SKELETON ECOSYSTEM

SMART CONTRACT AUDIT

Project X [$OXO]

ERC 20

0xe4F1BDF9E4f37F7DB5045129D983f005AaEd7AEA

Table of Contents

Global Disclaimer

This document serves as a disclaimer for the crypto smart contract audit conducted by Skeleton Ecosystem. The purpose of the audit was to review the codebase of the smart contracts for potential vulnerabilities and issues. It is important to note the following:

Limited Scope: The audit is based on the code and information available up to the audit completion date. It does not cover external factors, system interactions, or changes made after the audit. The audit itself can not guarantee 100% safaty and can not detect common scam methods like farming and developer sell-out.

No Guarantee of Security: While we have taken reasonable steps to identify vulnerabilities, it is impossible to guarantee the complete absence of security risks or issues. The audit report provides an assessment of the contract’s security as of the audit date.

Continued Development: Smart contracts and blockchain technology are evolving fields. Updates, forks, or changes to the contract post-audit may introduce new risks that were not present during the audit.

Third-party Code: If the smart contract relies on third-party libraries or code, those components were not thoroughly audited unless explicitly stated. Security of these dependencies is the responsibility of their respective developers.

Non-Exhaustive Testing: The audit involved automated analysis, manual review, and testing under controlled conditions. It is possible that certain vulnerabilities or issues may not have been identified.

Risk Evaluation: The audit report includes a risk assessment for identified vulnerabilities. It is recommended that the development team carefully reviews and addresses these risks to mitigate potential exploits.

Not Financial Advice: This audit report is not intended as financial or investment advice. Decisions regarding the use, deployment, or investment in the smart contract should be made based on a comprehensive assessment of the associated risks.

By accessing and using this audit report, you acknowledge and agree to the limitations outlined above. Skeleton Ecosystem and its auditors shall not be held liable for any direct or indirect damages resulting from the use of the audit report or the smart contract itself.

Please consult with legal, technical, and financial professionals before making any decisions related to the smart contract.

Overview

Contract Name	ProjectXPolygon
Ticker/Simbol	0x0
Blockchain	Polygon Chain ERC20
Contract Address	0xe4F1BDF9E4f37F7DB5045129D983f005AaEd7AEA
Creator Address	0xb3b6F71A72a47A6EE7deF98381c1035cB1187B82
Current Owner Address	0x9ba8648ff0b7ebbf584b879c9ad81977f27d55df
Contract Explorer	https://polygonscan.com/token/0xe4F1BDF9E4f37F7DB5045129D983f005AaEd7AEA
Compiler Version	v0.8.18+commit.87f61d96
License	MIT
Optimisation	Yes with 500 Runs
Total Supply	10,000,000,000 **0x0**
Decimals	18

Creation/Audit

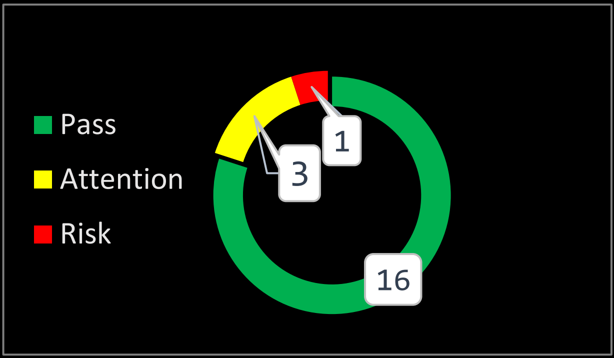
Contract Deployed	17 Sept 2023
Audit Created	03 Oct 2023
Audit Update	V 1.0

Verified Socials

Website	https://0x0me.me
Telegram	https://t.me/project0x0
Twitter (X)	https://x.com/Project_0X0

