CyStack

PUBLIC ACCESS

CYBERSECURITY AUDIT REPORT

Version v1.3

This document details the process and results of the smart contract audit performed independently by CyStack from 12/01/2022 to 14/01/2022.

Audited for

Doxy Finance

Audited by

Vietnam CyStack Joint Stock Company

Contents

Disclaimer

Smart Contract Audit only provides findings and recommendations for an exact commitment of a smart contract codebase. The results, hence, are not guaranteed to be accurate outside of the commitment, or after any changes or modifications made to the codebase. The evaluation result does not guarantee the nonexistence of any further findings of security issues.

Time-limited engagements do not allow for a comprehensive evaluation of all security controls, so this audit does not give any warranties on finding all possible security issues of the given smart contract(s). CyStack prioritized the assessment to identify the weakest security controls an attacker would exploit. We recommend Dezilink International Limited conducting similar assessments on an annual basis by internal, third-party assessors, or a public bug bounty program to ensure the security of smart contract(s).

This security audit should never be used as an investment advice.

Version History

Version	Date	Release notes
1.0	13/01/2022	The first report is sent to the client. All findings are in the open status.
1.1	14/01/2022	All the findings are approved. All found issues are resolved in the new codebase.
1.2	14/01/2022	Dezilink International Limited allowed CyStack to publish the audit report publicly.
1.3	10/02/2022	Dezilink International Limited requested CyStack to modify some information, including the reference link to the codebase, roadmap and tokenomics images, and description, for Audit Target in section 1.1. CyStack confirms these changes are reasonable and do not mislead the audit results.

Contact Information

Company	Representative	Position	Email address
Dezilink International Limited	Alexander Denzel	Project Coordinator	alex@doxyfinance.com
CyStack	Vo Huyen Nhi	Sales Manager	nhivh@cystack.net
CyStack	Nguyen Ngoc Anh	Sales Executive	anhntn@cystack.net

Auditors

Fullname	Role	Email address
Nguyen Huu Trung	Head of Security	trungnh@cystack.net
Ha Minh Chau	Auditor	
Vu Hai Dang	Auditor	
Nguyen Van Huy	Auditor	
Nguyen Trung Huy Son	Auditor	
Nguyen Ba Anh Tuan	Auditor	

Introduction

From 12/01/2022 to 14/01/2022, Dezilink International Limited engaged CyStack to evaluate the security posture of the Doxy Finance of their contract system. Our findings and recommendations are detailed here in this initial report.

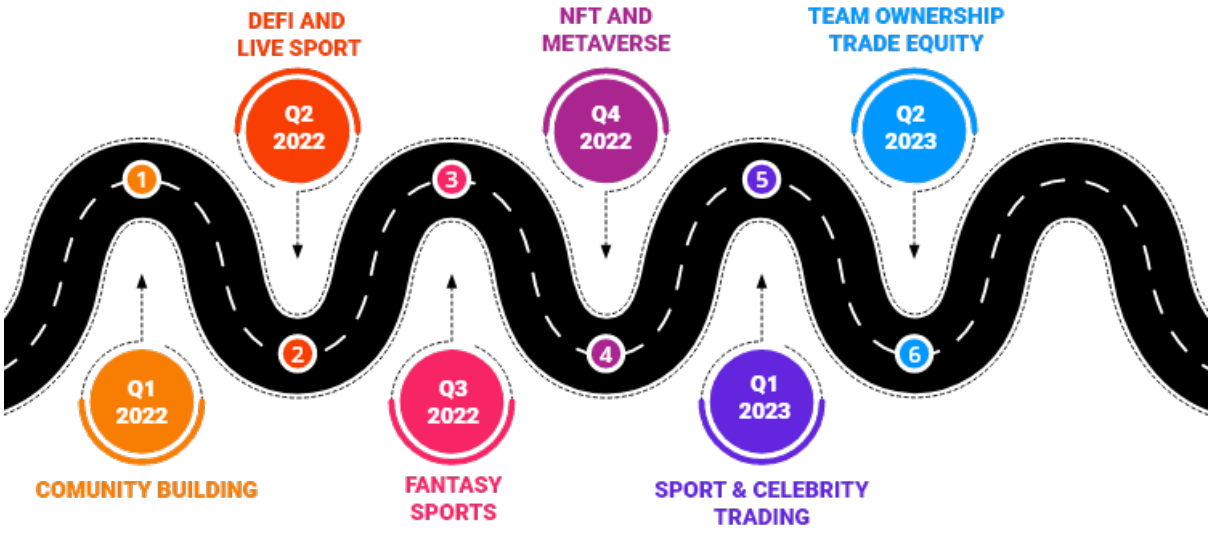
1.1 Audit Details

Audit Target

Doxy is a Live & Fantasy Sports platform powered by the blockchain technology. It's a decentralized social gaming experience in which people can express their passion for sports, compete against each other, and show "How much your sports knowledge is worth".

The sports industry through blockchain technology is a sun-rise industry all over the world. DOXY Sports public blockchain makes use of blockchain technology, which combines application and business situations of sports industry and sports ecosystem chain. Doxy will issue its native cryptographic token "DOXY TOKEN" for use on the DOXY Sports public blockchain.

The roadmap of Doxy Finance:



The basic information of DOXY is as follows:

Item	Description
Project Name	Doxy Finance
Issuer	Dezilink International Limited
Website	https://doxyfinance.com/
Platform	Binance Smart Contract
Language	Solidity
Codebase	https://bscscan.com/address/0xd6c7fbd02752e41d9b6000193668c16470fefd7d#code
Commit	N/A
Audit method	Whitebox

Token Doxy Finance is a token defined with the following information:

- Name: Doxy Finance
- Symbol: DOXY
- Decimals: 9
- Total Supply: 11,000,000 tokens

Overview for Doxy Finance tokenomics:

When the contract for Doxy Finance token is deployed:

- 11 million tokens will be transferred to wallets by this ratio: liquidityWallet (57.84%), marketingWallet (19.71%), strategicSalesWallet (5.39%), gameOperationsWallet (11.26%), teamWallet(4.40%), communityAirdropWallet (1.40%);
- The above wallets will be allowed to sell tokens (*includeInSell*) as well as make transactions without fees (*excludedFromFee*).

