SolyzerX
BLOCKCHAIN SECURITY

ApeGrow

Security Assesment

SEPTEMBER 2022

SolyzerX

SolyzerX.com

Prepared for:

Justin Aaron
ApeGrow Core Team

Prepared by:

Adrian Mikhail

SolyzerX

Introduction

Auditing Firm	SolyzerX
Client Firm	ApeGrow
Methodology	Automated Analysis, Manual Code Review
Language	Solidity
Contract	0x8832742f2c2FA9bB5b83BE6940888269Od810471
Blockchain	Binance Smart Chain
Centralization	Active Ownership
Commit	1eda057b4aba05bc48a6ff9cdf405d19
Website	https://apegrow.xyz
Telegram	https://t.me/apegrowcoinportal
Twitter	https://twitter.com/apegrowcoin
Report Date	November 5, 2022

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

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	3	2	6	7	1

Category	Denial of service	Data Validation	Arithmetic	Auditing and Logging	Undefined Behavior
Count	0	6	1	5	7

ApeGrow's smart contract source codes have achieved the following score: 8.9

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.

SolyzerX

Table of Contents

Scope of Work

SolyzerX was consulted by ApeGrow to conduct the smart contract audit of their solidity source codes.

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

- ApeGrow.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/address/0x8832742f2c2FA9bB5b83BE69408882690d810471#code	
Contract Name	ApeGrow
Compiler Version	0.8.8
License	Unlicensed

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
  - DEX Dependencies, e.g., Pancakeswap, Uniswap
- 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

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.

Automated Analysis

Symbol	Definition
	Internal
	External
	Public
	Private

```
+--------+----------+---------+--------+-------+--------------+--------------+
|Function|Visibility|Modifiers|  Read  | Write |Internal Calls|External Calls|
+--------+----------+---------+--------+-------+--------------+--------------+
| **IBEP20** | Interface | |||
| L | totalSupply | External   | | | | | |
| L | balanceOf | External   | | | | | |
| L | transfer | External   | | | | | |
| L | allowance | External   | | | | | |
| L | approve | External   | | | | | |
| L | transferFrom | External   | | | | | |
||||||
| **Context** | Implementation | |||
| L | _msgSender | Internal  |  | 'msg.sender' | | | |
| L | _msgData | Internal  |  | 'msg.data' | | | |
```

Ownable	Implementation	Context		
L	_msgSender	Internal		'_msg.sender'			
L	_msgData	Internal		'_msg.data'			
L	constructor	Internal				'_msgSender','_setOwner'	
L	owner	Public		'_owner'			
L	renounceOwnership	Public	'onlyOwner'			'onlyOwner','_setOwner'	
L	transferOwnership	Public	'onlyOwner'			'onlyOwner','require(bool,string)'	
L	_setOwner	Private		'_owner'	'_owner'		
IFactory	Interface			
L	createPair	External					
IRouter	Interface			
L	factory	External					
L	WETH	External					
L	addLiquidityETH	External					
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External				
Address	Implementation			
L	sendValue	Internal		'this'		'balance(address)','require(bool,string)'	'recipient.call{value:amount}()'

ApeGrow	Implementation	Ownable, IBEP20, Context		
L	constructor	Public				
L	name	Public		'_name'			
L	symbol	Public		'_symbol'			
L	decimals	Public		'_decimals'			
L	totalSupply	Public		'_tTotal'			
L	balanceOf	Public		'_isExcluded','rOwned'		'tokenFromReflection'		'_tOwned'
L	allowance	Public		'_name'		'_allowances'	
L	approve	Public				'_approve','_msgSender'	
L	transferFrom	Public		'_allowances'		'_approve','_transfer'	
L	increaseAllowance	Public		'_allowances'		'_approve','_msgSender'	
L	decreaseAllowance	Public		'_allowances'		'_approve','require(bool,string)'	
L	transfer	Public		'msg.sender'		'_transfer'	
L	isExcludedFromReward	Public		'_isExcluded'			
L	reflectionFromToken	Public		'_tTotal'		'getValues','require(bool,string)'	
L	EnableTrading	External	'onlyOwner'	'tradingEnabled','block.number'	'genesis_block','swapEnabled'	'onlyOwner','require(bool,string)'	
L	updatedeadline	External	'onlyOwner'	'tradingEnabled'	'deadline'	'onlyOwner','require(bool,string)'	
L	tokenFromReflection	Public		'_rTotal'		'_getRate','require(bool,string)'	
L	excludeFromReward	Public	'onlyOwner'	'_excluded','_isExcluded'	'_excluded','_isExcluded'	'onlyOwner','tokenFromReflection'	'_excluded.push(account)'	'_rOwned'	'_tOwned'	'require(bool,string)'	

