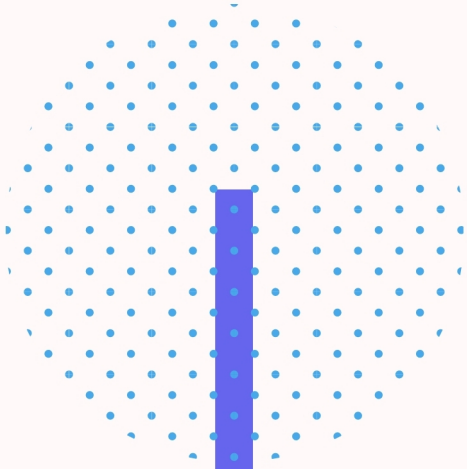
NIGHTWATCH

AUDIT REPORT



Contents

- 1. Introduction
  - 1.1. About Project
  - 1.2. Audit Goal
  - 1.3. Disclaimer
- 2. Findings
  - 2.1. Data Validation Issues - PASS
  - 2.2. Random Number Issues - PASS
  - 2.3. State Issues - PASS
  - 2.4. Error Conditions, Return Values, Status Codes - PASS
  - 2.5. Data Processing Errors - PASS
  - 2.6. Bad Coding Practices - PASS
  - 2.7. Permission Issues - PASS
  - 2.8. Business Logic Errors - PASS
  - 2.9. Complexity Issues - PASS
- 3. Conclusion
- 4. Appendix
  - 4.1. Function Graph
  - 4.2. Inheritance Chart

1. Introduction

1.1. About Project

- Project Name: **Beast DAO**
- The contract is deployed at **0xdbb2f12cb89af05516768c2c69a771d92a25d17c**.
- The total Supply of BEAST is 500,000. There are no accessible mint functions present within the smart contract.
- The current owner can transfer ownership to another address.
- The contract allows token holders to destroy their own tokens and those that they have an allowance for.
- Utilization of SafeMath to prevent overflows and ensure safe transfers and properly follows the ERC20 standard.

1.2. Audit Goal

Category	Content	Result
Data Validation Issues	Incorrect Behavior Order: Early Validation,Permissive List of Allowed Inputs,Unchecked Input for Loop Condition	PASS
Random Number Issues	Small Space of Random Values,Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)	PASS
State Issues	External Control of System or Configuration Setting,Incomplete Internal State Distinction,Passing Mutable Objects to an Untrusted Method	PASS
Error Conditions, Return Values, Status Codes	Unchecked Return Value,Unexpected Status Code or Return Value,Reachable Assertion,Detection of Error Condition Without Action	PASS
Data Processing Errors	Collapse of Data into Unsafe Value,Improper Handling of Parameters, Comparison of Incompatible Types	PASS
Bad Coding Practices	Missing Default Case in Switch Statement,Excessive Index Range Scan for a Data Resource,Excessive Platform Resource Consumption within a Loop	PASS
Permission Issues	Incorrect Default Permissions,Incorrect Execution-Assigned Permissions,Improper Preservation of Permissions	PASS
Business Logic Errors	Unverified Ownership,Incorrect Ownership Assignment,Allocation of Resources Without Limits or Throttling	PASS
Complexity Issues	Loop Condition Value Update within the Loop,Excessively Deep Nesting	PASS

1.3. Disclaimer

Note that this audit does not give any warranties on finding all possible security issues of the given smart contract(s), i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.

2. Findings

2.1. Data Validation Issues - PASS

Weaknesses in this category are related to a software system's components for input validation, output validation, or other kinds of validation. Validation is a frequently-used technique for ensuring that data conforms to expectations before it is further processed as input or output. There are many varieties of validation). Validation is distinct from other techniques that attempt to modify data before processing it, although developers may consider all attempts to product "safe" inputs or outputs as some kind of validation. Regardless, validation is a powerful tool that is often used to minimize malformed data from entering the system, or indirectly avoid code injection or other potentially-malicious patterns when generating output. The weaknesses in this category could lead to a degradation of the quality of data flow in a system if they are not addressed.

Test results: No related vulnerabilities in smart contract code. Safety advice:None.

2.2. Random Number Issues - PASS

Weaknesses in this category are related to a software system's random number generation. The number of possible random values is smaller than needed by the product, making it more susceptible to brute force attacks. The code uses a Pseudo-Random Number Generator (PRNG) that does not correctly manage seeds.

Test results: No related vulnerabilities in smart contract code. Safety advice:None.

2.3. State Issues - PASS

Weaknesses in this category are related to improper management of system state. One or more system settings or configuration elements can be externally controlled by a user. The software does not properly determine which state it is in, causing it to assume it is in state X when in fact it is in state Y, causing it to perform incorrect operations in a security-relevant manner.

Test results: No related vulnerabilities in smart contract code. Safety advice:None.

2.4. Error Conditions, Return Values, Status Codes - PASS

This category includes weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function. This type of problem is most often found in conditions that are rarely encountered during the normal operation of the product. Presumably, most bugs related to common conditions are found and eliminated during development and testing. In some cases, the attacker can directly control or influence the environment to trigger the rare conditions.

