AegisX

Smart Contract Audits | KYC

PALLADIUM

Security Assessment

DayOfDefeat Treasury

November 7, 2022

Table of Contents

AegisX

Assessment Summary

This report has been prepared for DayOfDefeat Treasury on the Binance Smart Chain network. AegisX provides both client-centered and user-centered examination of the smart contracts and their current status when applicable. This report represents the security assessment made to find issues and vulnerabilities on the source code along with the current liquidity and token holder statistics of the protocol.

A comprehensive examination has been performed, utilizing Cross Referencing, Static Analysis, In-House Security Tools, and line-by-line Manual Review.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Inspecting liquidity and holders statistics to inform the current status to both users and client when applicable.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Verifying contract functions that allow trusted and/or untrusted actors to mint, lock, pause, and transfer assets.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders
- Thorough line-by-line manual review of the entire codebase by industry experts.

Technical Findings Summary

Classification of Risk

Severity	Description
Critical	Risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
Major	Risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
Medium	Risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform
Minor	Risks can be any of the above but on a smaller scale. They generally do not compromise the overall integrity of the Project, but they may be less efficient than other solutions.
Informational	Errors are often recommended to improve the code's style or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

Findings

Severity	Found	Pending	Resolved
Critical	1	1	0
Major	2	2	0
Medium	0	0	0
Minor	3	3	0
Informational	1	1	0
Total	7	7	0

Project Overview

Contract Summary

Parameter	Result
Address	
Name	DayOfDefeat
Token Tracker	DayOfDefeat (DOD)
Decimals	N/A
Supply	N/A
Platform	Binance Smart Chain
compiler	v0.8.0^
Contract Name	Treasury
Optimization	N/A
LicenseType	MIT
Language	Solidity
Codebase	N/A
Payment Tx	

Main Contract Assessed Contract Name

Name	Contract	Live
DayOfDefeat		No

TestNet Contract was Not Assessed

Solidity Code Provided

SolID	File Sha-1	FileName
Treasury	2740e789c5fe42aad15b4994dee2f969b496ec7e	Treasury.sol

Call Graph

The contract for DayOfDefeat has the following call graph structure.

KYC Information

The Project Owners of DayOfDefeat are not KYC'd. .

The owner wallet has the power to call the functions displayed on the priviliged functions chart below, if the owner wallet is compromised this privileges could be exploited.

We recommend the team to renounce ownership at the right timing if possible, or gradually migrate to a timelock with governing functionalities in respect of transparency and safety considerations.

KYC Information Notes:

Auditor Notes: N/A

Project Owner Notes:

Smart Contract Vulnerability Checks

ID	Severity	Name	File	location
SWC-100	Pass	Function Default Visibility	Treasury.sol	L: 0 C: 0
SWC-101	Pass	Integer Overflow and Underflow.	Treasury.sol	L: 0 C: 0
SWC-102	Pass	Outdated Compiler Version file.	Treasury.sol	L: 0 C: 0
SWC-103	Pass	A floating pragma is set.	Treasury.sol	L: 0 C: 0
SWC-104	Pass	Unchecked Call Return Value.	Treasury.sol	L: 0 C: 0
SWC-105	Pass	Unprotected Ether Withdrawal.	Treasury.sol	L: 0 C: 0
SWC-106	Pass	Unprotected SELFDESTRUCT Instruction	Treasury.sol	L: 0 C: 0
SWC-107	Low	Read of persistent state following external call.	Treasury.sol	L: 85 C: 16, L: 116 C: 16
SWC-108	Pass	State variable visibility is not set..	Treasury.sol	L: 0 C: 0
SWC-109	Pass	Uninitialized Storage Pointer.	Treasury.sol	L: 0 C: 0
SWC-110	Pass	Assert Violation.	Treasury.sol	L: 0 C: 0
SWC-111	Pass	Use of Deprecated Solidity Functions.	Treasury.sol	L: 0 C: 0
SWC-112	Pass	Delegate Call to Untrusted Callee.	Treasury.sol	L: 0 C: 0

