DAO Paykiken Geo
Smart contracts security audit

(Completed August, 2023)

paykiken.io

Contents

Introduction

DAO Paykiken Geo is the first decentralized autonomous organization that implements the unique concept of collective investment in the commodity sectors of the global economy using blockchain technology and artificial intelligence.

Paykiken Geo architecture is a system of smart contracts that form the main structure of the DAO Paykiken Geo in blockchain. This architecture realizes functions of token SWAP (buy/sell), token price growth function, voting, founders team tokens freeze/unfreeze algorithm and token hold. This structure allows to join or leave the DAO at any given moment, stipulates token growth in geometrical progression in direct relation to the DAO USDT liquidity pool and utilizes a DAO voting mechanism for transactions made from DAO capital pool.

DAO Paykiken Geo engaged the DAO community members and developers to perform a source code review of DAO Paykiken Geo Protocol. The objective of the audit was to evaluate the security of the smart contracts. The assessment covered the repository at https://github.com/DAO-Paykiken-Geo/contracts/tree/testnet-version as of commit **d0b194c1b489e9b7e7bbd82a627840be28095859** tag audit, with focus on recent changes made to DAO Paykiken Geo smart contract architecture code.

The Paykiken Geo Protocol was deployed to Nile Testnet in TRON blockchain with the following contracts being the main point of focus for the audit:

- Governor.sol (https://nile.tronscan.org/#/address/TWkhJreRfSCCNAC6RK9UFgfh6upg8SggiH);
- Hold.sol (https://nile.tronscan.org/#/address/TTTmXMG4yWADpXPFqb5WfBGgUDiLibNnMs);
- Swap.sol (https://nile.tronscan.org/#/address/TFeJEt72MnghNQ21HMssq824u11V7i8yWB);
- Team.sol (https://nile.tronscan.org/#/address/TXQVUVDemwBCgozLhK57uSBPaJLeHq2Stt).

The information presented in this document is provided as is and without warranty. Vulnerability assessments are a “point in time” analysis and as such it is possible that something in the environment could have changed since the tests reflected in this report were run. This report should not be considered a perfect representation of the risks threatening the analyzed system, networks and applications.

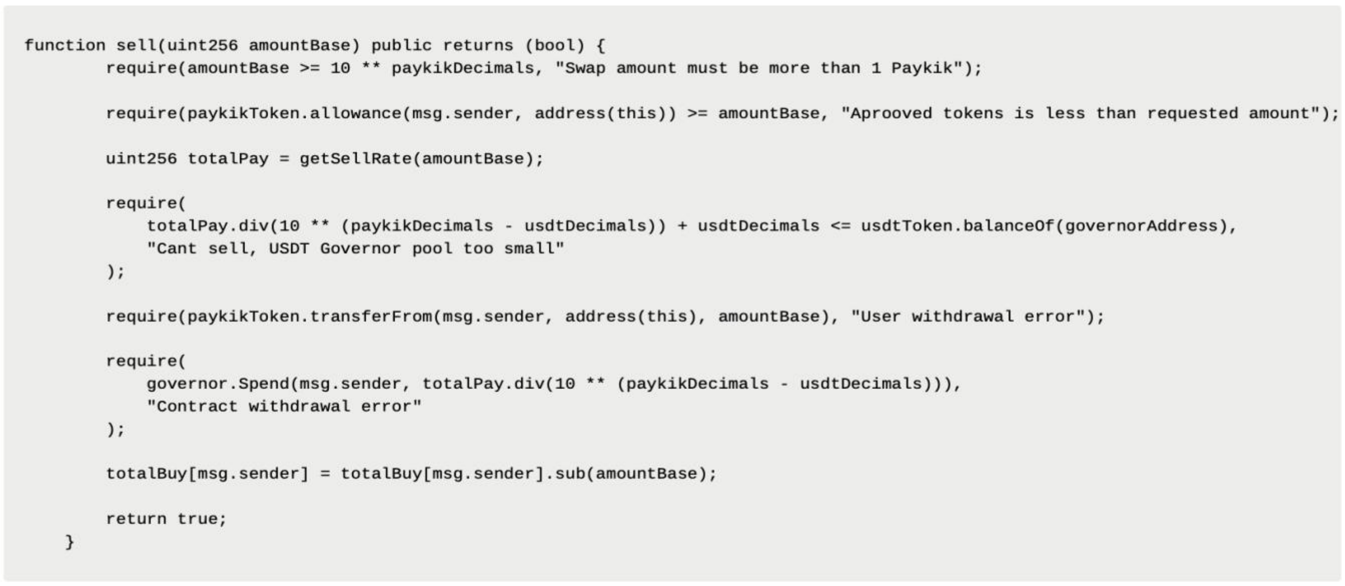
Identified and mitigated issues

Issue 1. Contract Swap.sol

1.1. Identified issue:

"Can't sell PAYKIK tokens that were not originally bought from the Swap smart contract".

- **Severity** - critical;
- **Type** – unrecorded tokens;
- **Vulnerable SHA1** - 927ef030bc47106473b406508fb4fb7ac6651915;
- **Mitigated SHA1** - ccf07974d6d1965fd62df7c1c8bef1fc441c6dd.



```
function sell(uint256 amountBase) public returns (bool) {
        require(amountBase >= 10 ** paykikDecimals, "Swap amount must be more than 1 Paykik");

        require(paykikToken.allowance(msg.sender, address(this)) >= amountBase, "Aprooved tokens is less than requested amount");

        uint256 totalPay = getSellRate(amountBase);

        require(
            totalPay.div(10 ** (paykikDecimals - usdtDecimals)) + usdtDecimals <= usdtToken.balanceOf(governorAddress),
            "Cant sell, USDT Governor pool too small"
        );

        require(paykikToken.transferFrom(msg.sender, address(this), amountBase), "User withdrawal error");

        require(
            governor.Spend(msg.sender, totalPay.div(10 ** (paykikDecimals - usdtDecimals))),
            "Contract withdrawal error"
        );

        totalBuy[msg.sender] = totalBuy[msg.sender].sub(amountBase);

        return true;
    }
```

