CONTRACT WOLF

Blockchain Security - Smart Contract Audits

Security Assessment

October 11, 2022

Disclaimer

ContractWolf.io audits and reports should not be considered as a form of project's "advertisement" and does not cover any interaction and assessment from "project's contract" to "external contracts" such as Pancakeswap or similar.

ContractWolf does not provide any warranty on its released reports.

ContractWolf should not be used as a decision to invest into an audited project and is not affiliated nor partners to its audited contract projects.

ContractWolf provides transparent report to all its "clients" and to its "clients participants" and will not claim any guarantee of bug-free code within its **SMART CONTRACT**.

ContractWolf presence is to analyze, audit and assess the client's smart contract's code.

Each company or projects should be liable to its security flaws and functionalities.

Scope of Work

PaperDao Money team agreed and provided us with the files that needs to be tested (Github, Bscscan, Etherscan, files, etc.). The scope of the audit is the main contract.

The goal of this engagement was to identify if there is a possibility of security flaws in the implementation of the contract or system.

ContractWolf will be focusing on contract issues and functionalities along with the projects claims from smart contract to their website, whitepaper and repository which has been provided by **PaperDao Money.**

Risk Level Classification

Risk Level represents the classification or the probability that a certain function or threat that can exploit vulnerability and have an impact within the system or contract.
Risk Level is computed based on CVSS Version 3.0

Level	Value	Vulnerability
Critical	9 - 10	An Exposure that can affect the contract functions in several events that can risk and disrupt the contract
High	7 - 8.9	An Exposure that can affect the outcome when using the contract that can serve as an opening in manipulating the contract in an unwanted manner
Medium	4 - 6.9	An opening that could affect the outcome in executing the contract in a specific situation
Low	0.1 - 3.9	An opening but doesn't have an impact on the functionality of the contract
Informational	0	An opening that consists of information's but will not risk or affect the contract

Auditing Approach

Every line of code along with its functionalities will undergo manual review to check its security issues, quality, and contract scope of inheritance. The manual review will be done by our team that will document any issues that there were discovered.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:

    - Review of the specifications, sources, and instructions provided to ContractWolf to make sure we understand the size, scope, and functionality of the smart contract.
    - Manual review of code, our team will have a process of reading the code line-by-line with the intention of identifying potential vulnerabilities and security flaws.

2. Testing and automated analysis that includes:

    - Testing the smart contract functions with common test cases and scenarios, to ensure that it returns the expected results.

3. Best practices review, the team will review the contract with the aim to improve efficiency, effectiveness, clarifications, maintainability, security, and control within the smart contract.

4. Recommendations to help the project take steps to secure the smart contract.

Used Code from other Frameworks/Smart Contracts (Direct Imports)

Imported Packages

- Comp
- GovernorAlpha
- GovernorBravoDelegate
- GovernorBravoDelegateG1
- GovernorBravoDelegateG2
- GovernorBravoDelegator
- GovernorBravoInterfaces

Description

Network: PoW

Symbol: PPR

Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	10	0	5	0

Capabilities

Version	Solidity Versions Observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	v0.5.16		Yes	Yes	No

Correct implementation of Token Standard

Tested	Verified
✓	✓

Overall Checkup (Smart Contract Security)

Tested	Verified
✓	✓

Function	Description	Exist	Tested	Verified
TotalSupply	Information about the total coin or token supply	✓	✓	✓
BalanceOf	Details on the account balance from a specified address	✓	✓	✓
Transfer	An action that transfers a specified amount of coin or token to a specified address	✓	✓	✓
TransferFrom	An action that transfers a specified amount of coin or token from a specified address	✓	✓	✓
Approve	Provides permission to withdraw specified number of coin or token from a specified address	✓	✓	✓

Verify Claims

Statement	Exist	Tested	Deployer
Renounce Ownership	—	—	—
Mint	—	—	—
Burn	—	—	—
Block	—	—	—
Pause	—	—	—

Legend

Attribute	Symbol
Verified / Can	✓
Verified / Cannot	✗
Unverified / Not checked	⚑
Not Available	—

SWC Attacks

ID	Title	Status
SWC-136	Unencrypted Private Data On-Chain	PASSED
SWC-135	Code With No Effects	PASSED
SWC-134	Message call with hardcoded gas amount	PASSED
SWC-133	Hash Collisions with Multiple Variable Length Arguments	PASSED
SWC-132	Unexpected Ether balance	PASSED
SWC-131	Presence of unused variables	PASSED
SWC-130	Right-To Left Override control character (U+202E)	PASSED
SWC-129	Typographical Error	PASSED
SWC-128	DoS With Block Gas Limit	PASSED
SWC-127	Arbitrary Jump with Function Type Variable	PASSED
SWC-126	Insufficient Gas Griefing	PASSED
SWC-125	Incorrect Inheritance Order	PASSED
SWC-124	Write to Arbitrary Storage Location	PASSED
SWC-123	Requirement Violation	PASSED
SWC-122	Lack of Proper Signature Verification	PASSED
SWC-121	Missing Protection against Signature Replay Attacks	PASSED
SWC-120	Weak Sources of Randomness from Chain Attributes	NOT PASSED
SWC-119	Shadowing State Variables	PASSED
SWC-118	Incorrect Constructor Name	PASSED
SWC-117	Signature Malleability	PASSED
SWC-116	Block values as a proxy for time	PASSED
SWC-115	Authorization through tx.origin	PASSED
SWC-114	Transaction Order Dependence	PASSED
SWC-113	DoS with Failed Call	PASSED
SWC-112	Delegate call to Untrusted Callee	PASSED
SWC-111	Use of Deprecated Solidity Functions	PASSED