L	includeInReward	External	'onlyOwner'	'_excluded','_isExcluded'	'_excluded','_isExcluded'	'onlyOwner','require(bool,string)'	'_excluded.pop()'		'_tOwned'	
L	excludeFromFee	Public	'onlyOwner'		'_isExcludedFromFee'	'onlyOwner'	
L	includeInFee	Public	'onlyOwner'		'_isExcludedFromFee'	'onlyOwner'	
L	isExcludedFromFee	Public		'_isExcludedFromFee'			
L	setTaxes	Public	'onlyOwner'		'taxes'	'onlyOwner','require(bool,string)'	
L	setSellTaxes	Public	'onlyOwner'		'sellTaxes'	'onlyOwner','require(bool,string)'	
L	_reflectRfi	Private		'_rTotal','totFeesPaid'	'_rTotal','totFeesPaid'		
L	_takeLiquidity	Private		'_isExcluded','_rOwned'	'_rOwned','_tOwned'			'_tOwned','totFeesPaid'	'totFeesPaid'		
L	_takeMarketing	Private		'_isExcluded','_rOwned'	'_rOwned','_tOwned'			'_tOwned','totFeesPaid'	'totFeesPaid'		
L	_takeDev	Private		'_isExcluded','_rOwned'	'_rOwned','_tOwned'			'_tOwned','totFeesPaid'	'totFeesPaid'		
L	_getValues	Private				'_getRValues2','_getTValues'						'_getRate'.'_getRValues1'
L	_getTValues	Private		'launchtax','sellTaxes'						'taxes'	
L	_getRValues1	Private					
L	_getRValues2	Private					
L	_getRate	Private			'_getCurrentSupply'	

| L | _getCurrentSupply | Private | | | | '_excluded','_rOwned' | | | | '_rTotal','_tOwned' | |

| L | _approve | Private | | | '_allowances' | 'require(bool,string)' | |

| L | _transfer | Private | | '_isExcludedFromFee', '_lastSell' | '_lastSell' | '_tokenTransfer', 'balanceOf' | | | | | | 'coolDownEnabled','coolDownTime' | | 'swapAndLiquify', 'require(bool,string)' | | | | | | 'maxBuyLimit', 'maxSellLimit' | | | | | | | 'maxWalletLimit', 'pair' | | | | | | | | 'sellTaxes', 'swapEnabled' | | | | | | | | 'swapTokensAtAmount','swapping' | | | | | | | 'taxes', 'tradingEnabled' | | | | | | | | 'block.timestamp', 'this' | | | |

| L | _tokenTransfer | Private | | '_isExcluded', '_isExcludedFromFee' | '_rOwned', '_tOwned' | '_takeMarketing', '_takeLiquidity' | | | | | '_rOwned', '_tOwned' | | '_reflectRfi', '_getValues' | | | | | | 'deadline', 'genesis_block' | | '_takeDev' | | | | | | 'block.number', 'this' | | | |

| L | swapAndLiquify | Private | 'lockTheSwap' | 'devWallet','marketingWallet' | | 'balance(address)','addLiquidity' | 'address(marketingWallet).sendValue(marketingAmt)','address(devWallet).sendValue(devAmt)' | | | | | 'this' | | 'swapTokensForBNB', 'lockTheSwap' | |

| L | addLiquidity | Private | | 'deadWallet', 'router' | |'_approve' | 'router.addLiquidityETH{value:bnbAmount}(address(this),tokenAmount,0,0,deadWallet,block.timestamp)' | | | | | 'block.timestamp', 'this' | | | |