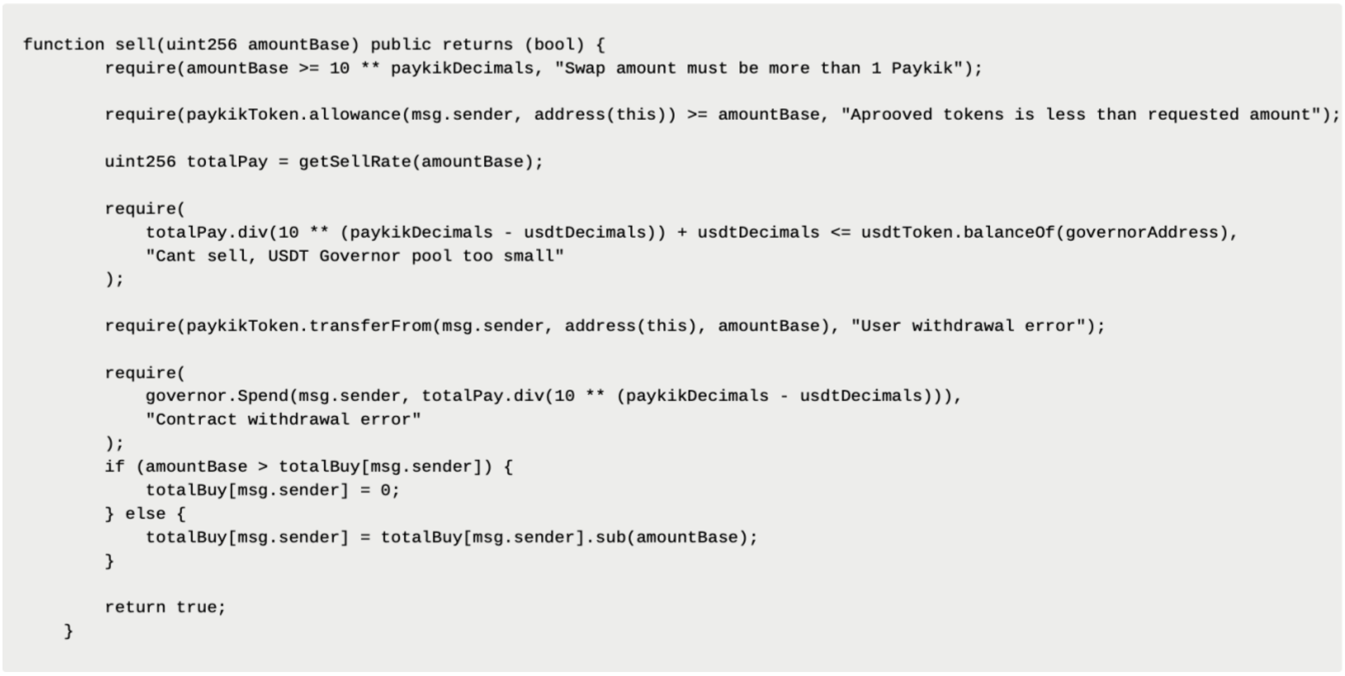
Picture 1. Vulnerable sell() method (927ef030bc47106473b406508fb4fb7ac6651915)

Vulnerability description:

Tokens that are frozen on the Team.sol contract cannot be sold on the **Swap.sol** contract because the balance of 32 holders of frozen PAYKIK tokens on the **Team.sol** contract has been identified as having a value of zero (**totalBuy[msg.sender]**). This is because these PAYKIK tokens were not originally purchased from the **Swap.sol** contract.

1.2. **Solution:**

The new logic stipulates that the users can sell more PAYKIK tokens than they have bought (**totalBuy[msg.sender]**). This situation is only possible if the tokens were purchased outside of the Swap.sol smart contract, for example, on a P2P platform. The creators of the PAYKIK protocol considered this scenario to be possible.



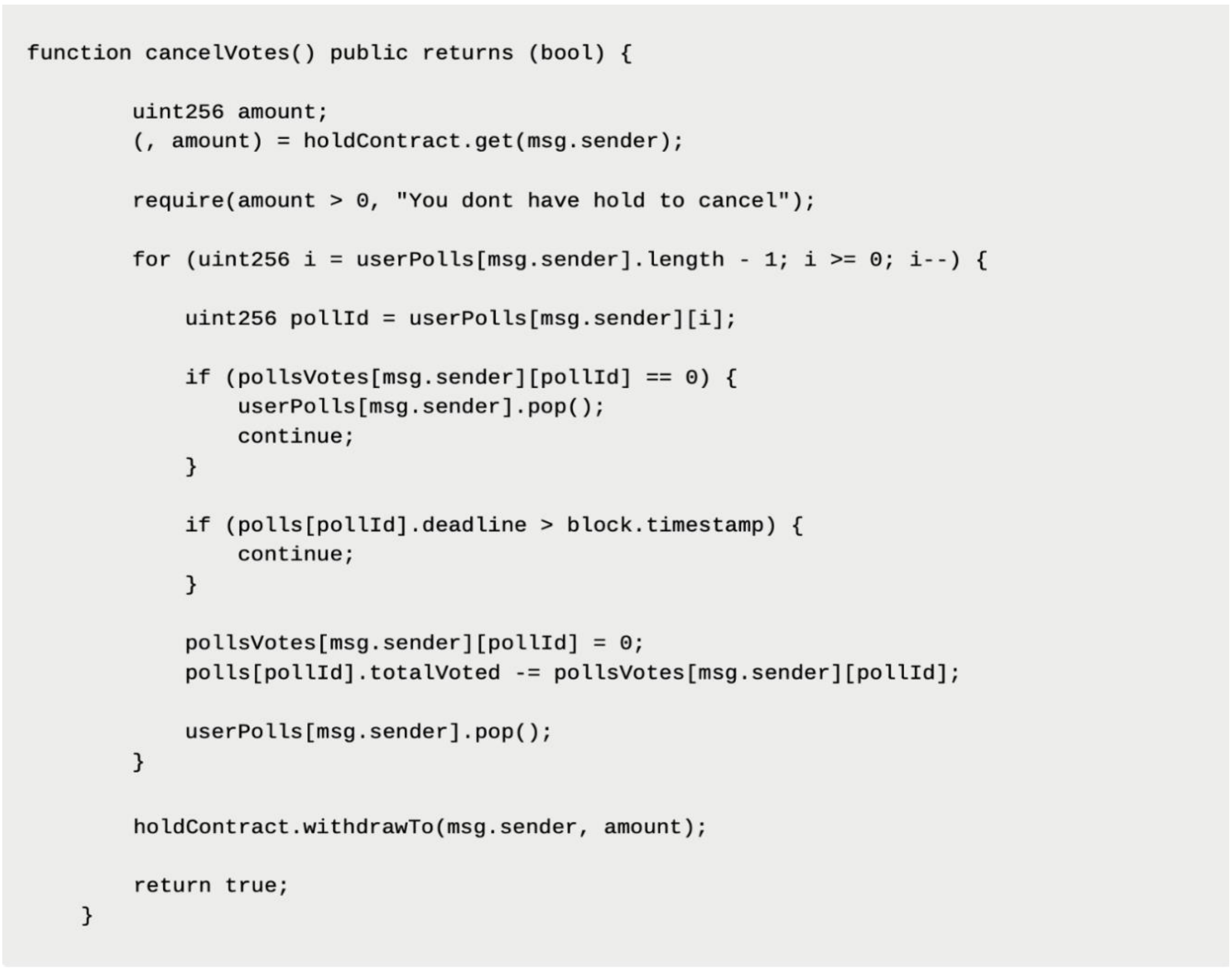
```
function sell(uint256 amountBase) public returns (bool) {
        require(amountBase >= 10 ** paykikDecimals, "Swap amount must be more than 1 Paykik");

        require(paykikToken.allowance(msg.sender, address(this)) >= amountBase, "Aprooved tokens is less than requested amount");

        uint256 totalPay = getSellRate(amountBase);

        require(
            totalPay.div(10 ** (paykikDecimals - usdtDecimals)) + usdtDecimals <= usdtToken.balanceOf(governorAddress),
            "Cant sell, USDT Governor pool too small"
        );

        require(paykikToken.transferFrom(msg.sender, address(this), amountBase), "User withdrawal error");

        require(
            governor.Spend(msg.sender, totalPay.div(10 ** (paykikDecimals - usdtDecimals))),
            "Contract withdrawal error"
        );
        if (amountBase > totalBuy[msg.sender]) {
            totalBuy[msg.sender] = 0;
        } else {
            totalBuy[msg.sender] = totalBuy[msg.sender].sub(amountBase);
        }

        return true;
    }
```

