Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	WP Smart Contracts
Website:	wpsmartcontracts.com
Platform:	ETH, BSC, and others
Language:	Solidity
Date:	April 22nd, 2022

Table of contents

Email: audit@EtherAuthority.io

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the WP Smart Contracts team to perform the Security audit of the Suika (ERC721), Matcha (ERC721), Almond (Staking) and Ube (staking) smart contracts 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 22nd, 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 WP Smart Contracts provides the smart contract solutions to the wordpress users. They develop various WP plugins which lets WP websites use the smart contract deployment quickly. We audited their Suika (ERC721), Matcha (ERC721), Almond (Staking) and Ube (staking) smart contracts.

Audit scope

Name	**Code Review and Security Analysis Report for WP Smart Contracts Protocol Smart Contracts**
Platform	**Multiple blockchain platforms / Solidity**
File 1	Suika - ERC721 NFT
File 2	Ube - Staking
File 3	Matcha - ERC721 NFT Marketplace
File 4	Almond - Staking
Audit Date	April 22nd, 2022
Revision Date	May 9th, 2022

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Ube - Staking ● Owner can set: ○ APY: annual percentage yield, or annual percentage interest, calculated per second ○ Maturity: users can claim rewards only if they remain staked for at least this number of days ○ Minimum amount to create a stake ○ ERC-20/BEP-20 token to stake ● If the owner does not provide allowance of the token or removes it afterwards, then it will not pay any interest to users.	**YES, This is valid.**
Suika - ERC721 NFT ● This contract has native & advanced features like: ○ Ownership ○ Transfer ○ Approval ○ Mint ○ Sell ○ Auctions, buy and sell using a standard ERC-20 or BEP-20 token ○ Royalty commissions for NFT creators ● Contract owner can change following: ○ Commission Rate ○ Royalties Commission Rate ○ Payment Token ○ Grant / Revoke roles ● Unlimited tokens can be minted.	**YES, This is valid.**

Almond - Staking ● Owner can set: ○ The first ERC-20/BEP-20, which is used to stake ○ A secondary ERC-20/BEP-20 token to accrue interest ○ APY: annual interest rate for the first token (optional), calculated per second ○ APY 2: the APY for the secondary token. ○ Maturity: users can claim rewards only if they remain staked for at least this number of days ○ Minimum amount to create a stake ● If the owner does not provide allowance of the token or removes it afterwards, then it will not pay any interest to users.	**YES, This is valid.**
Matcha - ERC721 NFT Marketplace ● This contract has native & advanced features like: ○ Ownership ○ Transfer ○ Approval ○ Mint ○ Sell ○ Auction ○ Auctions, buy and sell using a native coins (ETH, BNB, Matic, etc) ● Contract owner can change following: ○ Commission Rate ○ Change owner wallet ○ Disable/Enable public minting ● Unlimited tokens can be minted.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. Also, these contracts do contain owner control, which does not make them 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 2 critical, 0 high, 0 medium and 4 low and some very low level issues.
These are fixed / acknowledged in the revised contract code.

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	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	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	Passed
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 4 smart contract files. Smart contracts contain Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in the WP Smart Contracts Protocol 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 WP Smart Contracts Protocol.

The WP Smart Contracts team has not provided unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Some code parts are not well commented on smart contracts. We suggest using Ethereum's NatSpec style for the commenting.

Documentation

We were given a WP Smart Contracts Protocol smart contract code in the form of a BSCScan / Etherscan web link. The links of that code are mentioned above in the table.

As mentioned above, code parts are not well commented. But the logic is straightforward. So it is 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.

Use of Dependencies

As per our observation, the libraries are used in this smart contracts 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

Stakes.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	isOwner	modifier	Passed	No Issue
3	changeOwner	write	access by is Owner	No Issue
4	getOwner	read	Passed	No Issue
5	nonReentrant	modifier	Passed	No Issue
6	start	external	Passed	No Issue
7	end	external	Passed	No Issue
8	set	write	access by is Owner	No Issue
9	get_gains	read	Passed	No Issue
10	ledger_length	read	Passed	No Issue

ERC721Suika.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	nonReentrant	modifier	Passed	No Issue
3	autoMint	write	access only Minter	No Issue
4	mint	write	access only Minter	No Issue
5	safeMint	write	access only Minter	No Issue
6	isMinter	read	Passed	No Issue
7	safeMint	write	access only Minter	No Issue
8	_burn	internal	Passed	No Issue
9	_beforeTokenTransfer	internal	Passed	No Issue
10	tokenURI	write	Passed	No Issue
11	supportsInterface	write	Passed	No Issue
12	addMinter	write	access only Role	No Issue
13	canIMint	write	Passed	No Issue
14	onlyMinter	modifier	Passed	No Issue
15	canSell	read	Passed	No Issue
16	sell	write	Passed	No Issue
17	getPrice	read	Passed	No Issue
18	canBuy	read	Passed	No Issue
19	buy	write	No fraction value in commission rates	Acknowledged
20	canAuction	read	Passed	No Issue
21	createAuction	write	Passed	No Issue
22	canBid	read	Passed	No Issue
23	_mint	internal	Passed	No Issue
24	bid	write	Passed	No Issue

25	canWithdraw	read	Passed	No Issue
26	withdraw	write	Passed	No Issue
27	canFinalize	read	Passed	No Issue
28	auctionFinalize	write	No fraction value in commission rates	Acknowledged
29	highestBidder	read	Passed	No Issue
30	highestBid	read	Passed	No Issue
31	callOptionalReturn	write	Passed	No Issue
32	updateAdmin	write	Passed	No Issue

StakesAlmond.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	isOwner	modifier	Passed	No Issue
3	changeOwner	write	access by isOwner	No Issue
4	getOwner	read	Passed	No Issue
5	nonReentrant	modifier	Passed	No Issue
6	start	external	Passed	No Issue
7	end	write	Passed	No Issue
8	set	write	access by isOwner	No Issue
9	get_gains	read	Passed	No Issue
10	get_gains2	read	Passed	No Issue
11	ledger_length	read	Passed	No Issue

