AegisX

Smart Contract Audits | KYC

PALLADIUM

Security Assessment

DayOfDefeat Token

November 5, 2022

Assessment Summary

This report has been prepared for DayOfDefeat Token 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.
ℹ 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	2	2	0
🟠 Major	3	3	0
🟡 Medium	0	0	0
🟢 Minor	4	4	0
ℹ Informational	2	2	0
Total	11	11	0

Project Overview

Contract Summary

Parameter	Result
Address	
Name	DayOfDefeat
Token Tracker	DayOfDefeat (DOD)
Decimals	N/A
Supply	100,000,000,000,000
Platform	Binance Smart Chain
compiler	v0.7.4^
Contract Name	DayofdefeatToken
Optimization	N/A
LicenseType	MIT
Language	Solidity
Codebase	N/A
Payment Tx	

Project Overview

Risk Analysis Summary

Parameter	Result
Buy Tax	19%
Sale Tax	19%
Is honeypot?	Clean
Can edit tax?	No
Is anti whale?	No
Is blacklisted?	Yes
Is whitelisted?	Yes
Holders	Clean
Security Score	55/100
Auditor Score	55/100
Confidence Level	Low

The following quick summary it's added to the project overview; however, there are more details about the audit and its results. Please read every detail.

Main Contract Assessed
Contract Name

Name	Contract	Live
DayOfDefeat		No

TestNet Contract Assessed
Contract Name

Name	Contract	Live
DayOfDefeat	0x66d103b2f2b4f9d515da20fa911d459a48a21cb5	No

Solidity Code Provided

SolID	File Sha-1	FileName
DayofdefeatToken	b76f00a3355398009a3f93c3408ce711b7a14334	DayofdefeatToken.sol

Mint Check

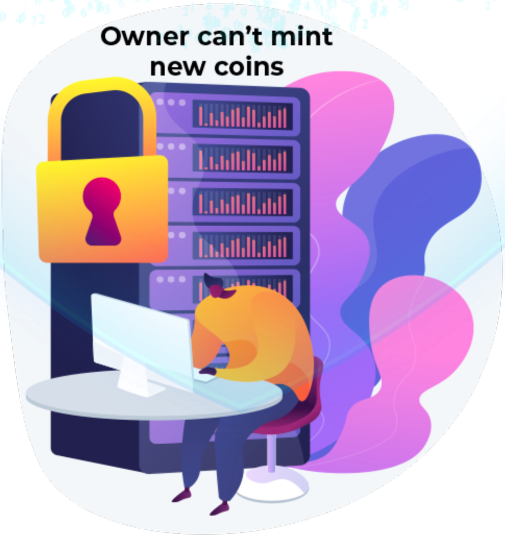
The project owners of DayOfDefeat do not have a mint function in the contract, owner cannot mint tokens after initial deploy.

The Project has a Total Supply of 100,000,000,000,000 and cannot mint any more than the Max Supply.

Mint Notes:

Auditor Notes: No Mint Function.

Project Owner Notes:



Fees Check

The project owners of DayOfDefeat do not have the ability to set fees higher than 25%.

The team May have fees defined; however, they can't set those fees higher than 25% or may not be able to configure the same.

Tax Fee Notes:

Auditor Notes: The contract does charge a fee of 19% and does not have a function to change it.

Project Owner Notes: .

Blacklist Check

The project owners of DayOfDefeat have the ability to Blacklist holders from transferring their tokens.

We recommend the Team be careful with a blacklist function as this can prevent a holder from buying/selling/transferring their assets. Malicious or compromised owners can trap contracts relying on tokens with a blacklist.

Blacklist Notes:

Auditor Notes: The contract does have a ban function.

Project Owner Notes:

MaxTx Check

The Project Owners of DayOfDefeat cannot set max tx amount

The Team allows any investors to swap, transfer or sell their total amount if needed.

MaxTX Notes:

Auditor Notes: N/A

Project Owner Notes:

Pause Trade Check

The Project Owners of DayOfDefeat don't have the ability to stop or pause trading.