ID	Severity	Name	File	location
SWC-113	Low	Multiple calls are executed in the same transaction.	Treasury.sol	L: 116 C: 16
SWC-114	Pass	Transaction Order Dependence.	Treasury.sol	L: 0 C: 0
SWC-115	Pass	Authorization through tx.origin.	Treasury.sol	L: 0 C: 0
SWC-116	Pass	A control flow decision is made based on The block.timestamp environment variable.	Treasury.sol	L: 0 C: 0
SWC-117	Pass	Signature Malleability.	Treasury.sol	L: 0 C: 0
SWC-118	Pass	Incorrect Constructor Name.	Treasury.sol	L: 0 C: 0
SWC-119	Pass	Shadowing State Variables.	Treasury.sol	L: 0 C: 0
SWC-120	Pass	Potential use of block.number as source of randonmness.	Treasury.sol	L: 0 C: 0
SWC-121	Pass	Missing Protection against Signature Replay Attacks.	Treasury.sol	L: 0 C: 0
SWC-122	Pass	Lack of Proper Signature Verification.	Treasury.sol	L: 0 C: 0
SWC-123	Low	Requirement Violation.	Treasury.sol	L: 85 C: 16
SWC-124	Pass	Write to Arbitrary Storage Location.	Treasury.sol	L: 0 C: 0
SWC-125	Pass	Incorrect Inheritance Order.	Treasury.sol	L: 0 C: 0
SWC-126	Pass	Insufficient Gas Griefing.	Treasury.sol	L: 0 C: 0

ID	Severity	Name	File	location
SWC-127	Pass	Arbitrary Jump with Function Type Variable.	Treasury.sol	L: 0 C: 0
SWC-128	Pass	DoS With Block Gas Limit.	Treasury.sol	L: 0 C: 0
SWC-129	Pass	Typographical Error.	Treasury.sol	L: 0 C: 0
SWC-130	Pass	Right-To-Left-Override control character (U+202E).	Treasury.sol	L: 0 C: 0
SWC-131	Pass	Presence of unused variables.	Treasury.sol	L: 0 C: 0
SWC-132	Pass	Unexpected Ether balance.	Treasury.sol	L: 0 C: 0
SWC-133	Pass	Hash Collisions with Multiple Variable Length Arguments.	Treasury.sol	L: 0 C: 0
SWC-134	Pass	Message call with hardcoded gas amount.	Treasury.sol	L: 0 C: 0
SWC-135	Pass	Code With No Effects (Irrelevant/Dead Code).	Treasury.sol	L: 0 C: 0
SWC-136	Pass	Unencrypted Private Data On-Chain.	Treasury.sol	L: 0 C: 0

We scan the contract for additional security issues using MYTHX and industry-standard security scanning tools.

Smart Contract Vulnerability Details

SWC-107 - Reentrancy.

CWE-841: Improper Enforcement of Behavioral Workflow.

Description:

One of the major dangers of calling external contracts is that they can take over the control flow. In the reentrancy attack (a.k.a. recursive call attack), a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways.

Remediation:

The best practices to avoid Reentrancy weaknesses are: Make sure all internal state changes are performed before the call is executed. This is known as the Checks-Effects-Interactions pattern Use a reentrancy lock.

References:

Ethereum Smart Contract Best Practices - Reentrancy

Smart Contract Vulnerability Details

SWC-113 - DoS with Failed Call

CWE-703: Improper Check or Handling of Exceptional Conditions

Description:

External calls can fail accidentally or deliberately, which can cause a DoS condition in the contract. To minimize the damage caused by such failures, it is better to isolate each external call into its own transaction that can be initiated by the recipient of the call. This is especially relevant for payments, where it is better to let users withdraw funds rather than push funds to them automatically (this also reduces the chance of problems with the gas limit).

Remediation:

It is recommended to follow call best practices:Avoid combining multiple calls in a single transaction, especially when calls are executed as part of a loop. Always assume that external calls can fail. Implement the contract logic to handle failed calls

References:

ConsenSys Smart Contract Best Practices

Smart Contract Vulnerability Details

SWC-123 - Requirement Violation

CWE-573: Improper Following of Specification by Caller

Description:

The Solidity require() construct is meant to validate external inputs of a function. In most cases, such external inputs are provided by callers, but they may also be returned by callees. In the former case, we refer to them as precondition violations. Violations of a requirement can indicate one of two possible issues:

A bug exists in the contract that provided the external input.
The condition used to express the requirement is too strong.

Remediation:

If the required logical condition is too strong, it should be weakened to allow all valid external inputs.Otherwise, the bug must be in the contract that provided the external input and one should consider fixing its code by making sure no invalid inputs are provided.