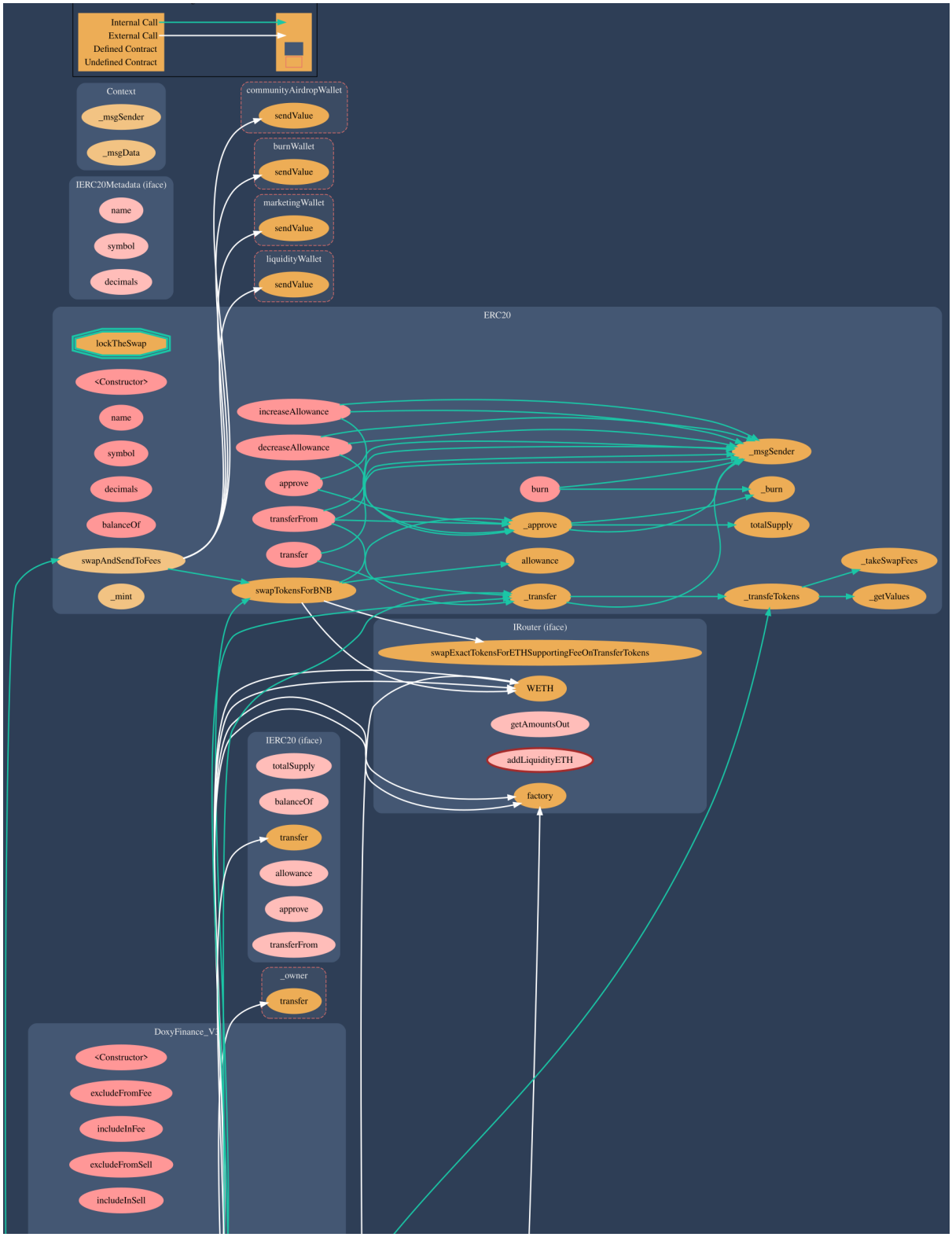
Features for users who own Doxy Finance tokens:

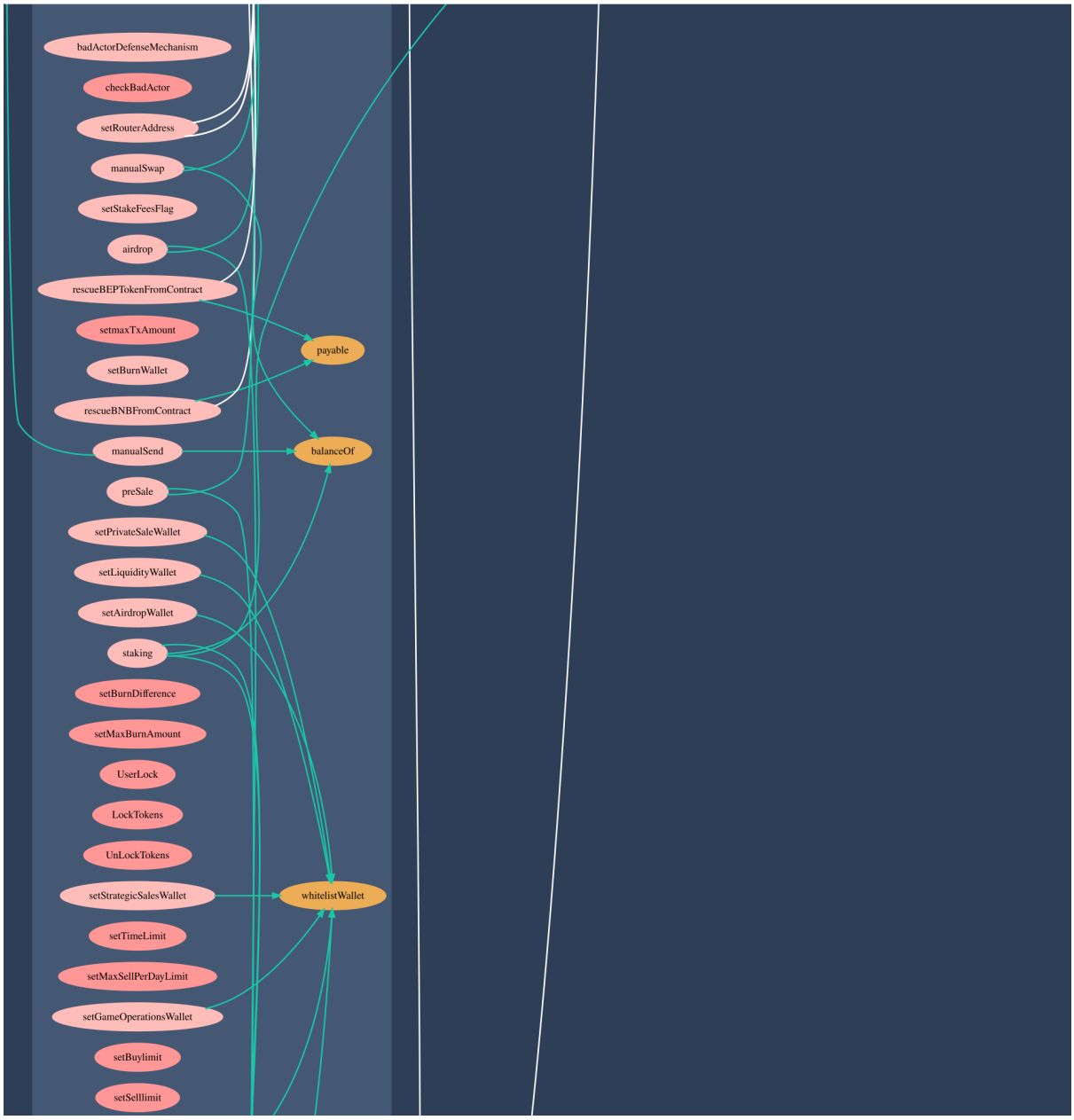
1. Buy and sell tokens on PancakeSwap:
    - Only wallets authorized by the owner are allowed to sell tokens (*includeInSell*);
    - With purchases on pancakeSwap, users can only buy up to 7,000 tokens (*buyLimit*) within every 24 hours;
    - With sales on pancakeSwap, users can only sell up to 2,000 (*sellLimit*) tokens within every 24 hours;
2. Transfer tokens:
    - For regular token transfers that aren't performed on pancakeSwap, regular users (without *excludedFromFee* permission) will have to pay a per-transaction fee of 2% and can only transfer up to 110,000 tokens per transaction.

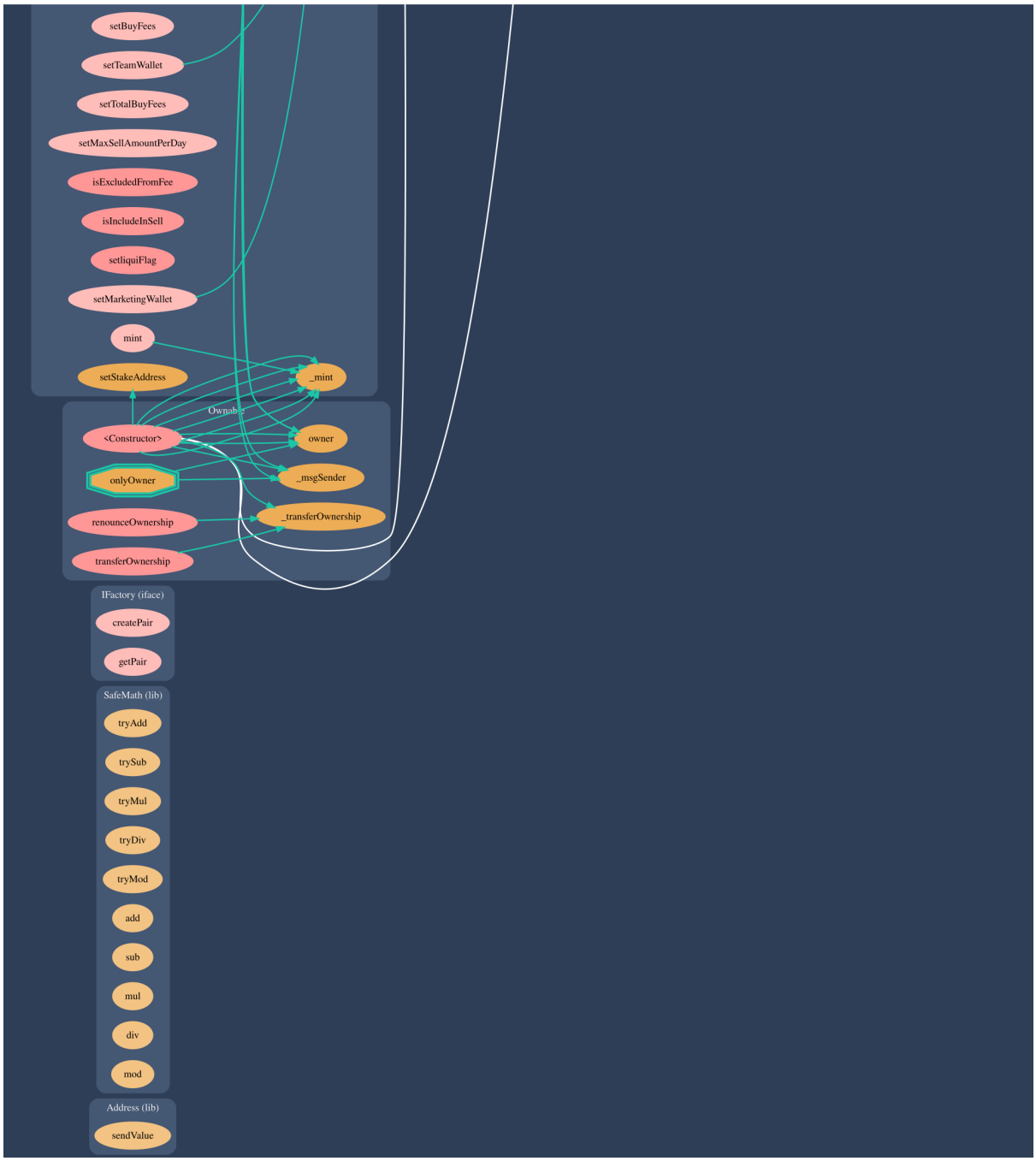
Some owner features that may lead to critical security issues:

- Owner can block any users' transaction;
- Owner can lock an amount of tokens of any users (not transferable);
- Owner can grant / revoke the right to sell tokens (*includeInSell*) of any users;
- Owner can grant/revoke transaction rights for any users without fees (ExcludedFromFee);
- Owner can change parameters *maxSellPerDay*, *buyLimit*, *sellLimit*;
- Owner can set the addresses of *marketingWallet*, *liquidityWallet*, *communityAirdropWallet*, *privateSaleWallet*, *strategicSalesWallet*, *gameOperationsWallet*, *teamWallet* and these addresses will be allowed to sell tokens (*includeInSell* permission) and make transactions for free (*excludedFromFee* permission).

The function calls in of DoxyFinance.sol is illustrated in the following graphs:







Audit Service Provider

CyStack is a leading security company in Vietnam with the goal of building the next generation of cybersecurity solutions to protect businesses against threats from the Internet. CyStack is a member of Vietnam Information Security Association (VNISA) and Vietnam Alliance for Cybersecurity Products Development.

CyStack's researchers are known as regular speakers at well-known cybersecurity conferences such as BlackHat USA, BlackHat Asia, Xcon, T2FI, etc. and are talented bug hunters who discovered critical vulnerabilities in global products and acknowledged by their vendors.

1.2 Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient and working according to its specifications. The audit activities can be grouped in the following three categories:

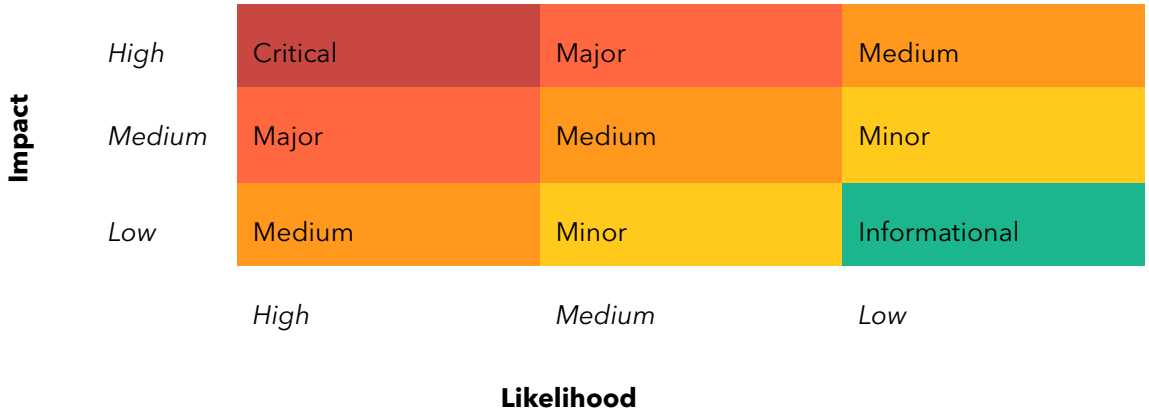
1. **Security:** Identifying security related issues within each contract and within the system of contracts.
2. **Sound Architecture:** Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. **Code Correctness and Quality:** A full review of the contract source code. The primary areas of focus include:
    - Correctness
    - Readability
    - Sections of code with high complexity
    - Improving scalability
    - Quantity and quality of test coverage

1.3 Audit Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology:

- **Likelihood** represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- **Impact** measures the technical loss and business damage of a successful attack;
- **Severity** demonstrates the overall criticality of the risk.

Likelihood and impact are categorized into three ratings: High, Medium and Low, i.e., H, M and L respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., Critical, Major, Medium, Minor and Informational (Info) as the table below:



CyStack firstly analyses the smart contract with open-source and also our own security assessment tools to identify basic bugs related to general smart contracts. These tools include Slither, securify, Mythril, Sūrya, Solgraph, Truffle, Geth, Ganache, Mist, Metamask, solhint, mythx, etc. Then, our security specialists will verify the tool results manually, make a description and decide the severity for each of them.

After that, we go through a checklist of possible issues that could not be detected with automatic tools, conduct test cases for each and indicate the severity level for the results. If no issues are found after manual analysis, the contract can be considered safe within the test case. Else, if any issues are found, we might further deploy contracts on our private testnet and run tests to confirm the findings. We would additionally build a PoC to demonstrate the possibility of exploitation, if required or necessary.

The standard checklist, which applies for every SCA, strictly follows the Smart Contract Weakness Classification Registry (SWC Registry). SWC Registry is an implementation of the weakness classification scheme proposed in The Ethereum Improvement Proposal project under the code EIP-1470. The checklist of testing according to SWC Registry is shown in Appendix A.

In general, the auditing process focuses on detecting and verifying the existence of the following issues:

- **Coding Specification Issues:** Focusing on identifying coding bugs related to general smart contract coding conventions and practices.
- **Design Defect Issues:** Reviewing the architecture design of the smart contract(s) and working on test cases, such as self-DoS attacks, incorrect inheritance implementations, etc.
- **Coding Security Issues:** Finding common security issues of the smart contract(s), for example integer overflows, insufficient verification of authenticity, improper use of cryptographic signature, etc.
- **Coding Design Issues:** Testing the code logic and error handlings in the smart contract code base, such as initializing contract variables, controlling the balance and flows of token transfers, verifying strong randomness, etc.
- **Coding Hidden Dangers:** Working on special issues, such as data privacy, data reliability, gas consumption optimization, special cases of authentication and owner permission, fallback functions, etc.

For better understanding of found issues' details and severity, each SWC ID is mapped to the most closely related Common Weakness Enumeration (CWE) ID. CWE is a category system for software weaknesses and vulnerabilities to help identify weaknesses surrounding software jargon. The list in Appendix B provides an overview on specific similar software bugs that occur in Smart Contract coding.

