EtherAuthority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	FAMPIT Token
Website:	www.fampit.com
Platform:	Ethereum Blockchain
Language:	Solidity
Date:	April 14th, 2022

Table of contents

`

Introduction

EtherAuthority was contracted by the FAMPIT team to perform the Security audit of the FAMPIT Token smart contract code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on April 14th, 2022.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

The FAMPIT is a NFT (ERC721A standard) smart Contract, which has functions like isWhitelisted, addNewWhitelistUsers, tokenURI, numberMinted, withdrawMoney, mint, etc.

Audit scope

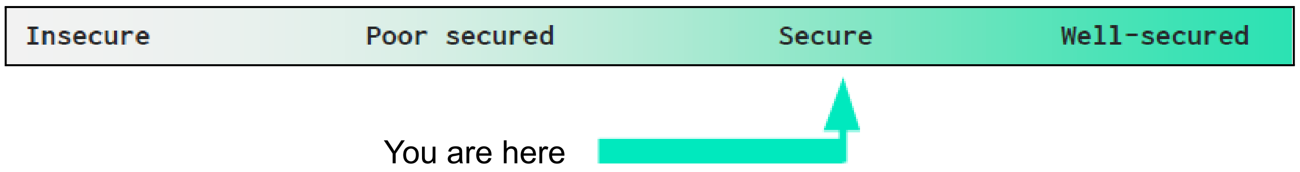
Name	**Code Review and Security Analysis Report for FAMPIT Token Smart Contract**
Platform	**Ethereum / Solidity**
File	FAMPIT.sol
File MD5 Hash	8166862EEB4A886CBB5C945C62F1A7B6
Online Code Link	0x0160f5dfb9a29Cb9aFF40Fefe6Bf8D659363f166
Audit Date	April 14th, 2022

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: ● Name: FAMPITFoundingMembership ● Symbol: FAMPIT ● Maximum per Transaction: 1 ● Maximum per Address: 1 ● Price: 0.2 ether ● Total number of NFTs: 500 ● Maximum tokens mint per batch: 50 ● Maximum Supply: 480 ● Reserve Supply: 20	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. This token contract does contain owner control, which does not make it fully decentralized.



We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 3 low and some very low level issues.

Investors Advice: Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Moderated
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Moderated
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: PASSED

Code Quality

This audit scope has 1 smart contract file. Smart contract contains Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in FAMPIT Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the FAMPIT Token.

The FAMPIT Token team has **not** provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are **not well** commented on smart contracts.

Documentation

We were given an FAMPIT Token smart contract code in the form of a Rinkeby Etherscan Web Link. The hash of that code is mentioned above in the table.

As mentioned above, code parts are **not well** commented. So it is not easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Another source of information was its official website https://www.fampit.com which provided rich information about the project architecture and tokenomics.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well known industry standard open source projects.

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	totalSupply	read	Passed	No Issue
8	tokenByIndex	read	Passed	No Issue
9	tokenOfOwnerByIndex	read	Passed	No Issue
10	supportsInterface	read	Passed	No Issue
11	balanceOf	read	Passed	No Issue
12	_numberMinted	internal	Passed	No Issue
13	ownershipOf	internal	Passed	No Issue
14	ownerOf	read	Passed	No Issue
15	name	read	Passed	No Issue
16	symbol	read	Passed	No Issue
17	tokenURI	read	Passed	No Issue
18	_baseURI	internal	Passed	No Issue
19	_getUriExtension	internal	Passed	No Issue
20	approve	write	Passed	No Issue
21	getApproved	read	Passed	No Issue
22	setApprovalForAll	write	Passed	No Issue
23	isApprovedForAll	read	Passed	No Issue
24	transferFrom	write	Passed	No Issue
25	safeTransferFrom	write	Passed	No Issue
26	_exists	internal	Passed	No Issue
27	_safeMint	internal	Passed	No Issue
28	_transfer	write	Passed	No Issue
29	_approve	write	Passed	No Issue
30	_checkOnERC721Received	write	Passed	No Issue
31	_beforeTokenTransfers	internal	Passed	No Issue
32	_afterTokenTransfers	internal	Passed	No Issue
33	callerIsUser	modifier	Passed	No Issue
34	mint	external	Passed	No Issue
35	tokenURI	read	Passed	No Issue
36	isWhitelisted	read	Passed	No Issue
37	conf	external	Critical operation lacks event log,Function input parameters lack of check	Refer Audit Findings
38	addNewWhitelistUsers	write	access only Owner	No Issue

39	setURIbeforeRevel	external	access only Owner	No Issue
40	setBaseURI	external	access only Owner	No Issue
41	_baseURI	internal	Passed	No Issue
42	numberMinted	read	Passed	No Issue
43	getOwnershipData	external	Passed	No Issue
44	withdrawMoney	external	Critical operation lacks event log	Refer Audit Findings
45	changeRevelStatus	external	access only Owner	No Issue
46	changeMintPrice	external	access only Owner	No Issue
47	changeMAX_PER_Transtion	external	access only Owner	No Issue
48	changeMAX_PER_Address	external	Function input parameters lack of check	Refer Audit Findings
49	setStatus	external	access only Owner	No Issue
50	getStatus	read	Passed	No Issue
51	getPrice	read	Passed	No Issue
52	giveaway	write	Function input parameters lack of check	Refer Audit Findings
53	setStop	external	access only Owner	No Issue
54	setReserve	external	access only Owner	No Issue
55	nonReentrant	modifier	Passed	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Critical operation lacks event log:

Missing event log for:

- conf
- withdrawMoney

Resolution: Please write an event log for listed events.

(2) Function input parameters lack of check:

Variable validation is not performed in below functions:

- conf – MaxPerAddress, MaxPerWallet , Price
- changeMAX_PER_Address – MAXPERAddress
- giveaway – address_

Resolution: We advise to put validation like integer type variables should be greater than 0 and address type variables should not be address(0).

