QuillAudits

Audit Report
July, 2021

Favecoin

Contents

Scope of Audit

The scope of this audit was to analyze and document the Favecoin Token smart contract codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.
SmartCheck.

Static Analysis
Static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step a series of automated tools are used to test security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of automated analysis were manually verified.

Gas Consumption
In this step we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

Every issue in this report has been assigned with a severity level. There are four levels of severity and each of them has been explained below.

High severity issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality and we recommend these issues to be fixed before moving to a live environment.

Medium level severity issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems and they should still be fixed.

Low level severity issues

Low level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity four issues which indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Closed	3	1	2	5

Introduction

During the period of **June 26, 2021 to July 03, 2021** - QuillAudits Team performed a security audit for Favecoin smart contracts.

The code for the audit was taken from following the official link: https://github.com/balajipachai/favecoin-sc/blob/master/contracts/FAVE.sol

Note	Date	Commit hash
Version 1	June	e590b6c5fedb69d22d4edf502ffdb65b543f7638
Version 2	July	841e8bce76b063c880df516025cd0414049e5685
Version 3	July	9f5249567af546e0f7bafa8eb09dcc1111f48517

Issues Found – Code Review/Manual Testing

High severity issues

1. Anyone can mint tokens

Line	Code

131-137	function mint(address account, uint256 amount) public whenNotPaused {
	// Calculate fee and transfer the amount - fee
	uint256 fee = calculateFee(amount);
	amount -= fee;
	super.transfer(project, fee);
	_mint(account, amount);
	}

Description:
There is no access control for the mint() function. Any user can call this function with any amount and mint those tokens to himself. This allows users to get unlimited tokens.

Remediation:
We recommend using the onlyOwner modifier for the mint() function.

Status: Closed

The mint() function was removed from the code in version 2.

2. Anyone can mint tokens

Line	Code

114-120	function burn(address account, uint256 amount) public whenNotPaused {
	// Calculate fee and transfer the amount - fee
	uint256 fee = calculateFee(amount);
	amount -= fee;
	super.transfer(project, fee);
	_burn(account, amount);
	}

Description:
User A can call the burn() function, passing the address of User B, thus burning the tokens of User B. This can be used to drain anybody's wallets.

Remediation:
We recommend, instead of accepting 'address account' as a parameter and burning tokens of that address, use msg.sender in its place. This will only burn the tokens of the msg.sender.
Or
If you want it to be called by only the owner, then use the onlyOwner modifier on this function.

Status: Closed

The code was updated in version 2, and now msg.sender's tokens are burned.

3. transferFrom() transfers token from the msg.sender

Line	Code

187-194	function transferFrom(
	//solhint-disable-next-line no-unused-vars
	address sender,
	address recipient,
	uint256 amount
	) public override returns (bool) {
	return transfer(recipient, amount);
	}

Description:
These are the issues in transferFrom() function :

1. The token transfer is supposed to happen from sender to recipient, if the allowance of msg.sender is higher than the amount. But the actual transfer is from msg.sender to recipient.

2. Allowance of msg.sender is not checked.

Remediation:
In this function, calculate and deduct fees, and then call the super.transferFrom() function.

Status: Closed

The code was updated in version 2, and now tokens are sent from ‘sender’ to ‘recipient’. Also, the transaction fee is paid by the ‘sender’.

Medium severity issues

4. Tokens are transferable even when paused

Line	Code

204-214	function transfer(address recipient, uint256 amount)
	public
	override
	returns (bool)
	{
	// Calculate fee and transfer the amount - fee
	uint256 fee = calculateFee(amount);
	amount -= fee;
	super.transfer(project, fee);
	return super.transfer(recipient, amount);
	}

187-194	function transferFrom(
	//solhint-disable-next-line no-unused-vars
	address sender,
	address recipient,
	uint256 amount
	) public override returns (bool) {
	return transfer(recipient, amount);
	}

Description:
The transfer() and transferFrom() function can still be called when the contract is paused. This means users can transfer tokens even if it has been paused by the owner.

Remediation:
We recommend overriding the _beforeTokenTransfer() function in the FAVE contract like this:

```
function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual override {
        super._beforeTokenTransfer(from, to, amount);

        require(!paused(), "ERC20Pausable: token transfer while paused");
    }
```

Status: Closed

The code was updated in version 2, whenNotPaused() modifier is used for both the functions.