Picture 2. Patch version of sell() method (ccf07974d6d1965fd62df7c1c8bef1fc441c6dd)

Issue 2. Contract Governor.sol.

2.1. **Identified issue:**
“cancelVotes() Integer — Underflow”.

- **Severity** - critical;
- **Type** - integer-underflow;
- **Vulnerable SHA1** - 6361528e7f39f2894bcb4be42f9690d1d977ca09;
- **Mitigated SHA1** - 026845dffee0493a0fbe2b03fda0893629fbf3c9.



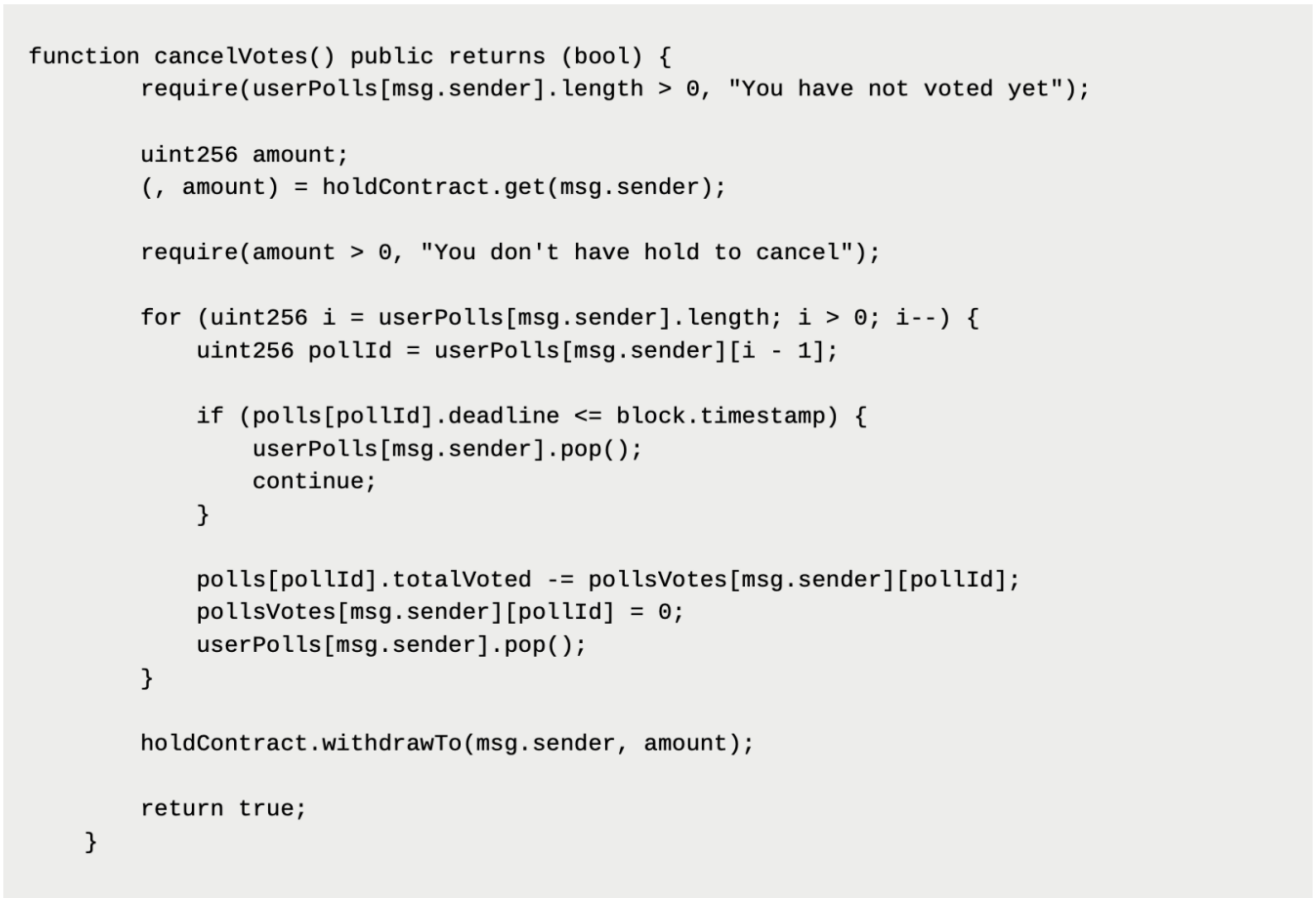
```
function cancelVotes() public returns (bool) {

        uint256 amount;
        (, amount) = holdContract.get(msg.sender);

        require(amount > 0, "You dont have hold to cancel");

        for (uint256 i = userPolls[msg.sender].length - 1; i >= 0; i--) {

            uint256 pollId = userPolls[msg.sender][i];

            if (pollsVotes[msg.sender][pollId] == 0) {
                userPolls[msg.sender].pop();
                continue;
            }

            if (polls[pollId].deadline > block.timestamp) {
                continue;
            }

            pollsVotes[msg.sender][pollId] = 0;
            polls[pollId].totalVoted -= pollsVotes[msg.sender][pollId];

            userPolls[msg.sender].pop();
        }

        holdContract.withdrawTo(msg.sender, amount);

        return true;
    }
```