(3) Price does not get update automatically for public:

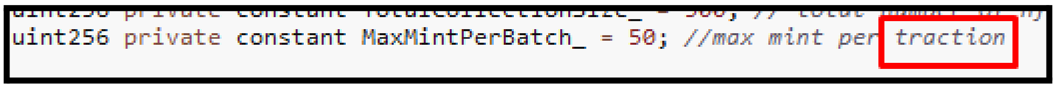
As mentioned in the requirement, the price for public sale is 0.25 ETH. There is no code for setting price when status changes from 1 to 2.

Resolution: We suggest setting the price on status changes from 1 to 2 or the owner needs to take care of this.

Very Low / Informational / Best practices:

(1) Spelling mistake:

MAX_PER_Transtion is misspelled



```
uint256 private constant MaxMintPerBatch_ = 50; //max mint per traction
```

Here traction is misspelled.

Resolution: We suggest correcting the words.

- MAX_PER_Transtion should be MAX_PER_Transaction
- traction should be transaction

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

- conf: Owner can update status, MaxPerAddress, MaxPerWallet , price, Stop_At values.
- addNewWhitelistUsers: Owner can add new user addresses in whitelist.
- setURIbeforeRevel: Owner can set URI before revel.
- setBaseURI: Owner can set BaseURI.
- withdrawMoney: Owner can withdraw money.
- changeRevelStatus: Owner can update revel status.
- changeMintPrice: Owner can update mint price.
- changeMAX_PER_Transtion: Owner can update maximum translation
- changeMAX_PER_Address: Owner can update max address.
- setStatus: Owner can update status.
- giveaway: Owner can update address and quantity values.
- setStop: Owner can update stop at value.
- setReserve: Owner can update reserve value.

Conclusion

We were given a contract code. And we have used all possible tests based on given objects as files. We have not observed any major issues. So, **it's good to go to production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security state of the reviewed contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:
In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:
Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

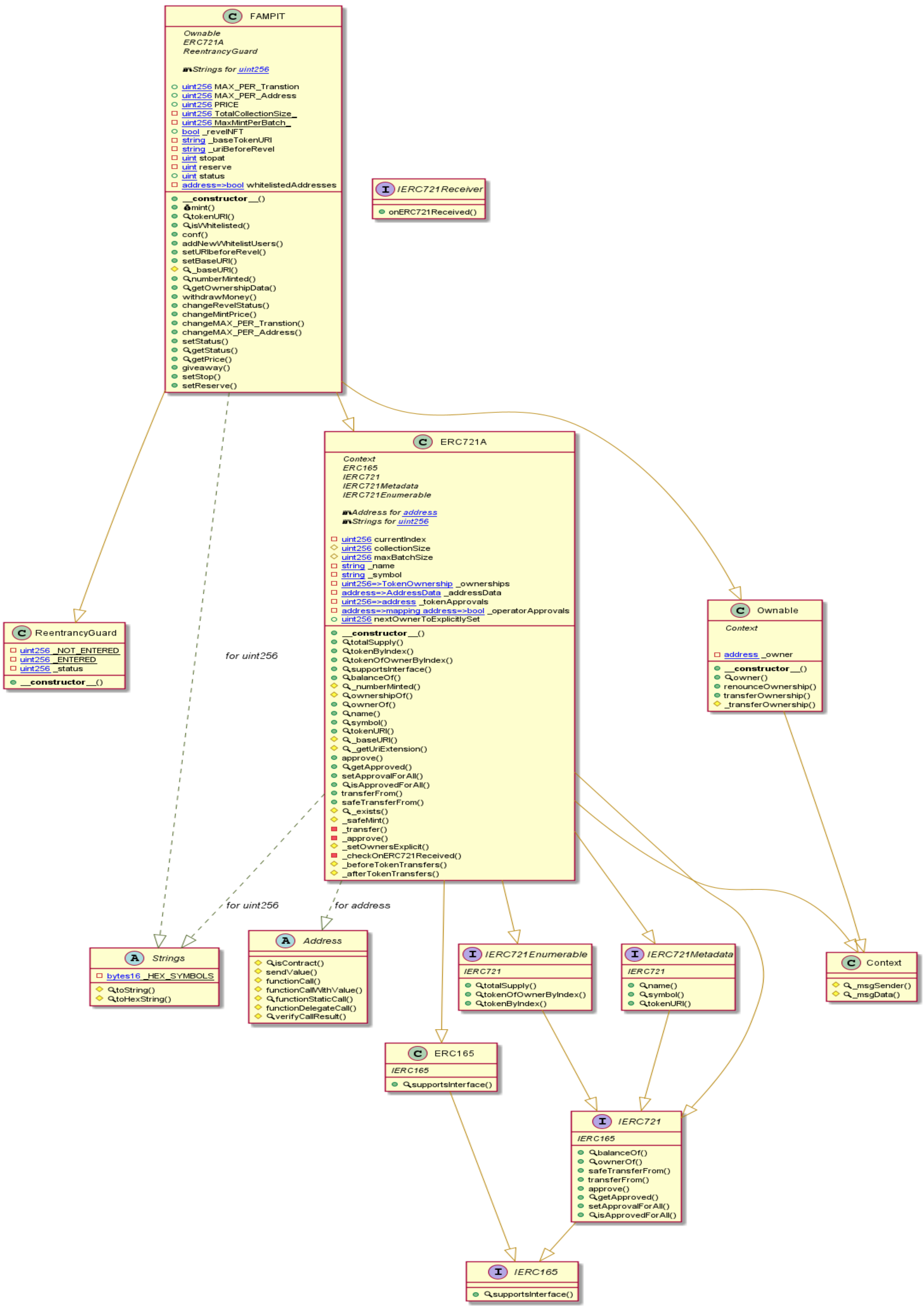
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