References:

The use of revert(), assert(), and require() in Solidity, and the new REVERT opcode in the EVM

Inheritance

The contract for DayOfDefeat has the following inheritance structure.

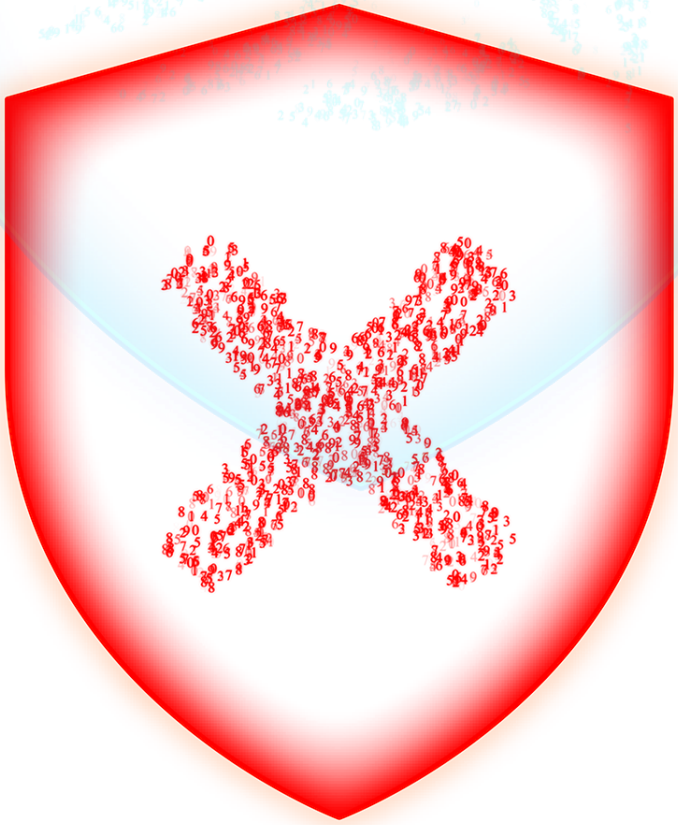
Privileged Functions (onlyOwner)

Function Name	Parameters	Visibility
setAccess		public
setToken		public
setBusd		public
setRouter		public
recoverToken		public

Assessment Results

- SafeTransferLib is not a standard library from OpenZeppelin. Please use with caution.
- The contract is not deployable due to missing Ownable.sol contract (it should be specified if it's a standard Ownable contract found on OpenZeppelin).
- Strong recommendation to use a multisig safe contract or something equivalent to carry the role of the owner's wallet.

Audit Failed



DOD-01 | Potential Sandwich Attacks.

Category	Severity	Location	Status
Security	Minor	Treasury.sol: 287,17, 314,13	Pending

Description

A sandwich attack might happen when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction being attacked) a transaction to purchase one of the assets and make profits by back running (after the transaction being attacked) a transaction to sell the asset. The following functions are called without setting restrictions on slippage or minimum output amount, so transactions triggering these functions are vulnerable to sandwich attacks, especially when the input amount is large:

- swapExactTokensForETHSupportingFeeOnTransferTokens()
- addLiquidityETH()

Remediation

We recommend setting reasonable minimum output amounts, instead of 0, based on token prices when calling the aforementioned functions.

Referrences:

What Are Sandwich Attacks in DeFi — and How Can You Avoid Them?.

DOD-02 | Function Visibility Optimization.

Category	Severity	Location	Status
Gas Optimization	Informational	Treasury.sol:	Pending

Description

The following functions are declared as public and are not invoked in any of the contracts contained within the projects scope:

Function Name	Parameters	Visibility
setAccess		public
setToken		public
setBusd		public
setRouter		public
recoverToken		public

The functions that are never called internally within the contract should have external visibility

Remediation

We advise that the function's visibility specifiers are set to external, and the array-based arguments change their data location from memory to calldata, optimizing the gas cost of the function.

References:

external vs public best practices.

DOD-03 | Lack of Input Validation.

Category	Severity	Location	Status
Volatile Code	Minor	Treasury.sol: 261,5, 349,5	Pending

Description

The given input is missing the check for the non-zero address.

Remediation

We advise the client to add the check for the passed-in values to prevent unexpected errors as below:

```
...
 require(receiver != address(0), "Receiver is the zero address");
...
```

DOD-04 | Centralized Risk In addLiquidity.

