Swych Token

AUDIT REPORT

Version 1.0.0

Serial No. 20230316000012020

Presented by Fairyproof

March 16, 2023

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Swych Token Issuance project.

Audit Start Time:

January 23, 2023

Audit End Time:

March 16, 2023

Audited Source Files:

The calculated SHA-256 value for the audited file when the audit was done is as follows:

```
Swych.sol:0x0d7254e1960dd318013df30a7081f9af299454ba43cf6a457a245021e438957d
sSwych.sol:0x1959110042d7b53152b0fd4e1d4fb23fc9c131f1c02cd1827b73452f46e48349
```

The source files audited include all the files as follows:

```
├── Swych.sol
├── interfaces
│   ├── IDEXFactory.sol
│   ├── IDEXPair.sol
│   └── IDEXRouter.sol
└── sSwych.sol

1 directory, 5 files
```

The goal of this audit is to review Swych's solidity implementation for its Token Issuance function, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

This audit only applies to the specified code, software or any materials supplied by the Swych team for specified versions. Whenever the code, software, materials, settings, environment etc is changed, the comments of this audit will no longer apply.

— Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding system security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from offchain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

— Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code Review, Including:

- Project Diagnosis

Understanding the size, scope and functionality of your project's source code based on the specifications, sources, and instructions provided to Fairyproof.

- Manual Code Review

Reading your source code line-by-line to identify potential vulnerabilities.

- Specification Comparison

Determining whether your project's code successfully and efficiently accomplishes or executes its functions according to the specifications, sources, and instructions provided to Fairyproof.

2. Testing and Automated Analysis, Including:

- Test Coverage Analysis

Determining whether the test cases cover your code and how much of your code is exercised or executed when test cases are run.

- Symbolic Execution

Analyzing a program to determine the specific input that causes different parts of a program to execute its functions.

3. Best Practices Review

Reviewing the source code to improve maintainability, security, and control based on the latest established industry and academic practices, recommendations, and research.

— Structure of the document

This report contains a list of issues and comments on all the above source files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

— Documentation

For this audit, we used the following source(s) of truth about how the token issuance function should work:

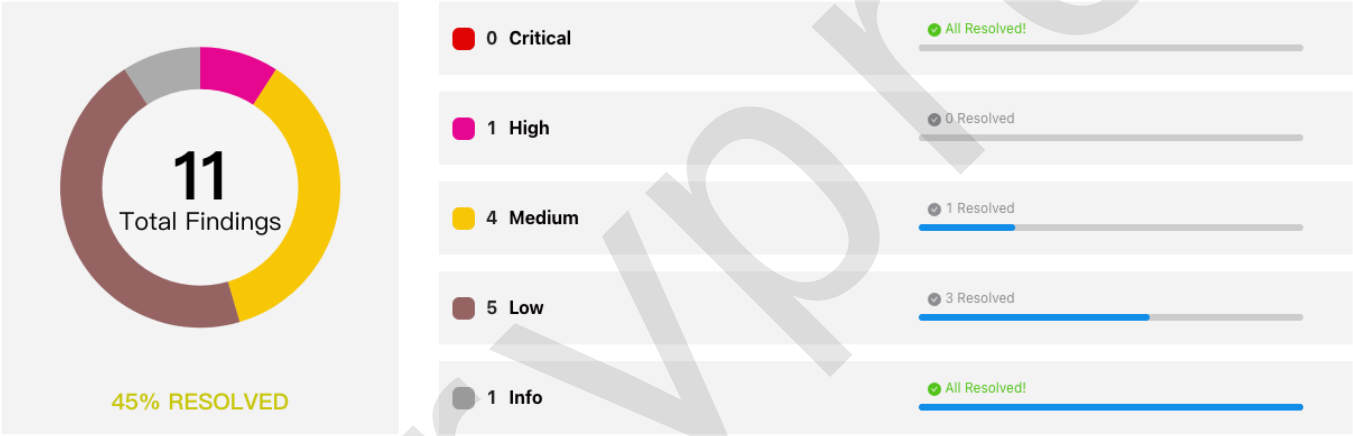
Website:https://swych.finance/

Source Code: SourceCode

These were considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the Swych team or reported an issue.

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2023031600012020	Fairyproof Security Team	Jan 23, 2023 - Mar 16, 2023	High Risk



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, one issue of high-severity, four issues of medium-severity, five issues of low-severity and one issue of info-severity were uncovered. The Swych team fixed one issue of medium, three issues of low and one issue of info, and acknowledged the remaining issues.

02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying blockchain applications.

03. Introduction to Swych

Swych is a powerful DeFi ecosystem of cutting edge products and services aimed at delivering new levels of rewards to users and projects.

The above description is quoted from relevant documents of Swych.

04. Major functions of audited code

The audited code mainly implements token issuance and farm function. Here are the details:

Token Issuance (Swych.sol):

- Token Standard: ERC20
- Token Name: Sywch
- Token Symbol: SYWCH
- Decimals: 18
- Current Supply: 100,000,000
- Max Supply: No Cap
- Mintable: Yes
- Pausable: Yes
- Blacklist: Yes

Note:

- The implementation is upgradeable and uses a proxy/implementation design pattern. When a contract upgrade is needed, the new code needs to be audited prior to the contract upgrade.
- The Swych's balance of an address changes according to Swych's price change. When the price goes up, the balance goes up accordingly. When the price goes down and the price drop reaches a preset value, the balance goes down accordingly.
- The owner can change all the core parameters of both `Swych` and `sSwych`.
- On every Sywch token transfer, a tax is charged. And depending on whether the token transfer interacts with a specified token pair, a tax rate charged for buy, sell or transfer operation is by default set as 12%, 15% or 35% respectively. 20% of the tax charged is sent to a blackhole address, 30% is reserved as `treasuryFee` and 50% is reserved as `LiquidityFee`. These tax rates can be changed.
- When an address transfers all its held Sywch tokens to an address whose balance is zero prior to this transfer, no tax will be charged however the tokens will be locked in the destination address for 10 days, regular buy or sell is disallowed and the address can only interfact with the `isFeeExempt` contract.
- When an address sells its Sywch tokens, an upper bound is applied to the total amount of tokens the address can sell in a consecutive 10 days. This upper bound is relevant to the ratio of the address' balance and a specified token pair's balance. The larger the ratio the smaller the upper bound. The min value of the upper bound is 2% and the max value of the upper bound is 20%. The upper bound is the address' balance multiplied by the ratio.

Token Farm(sSwych.sol):

A user can call `SYWCH` to stake its Swych tokens and get rewards in Swych (the reward is minted in `SYWCH`). When a user stakes its tokens it can fill in a referral (only once) which cannot be the user itself. The max number of referral levels is three.

When a user withdraws its staked tokens, 5% transaction fees will be charged and the fees will be sent to the blackhole address.

The implementation has an emergency withdrawal function which users can call to withdraw their staked tokens but not rewards.

Note:

The implementation uses a proxy/implementation pattern to allow contract upgrades. When a contract upgrade is to be done, new code needs to be audited prior to the contract upgrade.

The reward rate can be changed by the admin. Upon a rate change all unclaimed rewards will be re-calculated.

The system can be paused. When it is paused, staking, regular withdrawals or emergency withdrawals will not be allowed.

05. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Access Control
- Admin Rights
- Arithmetic Precision
- Code Improvement
- Contract Upgrade/Migration
- Delete Trap
- Design Vulnerability
- DoS Attack
- EOA Call Trap
- Fake Deposit
- Function Visibility
- Gas Consumption
- Implementation Vulnerability
- Inappropriate Callback Function
- Injection Attack
- Integer Overflow/Underflow
- IsContract Trap
- Miner's Advantage
- Misc
- Price Manipulation
- Proxy selector clashing
- Pseudo Random Number
- Re-entrancy Attack
- Replay Attack
- Rollback Attack
- Shadow Variable
- Slot Conflict
- Token Issuance
- Tx.origin Authentication
- Uninitialized Storage Pointer

06. Severity level reference

Every issue in this report was assigned a severity level from the following:

Critical severity issues need to be fixed as soon as possible.

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Informational is not an issue or risk but a suggestion for code improvement.

07. Major areas that need attention

Based on the provided source code the Fairyproof team focused on the possible issues and risks related to the following functions or areas.

- Function Implementation

We checked whether or not the functions were correctly implemented.
We found some issues, for more details please refer to [FP-4,FP-6,FP-9,FP-10] in "09. Issue description".

- Access Control

We checked each of the functions that could modify a state, especially those functions that could only be accessed by owner or administrator
We didn't find issues or risks in these functions or areas at the time of writing.

- Token Issuance & Transfer

We examined token issuance and transfers for situations that could harm the interests of holders.
We found some issues, for more details please refer to [FP-1,FP-2,FP-3,FP-5] in "09. Issue description".

- State Update

We checked some key state variables which should only be set at initialization.
We didn't find issues or risks in these functions or areas at the time of writing.

- Asset Security

We checked whether or not all the functions that transfer assets were safely handled.
We didn't find issues or risks in these functions or areas at the time of writing.

- Miscellaneous

We checked the code for optimization and robustness.
We found some issues, for more details please refer to [FP-7,FP-8,FP-11] in "09. Issue description".

08. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Token Selling Restricted	Token Issuance	High	Acknowledged
FP-2	Unlimited Token Issuance	Token Issuance	Medium	Acknowledged
FP-3	Relatively High transferFee	Token Issuance	Medium	Acknowledged
FP-4	Excessive Access Control	Admin Rights	Medium	Acknowledged
FP-5	Token transfers can be paused	Token Issuance	Low	Acknowledged
FP-6	`athPrice` Can Only Increase Monotonically	Design Vulnerability	Low	Acknowledged
FP-7	Incorrect Use of isContract	Implementation Vulnerability	Medium	✓ Fixed
FP-8	Typo	Code Improvement	Low	✓ Fixed
FP-9	Missing event argument	Code Improvement	Low	✓ Fixed
FP-10	The `launch` function can be called repeatedly	Design Vulnerability	Low	✓ Fixed
FP-11	Useless `getLiquidityBacking` function	Code Improvement	Info	✓ Fixed

09. Issue descriptions

[FP-1] Token Selling Restricted

Token Issuance High Acknowledged

Issue/Risk: Token Issuance

Description:

1. There is a ten-day sale's limit. If this limit is turned on, a user can only sell a maximum of 20% of SWYCHs every 10 days according to the proportion of his/her balance to the balance of a specified token pair.
   This is equivalent to a token locking mechanism where a user sells SWYCHs for a consecutive 50 days but still can not sell them all. `lastAvailableTradingBalanceAmount` is calculated based on the current balance. Let's say we have 100 tokens and the ten-day limit is 20%, then we can only sell 20 tokens in the first ten days and sell 16 tokens in the second ten-days . Although users can transfer the remaining tokens to other accounts to continue their selling operations, those selling

operations will be taxed and cost gas fees thus resulting in extra costs to users.

Recommendation:

If the ten-day limit is kept it might cause relatively economic risks to token holders.
Therefore this issue is marked as high-severity. Consider removing this restriction whenever appropriate in the future.

Update:

The Ten-day Take Profit feature is one of the core mechanisms of Swych and is set in place in order to guarantee sustainable daily passive income for the users. It prevents whales from dumping and makes sure everybody can take their share every day.

Status:

The Swych team has acknowledged this issue.

[FP-2] Unlimited Token Issuance

Token Issuance Medium Acknowledged

Issue/Risk: Token Issuance

Description:

In the current contract, tokens can be issued additionally and there is no cap on issuance,which may cause losses to token holders in certain scenarios.

Recommendation:

Consider setting a cap on token issuance.

Update:

The Swych team prefers to keep it now and will improve the code in the future.

Status:

The Swych team has acknowledged this issue.

[FP-3] Relatively High transferFee

Token Issuance Medium Acknowledged

Issue/Risk: Token Issuance

Description:

transferFee 's rate is relatively high.
When a user's transfer his SWYCH to others, the initial rate for TransferFee is 35% and can be set up to 50%

Recommendation:

Consider setting a relatively low rate instead of one as high as 50%.

Update:

The Transfer Fees are necessary to prevent that users play an equal part in the protocol and the same rules apply to every holder. The lack of a Transfer Fee (or an excessively low fee) would allow large holders to escape the DTP Ratio by splitting their wallets into smaller chunks, therefore increasing their total DTP Ratio at the expense of smaller holders. At the same time, any user who wishes to move their entire SWYCH holding from one wallet to another will be able to do this how many times they like and without paying any fee at all,

given that they transfer 100% of their tokens. This type of feeless transfers are designed for

- and especially useful to - users in case of a potentially compromised wallet or if they wish to move their SWYCH tokens to a hardware wallet for an added layer of security.

Status:

The Swych team has acknowledged this issue.

[FP-4] Excessive Access Control

Admin Rights Medium Acknowledged

Issue/Risk: Admin Rights

Description:

In both `Swych.sol` and `sSwych.sol`, all parameters can be changed by the admin. And the admin can pause token transactions, staking and withdrawals. This access control is too excessive.

Recommendation:

Consider removing part of the access control.

Update:

These management functions are to modify relevant parameters in time to make the contract run better when the environment changes, and some core parameter settings will be carefully changed.

Status:

The Swych team has acknowledged this issue.

[FP-5] Token transfers can be paused

Token Issuance Low Acknowledged

Issue/Risk: Token Issuance

Description:

In the current contract, token transfers can be paused, which may cause losses to token holders in certain scenarios.

Recommendation:

Consider properly using this function or removing it.

Update/Status:

The Swych team replied that it is a necessary function.

[FP-6] `athPrice` Can Only Increase Monotonically

Design Vulnerability Low Acknowledged

Issue/Risk: Design Vulnerability

Description:

Here the calculation formula of athPrice is

```
uint256 newPrice = _getTokenPriceInWBNB();

      if (newPrice > athPrice) {

        uint256 lastAthPrice = athPrice;

        athPrice = newPrice;

        emit NewAllTimeHigh(lastAthPrice, newPrice);

      }
```

If the price's initial value (such as artificially controlling the price) is high, athPrice 's value may no longer decrease. And this value will affect the subsequent deflation calculation. When its value is large, deflation is very huge.

Recommendation:

Consider changing this function.

Update/Status:

The Swych team replied that it is a necessary function.

[FP-7] Incorrect Use of isContract

Implementation Vulnerability | Medium | ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

Contract `sSwych` uses `isContract` to check if an address is a non-contract address. However when a contract is being contructed in its constructor, this check will fail.

Recommendation:

Consider using `require(msg.sender == tx.origin, "NOT_AN_EOA")` to replace `require(!isContract(msg.sender), "NOT_AN_EOA");`
For more details please refer to https://despac1to.medium.com/carefully-use-openzeppelins-address-iscontract-msg-sender-4136cc6ff66d

Update/Status:

The Swych team has fixed this issue.

[FP-8] Typo

Code Improvement | Low | ✓ Fixed

Issue/Risk: Code Improvement

Description:

In `Swych.sol`, the `setRewardRate` function's code `require(rewardRate > 0, "INVALID_REWARD_RATE");` should be e`require(rate > 0, "INVALID_REWARD_RATE");`

And `rate <= denominator` is required.

Recommendation:

Consider using `require(msg.sender == tx.origin, "NOT_AN_EOA")` to replace `require(!isContract(msg.sender), "NOT_AN_EOA");`
For more details please refer to https://despac1to.medium.com/carefully-use-openzeppelins-address-iscontract-msg-sender-4136cc6ff66d

Update:

Consider making changes as shown above.

Status:

The Swych team has fixed this issue.

[FP-9] Missing event argument

Code Improvement | Low | ✓ Fixed

Issue/Risk: Code Improvement

Description:

In `sSwych.sol`, the `setRewardRate` function triggers an event. However the event might miss an argument.

Recommendation:

It should take three arguments instead of two.

Update:

Consider making changes as shown above.

Status:

The Swych team has fixed this issue.

[FP-10] The `launch` function can be called repeatedly

Design Vulnerability | Low | ✓ Fixed

Issue/Risk: Design Vulnerability

Description:

In `Swych.sol`, repeatedly calling `launch` would reset `lastRewardTime`.

Recommendation:

Consider calling it with great caution or add a limit that can only be called once.

Update:

A require to not make it possible to call `launch` again was added.

Status:

The Swych team has fixed this issue.

[FP-11] Useless `getLiquidityBacking` function

Code Improvement | Info | ✓ Fixed

Issue/Risk: Code Improvement

Description:

In `Swych.sol`, the `getLiquidityBacking` function seems useless.

Recommendation:

Consider removing it.

Update:

Removed.

Status:

The Swych team has fixed this issue.

10. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- Consider managing the owner's access control with great care and transfering it to a multi-sig wallet or DAO when necessary.