Test results: No related vulnerabilities in smart contract code. Safety advice:None.

2.5. Data Processing Errors - PASS

Weaknesses in this category are typically found in functionality that processes data. Data processing is the manipulation of input to retrieve or save information. The software filters data in a way that causes it to be reduced or "collapsed" into an unsafe value that violates an expected security property. The software does not properly handle when

the expected number of values for parameters, fields, or arguments is not provided in input, or if those values are undefined.

Test results: No related vulnerabilities in smart contract code. Safety advice:None.

2.6. Bad Coding Practices - PASS

Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. These weaknesses do not directly introduce a vulnerability, but indicate that the product has not been carefully developed or maintained. If a program is complex, difficult to maintain, not portable, or shows evidence of neglect, then there is a higher likelihood that weaknesses are buried in the code.

Test results: No related vulnerabilities in smart contract code. Safety advice:None.

2.7. Permission Issues - PASS

Weaknesses in this category are related to improper assignment or handling of permissions. While it is executing, the software sets the permissions of an object in a way that violates the intended permissions that have been specified by the user. The software does not preserve permissions or incorrectly preserves permissions when copying, restoring, or sharing objects, which can cause them to have less restrictive permissions than intended.

Test results: No related vulnerabilities in smart contract code. Safety advice:None.

2.8. Business Logic Errors - PASS

Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application. They can be difficult to find automatically, since they typically involve legitimate use of the application's functionality. However, many business logic errors can exhibit patterns that are similar to well-understood implementation and design weaknesses.

Test results: No related vulnerabilities in smart contract code. Safety advice:None.

2.9. Complexity Issues - PASS

Weaknesses in this category are associated with things being overly complex. The code uses a loop with a control flow condition based on a value that is updated within the body of the loop or contains a callable or other code grouping in which the nesting / branching is too deep.

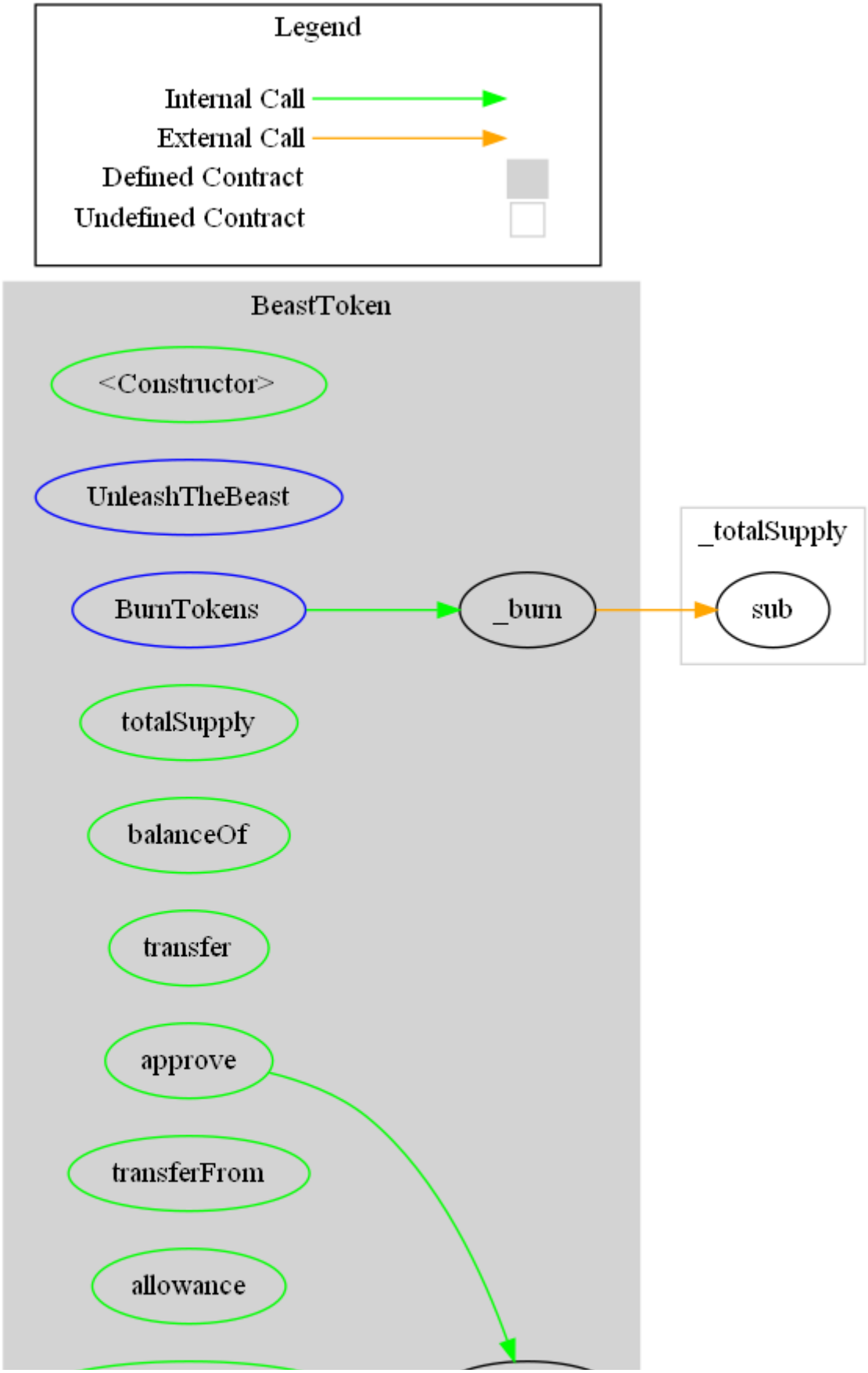
Test results: No related vulnerabilities in smart contract code. Safety advice:None.