Contract Function Analysis

✓ Pass ⚠ Attention Item ❗ Risky Item

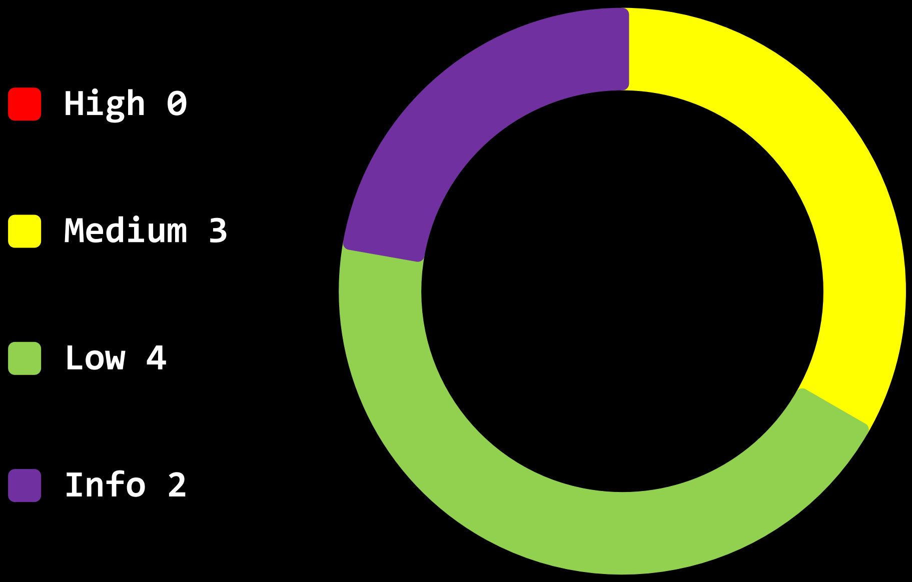


Contract Verified	✓	The contract source code is uploaded to blockchain explorer and is open source, so everybody can read it.
Contract Ownership	⚠	0x9ba8648ff0b7ebbf584b879c9ad81977f27d55df
Buy Tax	10 %	Shows the taxes for purchase transactions. Above 10% may be considered a high tax rate. More than 50% tax rate means may not be tradable.
Sell Tax	10 %	Shows the taxes for sell transactions. Above 10% may be considered a high tax rate. More than 50% tax rate means may not be tradable.
Honeypot Analyse	✓	Holder is able to buy and sell. If honeypot: The contract blocks sell transfer from holder wallet. Multiple events may cause honeypot. Trading disabled, extremely high tax
Liqudity Status		No LP Locker. Presale phase.
Trading Disable Functions	✓	No Trading suspendable function found. If a suspendable code is included, the token maybe neither be bought or sold (honeypot risk). If contract is renounced this function can't be used
Set Fees function	⚠	Fee Setting function found. The contract owner may contain the authority to modify the transaction tax. If the transaction tax is increased to more than 49%, the tokens may not be able to be traded (honeypot risk).
Proxy Contract	✓	Not a Proxy Contract. The proxy contract means contract owner can modifiy the function of the token and possibly effect the price. The Owner is not the creator but the creator may have authorisation to change functions.
Mint Function	✓	No mint Function found Mint function is transparent or non-existent. Hidden mint functions may increase the amount of tokens in circulation and effect the price of the token. Owner can mint new tokens and sell.

Balance Modifier Function	✅	No Balance Modifier function found. If there is a function for this, the contract owner can have the authority to modify the balance of tokens at other addresses. For example revoke the bought tokens from the holders wallet. Common form of scam: You buy the token, but it's disappearing from your wallet.
Blacklist Function	⚠	Blacklist function found If there is a blacklist, some addresses may not be able to trade normally. Example: you buy the token and right after your Wallet getting blacklisted. Like so you will be unable to sell. Honeypot Risk.
Whitelist Function	⚠	Whitelist function found If there is a function for this Developer can set zero fee or no max wallet size for adresses (for example team wallets can trade without fee. Can cause farming)
Hidden Owner Analysis	✅	No authorised hidden owner found. For contract with a hidden owner, developer can still manipulate the contract even if the ownership has been abandoned. Fake renounce.
Retrieve Ownership Function	✅	No functions found which can retrieve ownership of the contract. If this function exists, it is possible for the project owner to regain ownership even after relinquishing it. Also known as fake renounce.
Self Destruct Function	✅	No Self Destruct function found. If this function exists and is triggered, the contract will be destroyed, all functions will be unavailable, and all related assets will be erased.
Specific Tax Changing Function	✅	No Specific Tax Changing Functions found. If it exists, the contract owner may set a very outrageous tax rate for assigned address to block it from trading. Can assign all wallets at once!
Trading Cooldown Function	✅	No Trading Cooldown Function found. If there is a trading cooldown function, the user will not be able to sell the token within a certain time or block after buying. Like a temporary honeypot.
Max Transaction and Holding Modify Function	✅	No Max Transaction and Holding Modify function found If there is a function for this, the maximum trading amount or maximum position can be modified. Can cause honeypot
Transaction Limiting Function	✅	No Transaction Limiter Function Found. The number of overall token transactions may be limited (honeypot risk)

Contract Security

Total Findings: 7



🟥 **High Severity Issues:** High possibility to cause problems, need to be resolved.

🟨 **Medium Severity Issue:** Will likely cause problems, recommended to resolve.

🟩 **Low Severity Issues:** Won’t cause problems, but for improvement purposes could be adjusted.

🟪 **Informational Severity Issues:** Not harmful in any way, information for the developer team.

Contract Security

List of Found Issues

🟥 High severity Issues: (0)

🟨 Medium severity issues: (3)

- Approve of Front Running Attack
- Use of TX. Origin
- Unchecked Array Lenght

🟩 Low severity issues: (4)

- Missing Events
- Floating Pragma
- Long Number Literals
- Numeric Notation Best Practices

🟪 Informational severity issues: (2)

- Variables should be immutable
- Public Functions Should be Declared External

Contract Weakness Classisication

THE SMART CONTRACT WEAKNESS CLASSIFICATION REGISTRY (SWC REGISTRY) IS AN IMPLEMENTATION OF THE WEAKNESS CLASSIFICATION SCHEME PROPOSED IN EIP-1470. IT IS LOOSELY ALIGNED TO THE TERMINOLOGIES AND STRUCTURE USED IN THE COMMON WEAKNESS ENUMERATION (CWE) WHILE OVERLAYING A WIDE RANGE OF WEAKNESS VARIANTS THAT ARE SPECIFIC TO SMART CONTRACTS.

