Guard

Smart contracts security assessment

Final report

Tariff: Top

Warren Finance

November 2023

0xguard.com

hello@0xguard.com

Contents

Introduction

The report has been prepared for **Warren Finance**.

The project is a Uniswap V2 fork with a staking contract. The original Uniswap V2 code was modified to work with only one pair (supposedly the WarrenToken/Wrapped native token). The staking contract allows to create Bonds by providing liquidity to the pair and getting the Warren tokens after some time in return.

The SHA-1 hashes of audited files are:

warren_latest/WarrenToken.sol c0d65e6d2c25221b7313d0a154e7b05897db20bd

warren_latest/SimplifiedUniRouter.sol e69ca174aadd2e0649f3b722968f5bbd9edd80e3

warren_latest/WarrenProtocol.sol d320b364b61729987916ae00a78f4dc8e2b7391e

warren_latest/SimplifiedUniPair.sol 687df8b7973eeda8cb5b3ae54e74c4092dcc6ebe

A recheck was done for the following files with SHA-1 hashes:

Warren_latest recheck 2/WarrenToken.sol 31258a09d1311999d8c27a86a6f3057b4c28b612

Warren_latest recheck 2/SimplifiedUniRouter.sol
db5e31d190b90f6222991a7df0f0dde174dfe61d

Warren_latest recheck 2/WarrenProtocol.sol 59ab63807ceb4d854baf36edb6eba6dd32c87a06

Warren_latest recheck 2/SimplifiedUniPair.sol 687df8b7973eeda8cb5b3ae54e74c4092dcc6ebe

Name	Warren Finance
Audit date	2023-11-22 - 2023-11-24
Language	Solidity

Platform	Binance Smart Chain

Contracts checked

Name	Address
UniswapV2Pair	
UniswapV2Router02	
WARRENProtocol	
WARRENToken	

Procedure

We perform our audit according to the following procedure:

Automated analysis

- Scanning the project's smart contracts with several publicly available automated Solidity analysis tools
- Manual verification (reject or confirm) all the issues found by the tools

Manual audit

- Manually analyze smart contracts for security vulnerabilities
- Smart contracts' logic check

Known vulnerabilities checked

Title	Check result
Unencrypted Private Data On-Chain	passed
Code With No Effects	not passed
Message call with hardcoded gas amount	passed

Typographical Error	passed
DoS With Block Gas Limit	passed
Presence of unused variables	passed
Incorrect Inheritance Order	passed
Requirement Violation	passed
Weak Sources of Randomness from Chain Attributes	passed
Shadowing State Variables	passed
Incorrect Constructor Name	passed
Block values as a proxy for time	passed
Authorization through tx.origin	passed
DoS with Failed Call	passed
Delegatecall to Untrusted Callee	passed
Use of Deprecated Solidity Functions	passed
Assert Violation	passed
State Variable Default Visibility	passed
Reentrancy	passed
Unprotected SELFDESTRUCT Instruction	passed
Unprotected Ether Withdrawal	passed
Unchecked Call Return Value	passed
Floating Pragma	passed
Outdated Compiler Version	passed
Integer Overflow and Underflow	passed
Function Default Visibility	passed

Classification of issue severity

High severity	High severity issues can cause a significant or full loss of funds, change of contract ownership, major interference with contract logic. Such issues require immediate attention.
Medium severity	Medium severity issues do not pose an immediate risk, but can be detrimental to the client's reputation if exploited. Medium severity issues may lead to a contract failure and can be fixed by modifying the contract state or redeployment. Such issues require attention.
Low severity	Low severity issues do not cause significant destruction to the contract's functionality. Such issues are recommended to be taken into consideration.

Issues

High severity issues

1. Not transferred native token (WARRENProtocol)

Status: Fixed

The function `stake()` adds liquidity to the pair of the Warren token and the `DEFAULT_TOKEN`, but the amount of `DEFAULT` token to add to liquidity is never transferred to the contract.

```
  function stake(uint8 bondIdx, uint256 amount) external {
    ...

    WARRENToken(TOKEN_ADDRESS).mint(address(this), liquidityTokensAmount);
    WARRENToken(TOKEN_ADDRESS).increaseAllowance(UNISWAP_ROUTER_ADDRESS,
liquidityTokensAmount);

    (uint256 amountToken, uint256 amountETH, uint256 liquidity) =
IUniswapV2Router01(UNISWAP_ROUTER_ADDRESS).addLiquidity(
        TOKEN_ADDRESS,
        address(DEFAULT_TOKEN),
        liquidityTokensAmount,
```

```
        defaultTokenAmount,
        0,
        0,
        address(this),
        block.timestamp + 5 minutes
    );
...
}
```

Recommendation: Transfer the required amount of tokens to the Warren contract to add to liquidity.

