SAULIDITY

APEMAX

2023 | SMART CONTRACT SECURITY ANALYSIS

PREPARED BY

Saulidity

PRESENTED TO

ApeMax

2023

SAULIDITY

SECURITY REPORT

Smart Contract Audit

saulidity.com

Saulidity

@Saulidity

DISCLAIMER

This report does not constitute financial advice, and Saulidity is not accountable or liable for any negative consequences resulting from this report, nor may Saulidity be held liable in any way. You agree to the terms of this disclaimer by reading any part of the report. If you do not agree to the terms, please stop reading this report immediately and delete and destroy any and all copies of this report that you have downloaded and/or printed. This report was entirely based on information given by the audited party and facts that existed prior to the audit. Saulidity and/or its auditors cannot be held liable for any outcome, including modifications (if any) made to the contract(s) for the audit that was completed. No modifications have been made to the contract(s) by the Saulidity team unless it is indicated explicitly. The audit does not include the project team, website, logic, or tokenomics, but if it does, it will be indicated explicitly. The security is evaluated only on the basis of smart contracts only. There were no security checks performed on any apps or activities. There has not been a review of any product codes. It is assumed by Saulidity that the information and materials given were not tampered with, censored, or misrepresented. Even if this report exists and Saulidity makes every effort to uncover any security flaws, you should not rely completely on it and should conduct your own independent research. Saulidity hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Saulidity, for any amount or kind of loss or damage that may result to you or any other person or any kind of company, community, association and institution. Saulidity is the exclusive owner of this report, and it is published by Saulidity. Without Saulidity's express written authorization, use of this report for any reason other than a security interest in the individual contacts, or use of sections of this report, is forbidden.

Table of Contents

Saulidity

Saulidity is a renowned cybersecurity firm specializing in the analysis and development of Smart contracts. Saulidity, as a full-service security organization, can help with a variety of audits and project development.

In a market where confidence and trust are key, a genuine project may simply increase its user base enormously with an official audit performed by Saulidity.

Introduction

For a thorough understanding of the audit, please read the entire document.

The goal of the audit was to find any potential smart contract security problems and vulnerabilities.
The information in this report should be used to understand the smart contract's risk exposure and as a guide to improving the smart contract's security posture by addressing the concerns that were discovered.

During our audit, we conducted a thorough inquiry using automated analysis and manual review approaches.

The security specialists did a complete study independently of one another in order to uncover any security issues in the contracts as comprehensively as feasible. For optimum security and professionalism, all of our audits are undertaken by at least two independent auditors.

The project's website, logic, or tokenomics have not been vetted by the Saulidity team.

Scope

We analyze smart contracts for both well-known and more specific vulnerabilities.

Here are some of the most well-known vulnerabilities that are taken into account but not limited to:

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Style guide violation
- Transfer forwards all gas
- API violation
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Appendix

Vulnerabilities can be divided into four threat levels: Critical, High, Medium and Low. The classification is mainly based on the impact, likelihood of utilization and other factors.

Critical flaws can result in the loss of assets or the alteration of data and are often simple to exploit.

High-level vulnerabilities are challenging to exploit, but they can have a big influence on how smart contracts are executed, such as giving the public access to key features.

Although medium-level vulnerabilities should be fixed, they generally cannot result in the loss of assets or the manipulation of data.

Low-level flaws are typically caused by code fragments that are out-of-date, useless, etc. and cannot significantly affect execution.

SC Weakness Registry

ITEM	DESCRIPTION
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.
Unchecked Call Return Value	The return value of a message call should be checked.
Access Control & Authorization	Ownership takeover should not bepossible. All crucial functions should be protected. Users could not affect data that belongs to other users.
Selfdestruct	The contract should not be destroyed until it has funds belonging to users.
Check-Effect-Interaction	CEI pattern should be followed if the code performs any external call.

SC Weakness Registry

ITEM	DESCRIPTION
Uninitialized Storage Pointer	Storage type should be set explicitly if the compiler version is < 0.5.0.
Assert Violation	Properly functioning code should never reach a failing assert statement.
Deprecated Solidity Functions	Deprecated built-in functions should never be used.
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.
Denial of Service	Execution of the code should never be blocked by a specific contract state unless it is required.
Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.
Authorization through tx.origin	tx.origin should not be used for authorization.
Block values as a proxy for time	Block numbers should not be used for time calculations.

SC Weakness Registry

ITEM	DESCRIPTION
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.
Shadowing State Variable	State variables should not be shadowed.
Weak Sources of Randomness	Random values should never be generated from Chain Attributes.
Incorrect Inheritance Order	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.
Calls Only to Trusted Addresses	All external calls should be performed only to trusted addresses.
Presence of unused variables	The code should not contain unused variables if this is not justified by design.