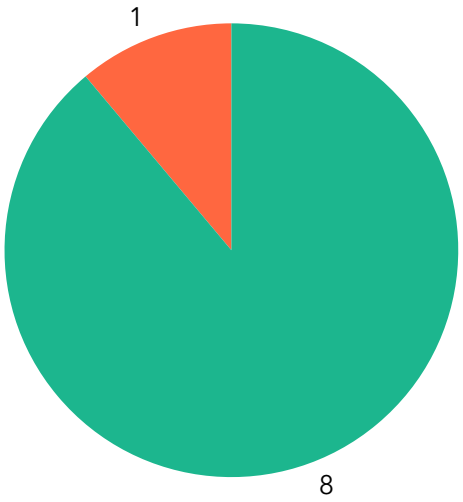
The final report will be sent to the smart contract issuer with an executive summary for overview and detailed results for acts of remediation.

1.4 Audit Scope

Assessment	Target	Type
Original target		
White-box testing	DoxyFinance_V3.sol	Solidity code file
Re-test target		
White-box testing	DoxyFinance.sol	Solidity code file
Official contract		
White-box testing	DoxyFinance.sol	Solidity code file

Executive Summary

Security issues by severity



Security issues by SWC

Function Default Visibility (SWC-100)	1	
Floating Pragma (SWC-103)	1	
State Variable Default Visibility (SWC-108)	2	
Lack of Proper Signature Verification (SWC-122)	1	
Typographical Error (SWC-129)	2	
Code With No Effects (SWC-135)	2	

Security issues by CWE

Insufficient Verification of Data Authenticity (CWE-345)	1	
Use of Incorrect Operator (CWE-480)	2	
Improper Control of a Resource Through its Lifetime (CWE-664)	1	
Improper Adherence to Coding Standards (CWE-710)	3	
Irrelevant Code (CWE-1164)	2	

Table of security issues

ID	Status	Vulnerability	Severity
#doxy-001	Resolved	Floating pragma	INFO
#doxy-002	Resolved	Unoptimized variable declarations	INFO
#doxy-003	Resolved	Struct and address declarations without usage	INFO
#doxy-004	Resolved	Unnecessary gas consumption due to improper variable declaration	INFO
#doxy-005	Resolved	Variable declarations without concrete visibility	INFO
#doxy-006	Resolved	Loose timeLimit validation for selling transactions on pancakeSwap	MAJOR
#doxy-007	Resolved	Ineffiecient function declaration	INFO
#doxy-008	Resolved	Mistyped argument names in functions	INFO
#doxy-009	Resolved	Too many digits	INFO

Recommendations

Based on the results of this smart contract audit, CyStack has the following high-level key recommendations:

Key recommendations	
Issues	CyStack conducted the third SCA for Doxy Finance Token after Doxy Finance development team had committed new codebases with mitigations for remaining issues from the second report. All the findings are resolved.
Recommendations	CyStack recommends Dezilink International Limited to evaluate the audit results with several different security audit third-parties for the most accurate conclusion.
References	• https://consensys.github.io/smart-contract-best-practices/known_attacks • https://consensys.github.io/smart-contract-best-practices/recommendations/ • https://medium.com/@knownsec404team/ethereum-smart-contract-audit-checklist-ba9d1159b901

Detailed Results

1. Floating pragma

Issue ID	#doxy-001
Category	SWC-103 - Floating Pragma
Description	Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.
Severity	INFO
Location(s)	DoxyFinance_V3.sol:3
Status	Resolved
Reference	CWE-664 - Improper Control of a Resource Through its Lifetime
Remediation	Lock the pragma version and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Description

The codeline where floating pragma is used:

```
…
3  pragma solidity ^0.8.0;
…
```

The code can be revised as following:

```
…
3  pragma solidity 0.8.0;
…
```

or:

```
…
3  pragma solidity 0.8.7;
…
```

2. Unoptimized variable declarations

Issue ID	#doxy-002
Category	SWC-108 - State Variable Default Visibility
Description	Unchanged variables should be declare as *constant* in order to reduce gas consumption.
Severity	INFO
Location(s)	DoxyFinance_V3.sol:282-284
Status	Resolved
Reference	CWE-710 - Improper Adherence to Coding Standards
Remediation	Add *constant* to the variable declaration.

Description

The codelines where the issue occurs:

```
...
282     address public pancakeSwapRouter =
283         address(0xD99D1c33F9fC3444f8101754aBC46c52416550D1); //
            ↪  10ed43c718714eb63d5aa57b78b54704e256024e for mainnet;
284     address public USDT = address(0x8301F2213c0eeD49a7E28Ae4c3e91722919B8B47);   //
        ↪  BUSDT tesnet : 0x8301F2213c0eeD49a7E28Ae4c3e91722919B8B47 // USDT ropsten :
        ↪  0xb03Ba6B311aaC34B06bdC97357E6f08BF2c12857
...
```

The code can be revised as following:

```
...
282     address public constant pancakeSwapRouter =
        ↪ 0xD99D1c33F9fC3444f8101754aBC46c52416550D1;
283     address public constant USDT = 0x8301F2213c0eeD49a7E28Ae4c3e91722919B8B47;
...
```

3. Struct and address declarations without usage

Issue ID	#doxy-003
Category	SWC-135 Code With No Effects
Description	The struct *antiwhale* and address *USDT* is declared but remain unused in the whole code base. This might involve more gas consumption.
Severity	INFO
Location(s)	DoxyFinance_V3.sol:284, 312-315
Status	Resolved
Reference	CWE-1164 - Irrelevant Code
Remediation	Remove these struct and address from the codebase.

Description

The codelines where the issue occurs:

```
…
284     address public USDT = address(0x8301F2213c0eeD49a7E28Ae4c3e91722919B8B47);
…
312     struct antiwhale {
313         uint256 selling_threshold; //this is value/1000 %
314         uint256 extra_tax; //this is value %
315
…       }
317
```

The codebase can be revised by removing those lines.