ERC721Matcha.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	exists	read	Passed	No Issue
3	tokensOfOwner	read	Passed	No Issue
4	setTokenURI	write	Anyone can set this	Removed in revised code
5	autoMint	write	access only Minter	No Issue
6	transfer	write	Passed	No Issue
7	nonReentrant	modifier	Passed	No Issue
8	canSell	read	Passed	No Issue
9	sell	write	Passed	No Issue
10	getPrice	read	Passed	No Issue
11	canBuy	read	Passed	No Issue
12	buy	write	Passed	No Issue
13	canAuction	read	Passed	No Issue

14	createAuction	write	Passed	No Issue
15	canBid	read	Passed	No Issue
16	bid	write	Bidding can be frozen	Fixed in the revised contract
17	canWithdraw	read	Passed	No Issue
18	withdraw	write	Passed	No Issue
19	canFinalize	read	Passed	No Issue
20	auctionFinalize	write	Passed	No Issue
21	highestBidder	read	Passed	No Issue
22	highestBid	read	Passed	No Issue
23	callOptionalReturn	write	Passed	No Issue
24	updateAdmin	write	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

Two critical vulnerabilities were found and fixed. WP Smart Contracts team desires to keep the bug details confidential, and thus are not revealed here. But we confirmed that those bugs were fixed in the revised contracts code.

High Severity

No high severity vulnerabilities were found.

Medium

No medium severity vulnerabilities were found.

Low

(1) No fractional commission amount possible - Suika and Matcha smart contracts

```
// calculate amounts
uint256 amount4admin = sellBidPrice[tokenId].mul(commissionRate).div(100);
uint256 amount4creator = sellBidPrice[tokenId].mul(royaltiesCommissionRate).div(100);
uint256 amount4owner = sellBidPrice[tokenId].sub(amount4admin).sub(amount4creator);
```

The commission for owner and creators can only be in whole amount and not in fraction. For example, it can only be 1,2,3,etc. It can not be 1.5% or other fractional value.

Resolution: If this is required logic, then this point can be safely ignored. On another hand, the commission value can be used after multiplying with 100 or any decimal amount. So, the owner can have the option to set the percentage in fraction if desired.

Status: Acknowledged

(2) Users may not gain the interest - Ube Smart Contract

```
// check that the owner can pay interest before trying to pay
if (asset.allowance(getOwner(), address(this)) >= _interest && asset.balanceOf(getOwner())
    asset.transferFrom(getOwner(), msg.sender, _interest);
} else {
    _interest = 0;
}
```

In case, the owner does not provide enough allowance, or he does not keep enough token balance into the owner wallet, then users will not receive any interest reward. This is a human factor, so it reduces the decentralization.

Resolution: The owner needs to acknowledge that he will provide enough allowance as well as keep enough balance so that users can receive their interest benefits. On another hand, to make this more trustless, enough tokens can be deposited in the contract for the purpose of interest payment.

Status: Acknowledged

(3) SafeMath is used - All 4 smart contracts

```
using SafeMath for uint256;
```

Solidity version above 0.8.0 has in-built integer overflow/underflow protection. So, it is recommended to avoid using safemath.

Resolution: We suggest avoiding safemath when the solidity version is over 0.8.0. This saves some gas as well.

Status: Fixed

(4) Older solidity version used - Matcha smart contract

Compiler Version	v0.5.7+commit.6da8b019

It is advisable to use the latest solidity version, as many security bugs are fixed in the latest version.

Status: Fixed

Very Low / Informational / Best practices:

(1) Input validations can be helpful - Suika and Matcha smart contracts

```
// update contract fields
function updateAdmin(address payable _admin, uint256 _commissionRate, uint256 _royaltiesCommissionRate,
    require(msg.sender==contract_owner, "Only contract owner can do this");
    admin = _admin;
    commissionRate = _commissionRate;
    royaltiesCommissionRate = _royaltiesCommissionRate;
    anyoneCanMint = _anyoneCanMint;
    payment_token = _payment_token;
}
```

The owner can set commission percentages. If the wrong amount has been set by mistake, then it creates discrepancy in the formula.

Resolution: We suggest adding a condition which specifies the expected percentage variable. This will make sure that the input params will be expected ones. On another hand, this can be acknowledged by the owner that he will make sure the correct amount before setting those values.

Status: Fixed

(2) Function suggestion - Ube smart contract

It is helpful to make a view function which outputs if a particular stake is matured or not. This will be helpful while unstaking, to make sure the premature staking is not withdrawn. This is a “nice to have” feature. And it will not create any issues if that is not present.

Status: Acknowledged

(3) Consider using ‘external’ visibility instead of ‘public’ - All 4 smart contracts

Although this is not a big problem, it is recommended to use the visibility ‘external’ over ‘public’. It saves some gas as well.

https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices

Status: Fixed

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:

- changeOwner - Stakes contract owner can change the owner.
- set: Stakes contract owner can set lower amount, maturity value, rate, penalization values.
- updateAdmin in ERC721Suika: The owner can change the commission percentages, payment token, etc.
- autoMint, mint, safeMint in ERC721Suika and ERC721Matcha: The minter can mint tokens as needed.
- grantRole in ERC721Suika: Any new role can be granted.
- revokeRole in ERC721Suika: The owner can revoke a particular role.
- renounceRole in ERC721Suika: The role can be given up completely.
- set: StakesAlmond owner can set values like: ratio1, ratio2, lower amount, maturity rate, interest rate, penalization, etc

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of files. And we have used all possible tests based on given objects as files. We had observed some issues in the smart contracts. And those issues are fixed / acknowledged in the revised contract code. **So, the smart contracts are ready for the mainnet deployment.**

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.