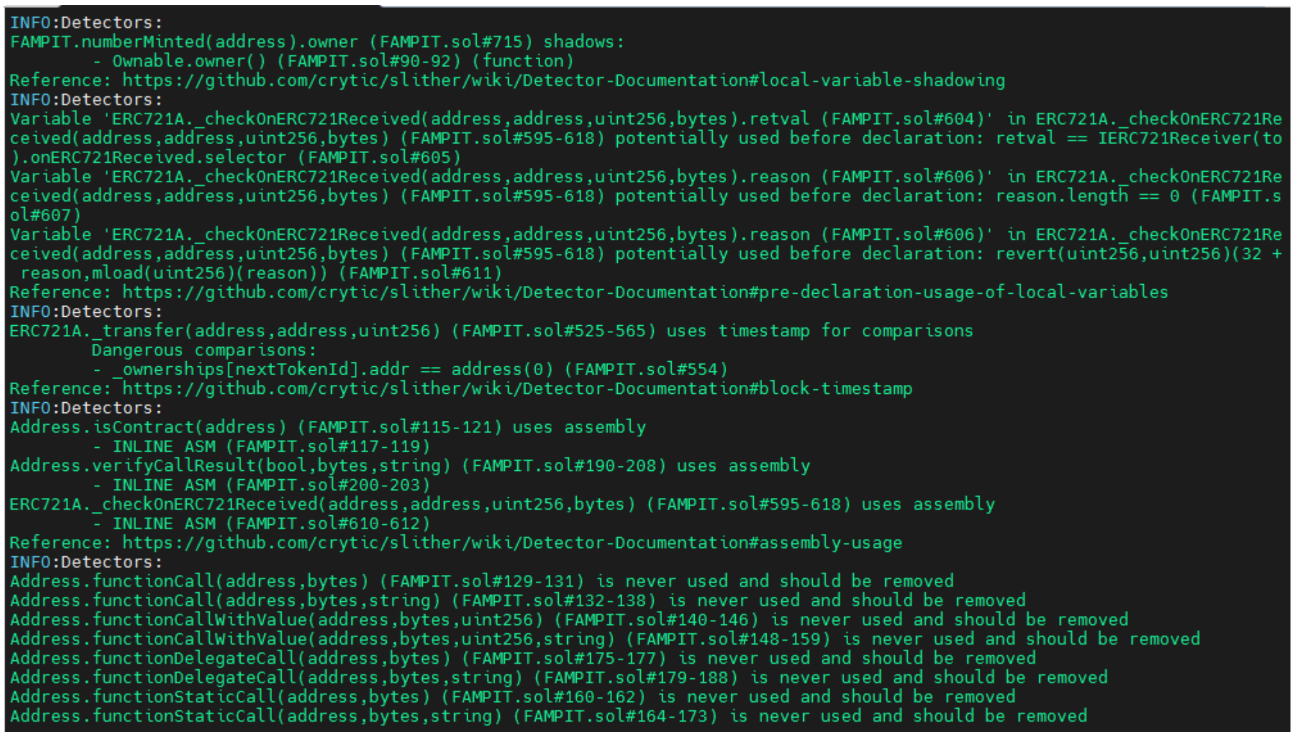
Appendix

Code Flow Diagram - FAMPIT Token

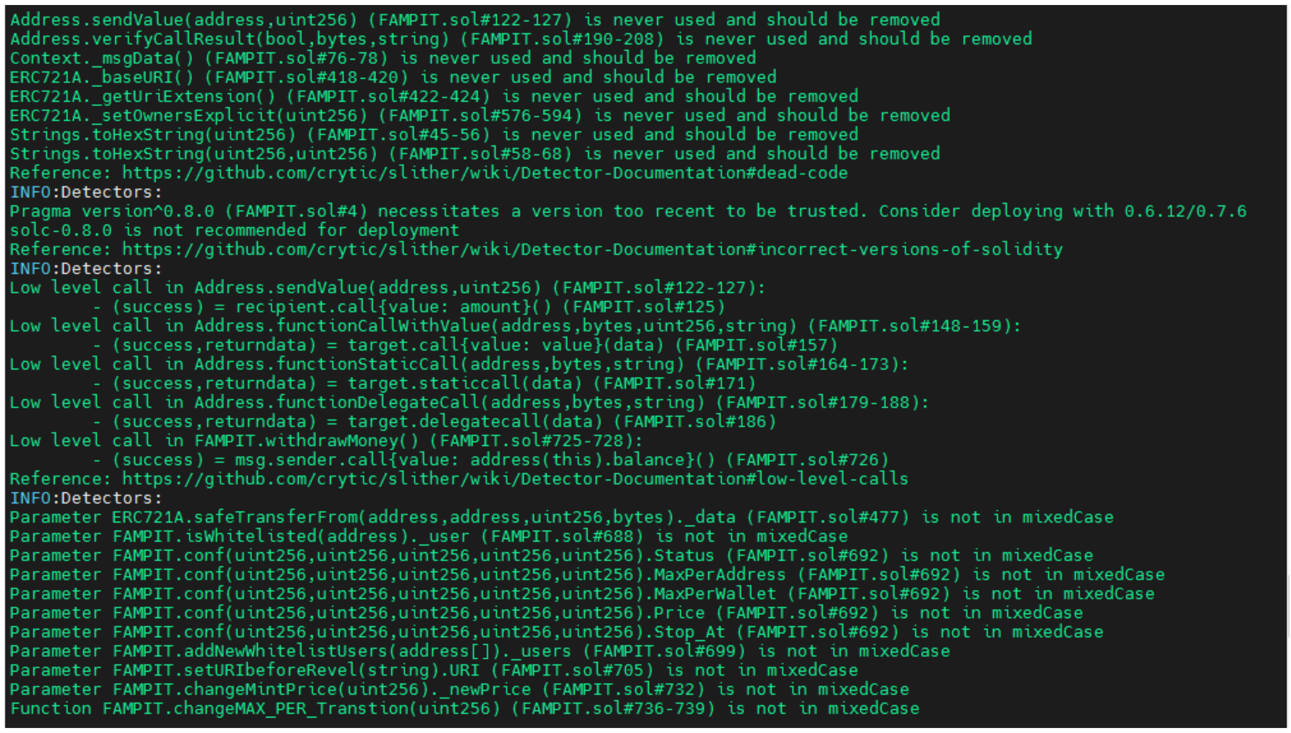


Slither Results Log

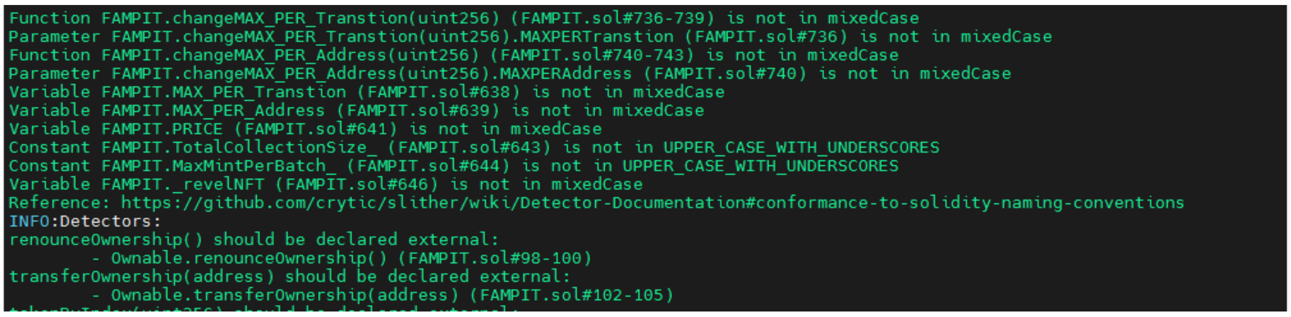
Slither log >> FAMPIT.sol



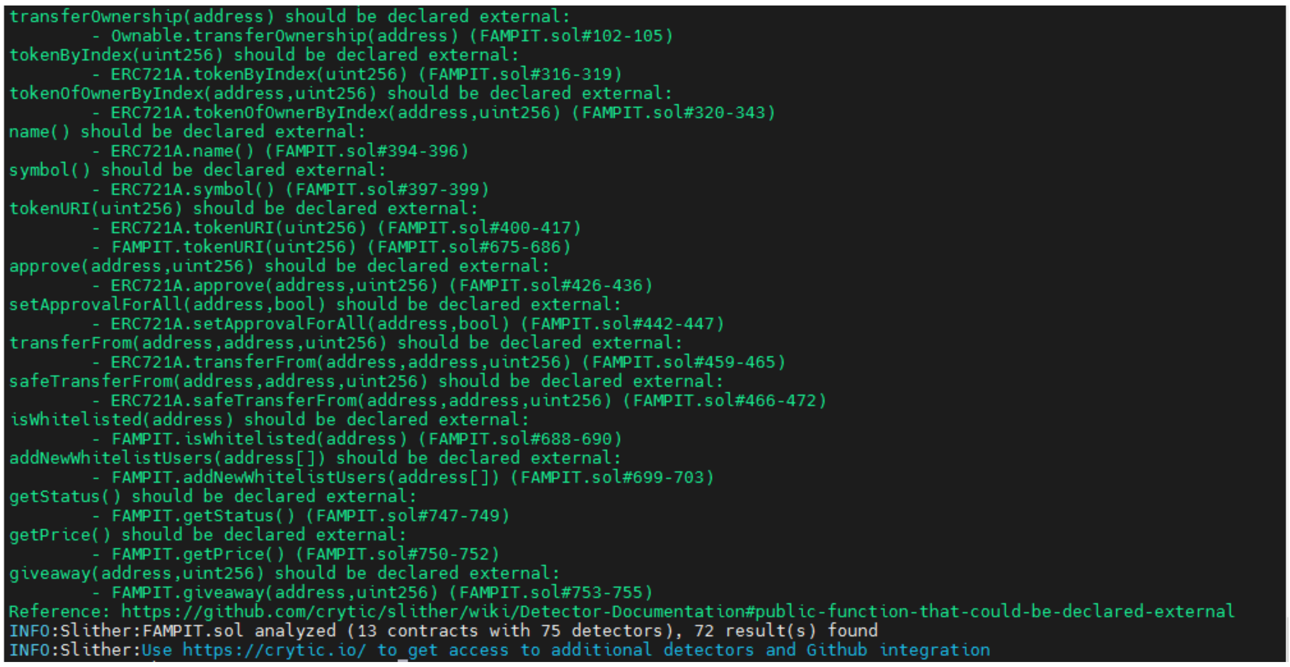
```
INFO:Detectors:
FAMPIT.numberMinted(address).owner (FAMPIT.sol#715) shadows:
        - Ownable.owner() (FAMPIT.sol#90-92) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Variable 'ERC721A._checkOnERC721Received(address,address,uint256,bytes).retval (FAMPIT.sol#604)' in ERC721A._checkOnERC721Re
ceived(address,address,uint256,bytes) (FAMPIT.sol#595-618) potentially used before declaration: retval == IERC721Receiver(to
).onERC721Received.selector (FAMPIT.sol#605)
Variable 'ERC721A._checkOnERC721Received(address,address,uint256,bytes).reason (FAMPIT.sol#606)' in ERC721A._checkOnERC721Re
ceived(address,address,uint256,bytes) (FAMPIT.sol#595-618) potentially used before declaration: reason.length == 0 (FAMPIT.s
ol#607)
Variable 'ERC721A._checkOnERC721Received(address,address,uint256,bytes).reason (FAMPIT.sol#606)' in ERC721A._checkOnERC721Re
ceived(address,address,uint256,bytes) (FAMPIT.sol#595-618) potentially used before declaration: revert(uint256,uint256)(32 +
 reason,mload(uint256)(reason)) (FAMPIT.sol#611)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables
INFO:Detectors:
ERC721A._transfer(address,address,uint256) (FAMPIT.sol#525-565) uses timestamp for comparisons
        Dangerous comparisons:
        - _ownerships[nextTokenId].addr == address(0) (FAMPIT.sol#554)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (FAMPIT.sol#115-121) uses assembly
        - INLINE ASM (FAMPIT.sol#117-119)
Address.verifyCallResult(bool,bytes,string) (FAMPIT.sol#190-208) uses assembly