Picture 3. Vulnerable cancelVotes() method (6361528e7f39f2894bcb4be42f9690d1d977ca09)

Vulnerability description:

During iteration, cycle (**for (uint256 i = userPolls[msg.sender].length - 1; i >= 0; i--)**) will get Integer-Underflow as, the last iteration will be equal to 0 and operation 0 - 1 will happen as the result of decrementation.

2.2. **Solution:**



```
function cancelVotes() public returns (bool) {
        require(userPolls[msg.sender].length > 0, "You have not voted yet");

        uint256 amount;
        (, amount) = holdContract.get(msg.sender);

        require(amount > 0, "You don't have hold to cancel");

        for (uint256 i = userPolls[msg.sender].length; i > 0; i--) {
            uint256 pollId = userPolls[msg.sender][i - 1];

            if (polls[pollId].deadline <= block.timestamp) {
                userPolls[msg.sender].pop();
                continue;
            }

            polls[pollId].totalVoted -= pollsVotes[msg.sender][pollId];
            pollsVotes[msg.sender][pollId] = 0;
            userPolls[msg.sender].pop();
        }

        holdContract.withdrawTo(msg.sender, amount);

        return true;
    }
```

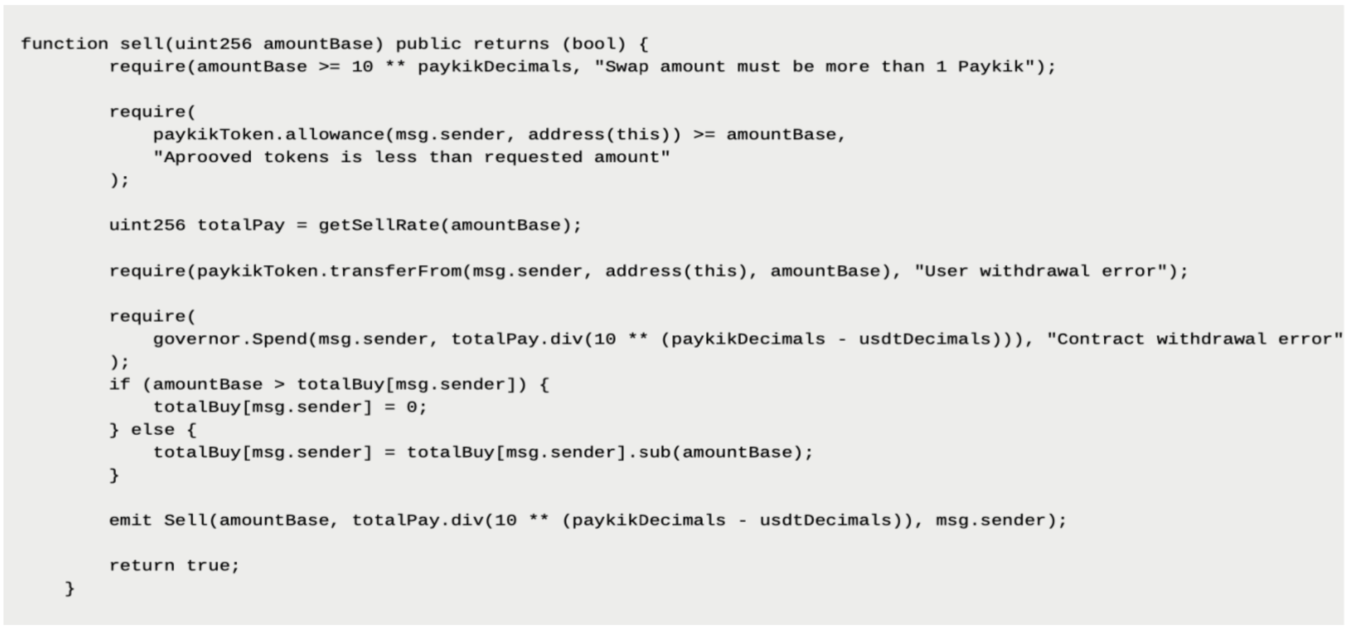
Picture 4. Patch version of cancelVotes() method (026845dffee0493a0fbe2b03fda0893629fbf3c9)

Issue 3. Contract Swap.sol.

3.1. Identified issue:

"getBuyRate() calculates price for cases that are not stipulated by the algorithm".