Audit & Project Information

Project Name	ApeMax
Contract Name	ApeMax_Production.sol BSC 0x0074E998e03D582108c8168e2a84A46aaaB42222
Report ID	apSAUL001 1.2
Website	Apemax.io
Contact	ApeMax Team
Contact Information	@ArchieHODL Telegram
Code language	Solidity

Summary Table

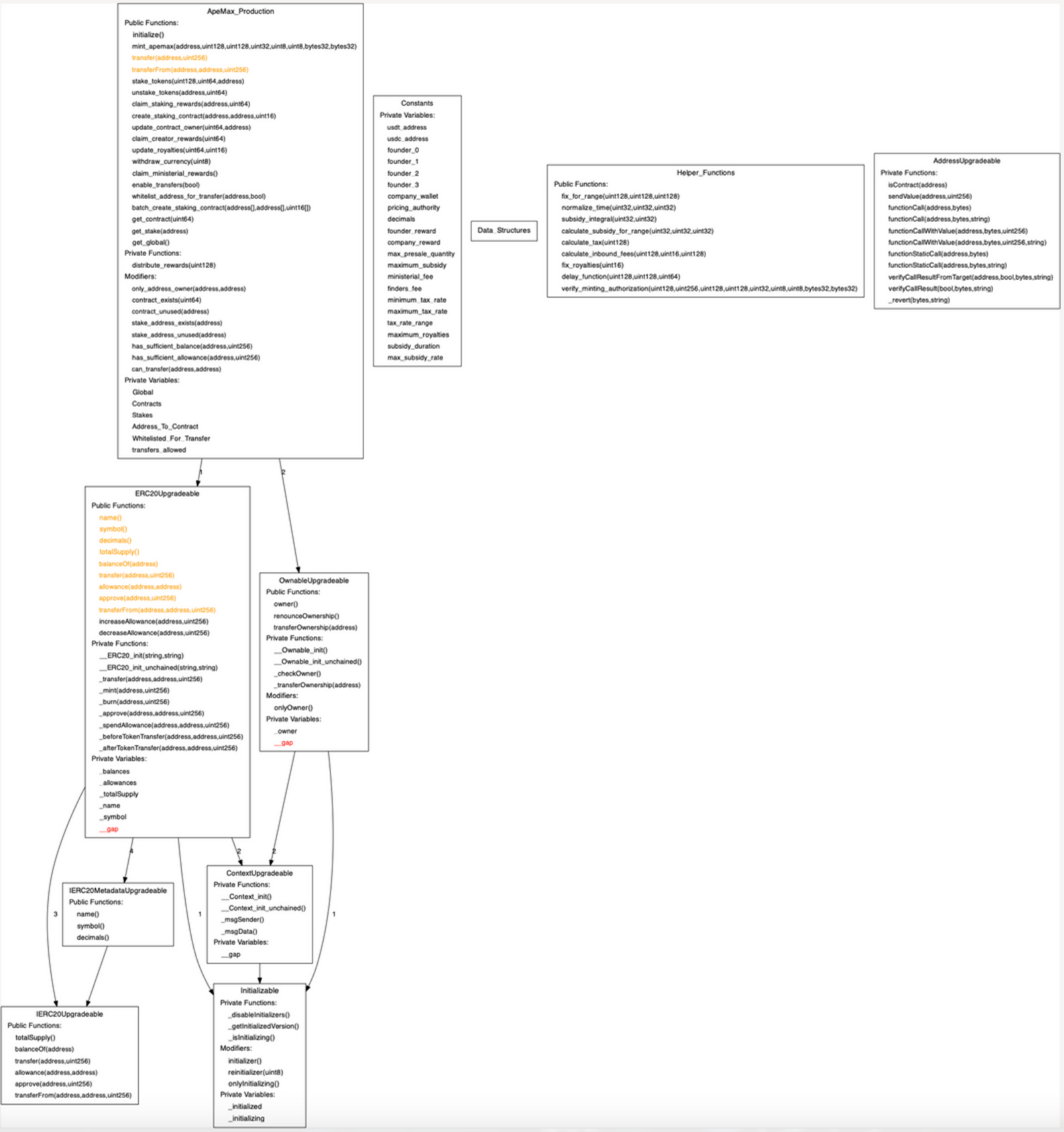
SEVERITY	FOUND
Critical	0
High	0
Medium	1
Low	2
Lowest / Code Style / Optimized Practice	0