Category	Severity	Location	Status
Coding Style	Major	Treasury.sol: 277,5, 297,5	Pending

Description

uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this), tokenAmount, 0, 0, owner(), block.timestamp);

The addLiquidity function calls the uniswapV2Router.addLiquidityETH function with the to address specified as owner() for acquiring the generated LP tokens from the DOD-WBNB pool.

As a result, over time the _owner address will accumulate a significant portion of LP tokens.If the _owner is an EOA (Externally Owned Account), mishandling of its private key can have devastating consequences to the project as a whole.

Remediation

We advise the to address of the uniswapV2Router.addLiquidityETH function call to be replaced by the contract itself, i.e. address(this) , and to restrict the management of the LP tokens within the scope of the contract's business logic. This will also protect the LP tokens from being stolen if the _owner account is compromised. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract based accounts with enhanced security practices, f.e. Multisignature wallets.

1. Indicatively, here are some feasible solutions that would also mitigate the potential risk:
2. Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
3. Assignment of privileged roles to multi-signature wallets to prevent single point of failure due to the private key;

Introduction of a DAO / governance / voting module to increase transparency and user involvement

Project Action

The contract adds liquidity to itself, the Treasury contract, and it is under control of its owner, carrying centralization risks. Strongly recommend to utilize at minimum, a multisig safe to reduce the risk.

DOD-05 | Missing Event Emission.

Category	Severity	Location	Status
Volatile Code	Minor	Treasury.sol: 261,5, 349,5	Pending

Description

Detected missing events for critical arithmetic parameters. There are functions that have no event emitted, so it is difficult to track off-chain changes.The linked code does not create an event for the transfer.

Remediation

Emit an event for critical parameter changes. It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

DOD-11 | Ownable.

Category	Severity	Location	Status
Missing Information	Critical	Treasury.sol: 4,1	Pending

Description

It was found that the contract isn't compilable in its current state with missing contract of Ownable.

Remediation

Provide Ownable contract for the complete review or replace with import from OpenZeppelin if it's a standard contract.

Project Action

Pending Customer Response

DOD-12 | Centralization Risks In The onlyOwner Role(s)

Category	Severity	Location	Status
Centralization / Privilege	Major	Treasury.sol: 241, 1	Pending

Description

In the contract Treasury, the role onlyOwner has authority over the functions that lead to centralization risks.

Any compromise to the onlyOwner account(s) may allow the hacker to take advantage of this authority.

Remediation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage.

We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked.

In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Project Action

Pending Customer Response

Social Media Checks

Social Media	URL	Result
Website	https://www.dayofdefeat.app/	Pass
Telegram	https://t.me/DayOfDefeatBSC	Pass
Twitter	https://twitter.com/dayofdefeatBSC	Pass
OtherSocial	https://titanservice.cn/dayofdefeatCN	Pass

We recommend to have 3 or more social media sources including a completed working websites.

Social Media Information Notes:

Auditor Notes: undefined

Project Owner Notes:

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that actagainst the nature of decentralization, such as explicit ownership or specialized access roles incombination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimalEVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on howblock.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functionsbeing invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that mayresult in a vulnerability.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to makethe codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code,such as a constructor assignment imposing different require statements on the input variables than a setterfunction.

Coding Best Practices

ERC 20 Conding Standards are a set of rules that each developer should follow to ensure the code meet a set of creterias and is readable by all the developers.

Disclaimer

AegisX has conducted an independent security assessment to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the reviewed code for the scope of this assessment. This report does not constitute agreement, acceptance, or advocation for the Project, and users relying on this report should not consider this as having any merit for financial advice in any shape, form, or nature. The contracts audited do not account for any economic developments that the Project in question may pursue, and the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude, and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are entirely free of exploits, bugs, vulnerabilities or deprecation of technologies.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence, regardless of the findings presented. Information is provided 'as is, and AegisX is under no covenant to audited completeness, accuracy, or solidity of the contracts. In no event will AegisX or its partners, employees, agents, or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions or actions with regards to the information provided in this audit report.

The assessment services provided by AegisX are subject to dependencies and are under continuing development. You agree that your access 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. Cryptographic tokens are emergent technologies with high levels of technical risk and uncertainty. The assessment reports could include false positives, negatives, and unpredictable results. The services may access, and depend upon, multiple layers of third parties.