public	
	symbol	Public	
	decimals	Public	
	totalSupply	Public	
	balanceOf	Public	
	transfer	Public	
	allowance	Public	
	approve	Public	
	transferFrom	Public	

	increaseAllowance	Public	
	decreaseAllowance	Public	
	_transfer	Internal	
	_mint	Internal	
	_burn	Internal	
	_approve	Internal	
	_beforeTokenTransfer	Internal	
IERC20	totalSupply	External	
	balanceOf	External	
	transfer	External	
	allowance	External	
	approve	External	
	transferFrom	External	
IERC20Metadata	totalSupply	External	
	balanceOf	External	
	transfer	External	
	allowance	External	

	approve	External	
	transferFrom	External	
	name	External	
	symbol	External	
	decimals	External	
IUniswapV2Factory	feeTo	External	
	feeToSetter	External	
	getPair	External	
	allPairs	External	
	allPairsLength	External	
	createPair	External	
	setFeeTo	External	
	setFeeToSetter	External	
IUniswapV2Pair	name	External	
	symbol	External	
	decimals	External	
	totalSupply	External	

	balanceOf	External	
	allowance	External	
	approve	External	
	transfer	External	
	transferFrom	External	
	DOMAIN_SEPARATOR	External	
	PERMIT_TYPEHASH	External	
	nonces	External	
	permit	External	
	MINIMUM_LIQUIDITY	External	
	factory	External	
	token0	External	
	token1	External	
	getReserves	External	
	price0CumulativeLast	External	
	price1CumulativeLast	External	
	kLast	External	
	mint	External	
	burn	External	

	swap	External	
	skim	External	
	sync	External	
	initialize	External	
IUniswapV2Router01	factory	External	
	WETH	External	
	addLiquidity	External	
	addLiquidityETH	External	
	removeLiquidity	External	
	removeLiquidityETH	External	
	removeLiquidityWithPermit	External	
	removeLiquidityETHWithPermit	External	
	swapExactTokensForTokens	External	
	swapTokensForExactTokens	External	
	swapExactETHForTokens	External	
	swapTokensForExactETH	External	
	swapExactTokensForETH	External	

	swapETHForExactTokens	External	
	quote	External	
	getAmountOut	External	
	getAmountIn	External	
	getAmountsOut	External	
	getAmountsIn	External	
IUniswapV2Router02	factory	External	
	WETH	External	
	addLiquidity	External	
	addLiquidityETH	External	
	removeLiquidity	External	
	removeLiquidityETH	External	
	removeLiquidityWithPermit	External	
	removeLiquidityETHWithPermit	External	
	swapExactTokensForTokens	External	
	swapTokensForExactTokens	External	
	swapExactETHForTokens	External	