        - INLINE ASM (FAMPIT.sol#200-203)
ERC721A._checkOnERC721Received(address,address,uint256,bytes) (FAMPIT.sol#595-618) uses assembly
        - INLINE ASM (FAMPIT.sol#610-612)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address.functionCall(address,bytes) (FAMPIT.sol#129-131) is never used and should be removed
Address.functionCall(address,bytes,string) (FAMPIT.sol#132-138) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (FAMPIT.sol#140-146) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (FAMPIT.sol#148-159) is never used and should be removed
Address.functionDelegateCall(address,bytes) (FAMPIT.sol#175-177) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (FAMPIT.sol#179-188) is never used and should be removed
Address.functionStaticCall(address,bytes) (FAMPIT.sol#160-162) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (FAMPIT.sol#164-173) is never used and should be removed
```



```
Address.sendValue(address,uint256) (FAMPIT.sol#122-127) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (FAMPIT.sol#190-208) is never used and should be removed
Context._msgData() (FAMPIT.sol#76-78) is never used and should be removed
ERC721A._baseURI() (FAMPIT.sol#418-420) is never used and should be removed
ERC721A._getUriExtension() (FAMPIT.sol#422-424) is never used and should be removed
ERC721A._setOwnersExplicit(uint256) (FAMPIT.sol#576-594) is never used and should be removed
Strings.toHexString(uint256) (FAMPIT.sol#45-56) is never used and should be removed
Strings.toHexString(uint256,uint256) (FAMPIT.sol#58-68) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.0 (FAMPIT.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (FAMPIT.sol#122-127):
        - (success) = recipient.call{value: amount}() (FAMPIT.sol#125)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (FAMPIT.sol#148-159):