The Team has done a great job to avoid stop trading, and investors has the ability to trade at any given time without any problems

Pause Trade Notes:

Auditor Notes: N/A

Project Owner Notes:

Contract Ownership

The contract DayOfDefeat is not live yet.

Liquidity Ownership

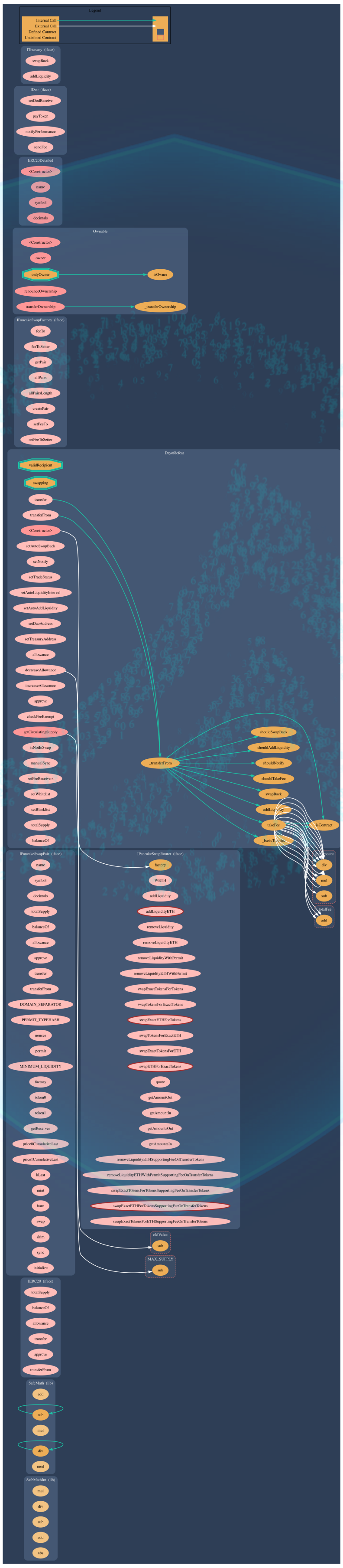
The token does not have liquidity at the moment of the audit, block N/A

If liquidity is unlocked, then the token developers can do what is infamously known as 'rugpull'. Once investors start buying token from the exchange, the liquidity pool will accumulate more and more coins of established value (e.g., ETH or BNB or Tether). This is because investors are basically sending these tokens of value to the exchange, to get the new token. Developers can withdraw this liquidity from the exchange, cash in all the value and run off with it. Liquidity is locked by renouncing the ownership of liquidity pool (LP) tokens for a fixed time period, by sending them to a time-lock smart contract. Without ownership of LP tokens, developers cannot get liquidity pool funds back. This provides confidence to the investors that the token developers will not run away with the liquidity money. It is now a standard practice that all token developers follow, and this is what really differentiates a scam coin from a real one.

Read More

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	DayofdefeatToken.sol	L: 0 C: 0
SWC-101	Pass	Integer Overflow and Underflow.	DayofdefeatToken.sol	L: 0 C: 0
SWC-102	Pass	Outdated Compiler Version file.	DayofdefeatToken.sol	L: 0 C: 0
SWC-103	Low	A floating pragma is set.	DayofdefeatToken.sol	L: 3 C: 0
SWC-104	Pass	Unchecked Call Return Value.	DayofdefeatToken.sol	L: 0 C: 0
SWC-105	Pass	Unprotected Ether Withdrawal.	DayofdefeatToken.sol	L: 0 C: 0
SWC-106	Pass	Unprotected SELFDESTRUCT Instruction	DayofdefeatToken.sol	L: 0 C: 0
SWC-107	Pass	Read of persistent state following external call.	DayofdefeatToken.sol	L: 0 C: 0
SWC-108	Low	State variable visibility is not set..	DayofdefeatToken.sol	L: 566 C: 29, L: 580 C: 12, L: 581 C: 12, L: 591 C: 9
SWC-109	Pass	Uninitialized Storage Pointer.	DayofdefeatToken.sol	L: 0 C: 0
SWC-110	Pass	Assert Violation.	DayofdefeatToken.sol	L: 0 C: 0