Low level severity issues

5. Avoid using .send() to transfer Ether

Line	Code

145-150	function withdrawAll() public payable onlyOwner whenNotPaused {
	require(
	payable(msg.sender).send(address(this).balance),
	"Withdraw failed"
	);
	}

Description:
Although transfer() and send() have been recommended as a security best-practice to prevent reentrancy attacks because they only forward 2300 gas, the gas repricing of opcodes may break deployed contracts. For reference, read more.

Remediation:
Use .call.value(...)("") instead, without hardcoded gas limits along with checks-effects-interactions pattern or reentrancy guards for reentrancy protection.

Status: Closed

According to version 3, .call() with hardcoded gas limits is used to send ether.

6. Missing zero address validation

Line	Code

74-79	function updateProject(address payable _project) external onlyOwner {
	require(project != _project, "New project can't be old project");
	address oldProject = project;
	project = _project;
	emit LogProjectChanged(oldProject, _project);
	}
	
	"Withdraw failed"
	);
	}

Description:
When updating the project address, it should be checked for zero address. Otherwise, tokens sent to the zero address may be burnt forever.

Remediation:
Use a require statement to check for zero address when updating the project address.

Status: Closed

According to version 3, the new project address is now checked for address zero.

Informational

7. Missing Events for Significant Transactions

Line	Code

74-79	function updateDecimals(uint8 noOfDecimals) public onlyOwner whenNotPaused {
	tokenDecimals = noOfDecimals;
	}

Description:
The missing event makes it difficult to track off-chain decimal changes. An event should be emitted for significant transactions calling the updateDecimals() function.

Remediation:
We recommend emitting an event to log the update of the tokenDecimals variable.

Status: Closed

According to version 3, 'LogDecimalsChanged' event is emitted by the updateDecimals() function.

8. Wrong comments

Line	Code
59	* - can only be invoked by the project */ function transferWithoutFeeDeduction(address recipient, uint256 amount) external onlyProjectOrOwner
72	* Requirements: * - can only be invoked by the project */ function updateProject(address payable _project) external onlyOwner {
112	* Requirements: * - invocation can be done, only by the contract owner. */ function burn(address account, uint256 amount) public whenNotPaused {
129	* Requirements: * * - invocation can be done, only by the contract owner. */ function mint(address account, uint256 amount) public whenNotPaused {

Description:
Wrong comments were found for the above code. A misunderstanding comment could influence code readability.

Remediation:
The comments for the above code should be corrected or removed properly.

Status: Closed

The code was updated in version 3 to include appropriate comments.

9. Public function that could be declared external

Description:
The following public functions that are never called by the contract should be declared external to save gas:

updateDecimals()
updateFee()
pause()
unpause()
withdrawAll()

Remediation:
Use the external attribute for functions never called from the contract.

Status: Closed

Function visibility was changed to external in version 3.

10. 'address project' defined as payable

Description:
The 'address project' is never sent any ether from the contract. So there is no need to define it as payable.

Remediation:
Do not use the keyword 'payable' for 'address project'.

Status: Closed

'payable' keyword was removed from the code in version 3.

11. **Unlocked pragma**

pragma solidity ^0.8.0;

Description:
Contracts should be deployed using the same compiler version/flags with which they have been tested. Locking the pragma (for e.g., by not using ^ in pragma solidity 0.8.0) ensures that contracts do not accidentally get deployed using an older compiler version with unfixed bugs.

Remediation:
Lock the pragma version.

Status: Closed

Solidity compiler version was locked to 0.8.4 in version 3.