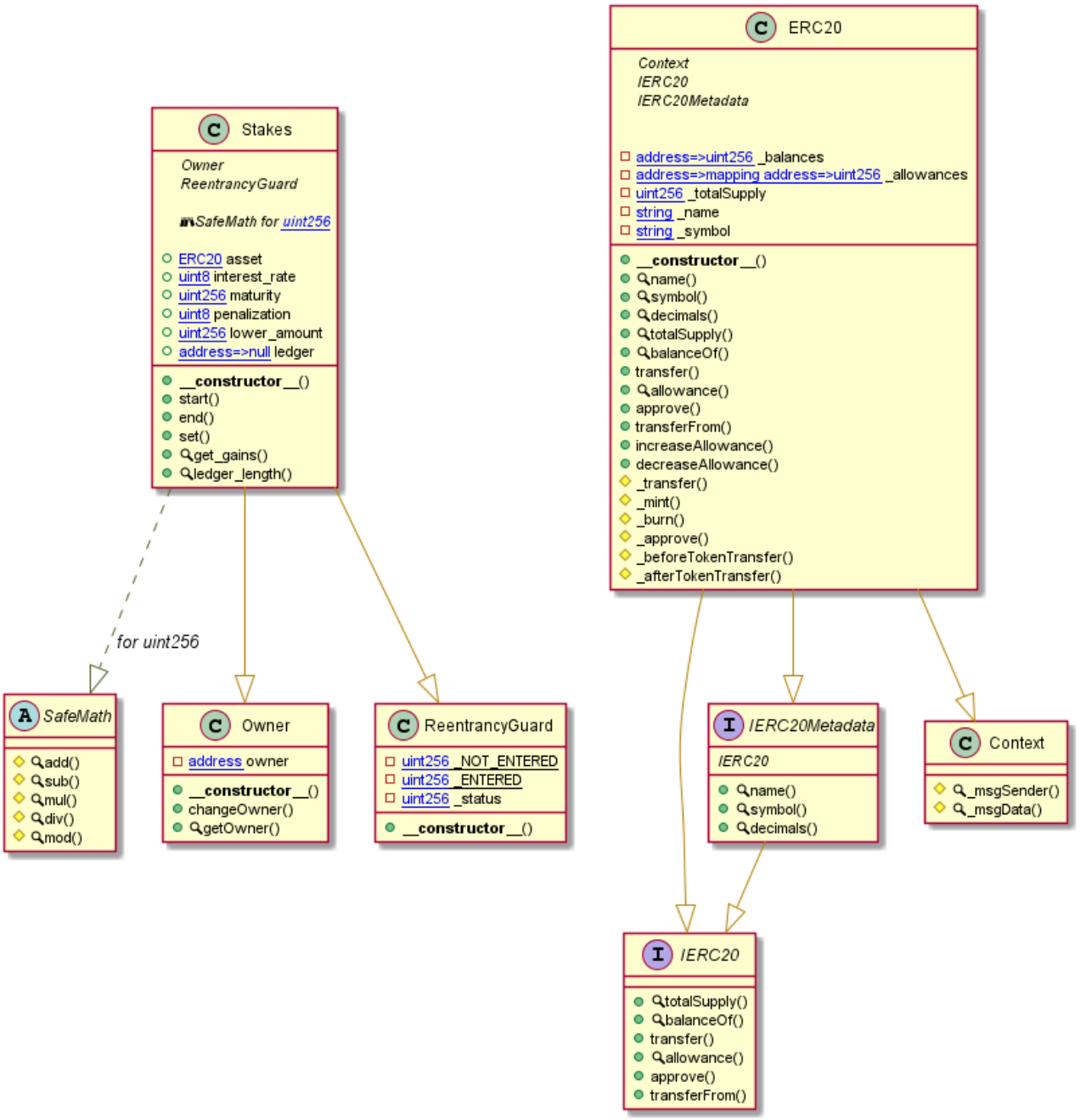
Technical Disclaimer

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.