ID	Description	AI	Manual	Result
SWC-100	Function Default Visibility	Passed	Passed	Passed
SWC-101	Integer Overflow and Underflow	Passed	Passed	Passed
SWC-102	Outdated Compiler Version	Passed	Passed	Passed
SWC-103	Floating Pragma	Low	Passed	Passed
SWC-104	Unchecked Call Return Value	Passed	Passed	Passed
SWC-105	Unprotected Ether Withdrawal	Passed	Passed	Passed
SWC-106	Unprotected SELFDESTRUCT Instruction	Passed	Passed	Passed
SWC-107	Reentrancy	Passed	Passed	Passed
SWC-108	State Variable Default Visibility	Passed	Passed	Passed
SWC-109	Uninitialized Storage Pointer	Passed	Passed	Passed
SWC-110	Assert Violation	Passed	Passed	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed	Passed	Passed
SWC-112	Delegatecall to Untrusted Callee	Passed	Passed	Passed
SWC-113	DoS with Failed Call	Passed	Passed	Passed
SWC-114	Transaction Order Dependence	Passed	Passed	Passed
SWC-115	Authorization through tx.origin	High	Medium	Medium
SWC-116	Block values as a proxy for time	Passed	Passed	Passed
SWC-117	Signature Malleability	Passed	Passed	Passed
SWC-118	Incorrect Constructor Name	Passed	Passed	Passed
SWC-119	Shadowing State Variables	Passed	Passed	Passed
SWC-120	Weak Sources of Randomness from Chain Attributes	Passed	Passed	Passed

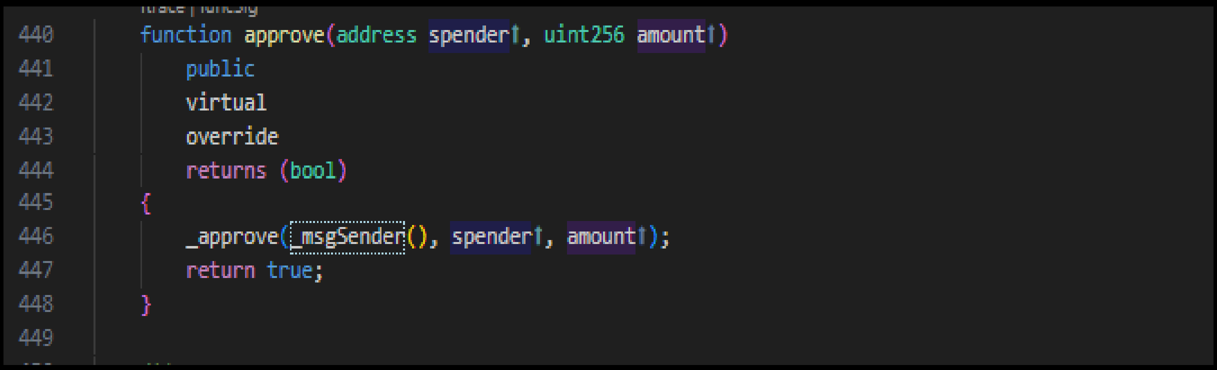
SWC-121	Missing Protection against Signature Replay Attacks	**Passed**	**Passed**	**Passed**
SWC-122	Lack of Proper Signature Verification	**Passed**	**Passed**	**Passed**
SWC-123	Requirement Violation	**Passed**	**Passed**	**Passed**
SWC-124	Write to Arbitrary Storage Location	**Passed**	**Passed**	**Passed**
SWC-125	Incorrect Inheritance Order	**Passed**	**Passed**	**Passed**
SWC-126	Insufficient Gas Griefing	**Passed**	**Passed**	**Passed**
SWC-127	Arbitrary Jump with Function Type Variable	**Passed**	**Passed**	**Passed**
SWC-128	DoS With Block Gas Limit	**Passed**	**Passed**	**Passed**
SWC-129	Typographical Error	**Passed**	**Passed**	**Passed**
SWC-130	Right-To-Left-Override control character (U+202E)	**Passed**	**Passed**	**Passed**
SWC-131	Presence of unused variables	**Passed**	**Passed**	**Passed**
SWC-132	Unexpected Ether balance	**Passed**	**Passed**	**Passed**
SWC-133	Hash Collisions With Multiple Variable Length Arguments	**Passed**	**Passed**	**Passed**
SWC-134	Message call with hardcoded gas amount	**Passed**	**Passed**	**Passed**
SWC-135	Code With No Effects	**Passed**	**Passed**	**Passed**
SWC-136	Unencrypted Private Data On-Chain	**Passed**	**Passed**	**Passed**

Detected High and Medium Severity Vulnerability Description

⚠ Approve of Front Running Attack (2 Items)