Executive Summary

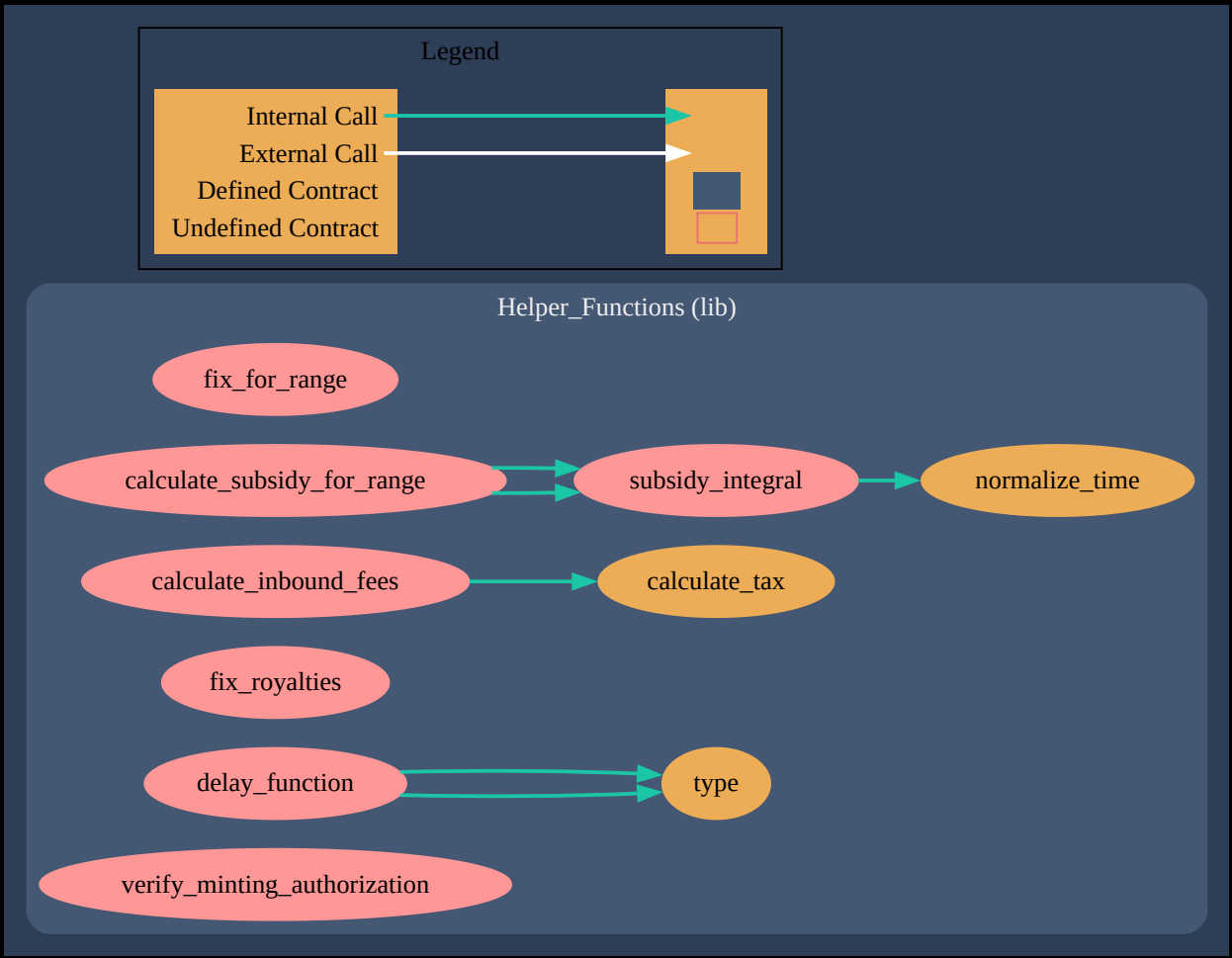
ALL ISSUES FOUND DURING ANALYSIS WERE REVIEWED, AND FALSE POSITIVES WERE ELIMINATED. THE FINDINGS ARE PRESENTED IN THE ANALYSIS SECTION OF THE REPORT.

IT SHOULD BE NOTED THAT ALL FINDINGS HAVE BEEN ACKNOWLEDGED AND/OR MITIGATED BY THE CLIENT.

Inheritance



Call Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Helper_Functions (lib)
fix_for_range
calculate_subsidy_for_range
subsidy_integral
normalize_time
calculate_inbound_fees
calculate_tax
fix_royalties
delay_function
type
verify_minting_authorization

Analysis

Issue: Upgradeable Contact

Severity: Note

Location: General

Description: Any upgradable smart contract carries a certain degree of risk as the process of upgrading the contract can introduce new vulnerabilities or flaws that were not present in the original code.