ID	Severity	Name	File	locatio n
SWC-111	Pass	Use of Deprecated Solidity Functions.	DayofdefeatToken.s ol	L: 0 C: 0
SWC-112	Pass	Delegate Call to Untrusted Callee.	DayofdefeatToken.s ol	L: 0 C: 0
SWC-113	Pass	Multiple calls are executed in the same transaction.	DayofdefeatToken.s ol	L: 0 C: 0
SWC-114	Pass	Transaction Order Dependence.	DayofdefeatToken.s ol	L: 0 C: 0
SWC-115	Pass	Authorization through tx.origin.	DayofdefeatToken.s ol	L: 0 C: 0
SWC-116	Pass	A control flow decision is made based on The block.timestamp environment variable.	DayofdefeatToken.s ol	L: 0 C: 0
SWC-117	Pass	Signature Malleability.	DayofdefeatToken.s ol	L: 0 C: 0
SWC-118	Pass	Incorrect Constructor Name.	DayofdefeatToken.s ol	L: 0 C: 0
SWC-119	Pass	Shadowing State Variables.	DayofdefeatToken.s ol	L: 0 C: 0
SWC-120	Pass	Potential use of block.number as source of randonmness.	DayofdefeatToken.s ol	L: 0 C: 0
SWC-121	Pass	Missing Protection against Signature Replay Attacks.	DayofdefeatToken.s ol	L: 0 C: 0
SWC-122	Pass	Lack of Proper Signature Verification.	DayofdefeatToken.s ol	L: 0 C: 0
SWC-123	Pass	Requirement Violation.	DayofdefeatToken.s ol	L: 0 C: 0
SWC-124	Pass	Write to Arbitrary Storage Location.	DayofdefeatToken.s ol	L: 0 C: 0

ID	Severity	Name	File	location
SWC-125	Pass	Incorrect Inheritance Order.	DayofdefeatToken.sol	L: 0 C: 0
SWC-126	Pass	Insufficient Gas Griefing.	DayofdefeatToken.sol	L: 0 C: 0
SWC-127	Pass	Arbitrary Jump with Function Type Variable.	DayofdefeatToken.sol	L: 0 C: 0
SWC-128	Pass	DoS With Block Gas Limit.	DayofdefeatToken.sol	L: 0 C: 0
SWC-129	Pass	Typographical Error.	DayofdefeatToken.sol	L: 0 C: 0
SWC-130	Pass	Right-To-Left-Override control character (U+202E).	DayofdefeatToken.sol	L: 0 C: 0
SWC-131	Pass	Presence of unused variables.	DayofdefeatToken.sol	L: 0 C: 0
SWC-132	Pass	Unexpected Ether balance.	DayofdefeatToken.sol	L: 0 C: 0
SWC-133	Pass	Hash Collisions with Multiple Variable Length Arguments.	DayofdefeatToken.sol	L: 0 C: 0
SWC-134	Pass	Message call with hardcoded gas amount.	DayofdefeatToken.sol	L: 0 C: 0
SWC-135	Pass	Code With No Effects (Irrelevant/Dead Code).	DayofdefeatToken.sol	L: 0 C: 0
SWC-136	Pass	Unencrypted Private Data On-Chain.	DayofdefeatToken.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-103 - Floating Pragma.

CWE-664: Improper Control of a Resource Through its Lifetime.

References:

Description:

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation:

Lock the pragma version and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package. Otherwise, the developer would need to manually update the pragma in order to compile locally.

References:

Ethereum Smart Contract Best Practices - Lock pragmas to specific compiler version.