Item: 1	Location:	Line 440-448	Severity:	🟨 Medium

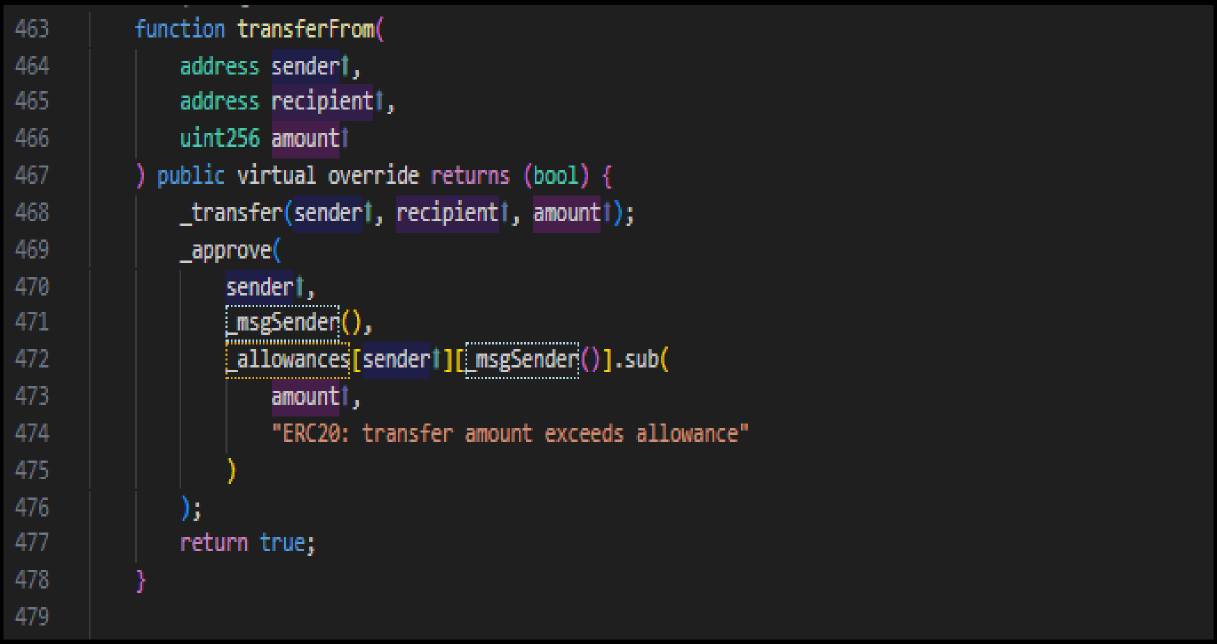
Function	The approve() method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account. This can be abused by a token receiver when they try to withdraw certain tokens from the sender's account. Meanwhile, if the sender decides to change the amount and sends another approve transaction, the receiver can notice this transaction before it's mined and can extract tokens from both the transactions, therefore, ending up with tokens from both the transactions. This is a front-running attack affecting the ERC20 Approve function. The function approve can be front-run by abusing the _approve function.
Remedation	1. Introduce mechanisms that limit the maximum acceptable gas price for transactions. This can help prevent front-runners from drastically increasing the gas fees to prioritize their transactions. 2. Use transaction taxes to prevent against front-run attack



```
440    function approve(address spender, uint256 amount)
441        public
442        virtual
443        override
444        returns (bool)
445    {
446        _approve(_msgSender(), spender, amount);
447        return true;
448    }
449
```

Item: 2	Location:	Line 463-478	Severity:	🟨 Medium

Function	The transferFrom() method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account. This can be abused by a token receiver when they try to withdraw certain tokens from the sender's account. Meanwhile, if the sender decides to change the amount and sends another approve transaction, the receiver can notice this transaction before it's mined and can extract tokens from both the transactions, therefore, ending up with tokens from both the transactions. This is a front-running attack affecting the ERC20 Approve function. The function transferFrom can be front-run by abusing the _approve function.
Remedation	1. Introduce mechanisms that limit the maximum acceptable gas price for transactions. This can help prevent front-runners from drastically increasing the gas fees to prioritize their transactions. 2. Use transaction taxes to prevent against front-run attack

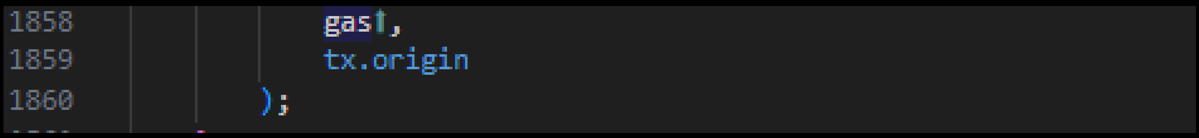


```
463    function transferFrom(
464        address sender,
465        address recipient,
466        uint256 amount
467    ) public virtual override returns (bool) {
468        _transfer(sender, recipient, amount);
469        _approve(
470            sender,
471            _msgSender(),
472            _allowances[sender][_msgSender()].sub(
473                amount,
474                "ERC20: transfer amount exceeds allowance"
475            )
476        );
477        return true;
478    }
479
```

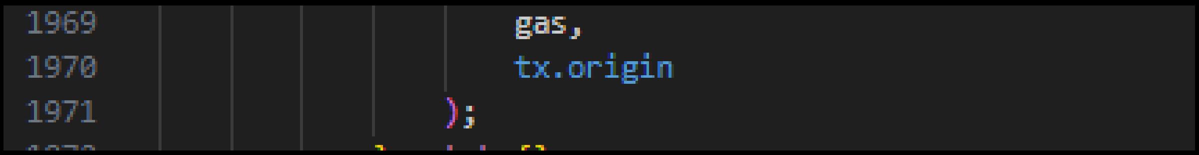
⚠ Use of TX. Origin (2 Items)

Item: 1	Location:	Line 1859	Severity:	🟨 Medium
Item: 2	Location:	Line 1970	Severity:	🟨 Medium

Function	In Solidity, tx.origin is a global variable that returns the address of the account that sent the transaction. Using the variable for authorization could make a contract vulnerable. For example, if an authorized account calls a malicious contract which triggers it to call the vulnerable contract that passes an authorization check since tx.origin returns the original sender of the transaction which in this case is the authorized account.
Remedation	tx.origin should not be used for authorization in smart contracts. It does have some legitimate use cases, for example, To prevent external contracts from calling the current contract, you can implement a require of the form require(tx.origin == msg.sender). This prevents intermediate contracts from calling the current contract, thus limiting the contract to regular codeless addresses.



```
1858                gas↑,
1859                tx.origin
1860            );
```