11. Appendices

11.1 Unit Test

1. Swych_test.js

```
1
2  const { expect,assert } = require("chai");
3  const { ethers } = require("hardhat");
4  const { Instruction } = require("hardhat/internal/hardhat-network/stack-traces/model");
5
6  describe("Swych Token Unit Test", function () {
7      let instance;
8      let owner, user1, user2, users;
9      let factory, router;
10     let weth,pair;
11     const one_day = 24 * 3600;
12     const init_supply = ethers.constants.WeiPerEther.mul(10**8);
13     const dead_address = "0x000000000000000000000000000000000000dEaD";
14
15     before(async () => {
16         [owner, user1, user2, ...users] = await ethers.getSigners();
17     });
18
19     async function deployUniswapAndToken() {
```

```
        // deploy weth
        const WETH9 = await ethers.getContractFactory("WETH9");
        weth = await WETH9.deploy();
        // deploy uniswap
        const UniswapV2Factory = await ethers.getContractFactory("UniswapV2Factory");
        factory = await UniswapV2Factory.deploy(owner.address);
        const UniswapV2Router02 = await ethers.getContractFactory("UniswapV2Router02");
        router = await UniswapV2Router02.deploy(factory.address, weth.address);
        const ERC1967Proxy = await ethers.getContractFactory("ERC1967Proxy");
        const Swych = await ethers.getContractFactory("Swych");
        let implement = await Swych.deploy();

        let proxy = await ERC1967Proxy.deploy(implement.address,"0x");
        let args = [
            router.address,weth.address,users[0].address,users[1].address
        ];
        instance = Swych.attach(proxy.address);
        await instance.initialize(...args);
        pair = await factory.getPair(weth.address,instance.address);

        await expect(implement.initialize(...args)).to.be.revertedWith("Initializable: contract is
already initialized");
    }

    async function getAmountOut(amountIn) {
        if(amountIn.eq(0)) {
            return ethers.constants.Zero;
        }
        let pair_contract_factory = await ethers.getContractFactory("UniswapV2Pair");
        let pair_contract = pair_contract_factory.attach(pair);
        let [reserve0,reserve1,] = await pair_contract.getReserves();
        let token0 = await pair_contract.token0();
        let [reserveIn,reserveOut] = token0 === instance.address ? [reserve0,reserve1] :
[reserve1,reserve0];
        let amountInWithFee = amountIn.mul(997);
        let numerator = amountInWithFee.mul(reserveOut);
        let denominator = reserveIn.mul(1000).add(amountInWithFee);
        return numerator.div(denominator);
    }

    async function addLiquid() {
        let balance = await instance.balanceOf(owner.address);
        let amount = balance.div(2);
        await instance.approve(router.address, ethers.constants.MaxUint256);
        await router.addLiquidityETH(instance.address,amount,1,1,owner.address,9876543210,{
            value:ethers.utils.parseEther("100")
        });
    }

    beforeEach(async () => {
        await deployUniswapAndToken();
    });

    describe("Initial State Unit Test", () => {
        it("Init twice should be failed", async () => {
            let args = [
                router.address,weth.address,users[0].address,users[1].address
            ];
            await expect(instance.initialize(...args)).to.be.revertedWith(
```

```
                "Initializable: contract is already initialized"
            );
        });
        it("Initial State should be checked", async () => {
            expect(await instance.athPriceDeltaPermille()).to.be.equal(10);
            expect(await instance.autoReward()).to.be.equal(false);
            expect(await instance.swapBackEnabled()).to.be.equal(false);
            expect(await instance.availableTradingBalanceEnabled()).to.be.equal(false);
            expect(await instance.transferFeeEnabled()).to.be.equal(false);
            expect(await instance.priceEnabled()).to.be.equal(false);

            expect(await instance.lastRewardTime()).to.be.equal(0);
            expect(await instance.launched()).to.be.equal(false);
            // meta
            expect(await instance.name()).to.be.equal("Swych");
            expect(await instance.symbol()).to.be.equal("SWYCH");
            expect(await instance.decimals()).to.be.equal(18);
            expect(await instance.getVersion()).to.be.equal("1.0");
            // supply and balance
            expect(await instance.totalSupply()).to.be.equal(init_supply);
            expect(await instance.balanceOf(owner.address)).to.be.equal(init_supply);
        });
    });

    describe("Interface of owner unit test", () => {
        it("launch should change state and emit event", async () => {
            await expect(instance.launch()).to.be.revertedWith("LIQUIDITY_NOT_ADDED");
            await addLiquid();
            await
expect(instance.launch()).to.be.revertedWith("INSUFFICIENT_WBNB_ALLOWANCE_FROM_LIQUIDITY_RECEIVER")
;
            await weth.connect(users[0]).approve(instance.address,ethers.constants.MaxUint256);
            let block = await ethers.provider.getBlockNumber();
            let {timestamp} = await ethers.provider.getBlock(block);
            await expect(instance.launch()).to.be.emit(
                instance,"Launched"
            ).withArgs(timestamp + 1);
            // check state
            expect(await instance.autoReward()).to.be.equal(true);
            expect(await instance.swapBackEnabled()).to.be.equal(true);
            expect(await instance.availableTradingBalanceEnabled()).to.be.equal(true);
            expect(await instance.transferFeeEnabled()).to.be.equal(true);
            expect(await instance.priceEnabled()).to.be.equal(true);
            expect(await instance.lastRewardTime()).to.be.equal(timestamp + 1);
            expect(await instance.launched()).to.be.equal(true);

            // launch again
            await expect(instance.launch()).to.be.revertedWith("ALREADY_LAUNCHED");
        });

        it("SetAthDeltaPermille should change state and emit event", async () => {
            await expect(instance.setAthDeltaPermille(0)).to.be.revertedWith("INVALID_PERMILLE");
            await
expect(instance.setAthDeltaPermille(1000)).to.be.revertedWith("INVALID_PERMILLE");
            await expect(instance.setAthDeltaPermille(500)).to.be.emit(
                instance,"SetAthDeltaPerMille"
            ).withArgs(500);
            expect(await instance.athPriceDeltaPermille()).to.be.equal(500);
        });
```

```
133
134         it("withdrawFeesToTreasury unit test", async () => {
135             await addLiquid();
136             await instance.transfer(instance.address,ethers.utils.parseUnits("500.0",18));
137             let balance = await instance.balanceOf(instance.address);
138             expect(balance).to.be.equal(ethers.utils.parseUnits("500.0",18));
139             // launch
140             await weth.connect(users[0]).approve(instance.address,ethers.constants.MaxUint256);
141             await instance.launch();
142             // withdraw
143             let tx = await instance.withdrawFeesToTreasury();
144             let receipts = await ethers.provider.getTransactionReceipt(tx.hash);
145             const {logs} = receipts;
146             const Swych = await ethers.getContractFactory("Swych");
147             for(let log of logs) {
148                 if(log.address === instance.address && log.topics.length === 1) {
149                     try {
150                         let {amount} =
    Swych.interface.decodeEventLog("WithdrawFeesToTreasury",log.data);
151                         console.log("data:",amount,ethers.utils.formatEther(amount));
152                     }catch(e) {
153                         continue;
154                     }
155                 }
156             }
157         });
158
159         it("setAuthorizedToMint unit test", async () => {
160             expect(await instance.authorizedToMint(user1.address)).to.be.false;
161             await instance.setAuthorizedToMint(user1.address,true);
162             expect(await instance.authorizedToMint(user1.address)).to.be.true;
163             await instance.setAuthorizedToMint(user1.address,false);
164             expect(await instance.authorizedToMint(user1.address)).to.be.false;
165         });
166
167         it("setBlockedPair unit test", async () => {
168             expect(await
    instance.blockedPairs(user1.address)).to.be.revertedWith("NOT_A_CONTRACT");
169             expect(await instance.blockedPairs(pair)).to.be.false;
170             await expect(instance.setBlockedPair(pair,true)).to.be.emit(
171                 instance,"SetBlockedPair"
172             ).withArgs(pair,true);
173             expect(await instance.blockedPairs(pair)).to.be.true;
174             await instance.setBlockedPair(pair,false);
175             expect(await instance.blockedPairs(pair)).to.be.false;
176         });
177
178         it("setRewardEnabled unit test", async () => {
179             expect(await instance.rebaseEnabled()).to.be.false;
180             expect(await instance.reboundEnabled()).to.be.false;
181             await expect(instance.setRewardEnabled(true,true)).to.be.emit(
182                 instance,"SetRewardEnabled"
183             ).withArgs(true,true);
184             expect(await instance.rebaseEnabled()).to.be.true;
185             expect(await instance.reboundEnabled()).to.be.true;
186         });
187
188         it("setPriceEnabled unit test", async () => {
189             expect(await instance.priceEnabled()).to.be.false;
```

```
            await expect(instance.setPriceEnabled(true)).to.be.emit(
                instance,"SetPriceEnabled"
            ).withArgs(true);
            expect(await instance.priceEnabled()).to.be.true;
        });

        it("setRewardFrequency unit test", async () => {
            await
expect(instance.setRewardFrequency(0)).to.be.revertedWith("INVALID_REWARD_FREQUENCY");
            await expect(instance.setRewardFrequency(one_day +
1)).to.be.revertedWith("INVALID_REWARD_FREQUENCY");
            await expect(instance.setRewardFrequency(300)).to.be.emit(
                instance,"SetRewardFrequency"
            ).withArgs(300);
            expect(await instance.rewardFrequency()).to.be.equal(300);
        });

        it("setAutoReward unit test", async () => {
            expect(await instance.autoReward()).to.be.false;
            await expect(instance.setAutoReward(true)).to.be.emit(
                instance,"SetAutoReward"
            ).withArgs(true);
            expect(await instance.autoReward()).to.be.true;
        });

        it("setAvailableTradingBalanceEnabled unit test", async () => {
            expect(await instance.availableTradingBalanceEnabled()).to.be.false;
            await expect(instance.setAvailableTradingBalanceEnabled(true)).to.be.emit(
                instance,"SetAvailableTradingBalanceEnabled"
            ).withArgs(true);
            expect(await instance.availableTradingBalanceEnabled()).to.be.true;
        });

        it("setNoCheckAvailableTradingBalance unit test", async () => {
            await
expect(instance.setNoCheckAvailableTradingBalance(user1.address,true)).to.be.emit(
                instance,"SetNoCheckAvailableTradingBalance"
            ).withArgs(user1.address,true);

        });

        it("setTransferFeeEnabled unit test", async () => {
            expect(await instance.transferFeeEnabled()).to.be.false;
            await expect(instance.setTransferFeeEnabled(true)).to.be.emit(
                instance,"SetTransferFeeEnabled"
            ).withArgs(true);
            expect(await instance.transferFeeEnabled()).to.be.true;
        });

        it("setSwapBackEnabled unit test", async () => {
            expect(await instance.swapBackEnabled()).to.be.false;