4. Unnecessary gas consumption due to improper variable declaration

Issue ID	#doxy-004
Category	SWC-135 Code With No Effects
Description	Unnecessary declaration of *_pancakeRouter*. *pancakeRouter* could be declared and assigned with the desire value directly. This leads to more gas consumption when deploying the smart contract.
Severity	INFO
Location(s)	Resolved
Status	DoxyFinance_V3.sol:809-818
Reference	CWE-1164 - Irrelevant Code
Remediation	Change the codebase following the description below.

Description

The codelines where the issue occurs:

```
…
809         IRouter _pancakeRouter = IRouter(
810             0xD99D1c33F9fC3444f8101754aBC46c52416550D1 // replace with
                ↪ 0x10ED43C718714eb63d5aA57B78B54704E256024E while deploying to mainnet
811         );
812
813         pancakePair = IFactory(_pancakeRouter.factory()).createPair(
814             address(this),
815             _pancakeRouter.WETH()
816         );
817
818         pancakeRouter = _pancakeRouter;
…
```

The code can be revised as following:

```
…
809     pancakeRouter = IRouter(0xD99D1c33F9fC3444f8101754aBC46c52416550D1);
…
811     pancakePair = IFactory(pancakeRouter.factory()).createPair(
812         address(this),
813         pancakeRouter.WETH()
814     );
815
```

5. Variable declarations without concrete visibility

Issue ID	#doxy-005
Category	SWC-108 - State Variable Default Visibility Severity
Description	Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.
Severity	INFO
Location(s)	DoxyFinance_V3.sol:270
Status	Resolved
Reference	CWE-710 - Improper Adherence to Coding Standards
Remediation	Variables can be specified as being *public*, *internal* or *private*. Explicitly define visibility for all state variables.

Description

The codelines where the issue occurs:

```
…
270     bool inSwap;
…
```

The code can be revised as following:

```
…
270     bool private inSwap;
…
```

6. Loose timeLimit validation for sell transactions on pancakeSwap

Issue ID	#doxy-006
Category	SWC-122 - Lack of Proper Signature Verification
Description	*block.timestamp* can be controlled by any users, which allows users to make transactions under the condition where **block.timestamp = timeLimit + 24 * 1 hours**. By that, users can sell the tokens without increasing *maxSellPerDayLimit*. In consequence, users can sell their tokens even when the value of *maxSellPerDayLimit* exceeds the value of *maxSellPerDay*.
Severity	MAJOR
Location(s)	DoxyFinance_V3.sol:497-502, 511-516
Status	Resolved
Reference	CWE-345 - Insufficient Verification of Data Authenticity
Remediation	Add additional verifications by comparing operations >= and >= or remove the conditions written in the following description.

Description

The codelines where the issue occurs:

```
…
497     if (block.timestamp < timeLimit + 24 * 1 hours) {
498         maxSellPerDayLimit += amount;
499     } else if (block.timestamp > timeLimit + 24 * 1 hours) {
500         maxSellPerDayLimit = 1000000000;
501         timeLimit = block.timestamp;
502     }
…
```

The code can be revised as following:

```
…
497     if (block.timestamp < timeLimit + 24 * 1 hours) {
498         maxSellPerDayLimit += amount;
499     } else {
500         maxSellPerDayLimit = 1000000000;
501         timeLimit = block.timestamp;
502     }
…
```

7. Ineffiecient function declaration

Issue ID	#doxy-007
Category	SWC-100 - Function Default Visibility
Description	public functions that are never called by the contract should be declared external to save gas.
Severity	INFO
Location(s)	DoxyFinance_V3.sol: 373-381, 393-411, 413-430, 432-443, 445-460, 679-695, 754-756, 758-764, 824-826, 828-830, 832-834, 836-838, 881-883, 972-974, 976-978, 980-982, 984-986, 988-990, 998-1000, 1002-1004, 1010-1012, 1014-1016, 1077-1079
Status	Resolved
Reference	CWE-710 - Improper Adherence to Coding Standards

Remediation	Declare the following functions as *external* to improve the codebase: • ERC20.transfer(address,uint256) • ERC20.approve(address,uint256) • ERC20.transferFrom(address,address,uint256) • ERC20.increaseAllowance(address,uint256) • ERC20.decreaseAllowance(address,uint256) • ERC20.burn(uint256) • Ownable.renounceOwnership() • Ownable.transferOwnership(address) • DoxyFinance_V3.excludeFromFee(address) • DoxyFinance_V3.includeInFee(address) • DoxyFinance_V3.excludeFromSell(address) • DoxyFinance_V3.includeInSell(address) • DoxyFinance_V3.setmaxTxAmount(uint256) • DoxyFinance_V3.setBurnDifference(uint256) • DoxyFinance_V3.setMaxBurnAmount(uint256) • DoxyFinance_V3.UserLock(address,bool) • DoxyFinance_V3.LockTokens(address,uint256) • DoxyFinance_V3.UnLockTokens(address) • DoxyFinance_V3.setTimeLimit(uint256) • DoxyFinance_V3.setMaxSellPerDayLimit(uint256) • DoxyFinance_V3.setBuylimit(uint256) • DoxyFinance_V3.setSelllimit(uint256) • DoxyFinance_V3.setliquiFlag()

8. Mistyped argument names in functions

Issue ID	#doxy-008
Category	SWC-129 - Typographical Error
Description	The functions that allow to make change on *liquidityWallet* , *strategicSalesWallet*, *gameOperationsWallet*, *teamWallet*, *communityAirdropWallet*, *privateSaleWallet* all make whitelist for *marketingWallet*.
Severity	INFO
Location(s)	DoxyFinance_V3.sol:918, 928, 938, 948, 958, 968
Status	Resolved
Reference	CWE-480 - Use of Incorrect Operator
Remediation	Perform whitelist for the correct type of wallet according to the functions.