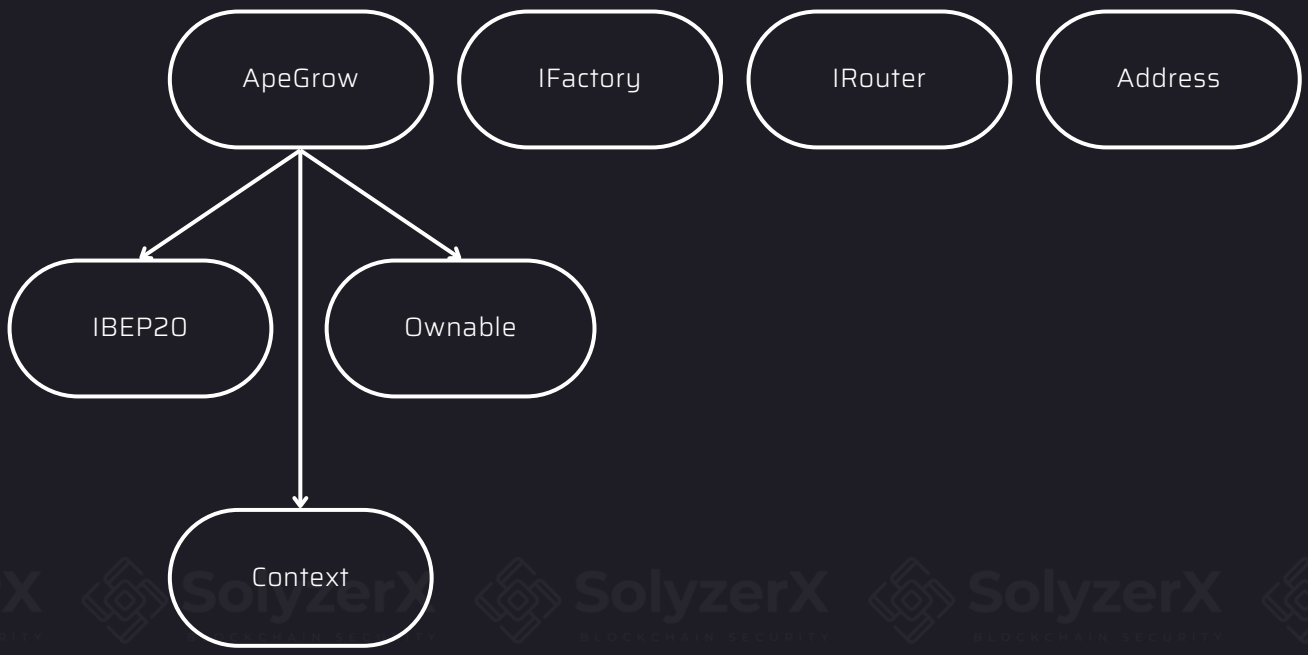
| L | swapTokensForBNB | Private | | 'router', 'block.timestamp' | | '_approve' | 'router.WETH()', 'router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)' | | | | | 'this' | | | 'new address[](2)' |

| L | bulkExcludeFee | External | 'onlyOwner' | | '_isExcludedFromFee' | 'onlyOwner' | |

L	updateMarketingWallet	External	'onlyOwner'		'marketingWallet'	'onlyOwner', 'require(bool,string)'	
L	updateDevWallet	External	'onlyOwner'		'devWallet'	'onlyOwner','require(bool,string)'	
L	bulkExcludeFee	External	'onlyOwner'		'_isExcludedFromFee'	'onlyOwner'	
L	updateMarketingWallet	External	'onlyOwner'		'marketingWallet'	'onlyOwner', 'require(bool,string)'	
L	updateDevWallet	External	'onlyOwner'		'devWallet'	'onlyOwner','require(bool,string)'	
L	updateCooldown	External	'onlyOwner'		'coolDownEnabled','coolDownTime'	'onlyOwner','require(bool,string)'	
L	updateSwapTokensAtAmount	External	'onlyOwner'	'_decimals'	'swapTokensAtAmount'	'onlyOwner','require(bool,string)'	
L	updateSwapEnabled	External	'onlyOwner'		'swapEnabled'	'onlyOwner'	
L	updateMaxTxLimit	External	'onlyOwner'		'maxBuyLimit','maxSellLimit'	'onlyOwner', 'decimals'						'maxWalletLimit'	'require(bool,string)'	
L	rescueBNB	External	'onlyOwner'	'msg.sender','this'		'balance(address)','onlyOwner'	'address(msg.sender).transfer(weiAmount)'						'require(bool,string)'	
L	rescueAnyBEP20Tokens	Public	'onlyOwner'	'this'		'onlyOwner','require(bool,string)'	'IBEP20(_tokenAddr).transfer(_to,_amount)'		receive()	external					