Smart Contract Vulnerability Details

SWC-108 - State Variable Default Visibility

CWE-710: Improper Adherence to Coding Standards

Description:

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

Remediation:

Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables.

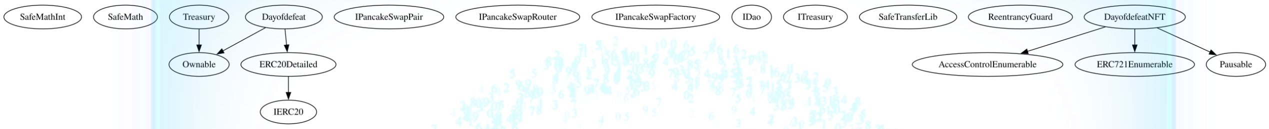
References:

Ethereum Smart Contract Best Practices - Explicitly mark visibility in functions and state variables

Inheritance

The contract for DayOfDefeat has the following inheritance structure.

The Project has a Total Supply of 100,000,000,000,000



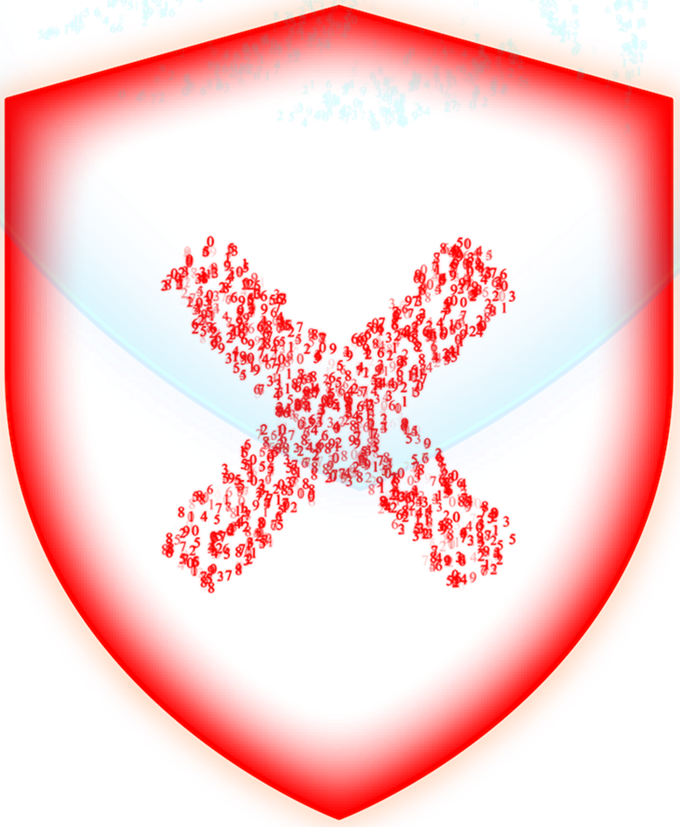
Privileged Functions (onlyOwner)

Function Name	Parameters	Visibility
renounceOwnership	none	Public
transferOwnership	none	Public
setAutoSwapBack	none	External
setNotify	none	External
setTradeStatus	none	External
setAutoLiquidityInterval	none	External
setAutoAddLiquidity	none	External
setDaoAddress	none	External
setTreasuryAddress	none	External
setFeeReceivers	none	External
setWhitelist	none	External
setBlacklist	none	External

Assessment Results

- Use of the most up-to-date compiler version is recommended to avoid known bugs and chances of exploits.
- There is a fee of 19% and cannot be changed.
- The owner can ban a user with the function setBlacklist.
- A complete audit cannot be done as key information behind the custom interface, IDao is missing.
- Division before multiplication will result in a loss of precision in arithmetic calculations, which can lead to a significant loss in assets.

Audit Failed



DOD-01 | Potential Sandwich Attacks.

Category	Severity	Location	Status
Security	● Minor	DayofdefeatToken.sol: 694,13, 698,13	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-03 | Lack of Input Validation.