Functional test

Function Names	Testing results
transferWithoutFeeDeduction()	Passed
updateProject()	Passed
updateDecimals()	Passed
updateFee()	Passed
burn()	Passed
withdrawAll()	Passed
pause()	Passed
unpause()	Passed
transferFrom()	Passed
transfer()	Passed
decimals()	Passed

Automated Testing

Slither

```
INFO:Detectors:
FAVE.withdrawAll() (FAVE.sol#121-123) sends eth to arbitrary user
        Dangerous calls:
        - address(msg.sender).transfer(address(this).balance) (FAVE.sol#122)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
Different versions of Solidity is used:
        - Version used: ['0.8.4', '>=0.4.22<0.9.0', '^0.8.0']
        - ^0.8.0 (openzeppelin-solidity\contracts\utils\Context.sol#3)
        - ^0.8.0 (openzeppelin-solidity\contracts\token\ERC20\ERC20.sol#3)
        - 0.8.4 (FAVE.sol#2)
        - ^0.8.0 (openzeppelin-solidity\contracts\token\ERC20\IERC20.sol#3)
        - ^0.8.0 (openzeppelin-solidity\contracts\token\ERC20\extensions\IERC20Metadata.sol#3)
        - >=0.4.22<0.9.0 (Migrations.sol#2)
        - ^0.8.0 (openzeppelin-solidity\contracts\access\Ownable.sol#3)
        - ^0.8.0 (openzeppelin-solidity\contracts\security\Pausable.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Pragma version^0.8.0 (openzeppelin-solidity\contracts\utils\Context.sol#3) necessitates a version too recent to be trusted. Consider de
ploying with 0.6.12/0.7.6
Pragma version^0.8.0 (openzeppelin-solidity\contracts\token\ERC20\ERC20.sol#3) necessitates a version too recent to be trusted. Conside
r deploying with 0.6.12/0.7.6
Pragma version0.8.4 (FAVE.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (openzeppelin-solidity\contracts\token\ERC20\IERC20.sol#3) necessitates a version too recent to be trusted. Consid
er deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (openzeppelin-solidity\contracts\token\ERC20\extensions\IERC20Metadata.sol#3) necessitates a version too recent to
 be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version>=0.4.22<0.9.0 (Migrations.sol#2) is too complex
Pragma version^0.8.0 (openzeppelin-solidity\contracts\access\Ownable.sol#3) necessitates a version too recent to be trusted. Consider d
eploying with 0.6.12/0.7.6
Pragma version^0.8.0 (openzeppelin-solidity\contracts\security\Pausable.sol#3) necessitates a version too recent to be trusted. Conside
r deploying with 0.6.12/0.7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter FAVE.updateProject(address)._project (FAVE.sol#76) is not in mixedCase
Parameter FAVE.updateFee(uint256)._newFee (FAVE.sol#109) is not in mixedCase
Variable Migrations.last_completed_migration (Migrations.sol#6) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (openzeppelin-solidity\contracts\utils\Context.sol#21)" inContext (openzeppelin-solidity\contracts\utils\Con
text.sol#15-24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
name() should be declared external:
        - ERC20.name() (openzeppelin-solidity\contracts\token\ERC20\ERC20.sol#60-62)
symbol() should be declared external:
        - ERC20.symbol() (openzeppelin-solidity\contracts\token\ERC20\ERC20.sol#68-70)
decimals() should be declared external:
        - ERC20.decimals() (openzeppelin-solidity\contracts\token\ERC20\ERC20.sol#85-87)
        - FAVE.decimals() (FAVE.sol#223-225)
totalSupply() should be declared external:
```

```
totalSupply() should be declared external:
        - ERC20.totalSupply() (openzeppelin-solidity\contracts\token\ERC20\ERC20.sol#92-94)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (openzeppelin-solidity\contracts\token\ERC20\ERC20.sol#99-101)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (openzeppelin-solidity\contracts\token\ERC20\ERC20.sol#119-121)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (openzeppelin-solidity\contracts\token\ERC20\ERC20.sol#130-133)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (openzeppelin-solidity\contracts\token\ERC20\ERC20.sol#170-173)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (openzeppelin-solidity\contracts\token\ERC20\ERC20.sol#189-195)
pause() should be declared external:
        - FAVE.pause() (FAVE.sol#145-147)
unpause() should be declared external:
        - FAVE.unpause() (FAVE.sol#157-159)
setCompleted(uint256) should be declared external:
        - Migrations.setCompleted(uint256) (Migrations.sol#16-18)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (openzeppelin-solidity\contracts\access\Ownable.sol#54-57)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (openzeppelin-solidity\contracts\access\Ownable.sol#63-67)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:. analyzed (8 contracts with 75 detectors), 29 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Results

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the Favecoin platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Favecoin Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.

No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract.

Numerous issues were discovered during the initial audit. However, all of them have been fixed and checked now.

Favecoin

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com