        - (success,returndata) = target.call{value: value}(data) (FAMPIT.sol#157)
Low level call in Address.functionStaticCall(address,bytes,string) (FAMPIT.sol#164-173):
        - (success,returndata) = target.staticcall(data) (FAMPIT.sol#171)
Low level call in Address.functionDelegateCall(address,bytes,string) (FAMPIT.sol#179-188):
        - (success,returndata) = target.delegatecall(data) (FAMPIT.sol#186)
Low level call in FAMPIT.withdrawMoney() (FAMPIT.sol#725-728):
        - (success) = msg.sender.call{value: address(this).balance}() (FAMPIT.sol#726)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter ERC721A.safeTransferFrom(address,address,uint256,bytes)._data (FAMPIT.sol#477) is not in mixedCase
Parameter FAMPIT.isWhitelisted(address)._user (FAMPIT.sol#688) is not in mixedCase
Parameter FAMPIT.conf(uint256,uint256,uint256,uint256,uint256).Status (FAMPIT.sol#692) is not in mixedCase
Parameter FAMPIT.conf(uint256,uint256,uint256,uint256,uint256).MaxPerAddress (FAMPIT.sol#692) is not in mixedCase
Parameter FAMPIT.conf(uint256,uint256,uint256,uint256,uint256).MaxPerWallet (FAMPIT.sol#692) is not in mixedCase
Parameter FAMPIT.conf(uint256,uint256,uint256,uint256,uint256).Price (FAMPIT.sol#692) is not in mixedCase
Parameter FAMPIT.conf(uint256,uint256,uint256,uint256,uint256).Stop_At (FAMPIT.sol#692) is not in mixedCase
Parameter FAMPIT.addNewWhitelistUsers(address[])._users (FAMPIT.sol#699) is not in mixedCase
Parameter FAMPIT.setURIbeforeRevel(string).URI (FAMPIT.sol#705) is not in mixedCase
Parameter FAMPIT.changeMintPrice(uint256)._newPrice (FAMPIT.sol#732) is not in mixedCase
Function FAMPIT.changeMAX_PER_Transtion(uint256) (FAMPIT.sol#736-739) is not in mixedCase
```