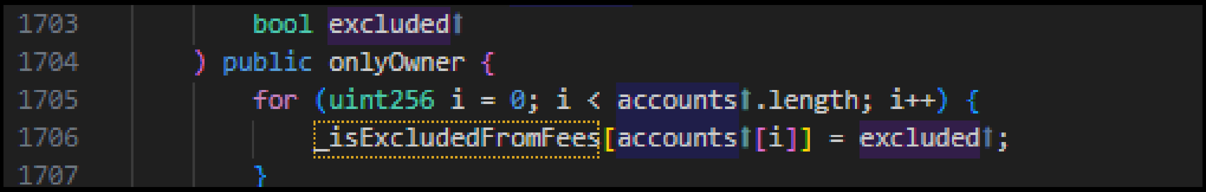


```
1969                    gas,
1970                    tx.origin
1971                );
```

⚠ Unchecked Array Lenght (1 Item)

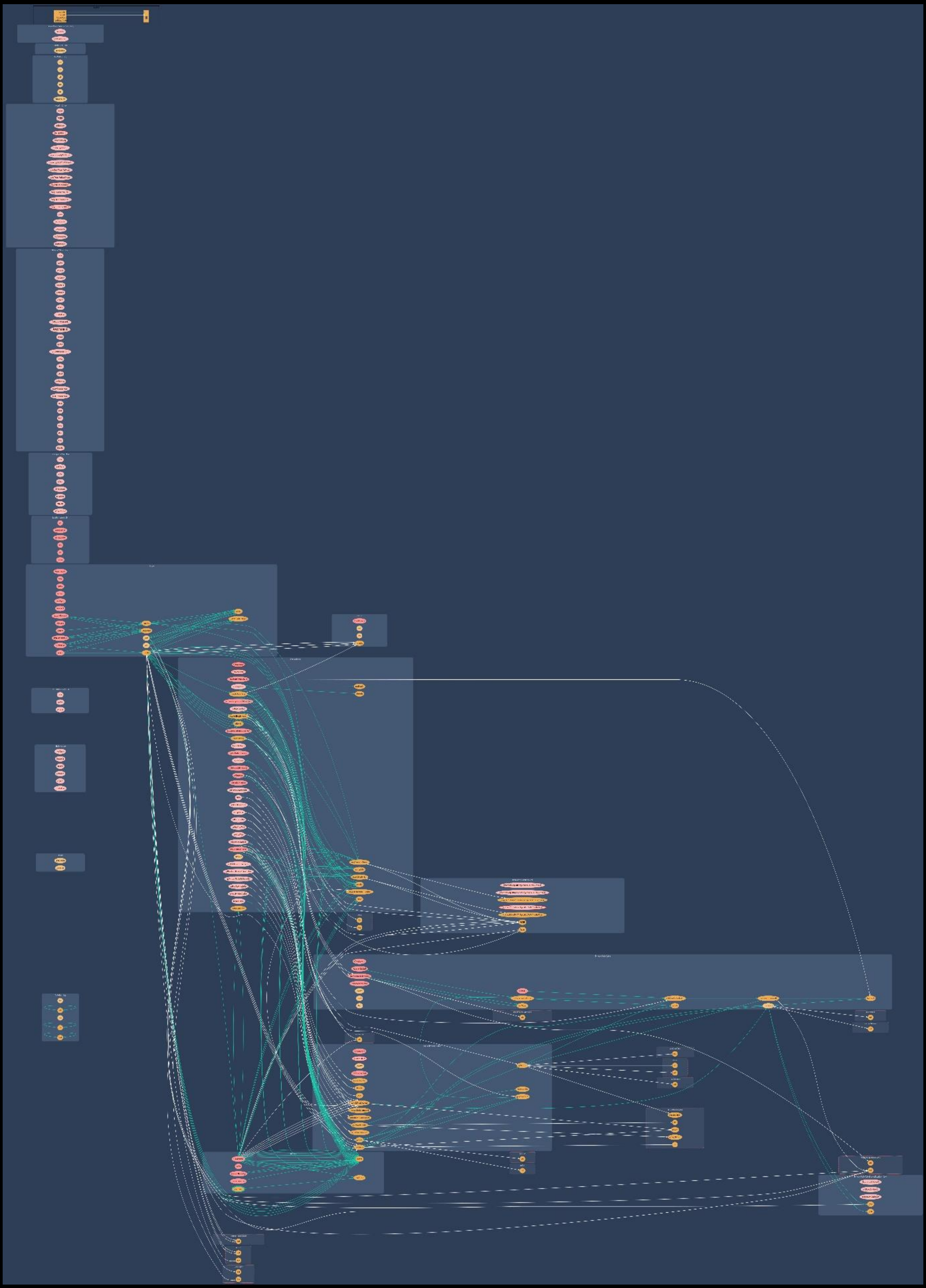
Item: 1	Location:	Line 1705	Severity:	🟨 Medium

Function	Ethereum is a very resource-constrained environment. Prices per computational step are orders of magnitude higher than with centralized providers. Moreover, Ethereum miners impose a limit on the total number of Gas consumed in a block. If array.length is large enough, the function exceeds the block gas limit, and transactions calling it will never be confirmed. for (uint256 i = 0; i < array.length ; i++) { cosltyFunc(); } This becomes a security issue if an external actor influences array.length. E.g., if an array enumerates all registered addresses, an adversary can register many addresses, causing the problem described above.
Remedation	Either explicitly or just due to normal operation, 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. Therefore, loops with a bigger or unknown number of steps should always be avoided.

