Gay Pepe Contract Audit

Security Assessment

May 26, 2023,

Project Dashboard

Application Summary

Name	Gay Pepe Contract
Version	v1.0
Type	Solidity Smart Contracts
Platforms	Ethereum

Engagement Summary

Dates	May 25 - May 26, 2023
Method	Black box
Consultants Engaged	1
Level of Effort	1 person-weeks

Vulnerability Summary

Total High-Severity Issues	7	■■■■■■■
Total	7	

Category Breakdown

Data Validation	0	
Patching	7	■■■■■■■
Error/Warning Reporting	1	■
Total	8	

Code Maturity

Category Name	Description
Access Controls	**Satisfactory.** The codebase has a strong access control mechanism I found only minor concerns.
Arithmetic	**Moderate.** The system relies on complex arithmetic. While the use of Solidity prevents overflow and underflow flaws, I found several issues related to code.
Centralization	**Moderate.** The contracts’ owners have significant privileges. Additionally, the deployer of ERC20 contracts will own all the tokens at deployment and will have a significant advantage.
Upgradeability	**Not Applicable.**
Function Composition	**Moderate.** Some components are written multiple times, and the codebase would benefit from greater code reuse.
Front-Running	**Satisfactory.** Most functions are not impacted by front-running, or the impact is expected. We found only one minor issue.
Monitoring	**Weak.** I’m not aware of any off-chain components that monitor the contracts.
Specification	**Moderate.** The provided documentation omitted several behaviors, and the codebase would benefit from more thorough documentation.
Testing & Verification	**Moderate.** The codebase has no unit tests. No verification of code was present.

Addresses:-

The contract code is obtained from the Binance Smart Chain Scan Explorer.
Gay Pepe Contract:-
https://bscscan.com/token/0x0158d3817c1391b4736be724b1e8e8553d615c57#code

Engagement Goals

I scoped the engagement to provide a security assessment of the Gay Pepe contract.

Specifically, I sought to answer the following questions:

- Is the contract functionality aligned with the stated objectives and requirements?
- Are the contract's mappings and variables properly defined and initialized?
- Are the total supply, name, symbol, and decimals accurately set for the token?
- Are the balance Of, transfer, transfer-from, and approve functions correctly implemented?
- Are the allowances being properly managed in the transfer From function?
- Is the approve function correctly setting the allowance for the spender?

Coverage

This review included the contracts comprising the Gay Pepe contract.

I reviewed contracts for common Solidity flaws, such as visibility modifiers, missing check statements, missing common functionality, and unprotected functions.

Comprehensive list of known attack vectors:-

These are the most common vulnerabilities in smart contracts.

Basic Coding Bugs

- Reentrancy
- Blackhole
- Revert DoS
- Unchecked External Call
- Costly Loop
- (Unsafe) Use Of Untrusted Libraries
- (Unsafe) Use Of Predictable Variables

Semantic Consistency Checks

- Semantic Consistency Checks

In particular, I perform the audit according to the following procedure:

• **Basic Coding Bugs:** I first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.

• **Semantic Consistency Checks:** I manually check the logic of implemented smart contracts and compare them with the description in the white paper.

• **Additional Recommendations:** I also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

Findings Summary

#	Title	Type	Severity
1	Add the latest version of the solidity	Patching	High
2	Add the Openzeppelin ERC20 Library instead of custom functions.	Patching	High
3	Don’t intitialize the decimal = 18	Patching	Medium
4	Remove the extra “balances” mapping	Patching	High
5	Remove the “allowance” mapping	Patching	High
6	Remove the balanceOf, transfer, TransfeFrom, approve Functions	Warning Reporting	High
7	Initialize the the Address type “Owner” variable	Patching	High
8	Add Burnable Library	Patching	High

Detailed Results

1. Add the latest version of the solidity

Severity: High

Type: Patching

Difficulty: Undetermined

Target: `Token.sol`

Description:
Using an old version of Solidity may make the contract more vulnerable to security attacks. It is recommended to use the latest version of Solidity when writing smart contracts.

Recommendation
You can use the latest version 0.8.19.

2. Add the Openzeppelin ERC20 Library

Severity: High

Type: Patching

Difficulty: Undetermined

Target: `Token.sol`

Description
it would be a good idea to use the ERC20 library instead of custom functions in your contract.

Here are some of the key benefits of using the ERC20 library:

- **Compatibility**: The ERC20 library is a standard library, so it is compatible with other ERC20 tokens and wallets. This means that your token will be able to be used with a wider range of applications.
- **Security**: The ERC20 library has been audited by security experts, so it is more secure than custom functions. This means that your token will be less vulnerable to attacks.

- **Ease of use**: The ERC20 library is well-documented and easy to use. This means that you will be able to develop your token more quickly and easily.

Note: One important Feature in ERC20 library is It has increaseAllowance and DecreaseAllowance functions to enhance the allowance feature strength.

Recommendation:

By Adding ERC20 Library you can remove the Custom Code functions balanceOf, transfer, transferFrom and approve which will save lots of gas.

Library:

```
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
```

3. Don’t Initialize the Decimal = 18

Severity: Medium

Difficulty: Undetermined

Type: Patching

Target: `Token.sol`

Description

It is not recommended to Redeclare the by default property.

Recommendation

When you use ERC20 library, It has by default 18 decimal property, This will save your Gas.

4. Remove the Extra “balances” mapping

Severity: High

Difficulty: Undetermined

Type: Patching

Target: `Token.sol`

Description

The Gay Pepe contract has extra mapping “balances” which is extra, In ERC20 we can achieve the purpose with the “balanceOf” builtin Read function.