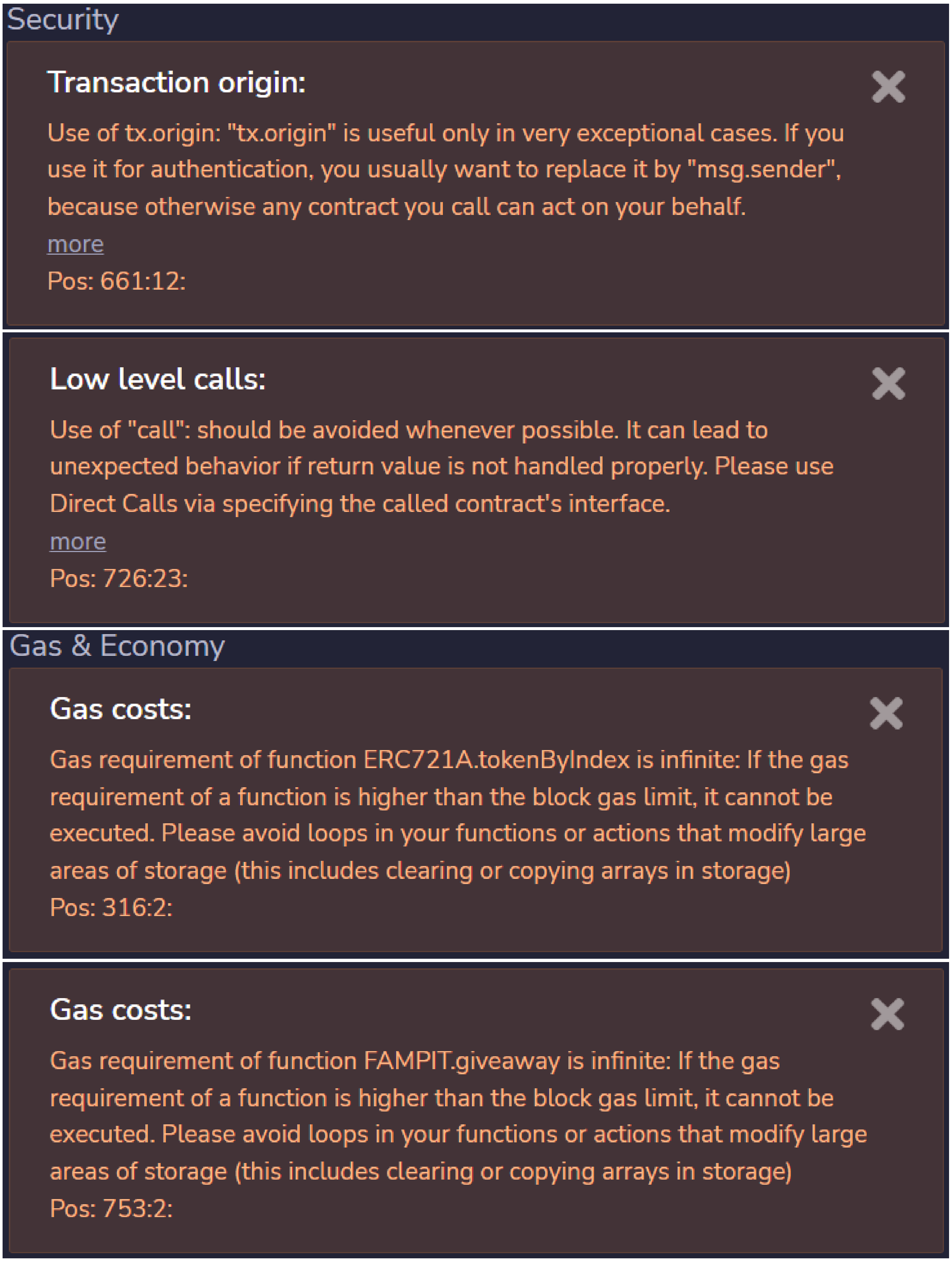
```
Function FAMPIT.changeMAX_PER_Transtion(uint256) (FAMPIT.sol#736-739) is not in mixedCase
Parameter FAMPIT.changeMAX_PER_Transtion(uint256).MAXPERTranstion (FAMPIT.sol#736) is not in mixedCase
Function FAMPIT.changeMAX_PER_Address(uint256) (FAMPIT.sol#740-743) is not in mixedCase
Parameter FAMPIT.changeMAX_PER_Address(uint256).MAXPERAddress (FAMPIT.sol#740) is not in mixedCase
Variable FAMPIT.MAX_PER_Transtion (FAMPIT.sol#638) is not in mixedCase
Variable FAMPIT.MAX_PER_Address (FAMPIT.sol#639) is not in mixedCase
Variable FAMPIT.PRICE (FAMPIT.sol#641) is not in mixedCase
Constant FAMPIT.TotalCollectionSize_ (FAMPIT.sol#643) is not in UPPER_CASE_WITH_UNDERSCORES
Constant FAMPIT.MaxMintPerBatch_ (FAMPIT.sol#644) is not in UPPER_CASE_WITH_UNDERSCORES
Variable FAMPIT._revelNFT (FAMPIT.sol#646) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (FAMPIT.sol#98-100)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (FAMPIT.sol#102-105)
```



```
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (FAMPIT.sol#102-105)
tokenByIndex(uint256) should be declared external:
        - ERC721A.tokenByIndex(uint256) (FAMPIT.sol#316-319)
tokenOfOwnerByIndex(address,uint256) should be declared external:
        - ERC721A.tokenOfOwnerByIndex(address,uint256) (FAMPIT.sol#320-343)
name() should be declared external:
        - ERC721A.name() (FAMPIT.sol#394-396)
symbol() should be declared external:
        - ERC721A.symbol() (FAMPIT.sol#397-399)
tokenURI(uint256) should be declared external:
        - ERC721A.tokenURI(uint256) (FAMPIT.sol#400-417)
        - FAMPIT.tokenURI(uint256) (FAMPIT.sol#675-686)
approve(address,uint256) should be declared external:
        - ERC721A.approve(address,uint256) (FAMPIT.sol#426-436)
setApprovalForAll(address,bool) should be declared external:
        - ERC721A.setApprovalForAll(address,bool) (FAMPIT.sol#442-447)
transferFrom(address,address,uint256) should be declared external:
        - ERC721A.transferFrom(address,address,uint256) (FAMPIT.sol#459-465)
safeTransferFrom(address,address,uint256) should be declared external:
        - ERC721A.safeTransferFrom(address,address,uint256) (FAMPIT.sol#466-472)
isWhitelisted(address) should be declared external:
        - FAMPIT.isWhitelisted(address) (FAMPIT.sol#688-690)
addNewWhitelistUsers(address[]) should be declared external:
        - FAMPIT.addNewWhitelistUsers(address[]) (FAMPIT.sol#699-703)
getStatus() should be declared external:
        - FAMPIT.getStatus() (FAMPIT.sol#747-749)
getPrice() should be declared external:
        - FAMPIT.getPrice() (FAMPIT.sol#750-752)
giveaway(address,uint256) should be declared external:
        - FAMPIT.giveaway(address,uint256) (FAMPIT.sol#753-755)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:FAMPIT.sol analyzed (13 contracts with 75 detectors), 72 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

FAMPIT.sol



Security

Transaction origin:

Use of tx.origin: "tx.origin" is useful only in very exceptional cases. If you use it for authentication, you usually want to replace it by "msg.sender", because otherwise any contract you call can act on your behalf.
more
Pos: 661:12:

Low level calls:

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.
more
Pos: 726:23:

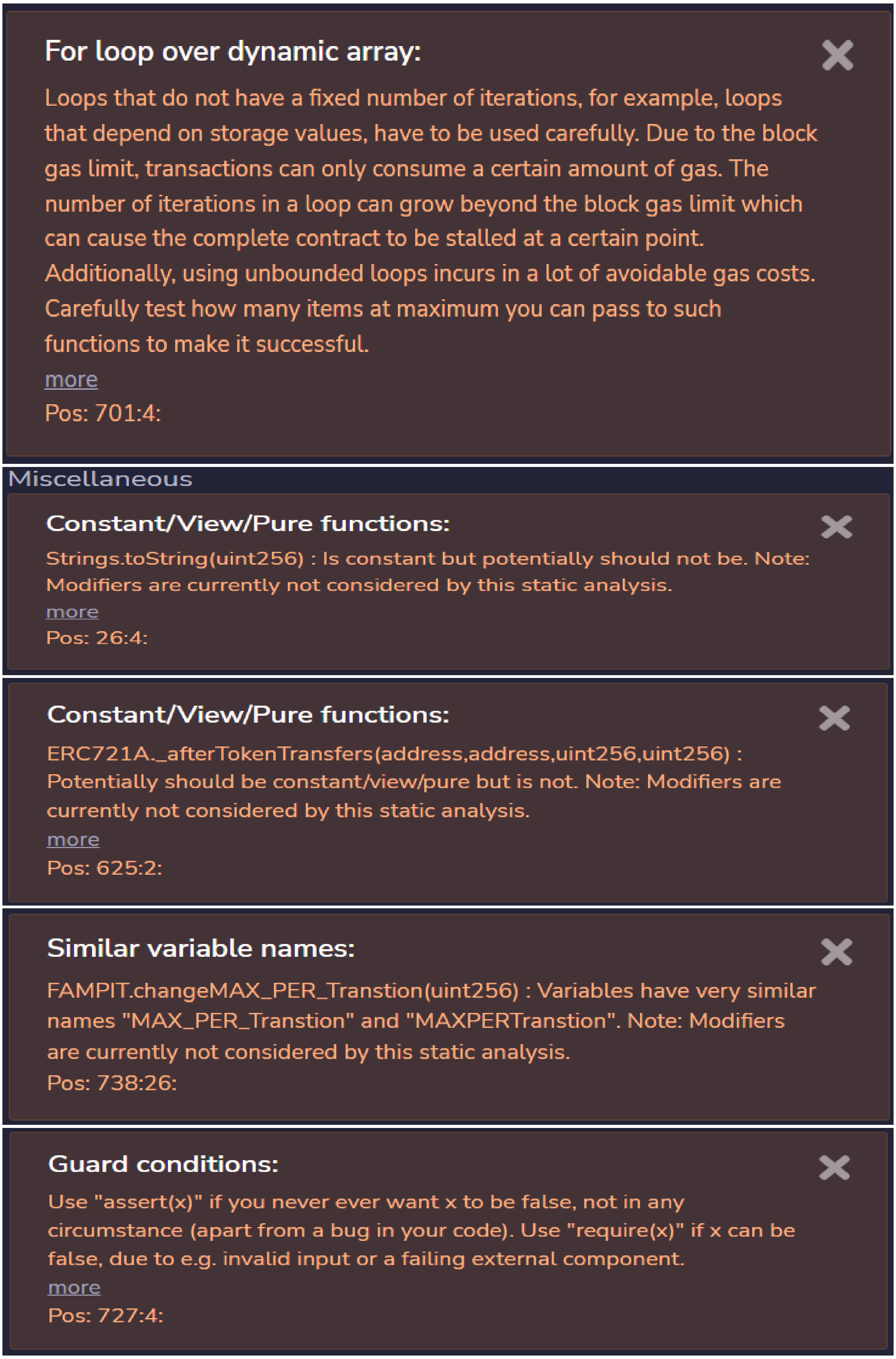
Gas & Economy

Gas costs:

Gas requirement of function ERC721A.tokenByIndex is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 316:2:

Gas costs:

Gas requirement of function FAMPIT.giveaway is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 753:2:



For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

more

Pos: 701:4:

Miscellaneous

Constant/View/Pure functions:

Strings.toString(uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 26:4:

Constant/View/Pure functions:

ERC721A._afterTokenTransfers(address,address,uint256,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 625:2:

Similar variable names:

FAMPIT.changeMAX_PER_Transtion(uint256) : Variables have very similar names "MAX_PER_Transtion" and "MAXPERTranstion". Note: Modifiers are currently not considered by this static analysis.

Pos: 738:26:

Guard conditions:

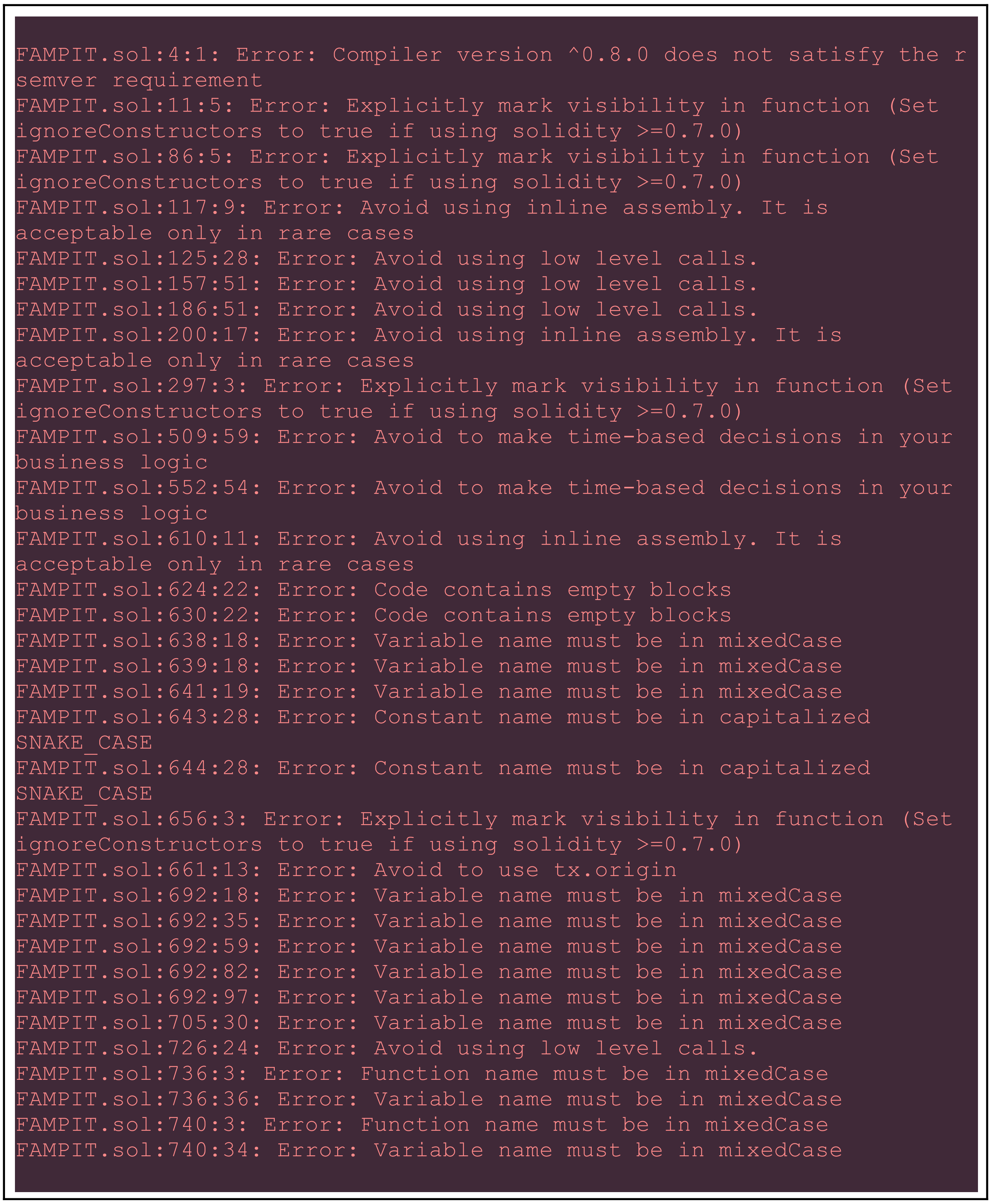
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 727:4:

Solhint Linter

FAMPIT.sol