Category	Severity	Location	Status
Volatile Code	● Minor	DayofdefeatToken.sol: 647,5, 1018,5	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	DayofdefeatToken.sol: 583,5, 831,5	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 a designated autoLiquidityReceiver address associated with ITreasury. And Treasury contract associated with ITreasury is under control of an owner of Treasury, 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	DayofdefeatToken.sol: 647,5, 1018,5	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-07 | State Variables could be Declared Constant.

Category	Severity	Location	Status
Coding Style	● Minor	DayofdefeatToken.sol: 575,5, 581,5	Pending

Description

Constant state variables should be declared constant to save gas.

```
liquidityFee
airdropFee
marketFee
feeDenominator
DEAD & ZERO
```

Remediation

Add the constant attribute to state variables that never changes.

https://docs.soliditylang.org/en/latest/contracts.html#constant-state-variables

DOD-10 | Initial Token Distribution.

Category	Severity	Location	Status
Centralization / Privilege	● Major	DayofdefeatToken.sol: 632,5	Pending

Description

All of the DayOfDefeat tokens are sent to the contract deployer when deploying the contract. This could be a centralization risk as the deployer can distribute tokens without obtaining the consensus of the community.

Remediation

We recommend the team to be transparent regarding the initial token distribution process, and the team shall make enough efforts to restrict the access of the private key.

Project Action

Considering the history of the project and the reason of deployment of V2 contract, it is recommended that the initial token distribution is sent to a verified multisig safe, not to a dev's deployer, to reduce a centralization risk.

DOD-11 | busdAddress.

Category	Severity	Location	Status
Custom Interface	🔴 Critical	DayofdefeatToken.sol: 621,9	Pending

Description

It was found that the contract isn't deployable in its current state with missing information of IDao interface. Line 621 prevents the constructor to work properly.

Remediation

Replace the line with plain BUSD CA, and/or provide further deatils about IDao interface.

Project Action

Pending Customer Response

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

Category	Severity	Location	Status
Centralization / Privilege	● Major	DayofdefeatToken.sol: 479, 9	Pending

Description

In the contract DayofdefeatToken, 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

DOD-13 | Extra Gas Cost For User

Category	Severity	Location	Status
Logical Issue	ⓘ Informational	DayofdefeatToken.sol: 694, 13	Pending

Description

The user may trigger a tax distribution during the transfer process, which will cost a lot of gas and it is unfair to let a single user bear it.

Remediation

We advise the client to make the owner responsible for the gas costs of the tax distribution.

Project Action

The functions addLiquidity and swapBack that calls the contract Treasury's corresponding functions; leads to an unfair situation of a single user bearing the fees incurring from the whole process. It is advised that the owner or an equivalent party bear the responsibility.

DOD-14 | Unnecessary Use Of SafeMath

Category	Severity	Location	Status
Logical Issue	ℹ Informational	DayofdefeatToken.sol: 5,1, 41,1	Pending

Description

The SafeMath library is used unnecessarily. With Solidity compiler versions 0.8.0 or newer, arithmetic operations will automatically revert in case of integer overflow or underflow.
An implementation of SafeMath library is found. SafeMath library is used for uint256 type in DayofdefeatToken contract.

Remediation

We advise removing the usage of SafeMath library and using the built-in arithmetic operations provided by the Solidity programming language

Project Action

The use of most up-to-date compiler version is advised, and eliminate SafeMath. The review revealed that there is no use of safemath specific functions, and only basic arithmetic calculations which can be replaced with (+-*/) when the most up-to-date compiler version is used.

DOD-15 | Divide Before Multiply.

Category	Severity	Location	Status
Mathematical Operations	🔴 Critical	DayofdefeatToken.sol: 707,13, 826,9	Pending

Description

Starting from line 707 to 826, it was found that divisions are being done before multiplication. Performing integer division before multiplication truncates the low bits, losing the precision of calculation.

Remediation

It is strongly advised to apply multiplication before division to avoid loss of precision that can result in a significant loss in assets

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.

AegisX