            await expect(instance.setSwapBackEnabled(true)).to.be.emit(
                instance,"SetSwapBackEnabled"
            ).withArgs(true);
            expect(await instance.swapBackEnabled()).to.be.true;
        });

        it("setCollectedFeeThreshold unit test", async () => {
            await expect(instance.setCollectedFeeThreshold(500)).to.be.emit(
```

```
246                 instance,"SetCollectedFeeThreshold"
247             ).withArgs(500);
248             expect(await instance.checkCollectedFeeThreshold()).to.be.equal(500);
249         });
250 
251         it("setAvailableTradingBalanceCoefficients unit test", async () => {
252            await expect(instance.setAvailableTradingBalanceCoefficients(50000,2000)).to.be.emit(
253             instance,"SetAvailableTradingBalanceCoefficients"
254            ).withArgs(50000,2000);
255            expect(await instance.coefficientA()).to.be.equal(50000);
256            expect(await instance.tradingBalanceDenominator()).to.be.equal(2000);
257         });
258 
259         it("setRangeHoldingPermyriadAvailableTradingBalanceApplied unit test" ,async () => {
260             await
expect(instance.setRangeHoldingPermyriadAvailableTradingBalanceApplied(10001,2000)).to.be.revertedW
ith(
261                 "INVALID_PERMYRIAD"
262             );
263             await
expect(instance.setRangeHoldingPermyriadAvailableTradingBalanceApplied(100,20000)).to.be.revertedWi
th(
264                 "INVALID_PERMYRIAD"
265             );
266             await
expect(instance.setRangeHoldingPermyriadAvailableTradingBalanceApplied(2000,1000)).to.be.revertedWi
th(
267                 "INVALID_RANGE"
268             );
269             await
expect(instance.setRangeHoldingPermyriadAvailableTradingBalanceApplied(1000,2000)).to.be.emit(
270                 instance,"SetRangeHoldingPermyriadAvailableTradingBalanceApplied"
271             ).withArgs(1000,2000);
272             expect(await
instance.minHoldingPermyriadAvailableTradingBalanceApplied()).to.be.equal(1000);
273             expect(await
instance.maxHoldingPermyriadAvailableTradingBalanceApplied()).to.be.equal(2000);
274         });
275 
276         it("setReboundTriggerFromAth unit test", async () => {
277             await expect(instance.setReboundTriggerFromAth(1000,2000)).to.be.emit(
278                 instance,"SetReboundFromAth"
279             ).withArgs(1000,2000);
280             expect(await instance.negativeFromAthPercent()).to.be.equal(1000);
281             expect(await instance.negativeFromAthPercentDenominator()).to.be.equal(2000);
282         });
283 
284         it("setRewardRate unit test", async () => {
285             expect(await instance.rewardRate()).to.be.equal(2073);
286             expect(await instance.rewardRateDenominator()).to.be.equal(10**7);
287             await expect(instance.setRewardRate(12073,10**8)).to.be.emit(
288                 instance,"SetRewardRate"
289             ).withArgs(12073,10**8);
290             expect(await instance.rewardRate()).to.be.equal(12073);
291             expect(await instance.rewardRateDenominator()).to.be.equal(10**8);
292         });
293 
294         it("setTreasuryWallet unit test", async () => {
295             expect(await instance.treasury()).to.be.equal(users[1].address);
```

```
            await expect(instance.setTreasuryWallet(users[2].address)).to.be.emit(
                instance,"SetTreasuryWallet"
            ).withArgs(users[2].address);
            expect(await instance.treasury()).to.be.equal(users[2].address);
        });

        it("setAutoLiquidityReceiver unit test", async () => {
            expect(await instance.autoLiquidityReceiver()).to.be.equal(users[0].address);
            await expect(instance.setAutoLiquidityReceiver(users[3].address)).to.be.emit(
                instance,"SetAutoLiquidityReceiver"
            ).withArgs(users[3].address);
            expect(await instance.autoLiquidityReceiver()).to.be.equal(users[3].address);
        });

        it("setFeeExemptAddress", async () => {
            let user = users[3].address;
            expect(await instance.isFeeExempt(user)).to.be.false;
            await expect(instance.setFeeExemptAddress(user,true)).to.be.emit(
                instance,"SetFeeExemptAddress"
            ).withArgs(user,true);
            expect(await instance.isFeeExempt(user)).to.be.true;
            await expect(instance.setFeeExemptAddress(user,false)).to.be.emit(
                instance,"SetFeeExemptAddress"
            ).withArgs(user,false);
            expect(await instance.isFeeExempt(user)).to.be.false;
        });

        it("setBackingLPToken unit test", async () => {
            expect(await instance.pair()).to.be.equal(pair);
            await expect(instance.setBackingLPToken(users[3].address)).to.be.emit(
                instance,"SetBackingLPToken"
            ).withArgs(users[3].address);
            expect(await instance.pair()).to.be.equal(users[3].address);
        });

        it("pause and unpause unit test", async () => {
            expect(await instance.paused()).to.be.false;
            await expect(instance.pause()).to.be.emit(
                instance,"Pause"
            );
            expect(await instance.paused()).to.be.true;
            await expect(instance.unpause()).to.be.emit(
                instance,"Unpause"
            );
            expect(await instance.paused()).to.be.false;
        });

        it("rescueToken unit test",async () => {
            await
expect(instance.rescueToken(instance.address,0)).to.be.revertedWith("CANNOT_WITHDRAW_SWYCH");
            await weth.deposit({
                value:100
            });
            // transfer token to instance
            await weth.transfer(instance.address,100);
            expect(await weth.balanceOf(instance.address)).to.be.equal(100);
            expect(await weth.balanceOf(owner.address)).to.be.equal(0);
            // rescueToken
            await instance.rescueToken(weth.address,100);
```

```
            expect(await weth.balanceOf(instance.address)).to.be.equal(0);
            expect(await weth.balanceOf(owner.address)).to.be.equal(100);
        });

        it("setFeeSplit unit test", async () => {
            let [fee0,fee1,fee2] = [20,30,50];
            fee2  += 10;
            await
expect(instance.setFeeSplit(fee0,fee1,fee2)).to.be.revertedWith("INVALID_FEE_SPLIT");
            fee2 -= 10;
            await expect(instance.setFeeSplit(fee0,fee1,fee2)).to.be.emit(
                instance,"SetFeeSplit"
            ).withArgs(fee0,fee1,fee2);
            expect(await instance.autoLiquidityFeePercent()).to.be.equal(fee0);
            expect(await instance.treasuryFeePercent()).to.be.equal(fee1);
            expect(await instance.burnFeePercent()).to.be.equal(fee2);
        });

        it("setFees unit test", async () => {
            await expect(instance.setFees(10,20,30)).to.be.emit(
                instance,"SetFees"
            ).withArgs(10,20,30);
            expect(await instance.buyFee()).to.be.equal(10);
            expect(await instance.sellFee()).to.be.equal(20);
            expect(await instance.transferFee()).to.be.equal(30);
        });

        it("setAutomatedMarketMakerPair unit test", async () => {
            let pairs = users.slice(0,7).map(user => user.address);
            // add
            for(let i=0;i<pairs.length;i++) {
                let pair_address = pairs[i];
                expect(await instance.automatedMarketMakerPairs(pair_address)).to.be.false;
                await expect(instance.setAutomatedMarketMakerPair(pair_address,true)).to.be.emit(
                    instance,"SetAutomatedMarketMakerPair"
                ).withArgs(pair_address,true);
                expect(await instance.automatedMarketMakerPairs(pair_address)).to.be.true;
            }
            //
            for(let i=0;i<pairs.length;i++) {
                expect(await instance.makerPairs(i + 1)).to.be.equal(pairs[i]);
            }
            // remove
            let remove_address = pairs[2];
            await instance.setAutomatedMarketMakerPair(remove_address,false);
            expect(await instance.makerPairs(3)).to.be.equal(pairs[pairs.length -1]);
        });
    });

    describe("View function unit test", () => {
        it("getAvailableTradingBalanceFactor unit test", async () => {
            // test calculateAvailableTradingBalanceFactor
            let min_applied = await instance.minHoldingPermyriadAvailableTradingBalanceApplied();
 // 100
            let max_applied = await instance.maxHoldingPermyriadAvailableTradingBalanceApplied();
 // 1000
            let min_factor = await instance.MIN_AVAILABLE_TRADING_BALANCE_FACTOR(); // 200
            let max_factor = await instance.MAX_AVAILABLE_TRADING_BALANCE_FACTOR(); // 2000
            let coefficientA = await instance.coefficientA(); //200000
```

```
410
411             function get_value(value,min,max) {
412                 value = ethers.BigNumber.from("" + value);
413                 return value.gt(max) ? max : ( value.lt(min) ? min : value);
414             }
415             let counter = 0;
416             for(let i=0;i<100;i++) {
417                 let rand = parseInt(Math.random() * 10000);
418                 if(rand > 100 && rand < 1000) {
419                     counter ++;
420                 }
421                 let applied = get_value(rand,min_applied,max_applied);
422                 let factor = get_value(coefficientA.div(applied),min_factor,max_factor);
423                 expect(await
    instance.calculateAvailableTradingBalanceFactor(rand)).to.be.equal(factor);
424             }
425             assert(counter > 0,"no matched number");
426
427             // test getAvailableTradingBalanceFactor
428             expect(await instance.getAvailableTradingBalanceFactor(owner.address)).to.be.equal(0);
429             await addLiquid();
430
431             expect(await
    instance.getAvailableTradingBalanceFactor(owner.address)).to.be.equal(min_factor);
432         });
433
434         it("getAvailableTradingBalanceAmount unit test", async () => {
435             await addLiquid();
436             let bal = await instance.balanceOf(owner.address);
437             let min_factor = await instance.MIN_AVAILABLE_TRADING_BALANCE_FACTOR(); // 200
438             let denominator = await instance.tradingBalanceDenominator();
439             let amount = bal.mul(min_factor).div(denominator);
440             expect(await
    instance.getAvailableTradingBalanceAmount(owner.address)).to.be.equal(amount);
441         });
442
443
444         it("getTriggerReboundPrice unit test", async () => {
445             await addLiquid();
446             await weth.connect(users[0]).approve(instance.address,ethers.constants.MaxUint256);
447             await instance.launch();
448             await instance.transfer(user1.address,10000);
449             await instance.connect(user1).transfer(user2.address, 4000);
450             let athPrice = await instance.athPrice();
451             let negativeFromAthPercent = 7;
452             let negativeFromAthPercentDenominator = 100;
453             let new_price =
    athPrice.sub(athPrice.mul(negativeFromAthPercent).div(negativeFromAthPercentDenominator));
454             expect(await instance.getTriggerReboundPrice()).to.be.equal(new_price);
455         });
456
457         it("getLiquidityBacking unit test",async () => {
458             // todo
459         });
460     });
461
462     describe("Reward unit test", () => {
463         it("Reward while price is reboundEnabled ", async () => {
464             await addLiquid();
```

```
            await instance.setRewardEnabled(true,true);
            await weth.connect(users[0]).approve(instance.address,ethers.constants.MaxUint256);
            await instance.launch();