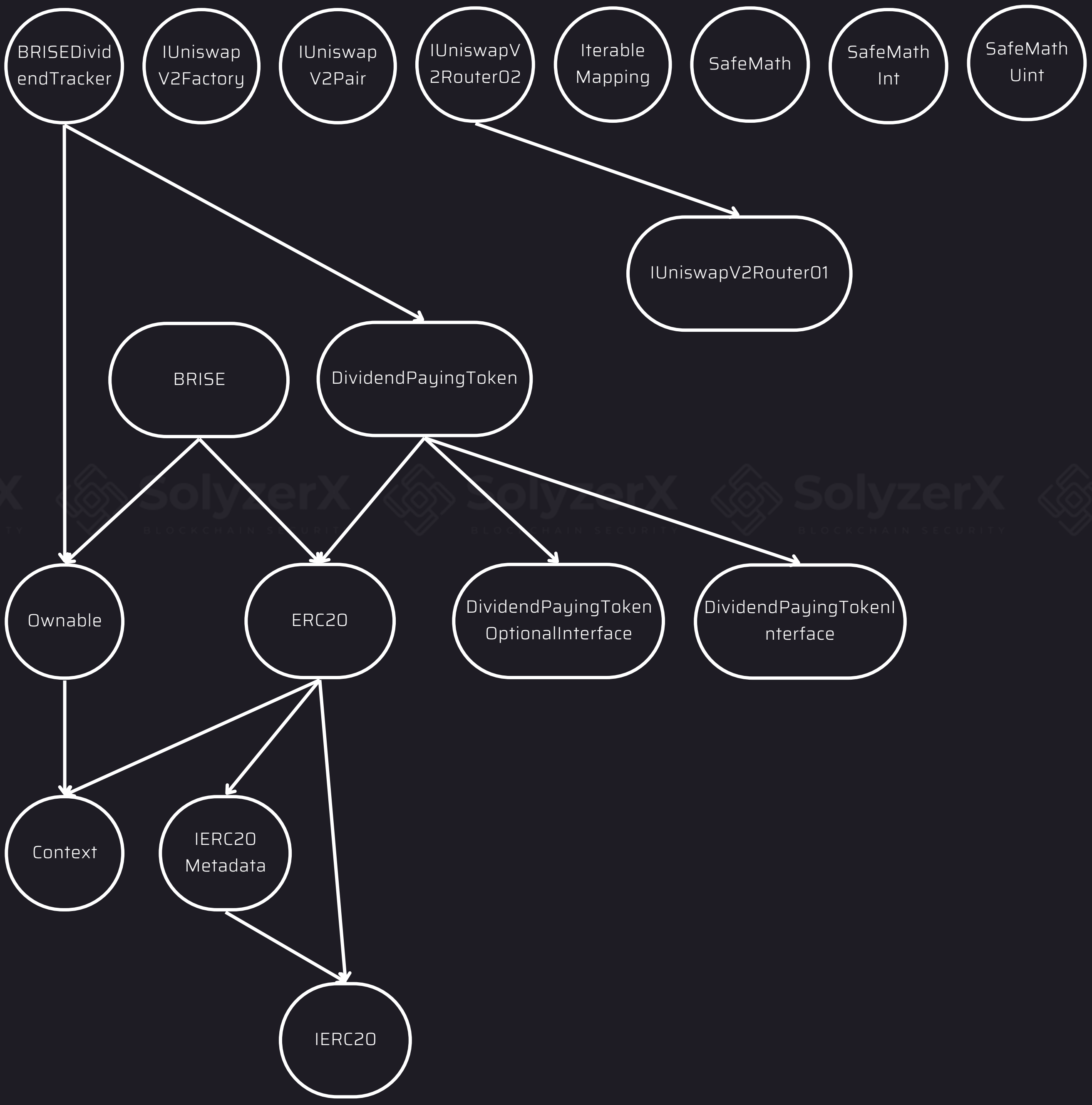
	swapTokensForExactETH	External	
	swapExactTokensForETH	External	
	swapETHForExactTokens	External	
	quote	External	
	getAmountOut	External	
	getAmountIn	External	
	getAmountsOut	External	
	getAmountsIn	External	
	removeLiquidityETHSupportingFeeOnTransferTokens	External	
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	

IterableMapping	get	Public	
	getIndexOfKey	Public	
	getKeyAtIndex	Public	
	size	Public	
	set	Public	
	remove	Public	
Ownable	_msgSender	Internal	
	_msgData	Internal	
	constructor	Public	
	owner	Public	
	renounceOwnership	Public	onlyOwner
	transferOwnership	Public	onlyOwner
SafeMath	add	Internal	
	sub	Internal	
	sub	Internal	
	mul	Internal	
	div	Internal	

	div	Internal	
	mod	Internal	
	mod	Internal	
SafeMathInt	mul	Internal	
	div	Internal	
	sub	Internal	
	add	Internal	
	abs	Internal	
	toUint256Safe	Internal	
SafeMathUint	toInt256Safe	Internal	

SolyzerX

Inheritance Graph



Findings Summary

#	Title	Type	Severity
1	Arbitrary-send-eth	Data Validation	High
2	Reentrancy vulnerabilities	Auditing and Logging	High
3	Reentrancy vulnerabilities	Auditing and Logging	Medium
4	Uninitialized-local	Undefined Behavior	Medium
5	Unused return	Undefined Behavior	Medium
6	Write after write	Undefined Behavior	Medium
7	Local variable shadowing	Auditing and Logging	Low
8	Events-maths	Arithmetic	Low
9	Missing-zero-check	Undefined Behavior	Low
10	Calls-loop	Auditing and Logging	Low