Recommendation
Remove the Extra mapping to save the Gas

Library:

```
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
```

5. Remove the Extra “allowance” mapping

Severity: High

Difficulty: Undetermined

Type: Error Reporting

Target: `Token.sol`

Description
The Gay Pepe contract has extra mapping “allowance” which is extra, In ERC20 we can achieve the purpose with the “allowance” builtin Read function.

Recommendation
Remove the Extra mapping to save the Gas

Library:

```
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
```

6. Remove the balanceOf, transfer, transferFrom, approve Functions

Severity: High

Difficulty: Undetermined

Type: Patching

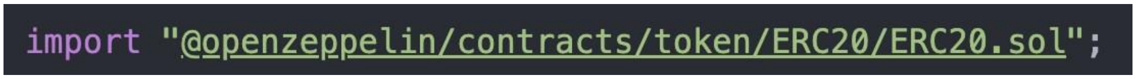
Target: `Token.sol`

Description
The Gay Pepe contract has extra balanceOf, transfer, transferFrom, approve functions. However the balanceOf function is not view so a warning will come. After Removing this

Custom Functions use ERC20 Library which has all the audited functions.

Recommendation
Remove All this Function and use ERC20 Library

Library:



```
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
```

Initialize the the Address type “Owner” state variable in the constructor

7. Initialize the Address type “Owner” Variable

Severity: High

Difficulty: Undetermined

Type: Patching

Target: `Token.sol`

Description
The Gay Pepe contract does not have any owner who owns tokens, therefore the initialization of the address type “Owner” state variable in the constructor is good.

Recommendation
It's Recommended that the contract should have the Owner who Owns the tokens.

8. Add Burnable Libraryy

Severity: High

Difficulty: Undetermined

Type: Patching

Target: `Token.sol`

Description
The Gay Pepe contract does not have a mechanism for burning tokens. This means that once tokens are created, they cannot be destroyed. This could lead to a situation where the supply of tokens becomes too large, which could reduce the value of each token.

Recommendation

It's Recommended that the contract should have the Burning library so that the owner can burn the Tokens

A. Vulnerability

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices or software
Cryptography	Related to protecting the privacy or integrity of data
Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Timing	Related to race conditions, locking or order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Undetermined	The extent of the risk was not determined during this engagement

Low	The risk is relatively small or is not a risk the customer has indicated is important
Medium	Individual user's information is at risk, exploitation would be bad for client's reputation, moderate financial impact, possible legal implications for client
High	Large numbers of users, very bad for client's reputation, or serious legal or financial implications

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploit was not determined during this engagement
Low	Commonly exploited, public tools exist or can be scripted that exploit this flaw
Medium	Attackers must write an exploit, or need an in-depth knowledge of a complex system
High	The attacker must have privileged insider access to the system, may need to know extremely complex technical details or must discover other weaknesses to exploit this issue

B. Code Quality Recommendations

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

Number of Issues by severty

Critical	High	Medium	Low	Note
0	0	0	0	0

Issues Checking Status

No	Issue Description	Checking status
1	Compiler warnings.	Passed
2	Race conditions and Reentrancy.Cross-function race conditions	Passed
3	Possible delays in data delivery.	Passed
4	Oracle calls.	Passed
5	Front running	Passed
6	Timestamp dependence	Passed
7	Integer Overflow and Underflow	Passed
8	Dos with Revert.	Passed
9	Dos with block gas limit.	Passed
10	Methods execution permissions.	Passed
11	Economy model.	Passed
12	The impact of the exchange rate on the logic	Passed
13	Private user data leaks.	Passed
14	Malicious event log	Passed
15	Scoping and declarations.	Passed
16	Uninitialized storage pointers.	Passed
17	Arithmetic accuracy	Passed
18	Design logic.	Passed
19	Cross-function race conditions	Passed
20	Safe Zeppelin	Passed
21	Fallback function security.	Passed

Manual Audit :

For this section the code was tested/read line by line by our developers,we also use Remix IDE's javascript VM and Kovan network to test the contract functionality

Critical severity issues

No critical severity issues found…..

High severity issues

No high severity issues found……

Medium severity issues

No Medium severity issues found..

Low severity isseus

No low severity issues found..

C. Code Maturity Classifications

Code Maturity Classes	
Category Name	Description

Access Controls	Related to the authentication and authorization of components.
Assembly Use	Related to the use of the inline assembly.
Centralization	Related to the existence of a single point of failure.
Upgrade ability	Related to contract upgrade ability.
Function Composition	Related to the separation of the logic into functions with a clear purpose.
Front-Running	Related to resilience against front-running.
Key Management	Related to the existence of proper procedures for key generation, distribution, and access.
Specification	Related to the expected codebase documentation.
Testing & Verification	Related to the use of testing techniques (unit tests, fuzzing, symbolic execution, etc.).

Rating Criteria	
Rating	**Description**
Strong	The component was reviewed and no concerns were found.
Satisfactory	The component had only minor issues.
Moderate	The component had some issues.
Weak	The component led to multiple issues; more issues might be present.
Missing	The component was missing.

Not Applicable	The component is not applicable.
Not Considered	The component was not reviewed.
Further Investigation Required	The component requires further investigation.

E. Fix Log

#	Title		Severity
1	Add the latest version of the solidity		High
2	Add the Openzeppelin ERC20 Library instead of custom functions.		High
3	Don’t intitialize the decimal = 18		Low
4	Remove the extra “balances” mapping		High
5	Remove the “allowance” mapping		High
6	Remove the balanceOf, transfer, TransfeFrom, approve Functions		High
7	Initialize the the Address type “Owner” variable		High
8	Add Burnable Library		

Summary

Smart Contract do not do not contain any severity issues.

Note
Please check the disclaimer above and note, the audit makes no statement or warranties models, investment attractiveness or code sustainability. The reports provide for only contract mention in the report