- **Severity** - critical;
- **Type** - manipulation over the price of a token;
- **Vulnerable SHA1** - dfca22e62aa17665dd00de1c49f087d2c77a6be7;
- **Mitigated SHA1** - 0ccf6f3c18e54441e2a80d2a4288e8558d93c1fd.



```
function sell(uint256 amountBase) public returns (bool) {
        require(amountBase >= 10 ** paykikDecimals, "Swap amount must be more than 1 Paykik");

        require(
            paykikToken.allowance(msg.sender, address(this)) >= amountBase,
            "Aprooved tokens is less than requested amount"
        );

        uint256 totalPay = getSellRate(amountBase);

        require(paykikToken.transferFrom(msg.sender, address(this), amountBase), "User withdrawal error");

        require(
            governor.Spend(msg.sender, totalPay.div(10 ** (paykikDecimals - usdtDecimals))), "Contract withdrawal error"
        );
        if (amountBase > totalBuy[msg.sender]) {
            totalBuy[msg.sender] = 0;
        } else {
            totalBuy[msg.sender] = totalBuy[msg.sender].sub(amountBase);
        }

        emit Sell(amountBase, totalPay.div(10 ** (paykikDecimals - usdtDecimals)), msg.sender);

        return true;
    }
```

Picture 5. Vulnerable sell() method (dfca22e62aa17665dd00de1c49f087d2c77a6be7)

Vulnerability description:

Before purchase **Swap** contract call's function (**getBuyRate()**). To ensure accurate calculation, the USDT balance in the **Governor** smart contract must exceed 1 USDT. Failure to meet this requirement may result in incorrect values and financial exploitation. Notably, the algorithm is initially calculated based on pools with a ratio of 1:1 or 1:1+ (the latter indicating the USDT pool).

In this case, we should not allow the user to buy PAYKIK if the USDT pool state is (**<1*10**usdtContract.decimals()**).

Picture 6.

```
function getBuyRate(uint256 amountBase) public returns (uint256) {
        uint256 paykikPool = paykikToken.balanceOf(address(this));

        uint256 usdtPool = usdtToken.balanceOf(governorAddress);

        uint256 totalPay = 0;
        uint256 amount = amountBase;

        require(amountBase <= paykikPool, "Requested amount exceeds Paykik availible for purchase");

        (amount, totalPay, paykikPool) =
            buyPoolReminderBefore(paykikPool, usdtPool * (10 ** (paykikDecimals - usdtDecimals)), amount);

        totalPay = totalPay.add(calculate(getCirculationPaykik(), usdtPool, amountBase, true));

        totalPay =
            buyPoolReminderAfter(amountBase, amount, usdtPool * (10 ** (paykikDecimals - usdtDecimals)), totalPay);

        return totalPay;
    }
```

Picture 6. getBuyRate() method without validation USDT pool exhaustion (0ccf6f3c18e54441e2a80d2a4288e8558d93c1fd)

3.2. ***Solution:***

Validation was added (**require(usdtPool >=1*1e6,"Governor pool should be more than 1 USDT"**). In this case smart contract won't conduct calculations if USDT pool is less than 1.

```
function getBuyRate(uint256 amountBase) public view returns (uint256) {
        uint256 paykikPool = paykikToken.balanceOf(address(this));

        uint256 usdtPool = usdtToken.balanceOf(governorAddress);

        uint256 totalPay = 0;
        uint256 amount = amountBase;

        require(usdtPool >= 1 * 1e6, "Governor pool should be more than 1 USDT");

        require(amountBase <= paykikPool, "Requested amount exceeds Paykik availible for purchase");

        (amount, totalPay, paykikPool) =
            buyPoolReminderBefore(paykikPool, usdtPool * (10 ** (paykikDecimals - usdtDecimals)), amount);

        totalPay = totalPay.add(calculate(getCirculationPaykik(), usdtPool, amountBase, true));

        totalPay =
            buyPoolReminderAfter(amountBase, amount, usdtPool * (10 ** (paykikDecimals - usdtDecimals)), totalPay);

        return totalPay;
    }
```

Picture 7. Patch version of getBuyRate() method (0ccf6f3c18e54441e2a80d2a4288e8558d93c1fd)

Issue 4. Contract Governor.sol, Swap.sol.

4.1. Identified issue:

“getBuyRate() calculates price for cases that are not stipulated by the algorithm”.

- **Severity** - critical;
- **Type** - Integer-Underflow;
- **Vulnerable SHA1** - 0c981511066be1a3c8888e5d7e5b9bb6922d0795;
- **Mitigated SHA1** - d0b194c1b489e9b7e7bbd82a627840be28095859.

```
uint256 public constant paykikEmissionOnSwap = 18 * 1e5 * 1e8;

function _getParticipationRate(uint256 pollId) private view returns (uint256) {
        uint256 currentSwapBalance = paykikToken.balanceOf(swapAddress);
        uint256 teamTotalSent = ITeam(teamAddress).getTotalSent();
        uint256 circulatingAmount = teamTotalSent + (paykikEmissionOnSwap - currentSwapBalance);

        (bool success, uint256 rate) = (polls[pollId].totalVoted).mul(100).tryDiv(circulatingAmount);
        require(success, "Something went wrong, when trying to calculate participationRate");

        require(rate > 0, string(abi.encodePacked("Current Swap balance: ", currentSwapBalance.toString())));

        return rate;
    }
```

Picture 8. Vulnerable _getParticipationRate() method (0c981511066be1a3c8888e5d7e5b9bb6922d0795)

Vulnerability description:

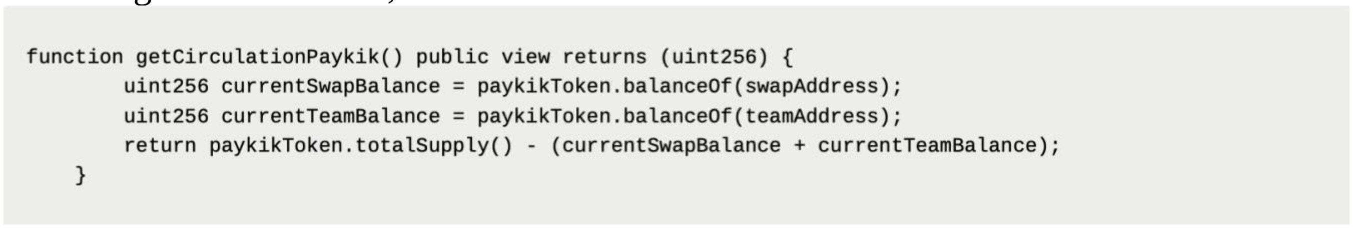
In this code block, circulation is not calculated relative to emission. This is an error as when PAYKIKs are unlocked from the **Team** contract pool, they migrate to **Swap's** balance.