            let block = await ethers.provider.getBlockNumber();
            let { timestamp } = await ethers.provider.getBlock(block);
            expect(await instance.lastRewardTime()).to.be.equal(timestamp);
            expect(await instance.totalSupplyIncludingBurnAmount()).to.be.equal(init_supply);
            let athPrice = await instance.athPrice();
            expect(athPrice).to.be.equal(0);
            let priceEnabled = await instance.priceEnabled();
            expect(priceEnabled).to.be.true;
            await instance.transfer(user1.address,10000);
            let new_price = await instance.athPrice();
            let price = await getAmountOut(ethers.constants.WeiPerEther);
            expect(price).to.be.equal(new_price);
            await instance.setNoCheckAvailableTradingBalance(owner.address,true);
            let amountIn = init_supply.div(4);
            await router.swapExactTokensForETHSupportingFeeOnTransferTokens(amountIn,1,[
                instance.address,weth.address],owner.address,9876543210
            );
            await expect(instance.reward()).to.be.revertedWith("REWARD_TOO_SOON");
            let target =  timestamp + 30 * 60 + 2;
            await ethers.provider.send("evm_mine", [target]);
            new_price = await instance.athPrice();
            let expect_auth_price = new_price.mul(1).div(100).add(new_price);
            expect(await instance.lastReboundTriggerAthPrice()).to.be.equal(0);
            await instance.reward();
            let auth_price = await instance.athPrice();
            if(expect_auth_price.gt(auth_price)) {
                expect(expect_auth_price.sub(auth_price)).to.be.lte(1)
            } else {
                expect(auth_price.sub(expect_auth_price)).to.be.lte(1)
            }
        });

        it("Reward while price is rebaseEnabled ", async () => {
            await addLiquid();
            await instance.setRewardEnabled(true,true);
            await weth.connect(users[0]).approve(instance.address,ethers.constants.MaxUint256);
            await instance.launch();

            let block = await ethers.provider.getBlockNumber();
            let { timestamp } = await ethers.provider.getBlock(block);
            expect(await instance.lastRewardTime()).to.be.equal(timestamp);
            expect(await instance.totalSupplyIncludingBurnAmount()).to.be.equal(init_supply);
            let athPrice = await instance.athPrice();
            expect(athPrice).to.be.equal(0);
            let priceEnabled = await instance.priceEnabled();
            expect(priceEnabled).to.be.true;
            await instance.transfer(user1.address,10000);
            let new_price = await instance.athPrice();
            let price = await getAmountOut(ethers.constants.WeiPerEther);
            expect(price).to.be.equal(new_price);
            await router.swapExactETHForTokensSupportingFeeOnTransferTokens(1,[
                weth.address,instance.address],owner.address,9876543210, {
                    value:ethers.utils.parseEther("1")
                }
            );
```

```
            await expect(instance.reward()).to.be.revertedWith("REWARD_TOO_SOON");
            let target =  timestamp + 30 * 60 + 2;
            await ethers.provider.send("evm_mine", [target]);
            await instance.reward();
            let rewardRate = 2073;
            let rewardRateDenominator = 10**7;
            expect(await
instance.totalSupplyIncludingBurnAmount()).to.be.equal(init_supply.mul(rewardRate +
rewardRateDenominator).div(rewardRateDenominator));
            expect(await instance.balanceOf(user1.address)).to.be.equal(parseInt(10000 *
(rewardRate + rewardRateDenominator) / rewardRateDenominator));
        });
    });

    describe("Transfer unit test", () => {
        it("User Transfer to User test", async () => {
            await instance.setBlockedPair(router.address,true);
            await
expect(instance.transfer(router.address,10000)).to.be.revertedWith("BLOCKED_DEX_PAIR");
            await instance.transfer(user1.address,10000);
            expect(await instance.balanceOf(user1.address)).to.be.equal(10000);
            await expect(instance.connect(user1).transfer(user2.address,60)).to.be.revertedWith(
                "TOKEN_NOT_LAUNCHED_YET"
            );
            await addLiquid();
            await weth.connect(users[0]).approve(instance.address,ethers.constants.MaxUint256);
            await instance.launch();

            await instance.connect(user1).transfer(user2.address, 4000);
            expect(await instance.balanceOf(user1.address)).to.be.equal(6000);
            expect(await instance.balanceOf(user2.address)).to.be.equal(4000/100 * 65);


            let burn_fee = 350 * 4 * 20 / 100;
            expect(await instance.balanceOf(dead_address)).to.be.equal(burn_fee);
            expect(await instance.balanceOf(instance.address)).to.be.equal(280 * 4);
            expect(await instance.totalSupply()).to.be.equal(init_supply.sub(burn_fee));
            expect(await instance.totalSupplyIncludingBurnAmount()).to.be.equal(init_supply);
        });

        it("User sell token test", async () => {
            let amountIn = 1000000000;
            await instance.transfer(user1.address,amountIn);
            await addLiquid();
            await weth.connect(users[0]).approve(instance.address,ethers.constants.MaxUint256);
            await instance.launch();
            let trading_amount = await
instance.getCurrentAvailableTradingBalanceAmount(user1.address);
            // get sell limit;
            let bal_can_trade = await instance.getAvailableTradingBalanceAmount(user1.address);
            expect(trading_amount).to.be.equal(bal_can_trade);
            let limit = amountIn * 2000 / 10000;
            expect(bal_can_trade).to.be.equal(limit);
            await instance.connect(user1).approve(router.address,ethers.constants.MaxUint256);
            let args = [
                amountIn,
                1,
                [instance.address,weth.address],
                user1.address,
```

```
                9876543210
            ]
            expect(await instance.availableTradingBalanceEnabled()).to.be.true;
            await
expect(router.connect(user1).swapExactTokensForETHSupportingFeeOnTransferTokens(...args)).to.be.rev
erted;
            args[0] = args[0] / 10;
            let block = await ethers.provider.getBlockNumber();
            let {timestamp} = await ethers.provider.getBlock(block);
            await
router.connect(user1).swapExactTokensForETHSupportingFeeOnTransferTokens(...args);
            let {lastAvailableTradingBalanceAmount,lastSoldTimestamp,totalSoldAmountLast24h} =
await instance.saleHistories(user1.address);
            expect(lastSoldTimestamp).to.be.equal(timestamp + 1);

 expect(totalSoldAmountLast24h).to.be.equal(ethers.constants.WeiPerEther.mul(amountIn/10));

 expect(lastAvailableTradingBalanceAmount).to.be.equal(ethers.constants.WeiPerEther.mul(limit));

            // sell again
            args[0] = args[0] / 10;
            await
router.connect(user1).swapExactTokensForETHSupportingFeeOnTransferTokens(...args);
            ({lastAvailableTradingBalanceAmount,totalSoldAmountLast24h} = await
instance.saleHistories(user1.address));

 expect(lastAvailableTradingBalanceAmount).to.be.equal(ethers.constants.WeiPerEther.mul(limit));

 expect(totalSoldAmountLast24h).to.be.equal(ethers.constants.WeiPerEther.mul(amountIn/100 * 11));
            // beyond limit
            args[0] = args[0] * 10;
            await
expect(router.connect(user1).swapExactTokensForETHSupportingFeeOnTransferTokens(...args)).to.be.rev
erted;
            block = await ethers.provider.getBlockNumber();
            ({timestamp} = await ethers.provider.getBlock(block));

            trading_amount = await instance.getCurrentAvailableTradingBalanceAmount(user1.address);
            expect(trading_amount).to.be.equal(bal_can_trade);
            // limit before sell
            let bal_source = await instance.balanceOf(user1.address);
            limit = bal_source.mul(20).div(100);
            let bal_limit = await instance.getAvailableTradingBalanceAmount(user1.address);
            expect(bal_limit).to.be.equal(limit);
            let target = timestamp + 10 * 24 * 3600 + 10;

            await ethers.provider.send("evm_mine", [target])
            await
router.connect(user1).swapExactTokensForETHSupportingFeeOnTransferTokens(...args);
            ({lastAvailableTradingBalanceAmount,lastSoldTimestamp,totalSoldAmountLast24h} = await
instance.saleHistories(user1.address));

 expect(lastAvailableTradingBalanceAmount).to.be.equal(ethers.constants.WeiPerEther.mul(bal_limit))
;

 expect(totalSoldAmountLast24h).to.be.equal(ethers.constants.WeiPerEther.mul(amountIn/10));
            expect(lastSoldTimestamp).to.be.equal(target + 1);
            trading_amount = await instance.getCurrentAvailableTradingBalanceAmount(user1.address);
            expect(trading_amount).to.be.equal(bal_limit);
```

```
        });
    });

    describe("Unit test of Others", () => {
        it("Call manualSync should emit event", async () => {
            let pair_contract_factory = await ethers.getContractFactory("UniswapV2Pair");
            let pair_contract = pair_contract_factory.attach(pair);
            await expect(instance.manualSync()).to.be.emit(
                pair_contract,"Sync"
            );
        });
        it("Mint should be failed without authorized",async () => {
            await expect(instance.mint(user1.address,1000000)).to.be.revertedWith(
                "UNAUTHORIZED_TO_MINT"
            );
        });

        it("Mint should change balance and totalSupply", async () => {
            await instance.setAuthorizedToMint(user1.address,true);
            await expect(instance.connect(user1).mint(user1.address,10000000)).to.be.emit(
                instance,"Transfer"
            ).withArgs(ethers.constants.AddressZero,user1.address,10000000);
            expect(await instance.balanceOf(user1.address)).to.be.equal(10000000);
            expect(await instance.totalSupply()).to.be.equal(init_supply.add(10000000));
        });

        it("decreaseAllowance and increaseAllowance unit test", async () => {
            await instance.increaseAllowance(user1.address,100000);
            expect(await instance.allowance(owner.address,user1.address)).to.be.equal(100000);
            await instance.decreaseAllowance(user1.address,10000);
            expect(await instance.allowance(owner.address,user1.address)).to.be.equal(90000);
            await weth.connect(users[0]).approve(instance.address,ethers.constants.MaxUint256);
            await addLiquid();
            await instance.launch();
            await instance.connect(user1).transferFrom(owner.address,user2.address,10000);
            expect(await instance.allowance(owner.address,user1.address)).to.be.equal(80000);
        });
    });

    describe("Airdrop unit test", async () => {
        it("Airdrop before launched should be failed", async () => {
            await weth.connect(users[0]).approve(instance.address,ethers.constants.MaxUint256);
            await addLiquid();
            await instance.launch();
            let args = [
                users.map(user => user.address),users.map(() => 100000000)
            ];
            await expect(instance.airdrop(...args)).to.be.revertedWith("TOKEN_ALREADY_LAUNCHED");
        });

        it("Airdrop should transfer tokens", async () => {
            let args = [
                users.map(user => user.address),users.map(() => 100000000)
            ];
            await instance.airdrop(...args);
            for(let i=0;i<users.length;i++) {
                let bal = await instance.balanceOf(users[i].address);
                expect(bal).to.be.equal(100000000);
            }
```

```
        });
    });
});
```