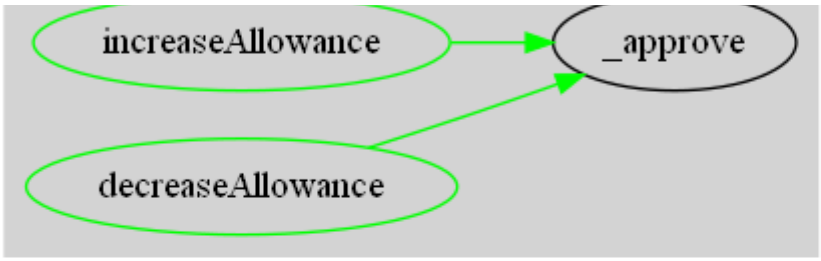
3. Conclusion

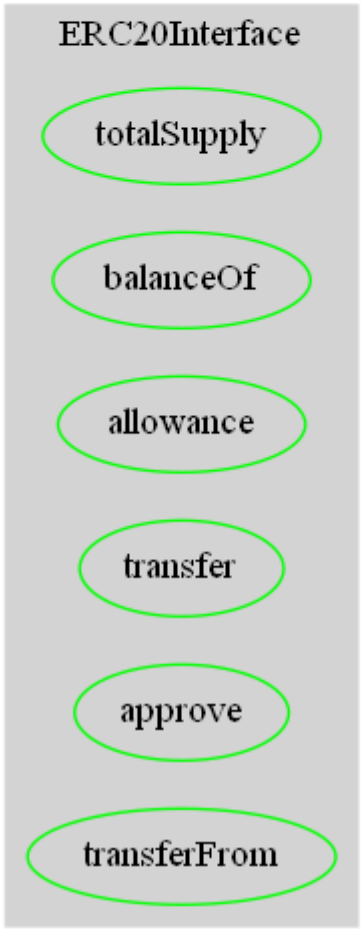
Use case of the smart contract is simple and the code is relatively small.Altogether, the code is written and demonstrates effective use of abstraction, separation of concerns, and modularity. No security issues from external attackers were identified and therefore the contract is good to be deployed on public networks as per the audit team’s analysis.

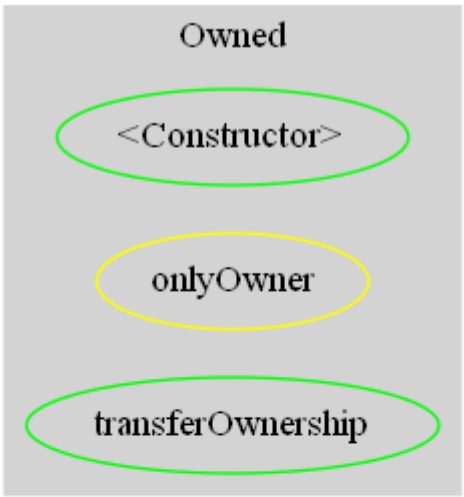
4. Appendix

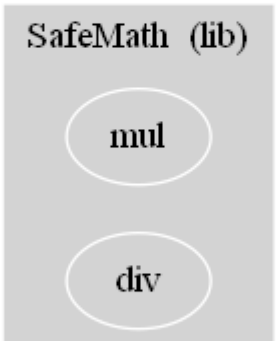
4.1. Function Graph

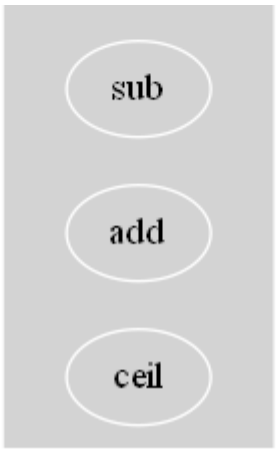



increaseAllowance
_approve
decreaseAllowance


ERC20Interface
totalSupply
balanceOf
allowance
transfer
approve
transferFrom


Owned
<Constructor>
onlyOwner
transferOwnership


SafeMath (lib)
mul
div



4.2. Inheritance Chart