Comment: Remember to protect initialize functions.

The Transparent Proxy Pattern is a method that enables upgrades to be made directly within the proxy contract itself. This is achieved by assigning an administrator role that has the power to communicate with the proxy contract and change the address of the referenced logic implementation. If a user without admin privileges tries to make a request, it will be routed to the implementation contract instead.

It's crucial to remember that the proxy admin should not have any important role in the logic implementation contract, as they cannot communicate directly with it. This ensures that the security and integrity of the implementation contract is not compromised by the actions of the proxy admin.

Analysis

Issue: Contract owner privileges

Severity: Note

Location: General

Description: The owner has control over these functions:

- claim_creator_rewards
- update_royalties
- withdraw_currency
- claim_ministerial_rewards
- batch_create_staking_contract
- create_staking_contract

Comment: Access to certain functions can be dangerous depending on the specific use case, as it could potentially centralize control and introduce security risks. To mitigate these risks, it is possible to renounce the ownership of the contract or to ensure that no single party can unilaterally control these functions with proper access control mechanisms, time-lock features, multi-signature requirements, etc.

Analysis

Issue: PRNG

Severity: **Medium**

Location: L517-605, L554, L571

Description: Weak PRNG due to a modulo on block.timestamp, block.number, now or blockhash. These can be influenced by miners to some extent so they should be avoided.

```
if (found_index == false) {
    contract_index = uint64(Global.random_seed % uint256(Global.contract_count));
}
```

Comment:To improve the randomness and security of your smart contract's random number generation, it is recommended to consider alternative methods of generating random numbers. You can use a verifiable source of randomness, such as Chainlink VRF, for the purpose of random number generation.

Status: Acknowledged - See Whitepaper

Analysis

Issue: Tautology or Contradiction

Severity: Low

Location: L463-480, L466

Description: Unnecessary comparison or superfluous check.

```
function withdraw_currency(uint8 currency_index) public onlyOwner {

    if (currency_index == 0) {
        require(address(this).balance >= 0, "Insufficient balance");
        payable(owner()).transfer(address(this).balance);
    }
    else if (currency_index == 1) {
        IERC20Upgradeable usdt = IERC20Upgradeable(Constants.usdt_address);
        uint256 usdt_balance = usdt.balanceOf(address(this));
        require(usdt.transfer(owner(), usdt_balance), "USDT token transfer failed");
    }
    else if (currency_index == 2) {
        IERC20Upgradeable usdc = IERC20Upgradeable(Constants.usdc_address);
        uint256 usdc_balance = usdc.balanceOf(address(this));
        require(usdc.transfer(owner(), usdc_balance), "USDC token transfer failed");
    }

}
```

Comment:It is possible to remove tautological checks without affecting functionality of a smart contract.

Status: Mitigated

Analysis

Issue: Division before multiplication

Severity: Low

Location:L304-373, L343-L346, L355-L358

Description: Integer division may result in a truncation. As a result, executing a multiplication before division can help to minimize accuracy loss in some cases.

```
uint256 rewards =
    Stake.amount_staked *
    relevant_units /
    Constants.decimals;
```

```
uint256 nerfed_rewards =
    rewards *
    (time_elapsed - Stake.delay_nerf) /
    time_elapsed;

Global.unclaimed_ministerial_rewards += uint128(rewards - nerfed_rewards);
```

Comment: We recommend to rearrange operations to ensure that multiplication is performed before division. This can help in minimizing potential rounding errors or loss of precision, especially when working with integers.

Status: Acknowledged

Testing Standards

The goal of the audit was to find any potential smart contract security problems and vulnerabilities.
The information in this report should be used to understand the smart contract's risk exposure and as a guide to improving the smart contract's security posture by addressing the concerns that were discovered.

The blockchain platform is used to deploy and execute smart contracts. The platform, its programming language, and other smart contract-related applications all have vulnerabilities that may be exploited. As a result, the audit cannot ensure the audited smart contract(s) explicit security.Audits can't make statements or warranties on security of the code.It also cannot be deemed an adequate assessment of the code's utility and safety, bug-free status, or any statements of the smart contract.While we did our best in completing the study and publishing this report, it is crucial to emphasize that you should not rely only on it; we advocate all projects doing many independent audits and participating in a public bug bounty program to assure smart contract security.

Testing Standards

1. Gather all relevant data.

2. Perform a preliminary visual examination of all documents and contracts.

3. Find security holes with specialist tools & manual review with independent experts.

4. Create and distribute a report.

SAULIDITY

Smart Contract Audit

saulidity.com
Saulidity
@Saulidity