11	Variable-scope	Auditing and Logging	Low
12	Reentrancy-benign	Auditing and Logging	Low
13	Reentrancy-events	Auditing and Logging	Low
14	Timestamp	Auditing and Logging	Low
15	Dead-code	Undefined Behavior	Informational
16	Solc-version	Auditing and Logging	Informational
17	Low-level-calls	Undefined Behavior	Informational
18	Naming-convention	Undefined Behavior	Informational
19	Redundant-statements	Undefined Behavior	Informational
20	Reentrancy-unlimited-gas	Auditing and Logging	Informational
21	Similar-names	Undefined Behavior	Informational
22	Too-many-digits	Undefined Behavior	Informational
23	Unused-state	Undefined Behavior	Informational

Detailed Findings

1. arbitrary-send-eth	
Severity: High	Difficulty: **Medium**
Type: Data Validation	Finding ID: Bitrise.sol# 417-425 & # 454-469
Target: Bitrise.sol	

Description

Unprotected call to a function sending Ether to an arbitrary address.

Exploit Scenario:

```
contract ArbitrarySendEth{
    address destination;
    function setDestination(){
        destination = msg.sender;
    }

    function withdraw() public{
        destination.transfer(this.balance);
    }
}
```

Bob calls setDestination and withdraw. As a result he withdraws the contract's balance.

Recommendation

Ensure that an arbitrary user cannot withdraw unauthorized funds.

2. Reentrancy vulnerabilities	
Severity: High	Difficulty: Medium
Type: Auditing and Logging	Finding ID: Bitrise.sol# 330-415
Target: Bitrise.sol	

Description

Detection of the reentrancy bug. Do not report reentrancies that don't involve Ether (see reentrancy-no-eth)

Exploit Scenario:

```
    function withdrawBalance(){
        // send userBalance[msg.sender] Ether to msg.sender
        // if mgs.sender is a contract, it will call its fallback
function
        if( ! (msg.sender.call.value(userBalance[msg.sender])() ) ){
            throw;
        }
        userBalance[msg.sender] = 0;
    }
```

Bob uses the re-entrancy bug to call withdrawBalance two times, and withdraw more than its initial deposit to the contract.

Recommendation

Apply the check-effects-interactions pattern.

SolyzerX

3. Reentrancy vulnerabilities	
Severity: Medium	Difficulty: Medium
Type: Auditing and Logging	Finding ID: Bitrise.sol#148-163
Target: Bitrise.sol	

Description

Detection of the reentrancy bug. Do not report reentrancies that don't involve Ether (see reentrancy-no-eth)

Exploit Scenario:

```
function bug(){
    require(not_called);
    if( ! (msg.sender.call() ) ){
        throw;
    }
    not_called = False;
}
```

Recommendation

Apply the check-effects-interactions pattern.

SolyzerX

4. uninitialized-local	
Severity: Medium	Difficulty: Medium
Type: Undefined Behavior	Finding ID: Bitrise.sol#148-163
Target: Bitrise.sol	

Description

Uninitialized local variables.

Exploit Scenario:

```
contract Uninitialized is Owner{
    function withdraw() payable public onlyOwner{
        address to;
        to.transfer(this.balance)
    }
}
```

Bob calls transfer. As a result, all Ether is sent to the address 0x0 and is lost.

Recommendation

Initialize all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero to improve code readability.

5. Unused return	
Severity: Medium	Difficulty: Medium
Type: Undefined Behavior	Finding ID: Bitrise.sol# 279-281
Target: Bitrise.sol	

Description

The return value of an external call is not stored in a local or state variable.

Exploit Scenario:

```
contract MyConc{
    using SafeMath for uint;
    function my_func(uint a, uint b) public{
        a.add(b);
    }
}
```

MyConc calls add of SafeMath, but does not store the result in a. As a result, the computation has no effect.

Recommendation

Ensure that all the return values of the function calls are used.

SolyzerX

6. Write after write	
Severity: Medium	Difficulty: High
Type: Undefined Behavior	Finding ID: Bitrise.sol#354
Target: Bitrise.sol	

Description

Detects variables that are written but never read and written again.

Exploit Scenario:

~~~
```solidity
contract Buggy{
    function my_func() external initializer{
        // ...
        a = b;
        a = c;
        // ..
    }
}
```
`a` is first asigned to `b`, and then to `c`. As a result the first write
does nothing.
~~~

Recommendation

Fix or remove the writes.

SolyzerX

7. Local variable shadowing	
Severity: Low	Difficulty: High
Type: Auditing and Logging	Finding ID: Bitrise.sol# 376
Target: Bitrise.sol	

Description

Detection of shadowing using local variables.