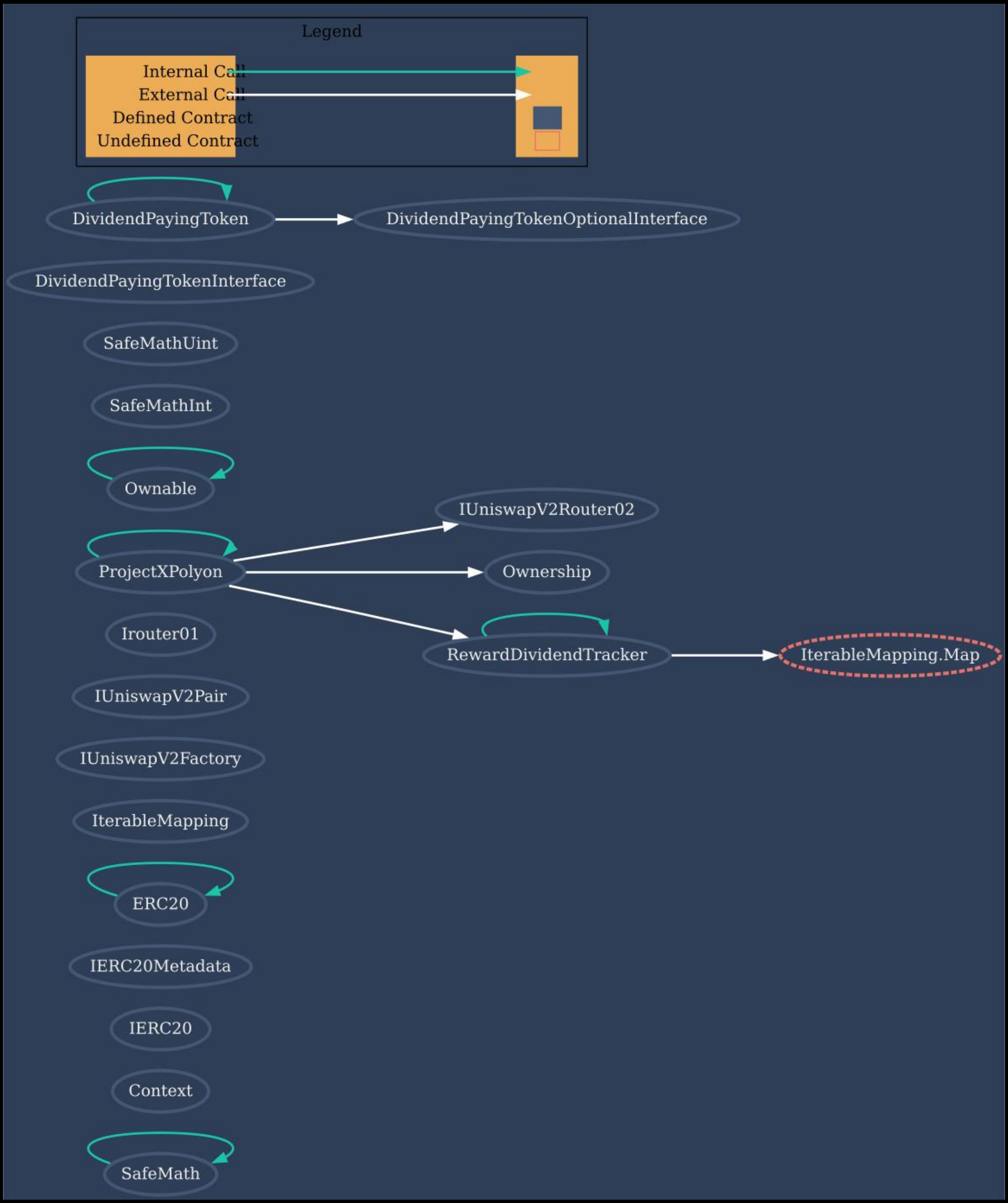


```
1703        bool excluded
1704    ) public onlyOwner {
1705        for (uint256 i = 0; i < accounts.length; i++) {
1706            _isExcludedFromFees[accounts[i]] = excluded;
1707        }
```