SolyzerX

Inheritance Graph



SolyzerX

Findings Summary

#	Title	Type	Severity
1	Send eth to arbitrary user	Data Validation	High
2	Reentrancy in ApeGrow. _transfer()	Data Validation	High
3	ApeGrow.rescueAnyBEP20Tokens ignores return value	Data Validation	High
4	Performs a multiplication on the result of a division	Undefined Behavior	Medium
5	ApeGrow.addLiquidity ignores return value by router	Data Validation	Medium
6	Local variable shadowing	Undefined Behavior	Low
7	Missing events arithmetic	Arithmetic	Low
8	Missing zero address validation	Auditing and Logging	Low
9	Reentrancy in ApeGrow._transfer - ties-2	Data Validation	Low
10	Reentrancy in ApeGrow._transfer - ties-3	Data Validation	Low
11	Block timestamp	Undefined Behavior	Low
12	Costly operations inside a loop	Auditing and Logging	Informational
13	Dead-code of Context._msgData()	Auditing and Logging	Informational
14	Function Initializing State	Undefined Behavior	Informational

15	Pragma version^0.8.7 allows old versions	Undefined Behavior	Informational
16	Low level call in Address.sendValue(address,uint256)	Auditing and Logging	Informational
17	Conformance to Solidity naming conventions	Auditing and Logging	Informational
18	Redundant expression "this"	Undefined Behavior	Informational
19	State variables that could be declared constant	Undefined Behavior	Optimization

SolyzerX

Detailed Findings

1. Send eth to arbitrary user	
Severity: **High**	Difficulty: **Medium**
Type: Data Validation	Finding ID: ApeGrow.sol# 693-706
Target: ApeGrow.sol	

Description

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

Dangerous calls:
- router.addLiquidityETH{value: bnbAmount}
(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (ApeGrow.sol# 698-705)

Exploit Scenario:

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

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

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

Recommendation

Ensure that an arbitrary user cannot withdraw unauthorized funds.

Carefully review the Solidity documentation, especially the Warnings section.

2. Reentrancy in ApeGrow. _transfer()	
Severity: **High**	Difficulty: **Medium**
Type: Data Validation	Finding ID: ApeGrow.sol#555-615
Target: ApeGrow.sol	

Description

Detection of the reentrancy bug.

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.

3. ApeGrow.rescueAnyBEP20Tokens ignores return value	
Severity: **High**	Difficulty: **Medium**
Type: Data Validation	Finding ID: ApeGrow.sol#773-776
Target: ApeGrow.sol	

Description

The return value of an external transfer/transferFrom call is not checked.

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;
}
```

Several tokens do not revert in case of failure and return false. If one of these tokens is used in MyBank, deposit will not revert if the transfer fails, and an attacker can call deposit for free..

Recommendation

Use SafeERC20, or ensure that the transfer/transferFrom return value is checked.

4. Performs a multiplication on the result of a division	
Severity: **Medium**	Difficulty: **Medium**
Type: Undefined Behavior	Finding ID: ApeGrow.sol#657-691
Target: ApeGrow.sol	

Description

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

Exploit Scenario:

```
contract A {
 function f(uint n) public {
        coins = (oldSupply / n) * interest;
    }
}
```

If n is greater than oldSupply, coins will be zero. For example, with oldSupply = 5; n = 10, interest = 2, coins will be zero.
If (oldSupply * interest / n) was used, coins would have been 1.
In general, it's usually a good idea to re-arrange arithmetic to perform multiplication before division, unless the limit of a smaller type makes this dangerous.

Recommendation

Consider ordering multiplication before division.

5. ApeGrow.addLiquidity ignores return value by router	
Severity: **Medium**	Difficulty: **Medium**
Type: Data Validation	Finding ID: ApeGrow.sol#657-691
Target: ApeGrow.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.

6. Local variable shadowing	
Severity: **Low**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: ApeGrow.sol#250 & #545
Target: ApeGrow.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.

7. Missing events arithmetic	
Severity: Low	Difficulty: Medium
Type: Arithmetic	Finding ID: ApeGrow.sol#320-324
Target: ApeGrow.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
    }
}
```

updateOwner() has no event, so it is difficult to track off-chain changes in the buy price.

Recommendation

Emit an event for critical parameter changes.