Exploit Scenario:

```
pragma solidity ^0.4.24;

contract Bug {
    uint owner;

    function sensitive_function(address owner) public {
        // ...
        require(owner == msg.sender);
    }

    function alternate_sensitive_function() public {
        address owner = msg.sender;
        // ...
        require(owner == msg.sender);
    }
}
```

sensitive_function.owner shadows Bug.owner. As a result, the use of owner in sensitive_function might be incorrect.

Recommendation

Rename the local variables that shadow another component.

SolyzerX

8. events-maths	
Severity: Low	Difficulty: Medium
Type: Arithmetic	Finding ID: Bitrise.sol# 291-293
Target: Bitrise.sol	

Description

Detect missing events for critical arithmetic parameters.

Exploit Scenario:

```
contract C {

    modifier onlyOwner {
        if (msg.sender != owner) throw;
        _;
    }

    function setBuyPrice(uint256 newBuyPrice) onlyOwner public {
        buyPrice = newBuyPrice;
    }

    function buy() external {
     ... // buyPrice is used to determine the number of tokens purchased
    }
}
```

setBuyPrice() does not emit an event, so it is difficult to track changes in the value of buyPrice off-chain.

Recommendation

Emit an event for critical parameter changes.

9. missing-zero-check	
Severity: Low	Difficulty: Medium
Type: Undefined Behavior	Finding ID: Bitrise.sol#307
Target: Bitrise.sol	

Description

Detect missing zero address validation.

Exploit Scenario:

```
contract C {

  modifier onlyAdmin {
    if (msg.sender != owner) throw;
    _;
  }

  function updateOwner(address newOwner) onlyAdmin external {
    owner = newOwner;
  }
}
```

setBuyPrice() does not emit an event, so it is difficult to track changes in the value of buyPrice off-chain.

Recommendation

Emit an event for critical parameter changes.

10. calls-loop	
Severity: Low	Difficulty: Medium
Type: Auditing and Logging	Finding ID: DividendPayingToken.sol# 86-102
Target: DividendPayingToken.sol	

Description

Calls inside a loop might lead to a denial-of-service attack.

Exploit Scenario:

```
contract CallsInLoop{

    address[] destinations;

    constructor(address[] newDestinations) public{
        destinations = newDestinations;
    }

    function bad() external{
        for (uint i=0; i < destinations.length; i++){
            destinations[i].transfer(i);
        }
    }

}
```

If one of the destinations has a fallback function that reverts, bad will always revert.

Recommendation

Favor pull over push strategy for external calls.

11. variable-scope	
Severity: Low	Difficulty: High
Type: Auditing and Logging	Finding ID: Bitrise.sol# 408
Target: Bitrise.sol	

Description

Detects the possible usage of a variable before the declaration is stepped over (either because it is later declared, or declared in another scope).

Exploit Scenario:

```
contract C {
    function f(uint z) public returns (uint) {
        uint y = x + 9 + z; // 'z' is used pre-declaration
        uint x = 7;

        if (z % 2 == 0) {
            uint max = 5;
            // ...
        }

        // 'max' was intended to be 5, but it was mistakenly declared in
a scope and not assigned (so it is zero).
        for (uint i = 0; i < max; i++) {
            x += 1;
        }

        return x;
    }
}
```

In the case above, the variable x is used before its declaration, which may result in unintended consequences. Additionally, the for-loop uses the variable max, which is declared in a previous scope that may not always be reached. This could lead to unintended consequences if the user mistakenly uses a variable prior to any intended declaration assignment. It also may indicate that the user intended to reference a different variable.

Recommendation

Move all variable declarations prior to any usage of the variable, and ensure that reaching a variable declaration does not depend on some conditional if it is used unconditionally.

12. reentrancy-benign	
Severity: Low	Difficulty: Medium
Type: Auditing and Logging	Finding ID: Bitrise.sol# 330-415
Target: Bitrise.sol	

Description

Detection of the reentrancy bug. Only report reentrancy that acts as a double call (see reentrancy-eth, reentrancy-no-eth).

Exploit Scenario:

```
function callme(){
    if( ! (msg.sender.call()() ) ){
        throw;
    }
    counter += 1
}
```

callme contains a reentrancy. The reentrancy is benign because it's exploitation would have the same effect as two consecutive calls.