Taking into account that (**paykikEmissionOnSwap**) is equal to (**18*1e5*1e8**) the current balance (**currentSwapBalance**) can tend to a number equal to the emission amount - in particular (**2*1e6*1e8**).

Therefore, this can lead to **((paykikEmissionOnSwap – currentSwapBalance) <0)**, that will result in Integer- Underflow, as it works with (unsigned integers).

4.2. **Solution:**

The line **(uint256 circulatingAmount = teamTotalSent + (paykikEmission + (paykikEmissionOnSwap – currentSwapBalance)**; was replaced with **uint256 circulatingAmount = getCirculationPaykik())**. In this case the circulation is calculated according to the emission, which is correct.



```
function getCirculationPaykik() public view returns (uint256) {
        uint256 currentSwapBalance = paykikToken.balanceOf(swapAddress);
        uint256 currentTeamBalance = paykikToken.balanceOf(teamAddress);
        return paykikToken.totalSupply() - (currentSwapBalance + currentTeamBalance);
    }
```

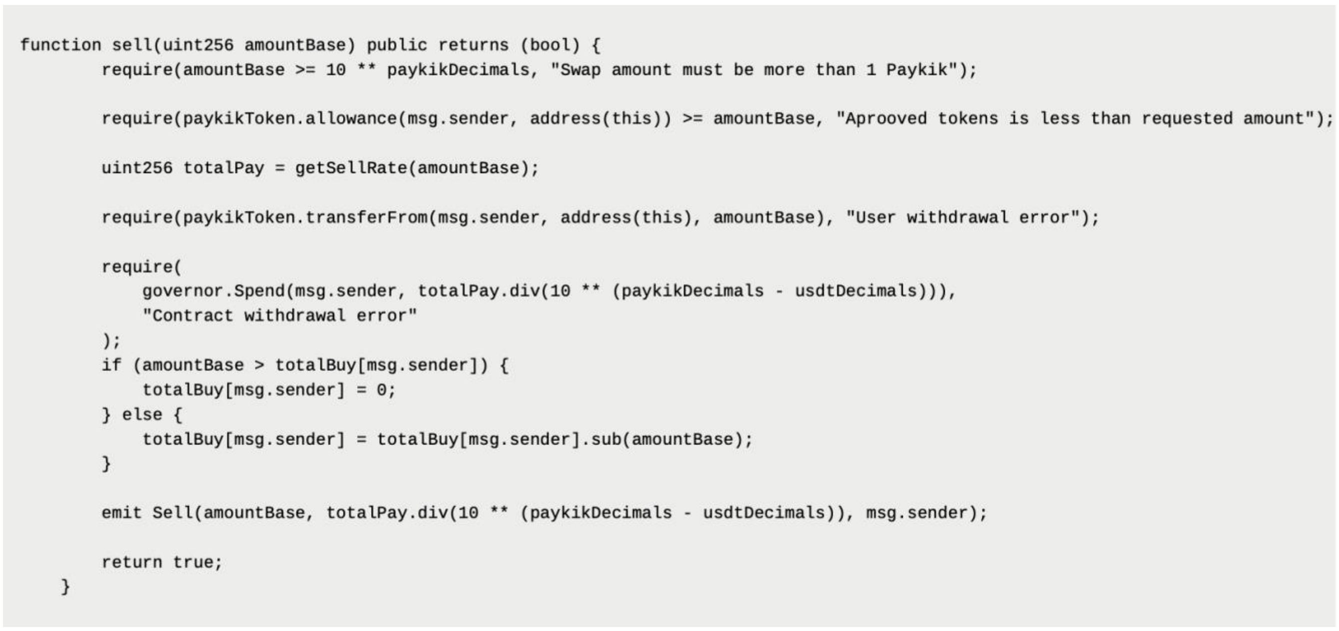
Picture 9. Patch version of calculation of paykiks circulation (d0b194c1b489e9b7e7bbd82a627840be28095859)

Issue 5. Contract Swap.sol

5.1. **Identified issue:**

"sell() leads to Denial Of Service by selling PAYKIK, without leaving USDT in the pool".

- **Severity** - high;
- **Type** - Dos;
- **Vulnerable SHA1** - 5cfb1e6cefeb9a8e5f4beb807b0b10160743a995;
- **Mitigated SHA1** - d0b194c1b489e9b7e7bbd82a627840be28095859.



```
function sell(uint256 amountBase) public returns (bool) {
        require(amountBase >= 10 ** paykikDecimals, "Swap amount must be more than 1 Paykik");

        require(paykikToken.allowance(msg.sender, address(this)) >= amountBase, "Aprooved tokens is less than requested amount");

        uint256 totalPay = getSellRate(amountBase);

        require(paykikToken.transferFrom(msg.sender, address(this), amountBase), "User withdrawal error");

        require(
            governor.Spend(msg.sender, totalPay.div(10 ** (paykikDecimals - usdtDecimals))),
            "Contract withdrawal error"
        );
        if (amountBase > totalBuy[msg.sender]) {
            totalBuy[msg.sender] = 0;
        } else {
            totalBuy[msg.sender] = totalBuy[msg.sender].sub(amountBase);
        }

        emit Sell(amountBase, totalPay.div(10 ** (paykikDecimals - usdtDecimals)), msg.sender);

        return true;
    }
```

Picture 10. Vulnerable sell()method (5cfb1e6cefeb9a8e5f4beb807b0b10160743a995)

Vulnerability description:

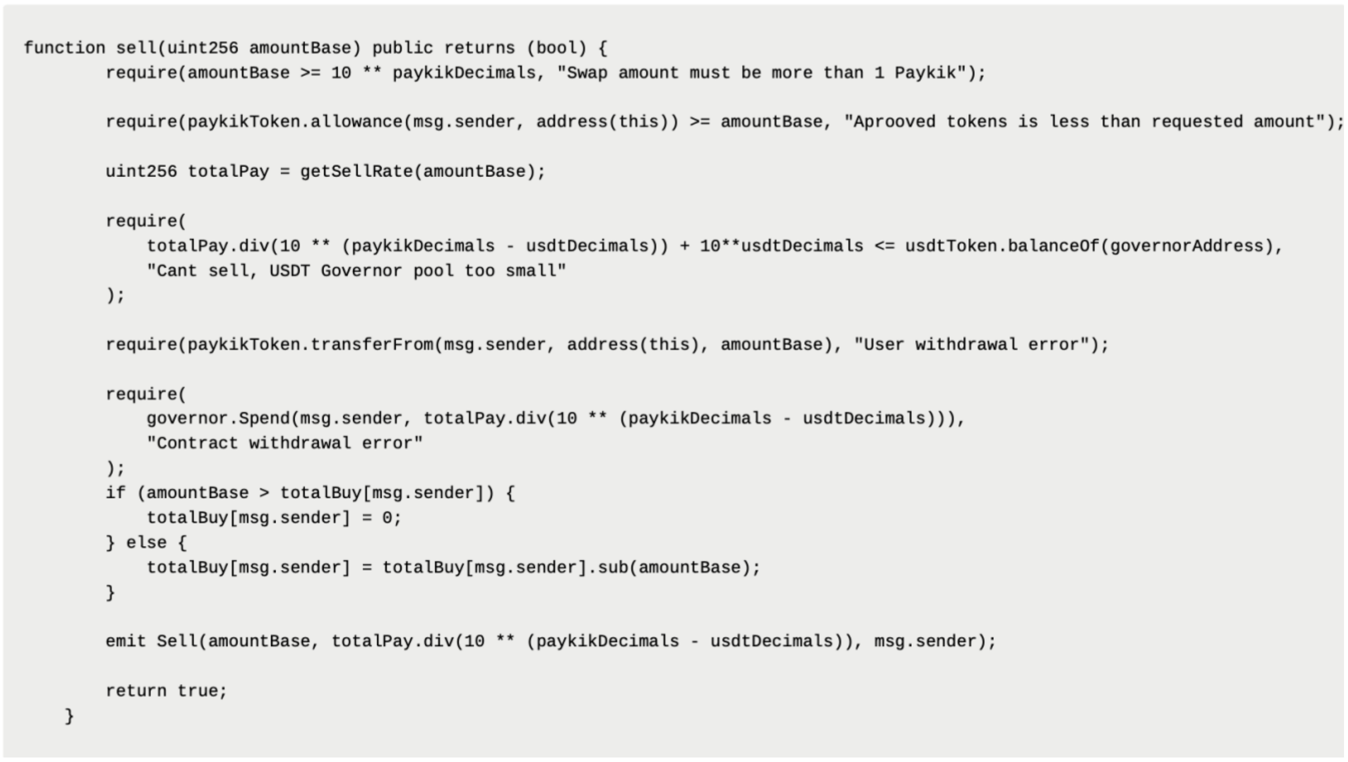
Price calculation method **getBuyRate()** stipulates a condition **require(usdtPool >= 1*1e6, "Governor pool should be more than 1 USDT")**. In case when this condition is not fulfilled, the user won't be able make a purchase.

In **sell()** method the user is not limited in terms of sell amount of the **PAYKIK → USDT** pair, that can lead to a condition when USDT on the balance is equal to **< 1 * 1e6**.

This case can lead to DoS, because until USDT balance on Governor contract is less than **1 * 1e6**, users won't be able to buy PAYKIK tokens.

5.2. **Solution:**

require(totalPay.div(10(paykikDecimals - usdtDecimals)) + 10**usdtDecimals <= usdtToken.balanceOf(governorAddress), "Cant sell, USDT Governor pool too small");** this function won’t allow USDT pool to be less than < 1 * 1e6



```
function sell(uint256 amountBase) public returns (bool) {
        require(amountBase >= 10 ** paykikDecimals, "Swap amount must be more than 1 Paykik");

        require(paykikToken.allowance(msg.sender, address(this)) >= amountBase, "Aprooved tokens is less than requested amount");

        uint256 totalPay = getSellRate(amountBase);

        require(
            totalPay.div(10 ** (paykikDecimals - usdtDecimals)) + 10**usdtDecimals <= usdtToken.balanceOf(governorAddress),
            "Cant sell, USDT Governor pool too small"
        );

        require(paykikToken.transferFrom(msg.sender, address(this), amountBase), "User withdrawal error");

        require(
            governor.Spend(msg.sender, totalPay.div(10 ** (paykikDecimals - usdtDecimals))),
            "Contract withdrawal error"
        );
        if (amountBase > totalBuy[msg.sender]) {
            totalBuy[msg.sender] = 0;
        } else {
            totalBuy[msg.sender] = totalBuy[msg.sender].sub(amountBase);
        }

        emit Sell(amountBase, totalPay.div(10 ** (paykikDecimals - usdtDecimals)), msg.sender);

        return true;
    }
```

Picture 11. Patch version of sell() method (d0b194c1b489e9b7e7bbd82a627840be28095859)

Picture 11. Patch block code

Issue 6. Contract Swap.sol, Team.sol, Hold.sol, Governor.sol

6.1. Identified issue:

"Transfer of all local mathematical operations to SafeMath.sol function".

- **Severity -** low;
- **Type -** mathematical vulnerabilities;
- **Vulnerable SHA1**-;
- **Mitigated SHA1** -.

Vulnerability description:

To maintain security standards, all of the mathematical calculations should be conducted using SafeMath.sol function.

6.2. Solution:

SafeMath.sol function was successfully integrated.

Reference: https://github.com/DAO-Paykiken-Geo/contracts/blob/testnet-version/contracts/base/SafeMath.sol

Issue 7. Contract Governor.sol.

7.1. Identified issue:

“Wrong msg.sender context in submitVote()”.