8. Missing zero address validation	
Severity: **Low**	Difficulty: **Medium**
Type: Auditing and Logging	Finding ID: ApeGrow.sol#207
Target: ApeGrow.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;
  }
}
```

Bob calls updateOwner without specifying the newOwner, so Bob loses ownership of the contract.

Recommendation

Check that the address is not zero.

9. Reentrancy in ApeGrow._transfer - ties-2	
Severity: **Low**	Difficulty: **Medium**
Type: Data Validation	Finding ID: ApeGrow.sol#555-615
Target: ApeGrow.sol	

Description

Detection of the reentrancy bug.

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.

10. Reentrancy in ApeGrow._transfer - ties-3	
Severity: **Low**	Difficulty: **Medium**
Type: Data Validation	Finding ID: ApeGrow.sol#555-615
Target: ApeGrow.sol	

Description

Detection of the reentrancy bug.

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.

11. Block timestamp	
Severity: **Low**	Difficulty: **Medium**
Type: Undefined Behavior	Finding ID: ApeGrow.sol#555-615
Target: ApeGrow.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.

12. Costly operations inside a loop	
Severity: Informational	Difficulty: Medium
Type: Auditing and Logging	Finding ID: ApeGrow.sol#342-353
Target: ApeGrow.sol	

Description

Costly operations inside a loop might waste gas, so optimizations are justified.

Exploit Scenario:

```
    contract CostlyOperationsInLoop{

    uint loop_count = 100;
    uint state_variable=0;

    function bad() external{
        for (uint i=0; i < loop_count; i++){
            state_variable++;
        }
    }

    function good() external{
      uint local_variable = state_variable;
      for (uint i=0; i < loop_count; i++){
        local_variable++;
      }
      state_variable = local_variable;
    }
}
```

Incrementing state_variable in a loop incurs a lot of gas because of expensive SSTOREs, which might lead to an out-of-gas.

Recommendation

Use a local variable to hold the loop computation result.

13. Dead-code of Context._msgData()	
Severity: **Informational**	Difficulty: **Medium**
Type: Auditing and Logging	Finding ID: ApeGrow.sol#35-38
Target: ApeGrow.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.

14. Function Initializing State	
Severity: Informational	Difficulty: High
Type: Undefined Behavior	Finding ID: ApeGrow.sol#35-38
Target: ApeGrow.sol	

Description

Detects the immediate initialization of state variables through function calls that are not pure/constant, or that use non-constant state variable.

Exploit Scenario:

```
contract StateVarInitFromFunction {

    uint public v = set(); // Initialize from function (sets to 77)
    uint public w = 5;
    uint public x = set(); // Initialize from function (sets to 88)
    address public shouldntBeReported = address(8);

    constructor(){
        // The constructor is run after all state variables are
initialized.
    }

    function set() public  returns(uint)  {
        // If this function is being used to initialize a state variable
declared
        // before w, w will be zero. If it is declared after w, w will be
set.
        if(w == 0) {
            return 77;
        }

        return 88;
    }
}
```

In this case, users might intend a function to return a value a state variable can initialize with, without realizing the context for the contract is not fully initialized. In the example above, the same function sets two different values for state variables because it checks a state variable that is not yet initialized in one case, and is initialized in the other. Special care must be taken when initializing state variables from an immediate function call so as not to incorrectly assume the state is initialized.

Recommendation

Remove any initialization of state variables via non-constant state variables or function calls. If variables must be set upon contract deployment, locate initialization in the constructor instead.

15. Pragma version^0.8.7 allows old versions	
Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: ApeGrow.sol#6
Target: ApeGrow.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.

16. Low level call in Address.sendValue(address,uint256)	
Severity: **Informational**	Difficulty: **High**
Type: Auditing and Logging	Finding ID: ApeGrow.sol#110-115
Target: ApeGrow.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.

17. Conformance to Solidity naming conventions	
Severity: **Informational**	Difficulty: **High**
Type: Auditing and Logging	Finding ID: ApeGrow.sol#82
Target: ApeGrow.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.

18. Redundant expression "this"	
Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: ApeGrow.sol#36
Target: ApeGrow.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.

18. State variables that could be declared constant	
Severity: **Optimization**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: ApeGrow.sol#82
Target: ApeGrow.sol	

Description

Constant state variables should be declared constant to save gas.

Recommendation

Add the constant attributes to state variables that never change.

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

 solyzerx

 solyzerx.com

 t.me/solyzerx

 t.me/solyzerxfoundation

SMART CONTRACT AUDITS | SOLIDITY DEVELOPMENT AND TESTING

RELENTLESSLY SECURING PUBLIC AND PRIVATE BLOCKCHAINS