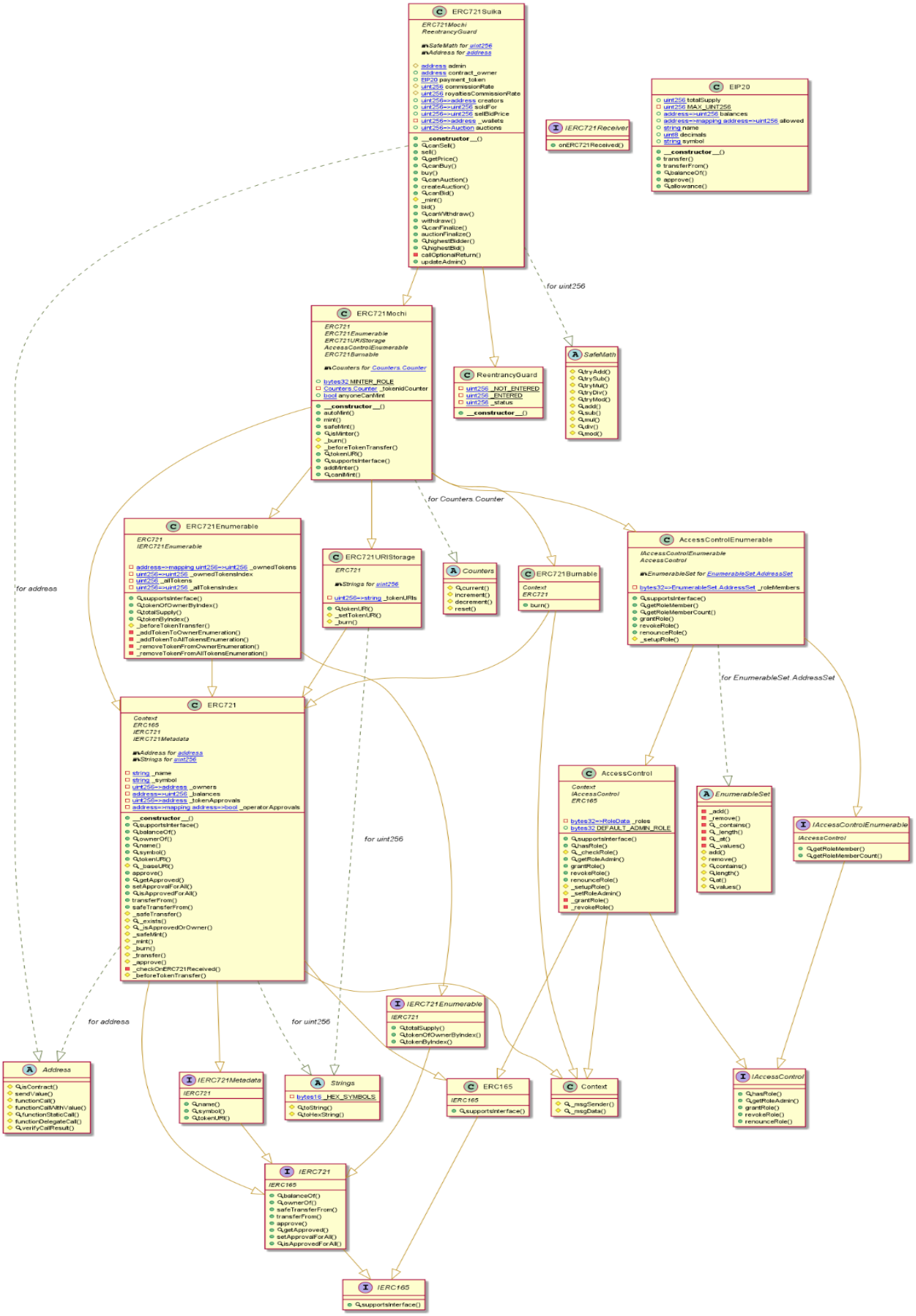
Appendix

Code Flow Diagram - WP Smart Contracts Protocol

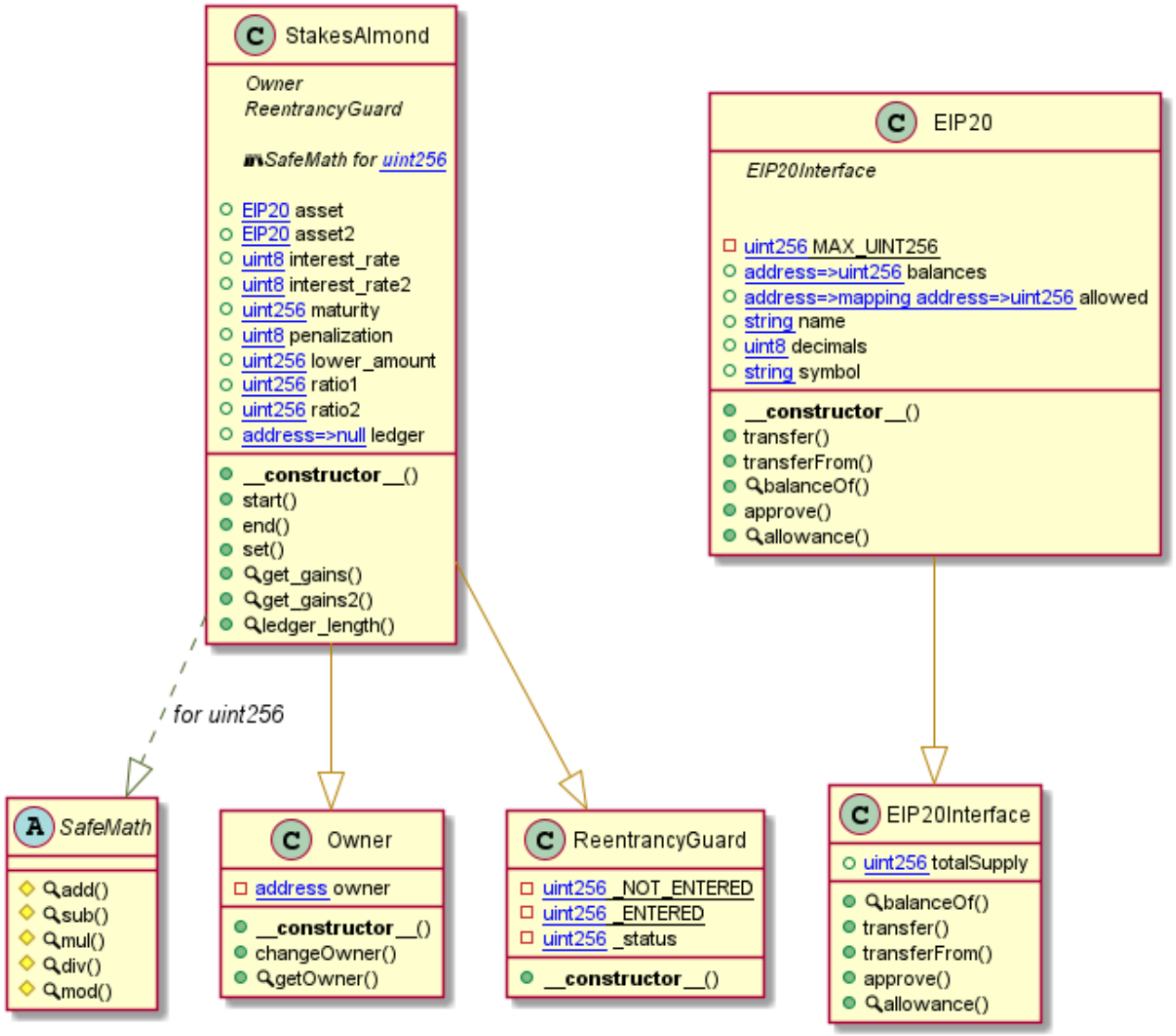
Stakes Diagram



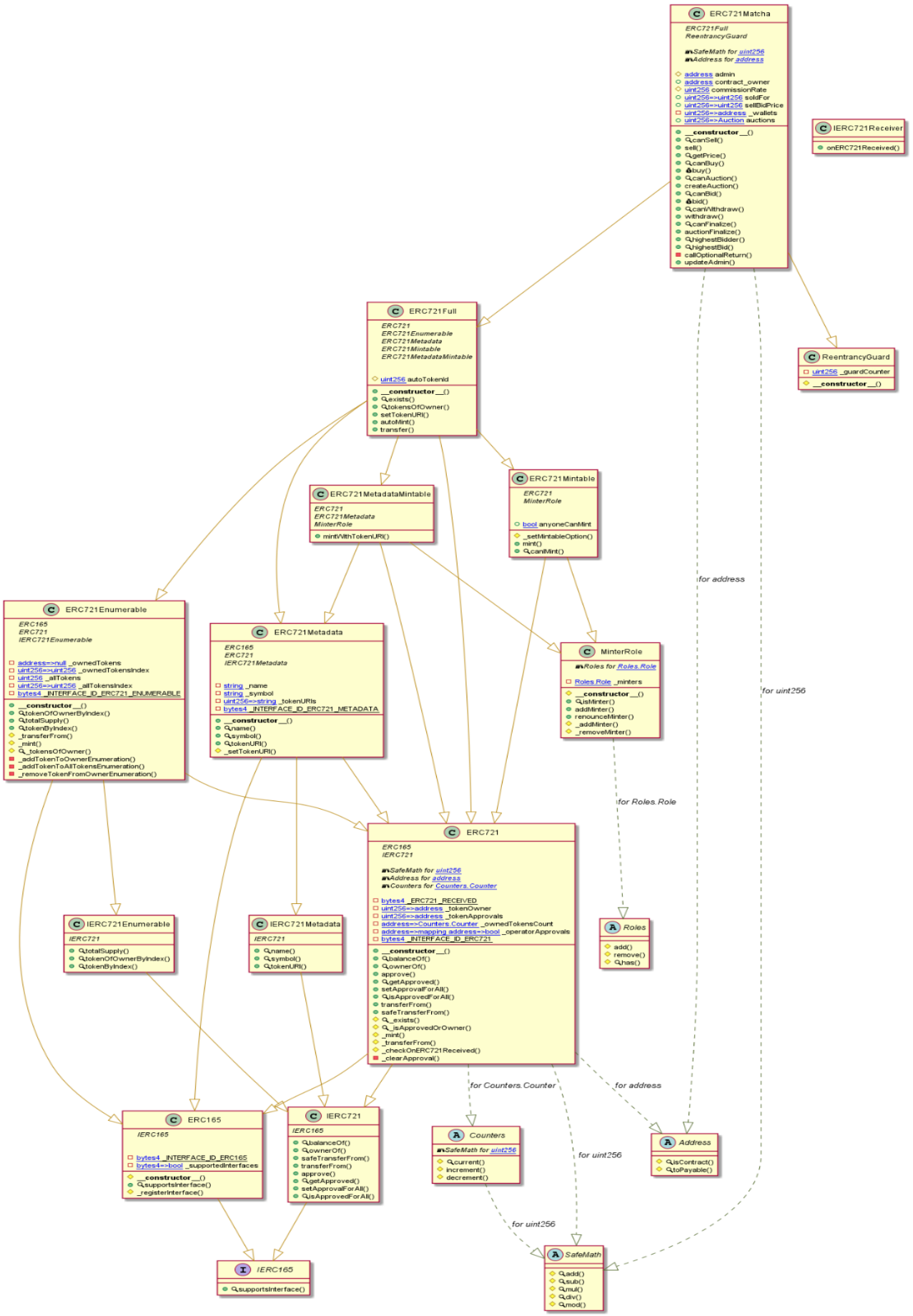
ERC721Suika Diagram



StakesAlmond Diagram



ERC721Matcha Diagram



Slither Results Log

```
INFO:Detectors:
Owner.constructor(address)._owner (Stakes.sol#179) lacks a zero-check on :
                - owner = _owner (Stakes.sol#180)
Owner.changeOwner(address).newOwner (Stakes.sol#188) lacks a zero-check on :
                - owner = newOwner (Stakes.sol#190)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in Stakes.start(uint256) (Stakes.sol#777-782):
        External calls:
        - asset.transferFrom(msg.sender,address(this),_value) (Stakes.sol#779)
        State variables written after the call(s):
        - ledger[msg.sender].push(Record(block.timestamp,_value,0,0,0,false)) (Stakes.sol#780)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in Stakes.end(uint256) (Stakes.sol#784-813):
        External calls:
        - asset.transfer(msg.sender,ledger[msg.sender][i].amount.sub(_penalization)) (Stakes.sol#792)
        - asset.transfer(getOwner(),_penalization) (Stakes.sol#793)
        Event emitted after the call(s):
        - StakeEnd(msg.sender,ledger[msg.sender][i].amount,_penalization,0,i) (Stakes.sol#797)
Reentrancy in Stakes.end(uint256) (Stakes.sol#784-813):
        External calls:
        - asset.transferFrom(getOwner(),msg.sender,_interest) (Stakes.sol#803)
        - asset.transfer(msg.sender,ledger[msg.sender][i].amount) (Stakes.sol#807)
        Event emitted after the call(s):
        - StakeEnd(msg.sender,ledger[msg.sender][i].amount,0,_interest,i) (Stakes.sol#811)
Reentrancy in Stakes.start(uint256) (Stakes.sol#777-782):
        External calls:
        - asset.transferFrom(msg.sender,address(this),_value) (Stakes.sol#779)
        Event emitted after the call(s):
        - StakeStart(msg.sender,_value,ledger[msg.sender].length - 1) (Stakes.sol#781)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