2. sSwych_test.js

```
const { expect,assert } = require("chai");
const { ethers } = require("hardhat");

describe("sSwych Unit Test", function () {
    let instance;
    let swych;
    let owner, user1, user2, users;
    let factory, router;
    let weth,pair;
    const dead_address = "0x000000000000000000000000000000000000dEaD";
    const one_day = 24 * 3600;
    const one_month = 30 * one_day;
    const one_minute = 60;
    let refer_rewards = [40,20,10];
    let [unstakeFee,perMinuteRewardRate,dailyRewardRate,monthlyRewardRate,rewardRateDenominator] =
[
        5,4563,6592340,217891580,10**9
    ];

    before(async () => {
        [owner, user1, user2, ...users] = await ethers.getSigners();
    });

    async function deployUniswapAndToken() {
        // deploy weth
        const WETH9 = await ethers.getContractFactory("WETH9");
        weth = await WETH9.deploy();
        // deploy uniswap
        const UniswapV2Factory = await ethers.getContractFactory("UniswapV2Factory");
        factory = await UniswapV2Factory.deploy(owner.address);
        const UniswapV2Router02 = await ethers.getContractFactory("UniswapV2Router02");
        router = await UniswapV2Router02.deploy(factory.address, weth.address);
        const Swych = await ethers.getContractFactory("Swych");
        let implement = await Swych.deploy();
        const ERC1967Proxy = await ethers.getContractFactory("ERC1967Proxy");
        let proxy = await ERC1967Proxy.deploy(implement.address,"0x");
        swych = Swych.attach(proxy.address);;
        let args = [
            router.address,weth.address,users[0].address,users[1].address
        ];
        await swych.initialize(...args);
        pair = await factory.getPair(weth.address,swych.address);
    }

    async function addLiquidAndLaunch() {
        let balance = await swych.balanceOf(owner.address);
        let amount = balance.div(2);
```

```
47          await swych.approve(router.address, ethers.constants.MaxUint256);
48          await router.addLiquidityETH(swych.address,amount,1,1,owner.address,9876543210,{
49              value:ethers.utils.parseEther("100")
50          });
51          await weth.connect(users[0]).approve(swych.address,ethers.constants.MaxUint256);
52          await swych.launch();
53      }
54
55      async function deploySSwychAndInit() {
56          const sSwych = await ethers.getContractFactory("sSwych");
57          let implement = await sSwych.deploy();
58          const ERC1967Proxy = await ethers.getContractFactory("ERC1967Proxy");
59          let proxy = await ERC1967Proxy.deploy(implement.address,"0x");
60          instance = sSwych.attach(proxy.address);
61          await instance.initialize(swych.address);
62          await expect(implement.initialize(swych.address)).to.be.revertedWith("Initializable:
    contract is already initialized");
63      }
64
65      beforeEach(async () => {
66          await deployUniswapAndToken();
67          await addLiquidAndLaunch();
68          await deploySSwychAndInit();
69          await swych.setFeeExemptAddress(instance.address,true);
70          await swych.setAuthorizedToMint(instance.address,true);
71      });
72
73      describe("Init state unit test", () => {
74          it("Init twice should be failed", async () => {
75              await expect(instance.initialize(swych.address)).to.be.revertedWith(
76                  "Initializable: contract is already initialized"
77              );
78          });
79
80          it("Initial state check", async () => {
81              expect(await instance.swych()).to.be.equal(swych.address);
82              for(let i=0;i<3;i++) {
83                  expect(await
    instance.referralPercentagePerLevel(i+1)).to.be.equal(refer_rewards[i]);
84              }
85              expect(await instance.swych()).to.be.equal(swych.address);
86              expect(await instance.unstakeFee()).to.be.equal(unstakeFee);
87              expect(await instance.perMinuteRewardRate()).to.be.equal(perMinuteRewardRate);
88              expect(await instance.dailyRewardRate()).to.be.equal(dailyRewardRate);
89              expect(await instance.monthlyRewardRate()).to.be.equal(monthlyRewardRate);
90              expect(await instance.rewardRateDenominator()).to.be.equal(rewardRateDenominator);
91              expect(await instance.getVersion()).to.be.equal("1.0");
92          });
93      });
94
95      describe("Interface of owner unit test", () => {
96          it("pause and unpause unit test", async () => {
97              expect(await instance.paused()).to.be.false;
98              await expect(instance.pause()).to.be.emit(
99                  instance,"Pause"
100             );
101             expect(await instance.paused()).to.be.true;
102             await expect(instance.unpause()).to.be.emit(
103                 instance,"Unpause"
```

```
            );
            expect(await instance.paused()).to.be.false;
        });

        // event
        it("setRewardRate test", async () => {
            let rates =[123,234,345];
            await expect(instance.setRewardRate(...rates)).to.be.emit(
                instance,"SetRewardRate"
            ).withArgs(rates[0],rates[1],rates[2]);
            expect(await instance.perMinuteRewardRate()).to.be.equal(rates[0]);
            expect(await instance.dailyRewardRate()).to.be.equal(rates[1]);
            expect(await instance.monthlyRewardRate()).to.be.equal(rates[2]);
        });

        it("setUnstakeFee test", async () => {
            await expect(instance.setUnstakeFee(25)).to.be.revertedWith("INVALID_FEE_PERCENTAGE");
            await expect(instance.setUnstakeFee(15)).to.be.emit(
                instance,"SetUnstakeFee"
            ).withArgs(15);
            expect(await instance.unstakeFee()).to.be.equal(15);
        });

        it("setReferralPercentages", async () => {
            let new_refer_rewards = refer_rewards.map(rate => parseInt(rate * 2.5));
            await expect(instance.setReferralPercentages(...new_refer_rewards)).to.be.emit(
                instance,"SetReferralPercentages"
            ).withArgs(...new_refer_rewards);
            for(let i=0;i<3;i++) {
                expect(await
instance.referralPercentagePerLevel(i+1)).to.be.equal(new_refer_rewards[i]);
            }

            for(let i=0;i<3;i++) {
                let error_refer_rewards = new_refer_rewards.map( rate => rate);
                error_refer_rewards[i] = 110;
                await
expect(instance.setReferralPercentages(...error_refer_rewards)).to.be.revertedWith(
                    "INVALID_PERCENTAGE"
                );
            }
        });

        it("setEmergencyWithdrawEnabled test", async () => {
            expect(await instance.emergencyWithdrawEnabled()).to.be.false;
            await expect(instance.setEmergencyWithdrawEnabled(true)).to.be.emit(
                instance,"SetEmergencyWithdrawEnabled"
            ).withArgs(true);
            expect(await instance.emergencyWithdrawEnabled()).to.be.true;
            await expect(instance.setEmergencyWithdrawEnabled(false)).to.be.emit(
                instance,"SetEmergencyWithdrawEnabled"
            ).withArgs(false);
            expect(await instance.emergencyWithdrawEnabled()).to.be.false;
        });

        it("setSuperCompounder test", async () => {
            let user = users[8].address;
            expect(await instance.superCompounder(user)).to.be.false;
            await expect(instance.setSuperCompounder(user,true)).to.be.emit(
```

```
                instance,"SetSuperCompounder"
            ).withArgs(user,true);
            expect(await instance.superCompounder(user)).to.be.true;
            await expect(instance.setSuperCompounder(user,false)).to.be.emit(
                instance,"SetSuperCompounder"
            ).withArgs(user,false);
            expect(await instance.superCompounder(user)).to.be.false;
        });
    });

    describe("Stake and unstake unit test", () => {
        let user;
        let referer;
        let stake_amount = ethers.utils.parseEther("10000");
        before(() => {
            user = user1;
            referer = users[5].address;
        });

        async function prepare(operator) {
            await swych.transfer(operator.address,stake_amount);
            await swych.connect(operator).approve(instance.address,ethers.constants.MaxUint256);
        }

        function get_rand() {
            return parseInt(10 * Math.random());
        }

        it("stake should failed while paused or having no tokens", async () => {
            await instance.pause();
            await expect(instance.connect(user).stake()).to.be.revertedWith("Pausable: paused");

            await instance.unpause();
            await expect(instance.connect(user).stake()).to.be.revertedWith("NO_SWYCH_TO_STAKE");
        });

        it("Stake without refer should change state and emit event", async () => {
            await prepare(user);
            let block = await ethers.provider.getBlockNumber();
            let {timestamp} = await ethers.provider.getBlock(block);
            await expect(instance.connect(user).stake()).to.be.emit(
                instance,"Stake"
            ).withArgs(user.address,stake_amount,stake_amount);
            // check state
            let {lastCheckpoint,gonsAccruedAmount,gonsPrincipal} = await
instance.stakers(user.address);
            let gonsPerFragment = await swych.gonsPerFragment();
            expect(lastCheckpoint).to.be.equal(timestamp + 1);
            expect(gonsAccruedAmount).to.be.equal(gonsPerFragment.mul(stake_amount));
            expect(gonsPrincipal).to.be.equal(gonsAccruedAmount);
            expect(await instance.hasStaked(user.address)).to.be.true;
            let params = [get_rand(),get_rand(),get_rand()];
            let rates = [monthlyRewardRate,dailyRewardRate,perMinuteRewardRate];

            let target = one_month * params[0] + one_day * params[1] + one_minute * params[2] +
timestamp + 1;
            await ethers.provider.send("evm_mine", [target]);
            await prepare(user);
            let sum = gonsAccruedAmount;
```

```