Description

The code can be revised as following, taking *setLiquidityWallet* as an example:

```
…
912     function setLiquidityWallet(address payable _address)
913         external
914         onlyOwner
915         returns (bool)
916     {
917         liquidityWallet = _address;
918         whitelistWallet(liquidityWallet) ;
919         return true;
920     }
…
```

9. Too many digits

Issue ID	#doxy-009
Category	SWC-129 - Typographical Error
Description	Literals with many digits are difficult to read and review.
Severity	INFO
Location(s)	DoxyFinance_V3.sol:339, 488, 500, 514, 658, 661, 780, 781, 782, 783, 784, 785, 821, 261, 262
Status	Resolved
Reference	CWE-480 - Use of Incorrect Operator
Remediation	Use ether suffix. A literal number can take a suffix of wei, gwei or ether to specify a subdenomination of Ether, where Ether numbers without a postfix are assumed to be wei.

Description

The codelines where the issue occurs, example DoxyFinance_V3.sol:261, 262:

```
…
261     uint256 public burnDifference = 11000000000000000;
262     uint256 public maxBurnAmount = 502700000000000;
…
```

The code can be revised as following:

```
…
261     uint256 public burnDifference = 11 * 10**6 * 10**9;
262     uint256 public maxBurnAmount = 502700 * 10**9;
…
```

Conclusion

CyStack had conducted a security audit for Doxy Finance Token. Total nine issues were found, including a major and eight informational issues. Right after receiving the first audit report, the Doxy Finance issuer immediately take action on addressing the issue based on their severity. CyStack confirmed that all found issues were resolved. Overall, Doxy Finance has included the best practices for smart contract development and has passed our security assessment for smart contracts.

To improve the quality for this report, and for CyStack's Smart Contract Audit report in general, we greatly appreciate any constructive feedback or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

Appendices

Appendix A – Security Issue Status Definitions

Status	Definition
Open	The issue has been reported and currently being review by the smart contract developers/issuer.
Unresolved	The issue is acknowledged and planned to be addressed in future. At the time of the corresponding report version, the issue has not been fixed.
Resolved	The issue is acknowledged and has been fully fixed by the smart contract developers/issuer.
Rejected	The issue is considered to have no security implications or to make only little security impacts, so it is not planned to be addressed and won't be fixed.

Appendix B – Severity Explanation

Severity	Definition
CRITICAL	Issues, considered as critical, are straightforwardly exploitable bugs and security vulnerabilities. It is advised to immediately resolve these issues in order to prevent major problems or a full failure during contract system operation.
MAJOR	Major issues are bugs and vulnerabilities, which cannot be exploited directly without certain conditions. It is advised to patch the codebase of the smart contract as soon as possible, since these issues, with a high degree of probability, can cause certain problems for operation of the smart contract or severe security impacts on the system in some way.
MEDIUM	In terms of medium issues, bugs and vulnerabilities exist but cannot be exploited without extra steps such as social engineering. It is advised to form a plan of action and patch after high-priority issues have been resolved.
MINOR	Minor issues are generally objective in nature but do not represent actual bugs or security problems. It is advised to address these issues, unless there is a clear reason not to.
INFO	Issues, regarded as informational (info), possibly relate to “guides for the best practices” or “readability”. Generally, these issues are not actual bugs or vulnerabilities. It is recommended to address these issues, if it make effective and secure improvements to the smart contract codebase.

Appendix C – Smart Contract Weakness Classification Registry (SWC Registry)

ID	Name	Description
	Coding Specification Issues	
SWC-100	Function Default Visibility	It is recommended to make a conscious decision on which visibility type (*external*, *public*, *internal* or *private*) is appropriate for a function. By default, functions without concrete specifiers are *public*.
SWC-102	Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler to avoid publicly disclosed bugs and issues in outdated versions.
SWC-103	Floating Pragma	It is recommended to lock the pragma to ensure that contracts do not accidentally get deployed using.
SWC-108	State Variable Default Visibility	Variables can be specified as being *public*, *internal* or *private*. Explicitly define visibility for all state variables.
SWC-111	Use of Deprecated Solidity Functions	Solidity provides alternatives to the deprecated constructions, the use of which might reduce code quality. Most of them are aliases, thus replacing old constructions will not break current behavior.
SWC-118	Incorrect Constructor Name	It is therefore recommended to upgrade the contract to a recent version of the Solidity compiler and change to the new constructor declaration (the keyword *constructor*).
	Design Defect Issues	
SWC-113	DoS with Failed Call	External calls can fail accidentally or deliberately, which can cause a DoS condition in the contract. It is better to isolate each external call into its own transaction and implement the contract logic to handle failed calls.

SWC-119	Shadowing State Variables	Review storage variable layouts for your contract systems carefully and remove any ambiguities. Always check for compiler warnings as they can flag the issue within a single contract.
SWC-125	Incorrect Inheritance Order	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order (from more /general/ to more /specific/).
SWC-128	DoS With Block Gas Limit	Modifying an array of unknown size, that increases in size over time, can lead to such a Denial of Service condition. Actions that require looping across the entire data structure should be avoided.
	Coding Security Issues	
SWC-101	Integer Overflow and Underflow	It is recommended to use safe math libraries for arithmetic operations throughout the smart contract system to avoid integer overflows and underflows.
SWC-107	Reentrancy	Make sure all internal state changes are performed before the call is executed or use a reentrancy lock.
SWC-112	Delegatecall to Untrusted Callee	Use *delegatecall* with caution and make sure to never call into untrusted contracts. If the target address is derived from user input ensure to check it against a whitelist of trusted contracts.
SWC-117	Signature Malleability	A signature should never be included into a signed message hash to check if previously messages have been processed by the contract.
SWC-121	Missing Protection against Signature Replay Attacks	In order to protect against signature replay attacks, store every message hash that has been processed by the smart contract, include the address of the contract that processes the message and never generate the message hash including the signature.
SWC-122	Lack of Proper Signature Verification	It is not recommended to use alternate verification schemes that do not require proper signature verification through *ecrecover()*.