- **Severity** - critical;
- **Type** - erroneous context;
- **Vulnerable SHA1** - 0c2f8eadfe44384a0b93087ed2c0f69dafa287f3;
- **Mitigated SHA1** - f3c7e29a242d2cdf2098126f193911cb147d905d.

```
function submitVote(uint256 id) public returns (bool) {
        if (msg.sender == address(this)) {
            revert("Contract can't vote");
        }

        require(!pollsVotes[id][msg.sender], "You already voted");

        // check if active
        if (polls[id].deadline <= block.timestamp) {
            revert("Poll expired");
        }

        uint256 amount = paykikToken.balanceOf(msg.sender);

        if (!holdContract.deposit(polls[id].deadline, amount)) {
            revert("Deposit to hold failed");
        }

        uint256 deadline;
        (amount, deadline) = holdContract.get(msg.sender);

        pollsVotes[id][msg.sender] = true;

        finishPoll(id);

        return true;
    }
```

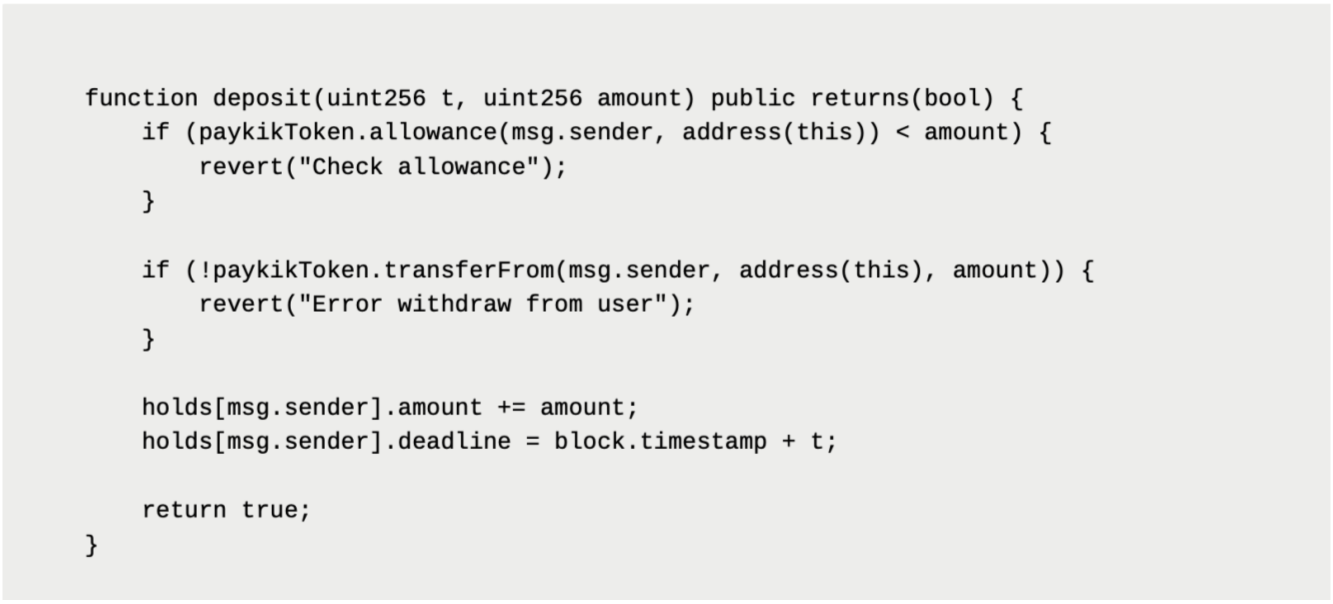
Picture 12. Vulnerable submitVote() method (0c2f8eadfe44384a0b93087ed2c0f69dafa287f3)

Vulnerability description:

```
if (!holdContract.deposit(polls[id].deadline, amount)) {
      revert("Deposit to hold failed");
}
```

Picture 13. Erroneuos context of deposit() method (0c2f8eadfe44384a0b93087ed2c0f69dafa287f3)

When deposit method of the **holdContract** is called, **Governor** contract is the transaction initiator, therefore the contract address will operate as **msg.sender** of built-in variable.



```
function deposit(uint256 t, uint256 amount) public returns(bool) {
    if (paykikToken.allowance(msg.sender, address(this)) < amount) {
        revert("Check allowance");
    }

    if (!paykikToken.transferFrom(msg.sender, address(this), amount)) {
        revert("Error withdraw from user");
    }

    holds[msg.sender].amount += amount;
    holds[msg.sender].deadline = block.timestamp + t;

    return true;
}
```

Picture 14. Implementation of deposit() method

As the result **Hold** smart contract will try to keep Paykik TRC20 from the **Governor** contract address, instead from the user that initially called the **submitVote()** method.

7.2. **Solution:**

A function could be utilized, which is similar to the methods realized in ERC20 standard **transfer()/transferFrom()**.

```
function depositFrom(
        address from,
        uint256 timePeriod,
        uint256 amount
    ) public returns (bool) {
        // Check if user approve fund transfer
        if (paykikToken.allowance(from, address(this)) < amount) {
            revert("Check allowance");
        }

        // Funds withdrawn
        if (!paykikToken.transferFrom(from, address(this), amount)) {
            // Error: couln't withdraw funds
            revert("Error withdraw from user");
        }

        holds[from].amount += amount;
        holds[from].deadline = block.timestamp + timePeriod;

        return true;
    }
```

Picture 15. New implementation of the deposit logic with the right context

And transfer when calling **sumbitVote()** to the **transferFrom()** method as an argument from the value of the variable **msg.sender** - which will be the user's address.

```
if (!holdContract.depositFrom(msg.sender, polls[id].deadline, amount)) {
            revert("Deposit to hold failed");
        }
```

Picture 16. Patch version of deposit logic (f3c7e29a242d2cdf2098126f193911cb147d905d)

Executive Summary

The DAO community tests identified 7 potential improvement points that were mitigated and are the main assessment subject under this audit. The modifications included:

- Swap.sol - “Unrecorded tokens”;
- Governor.sol - “Integer-Underflow;
- Swap.sol - “Manipulation over the price of tokens”;
- Governor.sol, Swap.sol - “Integer-Underflow”;
- Swap.sol - “DoS”;
- Swap.sol, Team.sol, Hold.sol, Governor.sol - “Mathematical vulnerabilities”;
- Governor.sol - “Erroneous context”.

The contracts are specified to be complied with Solidity 0.8.21 this is the latest maintenance release of the 0.8.x series.

The contracts compile without any errors or warnings. Linting does not show any problems. The code is very clean and well commented.

The repository contains a very comprehensive set of **66** unit tests for the smart contracts. After taking care of some minor hiccups (‘out of memory’ errors and timeouts), all tests passed.

Results:
All of the changes made to smart contracts were successful.
The Paykiken Geo Protocol is stable and secure.