            for(let i=0;i<params.length;i++) {
                let rate = rates[i];
                for(let j=0;j<params[i];j++) {
                    sum = sum.div(rewardRateDenominator).mul(rate).add(sum);
                }
            }
            let fee = sum.mul(unstakeFee).div(100);
            let amounts = await instance.getGonsAccruedAndFeeAmounts(user.address);
            expect(amounts.gonsAccruedAmount).to.be.equal(sum);
            expect(amounts.gonsFeeAmount).to.be.equal(fee);
            let totalAccruedAmount =
gonsPerFragment.mul(stake_amount).add(sum).div(gonsPerFragment);
            await expect(instance.connect(user).stake()).to.be.emit(
                instance,"Stake"
            ).withArgs(user.address,stake_amount,totalAccruedAmount);
            ({lastCheckpoint,gonsAccruedAmount,gonsPrincipal} = await
instance.stakers(user.address));
            expect(lastCheckpoint).to.be.equal(target + 3);
            expect(gonsAccruedAmount).to.be.equal( gonsPerFragment.mul(stake_amount).add(sum));
            expect(gonsPrincipal).to.be.equal(gonsPerFragment.mul(stake_amount).mul(2));
        });

        it("Stake with referer should be failed while has staked", async ()=> {
            await prepare(user);
            await instance.connect(user).stake();
            await prepare(user);
            await
expect(instance.connect(user).stakeWithReferrer(referer)).to.be.revertedWith("ALREADY_REGISTERED");
        });

        it("Stake with referer should add referer test", async () => {
            let Alice = users[0];
            let Bob = users[1];
            let Kite = users[2];
            await prepare(Alice);
            await instance.connect(Alice).stakeWithReferrer(Bob.address);
            await prepare(Bob);
            await instance.connect(Bob).stakeWithReferrer(Kite.address);
            await prepare(user);
            await instance.connect(user).stakeWithReferrer(Alice.address);

            let referees = await instance.connect(user).getReferees();
            expect(referees.length).to.be.equal(0);
            (referees = await instance.connect(Alice).getReferees());
            expect(referees.length).to.be.equal(1);
            (referees = await instance.connect(Bob).getReferees());
            expect(referees.length).to.be.equal(2);
            (referees = await instance.connect(Kite).getReferees());
            expect(referees.length).to.be.equal(2);

            let {referee,level} = await instance.referees(Alice.address,0);
            expect(referee).to.be.equal(user.address);
            expect(level).to.be.equal(1);
            ({referee,level} = await instance.referees(Bob.address,0));
            expect(referee).to.be.equal(Alice.address);
            expect(level).to.be.equal(1);
            ({referee,level} = await instance.referees(Bob.address,1));
            expect(referee).to.be.equal(user.address);
            expect(level).to.be.equal(2);
```

```
            ({referee,level} = await instance.referees(Kite.address,1));
            expect(referee).to.be.equal(user.address);
            expect(level).to.be.equal(3);
            await prepare(Alice);
            await
expect(instance.connect(Alice).stakeWithReferrer(Bob.address)).to.be.rejectedWith(
                "ALREADY_REGISTERED"
            );
        });

        it("Super Compound test", async () => {
            let Alice = users[9].address;
            prepare(user);
            await
expect(instance.connect(user).superCompound(Alice,100)).to.be.revertedWith("NOT_AUTHORIZED");
            await instance.setSuperCompounder(user.address,true);
            await
expect(instance.connect(user).superCompound(Alice,stake_amount.mul(2))).to.be.revertedWith("NO_SWYC
H_TO_STAKE");
            await instance.connect(user).superCompound(Alice,stake_amount);
            expect(await swych.balanceOf(user.address)).to.be.equal(0);
            expect(await swych.balanceOf(instance.address)).to.be.equal(stake_amount);
            let block = await ethers.provider.getBlockNumber();
            let {timestamp} = await ethers.provider.getBlock(block);
            let {lastCheckpoint,gonsAccruedAmount,gonsPrincipal} = await instance.stakers(Alice);
            let gonsPerFragment = await swych.gonsPerFragment();
            expect(lastCheckpoint).to.be.equal(timestamp);
            expect(gonsAccruedAmount).to.be.equal(gonsPerFragment.mul(stake_amount));
            expect(gonsPrincipal).to.be.equal(gonsAccruedAmount);

        });

        it("Unstake test", async () => {
            let Alice = users[0];
            let Bob = users[1];
            let Kite = users[2];
            await prepare(Alice);
            await instance.connect(Alice).stakeWithReferrer(Bob.address);
            await prepare(Bob);
            await instance.connect(Bob).stakeWithReferrer(Kite.address);
            await prepare(user);
            await instance.connect(user).stakeWithReferrer(Alice.address);

            let block = await ethers.provider.getBlockNumber();
            let {timestamp} = await ethers.provider.getBlock(block);
            expect(await
instance.lastCheckoutReferral(Alice.address,user.address)).to.be.equal(timestamp);

            let params = [get_rand(),get_rand(),get_rand()];
            // let rates = [monthlyRewardRate,dailyRewardRate,perMinuteRewardRate];
            let target = one_month * params[0] + one_day * params[1] + one_minute * params[2] +
timestamp + 1;
            await ethers.provider.send("evm_mine", [target]);
            // mint block;
            await swych.approve(users[9].address,0);
            let interest_alice = await instance.getInterestFromTimestamp(user.address,timestamp);
            let reward_alice = interest_alice.mul(40).div(100);
            let view_rewards = await instance.connect(Alice).viewReferralRewards();
            expect(view_rewards).to.be.equal(reward_alice);
```

```
            let alice_accure = await instance.getGonsAccruedAndFeeAmounts(Alice.address);
            alice_accure = alice_accure.gonsAccruedAmount;

            let withdraw_amount = await instance.getWithdrawAmount(user.address);
            await instance.connect(user).unstake();
            let fee = stake_amount.mul(unstakeFee).div(100);
            expect(await swych.balanceOf(dead_address)).to.be.equal(fee);
            let balance = await swych.balanceOf(user.address);
            if(withdraw_amount.gt(balance)) {
                expect(withdraw_amount.sub(balance).lt(10)).to.be.true;
            } else {
                expect(balance.sub(withdraw_amount).lt(10)).to.be.true;
            }
            let gonsPerFragment = await swych.gonsPerFragment();
            let {gonsPrincipal} = await instance.stakers(Alice.address);
            expect(stake_amount.add(reward_alice).mul(gonsPerFragment)).to.be.equal(gonsPrincipal);
        });

        it("collectReferralRewards test", async () => {
            let Alice = users[0];
            let Bob = users[1];
            let Kite = users[2];
            await prepare(Alice);
            await instance.connect(Alice).stakeWithReferrer(Bob.address);
            await prepare(Bob);
            await instance.connect(Bob).stakeWithReferrer(Kite.address);
            await prepare(user);
            await instance.connect(user).stakeWithReferrer(Alice.address);

            let block = await ethers.provider.getBlockNumber();
            let {timestamp} = await ethers.provider.getBlock(block);
            expect(await
instance.lastCheckoutReferral(Alice.address,user.address)).to.be.equal(timestamp);

            let params = [get_rand(),get_rand(),get_rand()];
            let target = one_month * params[0] + one_day * params[1] + one_minute * params[2] +
timestamp + 1;
            await ethers.provider.send("evm_mine", [target]);
            await swych.approve(users[9].address,0);

            let interest_alice = await instance.getInterestFromTimestamp(user.address,timestamp);
            let reward_alice = interest_alice.mul(40).div(100);
            let view_rewards = await instance.connect(Alice).viewReferralRewards();
            expect(view_rewards).to.be.equal(reward_alice);

            await expect(instance.connect(Alice).collectReferralRewards()).to.be.emit(
                instance,"ReferralReward"
            ).withArgs(Alice.address,user.address,reward_alice);
        });

        it("emergencyWithdraw test", async () => {
            let Alice = users[0];
            let Bob = users[1];
            let Kite = users[2];
            await prepare(Alice);
            await instance.connect(Alice).stakeWithReferrer(Bob.address);
            await prepare(Bob);
            await instance.connect(Bob).stakeWithReferrer(Kite.address);
            await prepare(user);
```

```
            await instance.connect(user).stakeWithReferrer(Alice.address);

            let block = await ethers.provider.getBlockNumber();
            let {timestamp} = await ethers.provider.getBlock(block);
            expect(await
instance.lastCheckoutReferral(Alice.address,user.address)).to.be.equal(timestamp);

            let params = [get_rand(),get_rand(),get_rand()];
            let target = one_month * params[0] + one_day * params[1] + one_minute * params[2] +
timestamp + 1;
            await ethers.provider.send("evm_mine", [target]);
            await swych.approve(users[9].address,0);

            await
expect(instance.connect(user).emergencyWithdraw()).to.be.rejectedWith("EMERGENCY_WITHDRAW_DISABLED"
);
            await instance.setEmergencyWithdrawEnabled(true);
            await expect(instance.connect(user).emergencyWithdraw()).to.be.emit(
                instance,"EmergencyWithdraw"
            ).withArgs(user.address,stake_amount);
            block = await ethers.provider.getBlockNumber();
            ({timestamp} = await ethers.provider.getBlock(block));
            let info = await instance.stakers(user.address);
            expect(info.gonsPrincipal).to.be.equal(0);
            expect(info.lastCheckpoint).to.be.equal(timestamp);
            expect(info.gonsAccruedAmount).to.be.equal(0);
        });

    });
});