Recommendation

Apply the check-effects-interactions pattern.

13. reentrancy-events	
Severity: **Low**	Difficulty: **Medium**
Type: Auditing and Logging	Finding ID: Bitrise.sol#193-202
Target: Bitrise.sol	

Description

Detection of the reentrancy bug. Only report reentrancies leading to out-of-order events.

Exploit Scenario:

```
function bug(Called d){
    counter += 1;
    d.f();
    emit Counter(counter);
}
```

If d.() re-enters, the Counter events will be shown in an incorrect order, which might lead to issues for third parties.

Recommendation

Apply the check-effects-interactions pattern.

14. timestamp	
Severity: Low	Difficulty: Medium
Type: Auditing and Logging	Finding ID: Bitrise.sol# 555-598
Target: Bitrise.sol	

Description

Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners.

Exploit Scenario:

"Bob's contract relies on block.timestamp for its randomness. Eve is a miner and manipulates block.timestamp to exploit Bob's contract.

Recommendation

Avoid relying on block.timestamp.

15. dead-code	
Severity: **Informational**	Difficulty: **Medium**
Type: Undefined Behavior	Finding ID: Context.sol# 20-23
Target: Context.sol	

Description

Functions that are not sued.

Exploit Scenario:

```
contract Contract{
    function dead_code() internal() {}
}
```

dead_code is not used in the contract, and make the code's review more difficult.

Recommendation

Remove unused functions.

SolyzerX

16. solc-version	
Severity: Informational	Difficulty: High
Type: Auditing and Logging	Finding ID: Bitrise.sol#3
Target: Bitrise.sol	

Description

solc frequently releases new compiler versions. Using an old version prevents access to new Solidity security checks. We also recommend avoiding complex pragma statement.

Recommendation

Deploy with any of the following Solidity versions:

- 0.5.16 - 0.5.17
- 0.6.11 - 0.6.12
- 0.7.5 - 0.7.6
- 0.8.16

The recommendations take into account:

- Risks related to recent releases
- Risks of complex code generation changes
- Risks of new language features
- Risks of known bugs

Use a simple pragma version that allows any of these versions. Consider using the latest version of Solidity for testing.

17. low-level-calls	
Severity: Informational	Difficulty: High
Type: Undefined Behavior	Finding ID: Bitrise.sol# 483-492
Target: Bitrise.sol	

Description

The use of low-level calls is error-prone. Low-level calls do not check for code existence or call success.

Recommendation

Avoid low-level calls. Check the call success. If the call is meant for a contract, check for code existence.

18. naming-convention	
Severity: Informational	Difficulty: High
Type: Undefined Behavior	Finding ID: Bitrise.sol#475
Target: Bitrise.sol	

Description

Solidity defines a naming convention that should be followed.

Rule exceptions

- Allow constant variable name/symbol/decimals to be lowercase (ERC20).
- Allow _ at the beginning of the mixed_case match for private variables and unused parameters.

Recommendation

Follow the Solidity naming convention.

19. redundant-statements	
Severity: Informational	Difficulty: High
Type: Undefined Behavior	Finding ID: Context.sol# 21
Target: Context.sol	

Description

Detect the usage of redundant statements that have no effect.

Exploit Scenario:

```
contract RedundantStatementsContract {

    constructor() public {
        uint; // Elementary Type Name
        bool; // Elementary Type Name
        RedundantStatementsContract; // Identifier
    }

    function test() public returns (uint) {
        uint; // Elementary Type Name
        assert; // Identifier
        test; // Identifier
        return 777;
    }
}
```

Each commented line references types/identifiers, but performs no action with them, so no code will be generated for such statements and they can be removed.

Recommendation

Remove redundant statements if they congest code but offer no value.

20. reentrancy-unlimited-gas	
Severity: Informational	Difficulty: Medium
Type: Auditing and Logging	Finding ID: Bitrise.sol# 330-415
Target: Bitrise.sol	

Description

Detection of the reentrancy bug. Only report reentrancy that is based on transfer or send.

Exploit Scenario:

```
function callme(){
    msg.sender.transfer(balances[msg.sender]):
    balances[msg.sender] = 0;
}
```

send and transfer do not protect from reentrancies in case of gas price changes.

Recommendation

Apply the check-effects-interactions pattern.