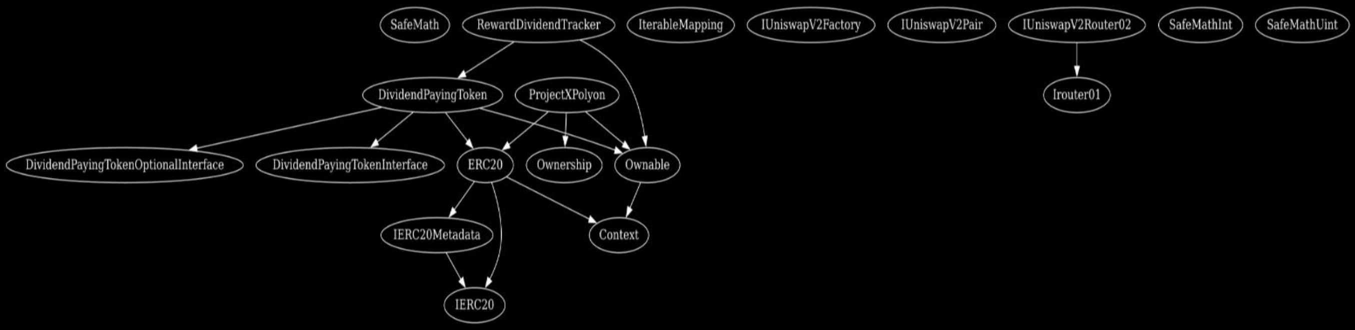
Contract Flow Graph



Interaction Graph



Inheritance Graph



Contract Functions

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
SafeMath	Library			
	add	Internal 🔒		
	sub	Internal 🔒		
	sub	Internal 🔒		
	mul	Internal 🔒		
	div	Internal 🔒		
	div	Internal 🔒		
	mod	Internal 🔒		
	mod	Internal 🔒		
Context	Implementation			
	_msgSender	Internal 🔒		
	_msgData	Internal 🔒		
IERC20	Interface			
	totalSupply	External ❗		NO❗
	balanceOf	External ❗		NO❗
	transfer	External ❗	🛑	NO❗
	allowance	External ❗		NO❗
	approve	External ❗	🛑	NO❗
	transferFrom	External ❗	🛑	NO❗
IERC20Metadata	Interface	IERC20		
	name	External ❗		NO❗
	symbol	External ❗		NO❗
	decimals	External ❗		NO❗
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public ❗	🛑	NO❗

	name	Public ❗		NO❗
	symbol	Public ❗		NO❗
	decimals	Public ❗		NO❗
	totalSupply	Public ❗		NO❗
	balanceOf	Public ❗		NO❗
	transfer	Public ❗	🛑	NO❗
	allowance	Public ❗		NO❗
	approve	Public ❗	🛑	NO❗
	transferFrom	Public ❗	🛑	NO❗
	increaseAllowance	Public ❗	🛑	NO❗
	decreaseAllowance	Public ❗	🛑	NO❗
	_transfer	Internal 🔒	🛑	
	_mint	Internal 🔒	🛑	
	_burn	Internal 🔒	🛑	
	_approve	Internal 🔒	🛑	
	_beforeTokenTransfer	Internal 🔒	🛑	
IterableMapping	Library			
	get	Public ❗		NO❗
	getIndexOfKey	Public ❗		NO❗
	getKeyAtIndex	Public ❗		NO❗
	size	Public ❗		NO❗
	set	Public ❗	🛑	NO❗
	remove	Public ❗	🛑	NO❗
IUniswapV2Factory	Interface			
	feeTo	External ❗		NO❗
	feeToSetter	External ❗		NO❗
	getPair	External ❗		NO❗
	allPairs	External ❗		NO❗
	allPairsLength	External ❗		NO❗
	createPair	External ❗	🛑	NO❗
	setFeeTo	External ❗	🛑	NO❗
	setFeeToSetter	External ❗	🛑	NO❗

IUniswapV2Pair	Interface			
	name	External ❗		NO❗
	symbol	External ❗		NO❗
	decimals	External ❗		NO❗
	totalSupply	External ❗		NO❗
	balanceOf	External ❗		NO❗
	allowance	External ❗		NO❗
	approve	External ❗	🛑	NO❗
	transfer	External ❗	🛑	NO❗
	transferFrom	External ❗	🛑	NO❗
	DOMAIN_SEPARATOR	External ❗		NO❗
	PERMIT_TYPEHASH	External ❗		NO❗
	nonces	External ❗		NO❗
	permit	External ❗	🛑	NO❗
	MINIMUM_LIQUIDITY	External ❗		NO❗
	factory	External ❗		NO❗
	token0	External ❗		NO❗
	token1	External ❗		NO❗
	getReserves	External ❗		NO❗
	price0CumulativeLast	External ❗		NO❗
	price1CumulativeLast	External ❗		NO❗
	kLast	External ❗		NO❗
	mint	External ❗	🛑	NO❗
	burn	External ❗	🛑	NO❗
	swap	External ❗	🛑	NO❗
	skim	External ❗	🛑	NO❗
	sync	External ❗	🛑	NO❗
	initialize	External ❗	🛑	NO❗
Irouter01	Interface			
	factory	External ❗		NO❗
	WETH	External ❗		NO❗
	addLiquidity	External ❗	🛑	NO❗
	addLiquidityETH	External ❗	💵	NO❗

	removeLiquidity	External ❗	🛑	NO❗
	removeLiquidityETH	External ❗	🛑	NO❗
	removeLiquidityWithPermit	External ❗	🛑	NO❗
	removeLiquidityETHWithPermit	External ❗	🛑	NO❗
	swapExactTokensForTokens	External ❗	🛑	NO❗
	swapTokensForExactTokens	External ❗	🛑	NO❗
	swapExactETHForTokens	External ❗	💵	NO❗
	swapTokensForExactETH	External ❗	🛑	NO❗
	swapExactTokensForETH	External ❗	🛑	NO❗
	swapETHForExactTokens	External ❗	💵	NO❗
	quote	External ❗		NO❗
	getAmountOut	External ❗		NO❗
	getAmountIn	External ❗		NO❗
	getAmountsOut	External ❗		NO❗
	getAmountsIn	External ❗		NO❗
IUniswapV2Router02	Interface	Irouter01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External ❗	🛑	NO❗

	swapExactTokensForTokensSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
	swapExactETHForTokensSupportingFeeOnTransferTokens	External ❗	💵	NO❗
	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
Ownership	Implementation			
		Public ❗	🛑	NO❗
	addr	Internal 🔒		
	fee	Internal 🔒		
Ownable	Implementation	Context		
		Public ❗	🛑	NO❗
	owner	Public ❗		NO❗
	renounceOwnership	Public ❗	🛑	onlyOwner
	transferOwnership	Public ❗	🛑	onlyOwner
SafeMathInt	Library			
	mul	Internal 🔒		
	div	Internal 🔒		
	sub	Internal 🔒		
	add	Internal 🔒		
	abs	Internal 🔒		
	toUint256Safe	Internal 🔒		
SafeMathUint	Library			
	toInt256Safe	Internal 🔒		