SWC-130	Right-To-Left-Override control character (U+202E)	The character *U+202E* should not appear in the source code of a smart contract.
	Coding Design Issues	
SWC-104	Unchecked Call Return Value	If you choose to use low-level call methods (e.g. *call()*), make sure to handle the possibility that the call fails by checking the return value.
SWC-105	Unprotected Ether Withdrawal	Implement controls so withdrawals can only be triggered by authorized parties or according to the specs of the smart contract system.
SWC-106	Unprotected SELFDESTRUCT Instruction	Consider removing the self-destruct functionality. If absolutely required, it is recommended to implement a multisig scheme so that multiple parties must approve the self-destruct action.
SWC-110	Assert Violation	Consider whether the condition checked in the *assert()* is actually an invariant. If not, replace the *assert()* statement with a *require()* statement.
SWC-116	Block values as a proxy for time	Developers should write smart contracts with the notion that block values are not precise, and the use of them can lead to unexpected effects. Alternatively, they may make use oracles.
SWC-120	Weak Sources of Randomness from Chain Attributes	To avoid weak sources of randomness, use commitment scheme, e.g. RANDAO, external sources of randomness via oracles, e.g. Oraclize, or Bitcoin block hashes.
SWC-123	Requirement Violation	If the required logical condition is too strong, it should be weakened to allow all valid external inputs. Otherwise, make sure no invalid inputs are provided.
SWC-124	Write to Arbitrary Storage Location	As a general advice, given that all data structures share the same storage (address) space, one should make sure that writes to one data structure cannot inadvertently overwrite entries of another data structure.

SWC-132	Unexpected Ether balance	Avoid strict equality checks for the Ether balance in a contract.
SWC-133	Hash Collisions With Multiple Variable Length Arguments	When using *abi.encodePacked()*, it's crucial to ensure that a matching signature cannot be achieved using different parameters. Alternatively, you can simply use *abi.encode()* instead. It is also recommended to use replay protection.
	Coding Hidden Dangers	
SWC-109	Uninitialized Storage Pointer	Uninitialized local storage variables can point to unexpected storage locations in the contract. If a local variable is sufficient, mark it with *memory*, else *storage* upon declaration. As of compiler version 0.5.0 and higher this issue has been systematically resolved.
SWC-114	Transaction Order Dependence	A possible way to remedy for race conditions in submission of information in exchange for a reward is called a commit reveal hash scheme. The best fix for the ERC20 race condition is to add a field to the inputs of approve which is the expected current value and to have approve revert or add a safe approve function.
SWC-115	Authorization through tx.origin	*tx.origin* should not be used for authorization. Use *msg.sender* instead.
SWC-126	Insufficient Gas Griefing	Insufficient gas griefing attacks can be performed on contracts which accept data and use it in a sub-call on another contract. To avoid them, only allow trusted users to relay transactions and require that the forwarder provides enough gas.
SWC-127	Arbitrary Jump with Function Type Variable	The use of assembly should be minimal. A developer should not allow a user to assign arbitrary values to function type variables.

SWC-129	Typographical Error	The weakness can be avoided by performing pre-condition checks on any math operation or using a vetted library for arithmetic calculations such as SafeMath developed by OpenZeppelin.
SWC-131	Presence of unused variables	Remove all unused variables from the code base.
SWC-134	Message call with hardcoded gas amount	Avoid the use of *transfer()* and *send()* and do not otherwise specify a fixed amount of gas when performing calls. Use *.call.value(...)("")* instead.
SWC-135	Code With No Effects	It's important to carefully ensure that your contract works as intended. Write unit tests to verify correct behaviour of the code.
SWC-136	Unencrypted Private Data On-Chain	Any private data should either be stored off-chain, or carefully encrypted.

Appendix D – Related Common Weakness Enumeration (CWE)

The SWC Registry loosely aligned to the terminologies and structure used in the CWE while overlaying a wide range of weakness variants that are specific to smart contracts.

CWE IDs *, to which SWC Registry is related, are listed in the following table:

CWE ID	Name	Related SWC IDs
CWE-284	**Improper Access Control**	SWC-105, SWC-106
CWE-294	Authentication Bypass by Capture-replay	SWC-133
CWE-664	**Improper Control of a Resource Through its Lifetime**	SWC-103
CWE-123	Write-what-where Condition	SWC-124
CWE-400	Uncontrolled Resource Consumption	SWC-128
CWE-451	User Interface (UI) Misrepresentation of Critical Information	SWC-130
CWE-665	Improper Initialization	SWC-118, SWC-134
CWE-767	Access to Critical Private Variable via Public Method	SWC-136
CWE-824	Access of Uninitialized Pointer	SWC-109
CWE-829	Inclusion of Functionality from Untrusted Control Sphere	SWC-112, SWC-116
CWE-682	**Incorrect Calculation**	SWC-101
CWE-691	**Insufficient Control Flow Management**	SWC-126
CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization (“Race Condition”)	SWC-114
CWE-480	Use of Incorrect Operator	SWC-129
CWE-667	Improper Locking	SWC-132
CWE-670	Always-Incorrect Control Flow Implementation	SWC-110
CWE-696	Incorrect Behavior Order	SWC-125
CWE-841	Improper Enforcement of Behavioral Workflow	SWC-107
CWE-693	**Protection Mechanism Failure**	

CWE-330	Use of Insufficiently Random Values	SWC-120
CWE-345	Insufficient Verification of Data Authenticity	SWC-122
CWE-347	Improper Verification of Cryptographic Signature	SWC-117, SWC-121
CWE-703	**Improper Check or Handling of Exceptional Conditions**	SWC-113
CWE-252	Unchecked Return Value	SWC-104
CWE-710	**Improper Adherence to Coding Standards**	SWC-100, SWC-108, SWC-119
CWE-477	Use of Obsolete Function	SWC-111, SWC-115
CWE-573	Improper Following of Specification by Caller	SWC-123
CWE-695	Use of Low-Level Functionality	SWC-127
CWE-1164	Irrelevant Code	SWC-131, SWC-135
CWE-937	**Using Components with Known Vulnerabilities**	SWC-102

** CWE IDs, which are presented in bold, are the greatest parent nodes of those nodes following it.*

All IDs in the CWE list above are relevant to the view "Research Concepts" (CWE-1000), except for CWE-937, which is relevant to the "Weaknesses in OWASP Top Ten (2013)" (CWE-928).