SWC-110	Assert Violation	**PASSED**
SWC-109	Uninitialized Storage Pointer	**PASSED**
SWC-108	State Variable Default Visibility	**PASSED**
SWC-107	Reentrancy	**PASSED**
SWC-106	Unprotected SELFDESTRUCT Instruction	**PASSED**
SWC-105	Unprotected Ether Withdrawal	**PASSED**
SWC-104	Unchecked Call Return Value	**PASSED**
SWC-103	Floating Pragma	**NOT PASSED**
SWC-102	Outdated Compiler Version	**PASSED**
SWC-101	Integer Overflow and Underflow	**PASSED**
SWC-100	Function Default Visibility	**PASSED**

Audit Comments (Governance)

Critical Issues

- Approved proposal may be impossible to queue, cancel or execute. The proposed function of the GovernorAlpha contract allows proposers to submit proposals with an unbounded number of actions. Specifically, the function does not impose a hard cap on the number of elements in the arrays passed as parameters (i.e., targets, values, signatures and calldatas).
- Queued proposal with repeated actions cannot be executed. The GovernorAlpha contract allows to propose and queue proposals with repeated actions. That is, two or more actions in a proposal can have the same set of target, value, signature, and data values. Assuming a proposal with repeated actions is approved by the governance system, then each action in the proposal will be queued individually in the Timelock contract via subsequent calls to its queueTransaction function. All queued actions are kept in the queuedTransactions mapping of the Timelock contract for future execution. While each action is identified by the keccak256 hash of its target, value, signature, data, and eta values, it must be noted that all actions in the same proposal share the same eta. As a consequence, repeated actions always produce the same identifier hash. So, a single entry will be created for them in the queuedTransactions mapping. When the time lock expires, the whole set of actions in a proposal can be executed atomically. In other words, the entire proposal must be aborted should one of its actions fail. To execute a proposal anyone can call the execute function of the GovernorAlpha contract. This will in turn call, for each action in the proposal, the executeTransaction function of the

Timelock contract. Considering a proposal with duplicated actions, the first of them will be executed normally and its entry in the queuedTransactions mapping will be set to false. However, the second repeated action will share the same identifier hash as the first action. As a result, its execution will inevitably fail due to the require statement in line 84 of Timelock.sol, thus reverting the execution of the entire proposal.

High Issues

- Cancel any non-executed proposal via cancel function

Medium Issues

- GovernorAlpha contract does not fully match specification, the proposalApproved(uint256): bool function mentioned in the specification is not implemented. According to the specification, a proposal can only succeed when, among other conditions, “For votes are greater than the quorum threshold”. However, in the implementation a proposal is considered successful when votes in favor are equal or greater than the quorum threshold. According to the specification, the GovernorAlpha contract should have a maximum number of operations that a proposal can contain. However, the audited implementation does not impose any limit on the number of actions (see propose function). This may allow proposers to submit proposals that may never be queued, canceled, or executed (as explained in issue [H01] Approved proposal may be impossible to queue, cancel or execute). Consider applying the necessary modifications to the code and / or to the specification so

that they fully match. Should any deviation be intentional, consider explicitly documenting it with docstrings and inline comments.

- Proposal execution not handling returned data, the public execute function of the GovernorAlpha contract allows anyone to execute a queued proposal. Each action contained in the proposal will trigger a call to the executeTransaction function of the Timelock contract. The executeTransaction function returns a bytes value containing whatever data is returned by the call to the target address. It is important to note that the data is never logged in the emitted ExecuteTransaction event, thus it should be handled by the caller to avoid losing it. However, the returned data is not handled by the execute function of the GovernorAlpha contract. As a consequence, relevant data returned by the proposal's actions may be lost. Consider handling the data returned by the subsequent calls to the executeTransaction function. Potential courses of action to be analyzed include logging the data in events or returning it to the execute function's caller in an array of bytes values.

Low Issues

- Lack of indexed parameters in events. None of the parameters in the events defined in the GovernorAlpha contract are indexed. Consider indexing event parameters to avoid hindering the task of off-chain services searching and filtering for specific events.

- Storage modification in require statement. Inside the require statement in line 143 of Comp.sol, the signatory's nonce is incremented right after being compared with the given nonce. In other words, the require statement fails if the given nonce is different from the one stored in the nonces mapping before it is incremented by one. Yet this subtlety of the language might no be caught by all readers, which can lead to confusions and errors in future changes to the code base. To favor readability, consider incrementing the nonce outside the mentioned require statement, right after it has been verified.

- Not declaring return types in functions with return statements. Public functions delegate and delegateBySig of the Comp contract include a return statement at the end of their execution, although they do not explicitly declare return types in their definition. Moreover, both functions attempt to return the result of the internal _delegate function, which does not declare return types nor returns any value. Consider removing the return statements of the delegate and delegateBySig functions, keeping in both cases the internal call to the _delegate function.

- Undocumented, untested, custom behavior in transfer of ERC20 token. When the transfer and transferFrom functions of the Comp token are called, they internally call the _transferTokens function. This internal function can execute additional actions that are not part of the ERC20 standard. In particular, if the source and destination have different delegates registered, the _decreaseVotes

and _increaseVotes functions are executed. This means that upon a transfer of tokens, delegates' votes amounts may be updated. While the described custom behavior is fundamental to Compound's governance system, it was found to be undocumented and untested. Consider explicitly explaining that delegates' votes can be updated in the docstrings of transfer and transferFrom functions. Furthermore, consider adding related unit tests in CompTest.js to ensure this sensitve feature works as expected.

CONTRACTWOLF

Blockchain Security - Smart Contract Audits