```
INFO:Detectors:
Stakes.end(uint256) (Stakes.sol#784-813) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp.sub(ledger[msg.sender][i].from) < maturity (Stakes.sol#790)
        - asset.allowance(getOwner(),address(this)) >= _interest && asset.balanceOf(getOwner()) >= _interest (Stakes.sol#802)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Stakes.end(uint256) (Stakes.sol#784-813) compares to a boolean constant:
        -require(bool,string)(ledger[msg.sender][i].ended == false,Invalid stake) (Stakes.sol#787)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Context._msgData() (Stakes.sol#376-378) is never used and should be removed
ERC20._burn(address,uint256) (Stakes.sol#646-661) is never used and should be removed
ERC20._mint(address,uint256) (Stakes.sol#623-633) is never used and should be removed
SafeMath.add(uint256,uint256) (Stakes.sol#28-33) is never used and should be removed
SafeMath.mod(uint256,uint256) (Stakes.sol#133-135) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (Stakes.sol#148-151) 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 (Stakes.sol#3) 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:
Parameter Stakes.start(uint256)._value (Stakes.sol#777) is not in mixedCase
Parameter Stakes.set(uint256,uint256,uint8,uint8)._lower (Stakes.sol#815) is not in mixedCase
Parameter Stakes.set(uint256,uint256,uint8,uint8)._maturity (Stakes.sol#815) is not in mixedCase
Parameter Stakes.set(uint256,uint256,uint8,uint8)._rate (Stakes.sol#815) is not in mixedCase
Parameter Stakes.set(uint256,uint256,uint8,uint8)._penalization (Stakes.sol#815) is not in mixedCase
Function Stakes.get_gains(address,uint256) (Stakes.sol#824-830) is not in mixedCase
Parameter Stakes.get_gains(address,uint256)._address (Stakes.sol#824) is not in mixedCase
```

```
INFO:Detectors:
changeOwner(address) should be declared external:
        - Owner.changeOwner(address) (Stakes.sol#188-191)
name() should be declared external:
        - ERC20.name() (Stakes.sol#433-435)
symbol() should be declared external:
        - ERC20.symbol() (Stakes.sol#441-443)
decimals() should be declared external:
        - ERC20.decimals() (Stakes.sol#458-460)
totalSupply() should be declared external:
        - ERC20.totalSupply() (Stakes.sol#465-467)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (Stakes.sol#472-474)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (Stakes.sol#484-487)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (Stakes.sol#492-494)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (Stakes.sol#503-506)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (Stakes.sol#521-535)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (Stakes.sol#549-552)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (Stakes.sol#568-576)
set(uint256,uint256,uint8,uint8) should be declared external:
        - Stakes.set(uint256,uint256,uint8,uint8) (Stakes.sol#815-821)
ledger_length(address) should be declared external:
        - Stakes.ledger_length(address) (Stakes.sol#832-834)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:Stakes.sol analyzed (8 contracts with 75 detectors), 51 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

```
INFO:Detectors:
ERC721Mochi.constructor(address,string,string,bool).name (ERC721Suika.sol#1959) shadows:
        - ERC721.name() (ERC721Suika.sol#582-584) (function)
        - IERC721Metadata.name() (ERC721Suika.sol#185) (function)
ERC721Mochi.constructor(address,string,string,bool).symbol (ERC721Suika.sol#1959) shadows:
        - ERC721.symbol() (ERC721Suika.sol#589-591) (function)
        - IERC721Metadata.symbol() (ERC721Suika.sol#190) (function)
ERC721Suika.constructor(EIP20,address,address,uint256,uint256,string,string,bool).name (ERC721Suika.sol#2470) shadows:
        - ERC721.name() (ERC721Suika.sol#582-584) (function)
        - IERC721Metadata.name() (ERC721Suika.sol#185) (function)
ERC721Suika.constructor(EIP20,address,address,uint256,uint256,string,string,bool).symbol (ERC721Suika.sol#2470) shadows:
        - ERC721.symbol() (ERC721Suika.sol#589-591) (function)
        - IERC721Metadata.symbol() (ERC721Suika.sol#190) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
ERC721Suika.updateAdmin(address,uint256,uint256,bool,EIP20) (ERC721Suika.sol#2798-2805) should emit an event for:
        - commissionRate = _commissionRate (ERC721Suika.sol#2801)
        - royaltiesCommissionRate = _royaltiesCommissionRate (ERC721Suika.sol#2802)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
ERC721Suika.constructor(EIP20,address,address,uint256,uint256,string,string,bool)._admin (ERC721Suika.sol#2469) lacks a zero-ch
eck on :
                - admin = _admin (ERC721Suika.sol#2473)
ERC721Suika.constructor(EIP20,address,address,uint256,uint256,string,string,bool)._owner (ERC721Suika.sol#2469) lacks a zero-ch
eck on :
                - contract_owner = _owner (ERC721Suika.sol#2474)
ERC721Suika.updateAdmin(address,uint256,uint256,bool,EIP20)._admin (ERC721Suika.sol#2798) lacks a zero-check on :
                - admin = _admin (ERC721Suika.sol#2800)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