```

3. UnitTestOutput

```
Swych Token Unit Test
    Initial State Unit Test
      ✔ Init twice should be failed
      ✔ Initial State should be checked (85ms)
    Interface of owner unit test
      ✔ launch should change state and emit event (159ms)
      ✔ SetAthDeltaPermille should change state and emit event
data: BigNumber { value: "996990060009101" } 0.000996990060009101
      ✔ withdrawFeesToTreasury unit test (137ms)
      ✔ setAuthorizedToMint unit test
      ✔ setBlockedPair unit test
      ✔ setRewardEnabled unit test
      ✔ setPriceEnabled unit test
      ✔ setRewardFrequency unit test
      ✔ setAutoReward unit test
      ✔ setAvailableTradingBalanceEnabled unit test
      ✔ setNoCheckAvailableTradingBalance unit test
      ✔ setTransferFeeEnabled unit test
      ✔ setSwapBackEnabled unit test
      ✔ setCollectedFeeThreshold unit test
      ✔ setAvailableTradingBalanceCoefficients unit test
      ✔ setRangeHoldingPermyriadAvailableTradingBalanceApplied unit test (38ms)
```

```
      ✔ setReboundTriggerFromAth unit test
      ✔ setRewardRate unit test
      ✔ setTreasuryWallet unit test
      ✔ setAutoLiquidityReceiver unit test
      ✔ setFeeExemptAddress
      ✔ setBackingLPToken unit test
      ✔ pause and unpause unit test
      ✔ rescueToken unit test (41ms)
      ✔ setFeeSplit unit test
      ✔ setFees unit test
      ✔ setAutomatedMarketMakerPair unit test (166ms)
    View function unit test
      ✔ getAvailableTradingBalanceFactor unit test (469ms)
      ✔ getAvailableTradingBalanceAmount unit test (62ms)
      ✔ getTriggerReboundPrice unit test (103ms)
      ✔ getLiquidityBacking unit test
    Reward unit test
      ✔ Reward while price is reboundEnabled  (215ms)
      ✔ Reward while price is rebaseEnabled  (198ms)
    Transfer unit test
      ✔ User Transfer to User test (137ms)
      ✔ User sell token test (329ms)
    Unit test of Others
      ✔ Call manualSync should emit event
      ✔ Mint should be failed without authorized
      ✔ Mint should change balance and totalSupply
      ✔ decreaseAllowance and increaseAllowance unit test (103ms)
    Airdrop unit test
      ✔ Airdrop before launched should be failed (75ms)
      ✔ Airdrop should transfer tokens (105ms)

  sSwych Unit Test
    Init state unit test
      ✔ Init twice should be failed
      ✔ Initial state check (42ms)
    Interface of owner unit test
      ✔ pause and unpause unit test
      ✔ setRewardRate test
      ✔ setUnstakeFee test
      ✔ setReferralPercentages (42ms)
      ✔ setEmergencyWithdrawEnabled test
      ✔ setSuperCompounder test
    Stake and unstake unit test
      ✔ stake should failed while paused or having no tokens
      ✔ Stake without refer should change state and emit event (135ms)
      ✔ Stake with referer should be failed while has staked (91ms)
      ✔ Stake with referer should add referer test (233ms)
      ✔ Super Compound test (84ms)
      ✔ Unstake test (331ms)
      ✔ collectReferralRewards test (218ms)
      ✔ emergencyWithdraw test (295ms)


  59 passing (18s)

```

11.2 External Functions Check Points

1. Swych.sol_output.md

File: contracts/Swych.sol

(Empty fields in the table represent things that are not required or relevant)

contract: Swych is IERC20, Initializable, OwnableUpgradeable, PausableUpgradeable, UUPSUpgradeable

Index	Function	Visibility	StateMutability	Permission Check	IsUserInterface	Unit Test	Notes
1	initialize(address,address,address,address)	public				Passed	initializer
2	getVersion()	external	pure			Passed	
3	transfer(address,uint256)	external			True	Passed	whenNotPaused
4	transferFrom(address,address,uint256)	external			True	Passed	whenNotPaused
5	decreaseAllowance(address,uint256)	external			True	Passed	whenNotPaused
6	increaseAllowance(address,uint256)	external			True	Passed	whenNotPaused
7	approve(address,uint256)	public			True	Passed	whenNotPaused
8	launch()	external		onlyOwner		Passed	onlyOnce
9	setAthDeltaPermille(uint256)	external		onlyOwner		Passed	
10	reward()	external			True	Passed	whenNotPaused
11	withdrawFeesToTreasury()	external		onlyOwner		Passed	
12	setAuthorizedToMint(address,bool)	external		onlyOwner		Passed	
13	setBlockedPair(address,bool)	external		onlyOwner		Passed	
14	setRewardEnabled(bool,bool)	external		onlyOwner		Passed	
15	setPriceEnabled(bool)	external		onlyOwner		Passed	
16	setRewardFrequency(uint256)	external		onlyOwner		Passed	
17	setAutoReward(bool)	external		onlyOwner		Passed	
18	setAvailableTradingBalanceEnabled(bool)	external		onlyOwner		Passed	
19	setNoCheckAvailableTradingBalance(address,bool)	external		onlyOwner		Passed	
20	setTransferFeeEnabled(bool)	external		onlyOwner		Passed	
21	setSwapBackEnabled(bool)	external		onlyOwner		Passed	
22	setCollectedFeeThreshold(uint256)	external		onlyOwner		Passed	
23	setAvailableTradingBalanceCoefficients(uint256,uint256)	external		onlyOwner		Passed	
24	setRangeHoldingPermyriadAvailableTradingBalanceApplied(uint256,uint256)	external		onlyOwner		Passed	
25	setReboundTriggerFromAth(uint256,uint256)	external		onlyOwner		Passed	
26	setRewardRate(uint256,uint256)	external		onlyOwner		Passed	
27	setTreasuryWallet(address)	external		onlyOwner		Passed	
28	setAutoLiquidityReceiver(address)	external		onlyOwner		Passed	
29	setFeeExemptAddress(address,bool)	external		onlyOwner		Passed	
30	setBackingLPToken(address)	external		onlyOwner		Passed	
31	pause()	external		onlyOwner		Passed	
32	unpause()	external		onlyOwner		Passed	
33	rescueToken(address,uint256)	external		onlyOwner		Passed	
34	setFeeSplit(uint256,uint256,uint256)	external		onlyOwner		Passed	
35	setFees(uint256,uint256,uint256)	external		onlyOwner		Passed	
36	balanceOf(address)	public	view			Passed	
37	checkCollectedFeeThreshold()	external	view			Passed	
38	setAutomatedMarketMakerPair(address,bool)	public		onlyOwner		Passed	
39	calculateAvailableTradingBalanceFactor(uint256)	public	view			Passed	
40	getAvailableTradingBalanceFactor(address)	public	view			Passed	
41	getAvailableTradingBalanceAmount(address)	public	view			Passed	
42	getRemaningAvailableTradingBalanceAmount(address)	public	view				
43	getCurrentAvailableTradingBalanceAmount(address)	public	view			Passed	
44	getTriggerReboundPrice()	public	view			Passed	
45	allowance(address,address)	public	view			Passed	
46	manualSync()	public			True	Passed	
47	totalSupplyIncludingBurnAmount()	public	view			Passed	
48	totalSupply()	public	view			Passed	
49	name()	public	pure			Passed	
50	symbol()	public	pure			Passed	
51	decimals()	public	pure			Passed	
52	airdrop(address[],uint256[])	external				Passed	beforeLaunched
53	mint(address,uint256)	public		isAuthorizedToMint		Passed	

2. sSwych.sol_output.md

File: contracts/sSwych.sol

(Empty fields in the table represent things that are not required or relevant)

contract: sSwych is Initializable, OwnableUpgradeable, PausableUpgradeable, UUPSUpgradeable, ReentrancyGuardUpgradeable

Index	Function	Visibility	StateMutability	Permission Check	IsUserInterface	Unit Test	Notes
1	initialize(ISwych)	public				Passed	initializer
2	getVersion()	external	pure			Passed	
3	pause()	external		onlyOwner		Passed	
4	unpause()	external		onlyOwner		Passed	
5	setRewardRate(uint256,uint256,uint256)	external		onlyOwner		Passed	
6	setUnstakeFee(uint256)	external		onlyOwner		Passed	
7	setReferralPercentages(uint256,uint256,uint256)	external		onlyOwner		Passed	
8	setEmergencyWithdrawEnabled(bool)	external		onlyOwner		Passed	
9	setSuperCompounder(address,bool)	external		onlyOwner		Passed	
10	getGonsAccruedAndFeeAmounts(address)	public	view			Passed	
11	getInterestFromTimestamp(address,uint256)	public	view			Passed	
12	getAccruedAndFeeAmounts(address)	public	view				
13	getWithdrawAmount(address)	external	view			Passed	
14	stake()	external			Yes	Passed	nonReentrant,whenNotPaused
15	stakeWithReferrer(address)	external			Yes	Passed	nonReentrant,whenNotPaused
16	superCompound(address,uint256)	external		onlySuperCompounder		Passed	nonReentrant,whenNotPaused
17	unstake()	external			Yes	Passed	nonReentrant,whenNotPaused
18	collectReferralRewards()	external			Yes	Passed	nonReentrant,whenNotPaused
19	viewReferralRewards()	public	view			Passed	
20	getReferees()	public	view			Passed	
21	emergencyWithdraw()	external			Yes	Passed	nonReentrant,whenNotPaused

FAIRYPROOF

https://medium.com/@FairyproofT

https://twitter.com/FairyproofT

https://www.linkedin.com/company/fairyproof-tech

https://t.me/Fairyproof_tech

Reddit: https://www.reddit.com/user/FairyproofTech