21. similar-names	
Severity: **Informational**	Difficulty: Medium
Type: Undefined Behavior	Finding ID: DividendPayingToken.sol# 87
Target: DividendPayingToken.sol	

Description

Detect variables with names that are too similar.

Exploit Scenario:

Bob uses several variables with similar names. As a result, his code is difficult to review.

Recommendation

Prevent variables from having similar names.

22. too-many-digits	
Severity: Informational	Difficulty: Medium
Type: Undefined Behavior	Finding ID: Bitrise.sol#97-138
Target: Bitrise.sol	

Description

Literals with many digits are difficult to read and review.

Exploit Scenario:

```
contract MyContract{
    uint 1_ether = 10000000000000000000;
}
```

While 1_ether looks like 1 ether, it is 10 ether. As a result, it's likely to be used incorrectly.

Recommendation

Use:

- Ether suffix,
- Time suffix, or
- The scientific notation

23. unused-state	
Severity: Informational	Difficulty: High
Type: Undefined Behavior	Finding ID: SafeMathInt.sol#36
Target: SafeMathInt.sol	

Description

Unused state variable.

Recommendation

Remove unused state variables.

Disclaimers

SolyzerX provides the easy-to-understand audit of solidity source codes (commonly known as smart contracts).

The smart contract for this particular audit was analyzed for common contract vulnerabilities, and centralization exploits. This audit report makes no statements or warranties on the security of the code. This audit report does not provide any warranty or guarantee regarding the absolute bug-free nature of the smart contract analyzed, nor do they provide any indication of the client's business, business model or legal compliance. This audit report does not extend to the compiler layer, any other areas beyond the programming language, or other programming aspects that could present security risks. Cryptographic tokens are emergent technologies, they carry high level of technical risks and uncertainty. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. This audit report could include false positives, false negatives, and other unpredictable results.

Confidentiality

This report is subject to the terms and conditions (including without limitations, description of services, confidentiality, disclaimer and limitation of liability) outlined in the scope of the audit provided to the client. This report should not be transmitted, disclosed, referred to, or relied upon by any individual for any purpose without SolyzerX's prior written consent.

No Financial Advice

This audit report does not indicate the endorsement of any particular project or team, nor guarantees its security. No third party should rely on the reports in any way, including to make any decisions to buy or sell a product, service or any other asset. The information provided in this report does not constitute investment advice, financial advice, trading advice, or any other sort of advice and you should not treat any of the report's content as such. This audit report should not be used in any way to make decisions around investment or involvement. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort.

FOR AVOIDANCE OF DOUBT, SERVICES, INCLUDING ANY ASSOCIATED AUDIT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Technical Disclaimer

ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OTHER MATERIALS, OR ANY PRODUCTS OR RESULT OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, SOLYZERX HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO SERVICES, AUDIT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, SOLYZERX SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM THE COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

WITHOUT LIMITING THE FOREGOING, SOLYZERX MAKES NO WARRANTY OF ANY KIND THAT ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET THE CLIENT'S OR ANY OTHER INDIVIDUAL'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE.

Timeliness Of Content

The content contained in this audit report is subject to change without any prior notice. SolyzerX does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following the publication.

Links To Other Websites

This audit report provides, through hypertext or other computer links, access to websites and social accounts operated by individuals other than SolyzerX. Such hyperlinks are provided for your reference and convenience only and are the exclusive responsibility of such website's and social account's owners. You agree that SolyzerX is not responsible for the content or operation of such websites and social accounts and that SolyzerX shall have no liability to you or any other person or entity for the use of third-party websites and social accounts. You are solely responsible for determining the extent to which you may use any content at any other websites and social accounts to which you link from the report.

SolyzerX

About SolyzerX

Founded in 2022 and headquartered in Malaysia, SolyzerX provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code.

We provide solidity development, testing, and auditing services. We work on major public blockchains e.g., Ethereum, Binance, Cronos, Doge, Polygon, Avalanche, Metis, Fantom, Velas, Oasis, etc.

SolyzerX is built by engineers, developers, UI experts, and blockchain enthusiasts. Our team currently consists of 4 core members, and 5+ casual contributors.

Website: https://solyzerx.com

Email: support@solyzerx.com

Github: https://github.com/SolyzerX

Telegram (Channel): https://t.me/SolyzerX

Telegram (Foundation): https://t.me/SolyzerXFoundation