```
ceived(address,address,uint256,bytes) (ERC721Suika.sol#873-894) potentially used before declaration: reason.length == 0 (ERC721
Suika.sol#883)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (ERC721Suika.sol#882)' in ERC721._checkOnERC721Re
ceived(address,address,uint256,bytes) (ERC721Suika.sol#873-894) potentially used before declaration: revert(uint256,uint256)(32
 + reason,mload(uint256)(reason)) (ERC721Suika.sol#887)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables
INFO:Detectors:
Reentrancy in ERC721Suika.auctionFinalize(uint256) (ERC721Suika.sol#2712-2759):
        External calls:
        - require(bool,string)(payment_token.transfer(auctions[tokenId].beneficiary,amount4owner),Transfer failed.) (ERC721Suik
a.sol#2727)
        - require(bool,string)(payment_token.transfer(creators[tokenId],amount4creator),Transfer failed.) (ERC721Suika.sol#2731
)
        - require(bool,string)(payment_token.transfer(admin,amount4admin),Transfer failed.) (ERC721Suika.sol#2736)
        - callOptionalReturn(this,abi.encodeWithSelector(this.transferFrom.selector,owner,_highestBidder,tokenId)) (ERC721Suika
.sol#2748)
                - (success,returndata) = address(token).call(data) (ERC721Suika.sol#2788)
        State variables written after the call(s):
        - soldFor[tokenId] = auctions[tokenId].highestBid (ERC721Suika.sol#2750)
Reentrancy in ERC721Suika.buy(uint256) (ERC721Suika.sol#2527-2568):
        External calls:
        - callOptionalReturn(this,abi.encodeWithSelector(this.transferFrom.selector,owner,msg.sender,tokenId)) (ERC721Suika.sol
#2538)
                - (success,returndata) = address(token).call(data) (ERC721Suika.sol#2788)
        - require(bool,string)(payment_token.transferFrom(msg.sender,_wallets[tokenId],amount4owner),Transfer failed.) (ERC721S
uika.sol#2546)
        - require(bool,string)(payment_token.transferFrom(msg.sender,creators[tokenId],amount4creator),Transfer failed.) (ERC72
1Suika.sol#2550)
        - require(bool,string)(payment_token.transferFrom(msg.sender,admin,amount4admin),Transfer failed.) (ERC721Suika.sol#255
5)
        State variables written after the call(s):
        - soldFor[tokenId] = sellBidPrice[tokenId] (ERC721Suika.sol#2562)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in ERC721Suika.auctionFinalize(uint256) (ERC721Suika.sol#2712-2759):
        External calls:
        - require(bool,string)(payment_token.transfer(auctions[tokenId].beneficiary,amount4owner),Transfer failed.) (ERC721Suik
```

```
        Event emitted after the call(s):
        - Commission(tokenId,owner,sellBidPrice[tokenId],commissionRate,amount4admin) (ERC721Suika.sol#2559)
        - Royalty(tokenId,owner,sellBidPrice[tokenId],royaltiesCommissionRate,amount4creator) (ERC721Suika.sol#2560)
        - Sale(tokenId,owner,msg.sender,sellBidPrice[tokenId]) (ERC721Suika.sol#2558)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
ERC721Suika.canBid(uint256) (ERC721Suika.sol#2591-2602) uses timestamp for comparisons
        Dangerous comparisons:
        - ! msg.sender.isContract() && auctions[tokenId].open && block.timestamp <= auctions[tokenId].auctionEnd && msg.sender
!= ownerOf(tokenId) && getApproved(tokenId) == address(this) (ERC721Suika.sol#2592-2596)
ERC721Suika.bid(uint256,uint256) (ERC721Suika.sol#2614-2659) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= auctions[tokenId].auctionEnd,Auction already ended.) (ERC721Suika.sol#2627-26
30)
ERC721Suika.canWithdraw(uint256) (ERC721Suika.sol#2665-2680) uses timestamp for comparisons
        Dangerous comparisons:
        - auctions[tokenId].open && ((block.timestamp >= auctions[tokenId].auctionEnd && auctions[tokenId].highestBid > 0 && au
ctions[tokenId].highestBid < auctions[tokenId].reserve) || getApproved(tokenId) != address(this)) (ERC721Suika.sol#2666-2674)
ERC721Suika.canFinalize(uint256) (ERC721Suika.sol#2697-2709) uses timestamp for comparisons
        Dangerous comparisons:
        - auctions[tokenId].open && block.timestamp >= auctions[tokenId].auctionEnd && (auctions[tokenId].highestBid >= auction
s[tokenId].reserve || auctions[tokenId].highestBid == 0) (ERC721Suika.sol#2698-2703)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (ERC721Suika.sol#219-229) uses assembly
        - INLINE ASM (ERC721Suika.sol#225-227)
Address.verifyCallResult(bool,bytes,string) (ERC721Suika.sol#388-408) uses assembly
        - INLINE ASM (ERC721Suika.sol#400-403)
ERC721._checkOnERC721Received(address,address,uint256,bytes) (ERC721Suika.sol#873-894) uses assembly
        - INLINE ASM (ERC721Suika.sol#886-888)
EnumerableSet.values(EnumerableSet.AddressSet) (ERC721Suika.sol#1755-1764) uses assembly
        - INLINE ASM (ERC721Suika.sol#1759-1761)
EnumerableSet.values(EnumerableSet.UintSet) (ERC721Suika.sol#1828-1837) uses assembly
        - INLINE ASM (ERC721Suika.sol#1832-1834)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

Solidity Static Analysis

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Stakes.start(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 777:4:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 790:11:

Gas costs:

Gas requirement of function Stakes.set 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: 815:4:

Gas costs:

Gas requirement of function Stakes.get_gains 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: 824:4:

Miscellaneous

Constant/View/Pure functions:

SafeMath.sub(uint256,uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 44:4:

Constant/View/Pure functions:

ERC20._afterTokenTransfer(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 722:4:

Similar variable names:

Stakes.(contract ERC20,address,uint8,uint256,uint8,uint256) : Variables have very similar names "_owner" and "_lower". Note: Modifiers are currently not considered by this static analysis.
Pos: 774:23:

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: 787:8:

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: 816:8:

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Address.functionCallWithValue(address,bytes,uint256,string): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 315:4:

Gas costs:

Gas requirement of function ERC721Suika.withdraw 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: 2683:4:

Solhint Linter

Ube - Stakes.sol

```
Stakes.sol:530:18: Error: Parse error: missing ';' at '{'
Stakes.sol:571:18: Error: Parse error: missing ';' at '{'
Stakes.sol:604:18: Error: Parse error: missing ';' at '{'
Stakes.sol:653:18: Error: Parse error: missing ';' at '{'
```

ERC721Suika.sol

```
ERC721Suika.sol:1933:18: Error: Parse error: missing ';' at '{'
ERC721Suika.sol:1942:18: Error: Parse error: missing ';' at '{'
ERC721Suika.sol:2118:18: Error: Parse error: missing ';' at '{'
ERC721Suika.sol:2131:18: Error: Parse error: missing ';' at '{'
ERC721Suika.sol:2143:18: Error: Parse error: missing ';' at '{'
ERC721Suika.sol:2160:18: Error: Parse error: missing ';' at '{'
ERC721Suika.sol:2172:18: Error: Parse error: missing ';' at '{'
ERC721Suika.sol:2268:18: Error: Parse error: missing ';' at '{'
ERC721Suika.sol:2291:18: Error: Parse error: missing ';' at '{'
ERC721Suika.sol:2317:18: Error: Parse error: missing ';' at '{'
```

StakesAlmond.sol

```
StakesAlmond.sol:60:1: Error: Compiler version ^0.8.0 does not
satisfy the r semver requirement
StakesAlmond.sol:237:5: Error: Explicitly mark visibility in function
(Set ignoreConstructors to true if using solidity >=0.7.0)
StakesAlmond.sol:293:5: Error: Explicitly mark visibility in function
(Set ignoreConstructors to true if using solidity >=0.7.0)
StakesAlmond.sol:389:5: Error: Explicitly mark visibility in function
(Set ignoreConstructors to true if using solidity >=0.7.0)
StakesAlmond.sol:411:9: Error: Variable name must be in mixedCase
StakesAlmond.sol:472:18: Error: Variable name must be in mixedCase
StakesAlmond.sol:473:18: Error: Variable name must be in mixedCase
StakesAlmond.sol:476:20: Error: Variable name must be in mixedCase
StakesAlmond.sol:490:5: Error: Explicitly mark visibility in function
(Set ignoreConstructors to true if using solidity >=0.7.0)
StakesAlmond.sol:491:23: Error: Variable name must be in mixedCase
StakesAlmond.sol:508:40: Error: Avoid to make time-based decisions in
your business logic
StakesAlmond.sol:518:12: Error: Avoid to make time-based decisions in
your business logic
StakesAlmond.sol:523:13: Error: Possible reentrancy vulnerabilities.
Avoid state changes after transfer.
StakesAlmond.sol:524:13: Error: Possible reentrancy vulnerabilities.
Avoid state changes after transfer.
StakesAlmond.sol:524:40: Error: Avoid to make time-based decisions in
```

```
your business logic
StakesAlmond.sol:525:13: Error: Possible reentrancy vulnerabilities.
Avoid state changes after transfer.
StakesAlmond.sol:553:13: Error: Possible reentrancy vulnerabilities.
Avoid state changes after transfer.
StakesAlmond.sol:554:13: Error: Possible reentrancy vulnerabilities.
Avoid state changes after transfer.
StakesAlmond.sol:555:13: Error: Possible reentrancy vulnerabilities.
Avoid state changes after transfer.
StakesAlmond.sol:555:40: Error: Avoid to make time-based decisions in
your business logic
StakesAlmond.sol:556:13: Error: Possible reentrancy vulnerabilities.
Avoid state changes after transfer.
StakesAlmond.sol:562:32: Error: Variable name must be in mixedCase
StakesAlmond.sol:576:5: Error: Function name must be in mixedCase
StakesAlmond.sol:576:42: Error: Variable name must be in mixedCase
StakesAlmond.sol:577:9: Error: Variable name must be in mixedCase
StakesAlmond.sol:577:35: Error: Avoid to make time-based decisions in
your business logic
StakesAlmond.sol:587:9: Error: Variable name must be in mixedCase
StakesAlmond.sol:590:9: Error: Variable name must be in mixedCase
StakesAlmond.sol:600:5: Error: Function name must be in mixedCase
```

ERC721Matcha.sol

```
ERC721Matcha.sol:63:1: Error: Compiler version ^0.5.7 does not
satisfy the r semver requirement
ERC721Matcha.sol:1159:5: Error: Explicitly mark visibility of state
ERC721Matcha.sol:1258:5: Error: Explicitly mark visibility of state
ERC721Matcha.sol:1260:20: Error: Variable name must be in mixedCase
ERC721Matcha.sol:1263:5: Error: Explicitly mark visibility of state
ERC721Matcha.sol:1397:28: Error: Avoid to use ".call.value()()"
ERC721Matcha.sol:1397:28: Error: Avoid using low level calls.
ERC721Matcha.sol:1401:29: Error: Avoid to use ".call.value()()"
ERC721Matcha.sol:1401:29: Error: Avoid using low level calls.
ERC721Matcha.sol:1439:13: Error: Avoid to make time-based decisions
in your business logic
ERC721Matcha.sol:1472:13: Error: Avoid to make time-based decisions
in your business logic
ERC721Matcha.sol:1528:32: Error: Avoid to use ".call.value()()"
ERC721Matcha.sol:1528:32: Error: Avoid using low level calls.
ERC721Matcha.sol:1539:13: Error: Avoid to make time-based decisions
in your business logic
ERC721Matcha.sol:1566:32: Error: Avoid to use ".call.value()()"
ERC721Matcha.sol:1566:32: Error: Avoid using low level calls.
ERC721Matcha.sol:1570:33: Error: Avoid to use ".call.value()()"
ERC721Matcha.sol:1570:33: Error: Avoid using low level calls.
```

Software analysis result:

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

Email: audit@EtherAuthority.io