```
FAMPIT.sol:4:1: Error: Compiler version ^0.8.0 does not satisfy the r
semver requirement
FAMPIT.sol:11:5: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)
FAMPIT.sol:86:5: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)
FAMPIT.sol:117:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
FAMPIT.sol:125:28: Error: Avoid using low level calls.
FAMPIT.sol:157:51: Error: Avoid using low level calls.
FAMPIT.sol:186:51: Error: Avoid using low level calls.
FAMPIT.sol:200:17: Error: Avoid using inline assembly. It is
acceptable only in rare cases
FAMPIT.sol:297:3: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)
FAMPIT.sol:509:59: Error: Avoid to make time-based decisions in your
business logic
FAMPIT.sol:552:54: Error: Avoid to make time-based decisions in your
business logic
FAMPIT.sol:610:11: Error: Avoid using inline assembly. It is
acceptable only in rare cases
FAMPIT.sol:624:22: Error: Code contains empty blocks
FAMPIT.sol:630:22: Error: Code contains empty blocks
FAMPIT.sol:638:18: Error: Variable name must be in mixedCase
FAMPIT.sol:639:18: Error: Variable name must be in mixedCase
FAMPIT.sol:641:19: Error: Variable name must be in mixedCase
FAMPIT.sol:643:28: Error: Constant name must be in capitalized
SNAKE_CASE
FAMPIT.sol:644:28: Error: Constant name must be in capitalized
SNAKE_CASE
FAMPIT.sol:656:3: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)
FAMPIT.sol:661:13: Error: Avoid to use tx.origin
FAMPIT.sol:692:18: Error: Variable name must be in mixedCase
FAMPIT.sol:692:35: Error: Variable name must be in mixedCase
FAMPIT.sol:692:59: Error: Variable name must be in mixedCase
FAMPIT.sol:692:82: Error: Variable name must be in mixedCase
FAMPIT.sol:692:97: Error: Variable name must be in mixedCase
FAMPIT.sol:705:30: Error: Variable name must be in mixedCase
FAMPIT.sol:726:24: Error: Avoid using low level calls.
FAMPIT.sol:736:3: Error: Function name must be in mixedCase
FAMPIT.sol:736:36: Error: Variable name must be in mixedCase
FAMPIT.sol:740:3: Error: Function name must be in mixedCase
FAMPIT.sol:740:34: Error: Variable name must be in mixedCase
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.