DividendPayingTokenInterface	Interface			
	dividendOf	External ❗		NO❗
	withdrawDividend	External ❗	🛑	NO❗
DividendPayingTokenOptionalInterface	Interface			
	withdrawableDividendOf	External ❗		NO❗
	withdrawnDividendOf	External ❗		NO❗
	accumulativeDividendOf	External ❗		NO❗
DividendPayingToken	Implementation	ERC20, Ownable, DividendPayingTokenInterface, DividendPayingTokenOptionalInterface		
		Public ❗	🛑	ERC20
	distributeRewardDividends	Public ❗	🛑	onlyOwner
	withdrawDividend	Public ❗	🛑	NO❗
	_withdrawDividendOfUser	Internal 🔒	🛑	
	dividendOf	Public ❗		NO❗
	withdrawableDividendOf	Public ❗		NO❗
	withdrawnDividendOf	Public ❗		NO❗
	accumulativeDividendOf	Public ❗		NO❗
	_transfer	Internal 🔒	🛑	
	_mint	Internal 🔒	🛑	
	_burn	Internal 🔒	🛑	

	_setBalance	Internal 🔒	🛑	
ProjectXPolyon	Implementation	ERC20, Ownable, Ownership		
		Public ❗	💵	ERC20 Ownership
		External ❗	💵	NO❗
	updateDividendTracker	Public ❗	🛑	onlyOwner
	updaterouter	Public ❗	🛑	onlyOwner
	excludeFromFees	Public ❗	🛑	onlyOwner
	excludeMultipleAccountsFromFees	Public ❗	🛑	onlyOwner
	setMarketingWallet	External ❗	🛑	onlyOwner
	setTokenRewardsFee	External ❗	🛑	onlyOwner
	setLiquiditFee	External ❗	🛑	onlyOwner
	setMarketingFee	External ❗	🛑	onlyOwner
	setAutomatedMarketMakerPair	Public ❗	🛑	onlyOwner
	blacklistAddress	External ❗	🛑	onlyOwner
	_setAutomatedMarketMakerPair	Private 🔐	🛑	
	updateGasForProcessing	Public ❗	🛑	onlyOwner
	updateClaimWait	External ❗	🛑	onlyOwner
	getClaimWait	External ❗		NO❗
	getTotalDividendsDistributed	External ❗		NO❗
	isExcludedFromFees	Public ❗		NO❗

	withdrawableDividendOf	Public ❗		NO❗
	dividendTokenBalanceOf	Public ❗		NO❗
	excludeFromDividends	External ❗	🛑	onlyOwner
	getAccountDividendsInfo	External ❗		NO❗
	getAccountDividendsInfoAtIndex	External ❗		NO❗
	processDividendTracker	External ❗	🛑	NO❗
	claim	External ❗	🛑	NO❗
	getLastProcessedIndex	External ❗		NO❗
	getNumberOfDividendTokenHolders	External ❗		NO❗
	_transfer	Internal 🔒	🛑	
	swapAndSendToFee	Private 🔐	🛑	
	swapAndLiquify	Private 🔐	🛑	
	swapTokensForEth	Private 🔐	🛑	
	swapTokensForReward	Private 🔐	🛑	
	addLiquidity	Private 🔐	🛑	
	swapAndSendDividends	Private 🔐	🛑	
	recoverTokens	External ❗	🛑	onlyOwner
	recoverFunds	External ❗	🛑	onlyOwner
	setSwapAmount	External ❗	🛑	onlyOwner
RewardDividendTracker	Implementation	Ownable, DividendPayingToken		
		Public ❗	🛑	DividendPayingToken
		External ❗	💵	NO❗

	_transfer	Internal 🔒		
	withdrawDividend	Public ❗		NO❗
	excludeFromDividends	External ❗	🛑	onlyOwner
	updateClaimWait	External ❗	🛑	onlyOwner
	getLastProcessedIndex	External ❗		NO❗
	getNumberOfTokenHolders	External ❗		NO❗
	getAccount	Public ❗		NO❗
	getAccountAtIndex	Public ❗		NO❗
	canAutoClaim	Private 🔐		
	setBalance	External ❗	🛑	onlyOwner
	process	Public ❗	🛑	NO❗
	processAccount	Public ❗	🛑	onlyOwner

🛑	Function can modify state	💵	Function is payable

Audit Scope

Audit Method.

Our smart contract audit is an extensive methodical examination and analysis of the smart contract’s code that is used to interact with the blockchain. Goal: discover errors, issues and security vulnaribilities in the code. Findings getting reported and improvements getting suggested.

Automatic and Manual Review
We are using automated tools to scan functions and weeknesses of the contract. Transfers, integer over-undeflow checks such as all CWE events.

Tools we use:
Visual Studio Code
CWE
SWC
Solidity Scan
SVD

In manual code review our auditor looking at source code and performing line by line examination. This method helps to clarify developer’s coding decisions and business logic.

Skeleton Ecosystem

https://skeletonecosystem.com

https://github.com/SkeletonEcosystem/Audits