Update: Transfer of the default token was added to the function.

2. Possible mixed up reserves calculations (WARRENProtocol)

Status: Fixed

The contract calculates amounts to be minted by checking the reserves of the pair.

```
  function getETHAmount(uint256 tokensAmount) public view returns(uint256) {
    (uint256 reserve0, uint256 reserve1, ) =
IUniswapV2Pair(LP_TOKEN_ADDRESS).getReserves();

    return tokensAmount * reserve0 / reserve1;
  }

  function getTokensAmount(uint256 amount) public view returns(uint256) {
    (uint256 reserve0, uint256 reserve1, ) =
IUniswapV2Pair(LP_TOKEN_ADDRESS).getReserves();

    return amount * reserve1 / reserve0;
  }
```

Based on the deployed addresses the pair for the reserve 0 may be returned reserve of Warren token or the DEFAULT token. Which token reserve is returned it not checked in the code.

Recommendation: Ensure that the code uses correct reserves for calculations.

Update: Conditions to select the right reserve for calculations were added to the functions.

3. The owner of the contract can stop token transfers (WARRENToken)

Status: Fixed

The owner of the contract can stop token transfers with function `lockBuy()`. All token transfers except with recipients of the WarrenContract, router, zero address, or the pair would be reverted.

```
  function lockBuy() external onlyOwner  {//EDIT::KEEP OR REMOVE?
    buyLocked = true;
  }

 function _beforeTokenTransfer(address from, address to, uint256 ) internal view
override {
    if (LP_TOKEN_ADDRESS == address(0) || !buyLocked) {
      return;
    }

    if (from == LP_TOKEN_ADDRESS || from == PULSEX_ROUTER_ADDRESS) {
      require(
           to == mainContractAddress
        || to == PULSEX_ROUTER_ADDRESS
        || to == LP_TOKEN_ADDRESS
        || to == address(0),
        "Transfer: only main contract can buy tokens"
      );
    }
  }
```

Recommendation: Use multisig and timelock for the owner with at least 48 hours or remove the function `lockBuy()`

Update: The lock functionality was removed from the token code.

Medium severity issues

1. Potential lock of funds (UniswapV2Router02)

Status: Fixed

The UniswapV2Router02 router is a simplified version of the original UniswapV2Router02 contract. It works only with one pair but still receives token addresses as parameters. Passing a wrong token address may result in transferring these tokens to the contract and these tokens will be locked on the contract.

```
    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint amountA, uint amountB,
uint liquidity) {
        (amountA, amountB) = _addLiquidity(tokenA, tokenB, amountADesired,
amountBDesired, amountAMin, amountBMin);
        TransferHelper.safeTransferFrom(tokenA, msg.sender, hardCodedPair, amountA);
        TransferHelper.safeTransferFrom(tokenB, msg.sender, hardCodedPair, amountB);
        liquidity = IUniswapV2Pair(hardCodedPair).mint(to);
    }
```

Recommendation: Remove the token addresses from parameters from and use tokens from the hardcoded pair, or add require statements to ensure that the right addresses are passed as parameters.

Update: The issue was fixed in the code update. Checks were added to ensure the right token parameters are passed inside `getReserves()`;

2. Possible locked native currency (WARRENProtocol)

Status: Fixed

The contract has the function `receive()` to allow sending native currency to it. However, there is no way to use the sent currency. Any sent to the contract native currency would be locked.

```
  receive() external payable {

  }
```

Recommendation: Remove the `receive()` function.

Update: The function was removed in the update.

3. Incorrect calculations for amount of liquidity withdrawal in the sell() function (WARRENProtocol)

Status: Fixed

The function `sell()` collects a user's rewards and sells them to get native currency. Then, it calculates the amount of liquidity to be used as a buyback. If, for example, the amount of the Warren tokens is big enough to get half of the native tokens from liquidity, the function will calculate the amount of liquidity tokens for the buyback bigger than the liquidity tokens total supply.

```
  function sell(uint256 tokensAmount) external {
    ...
    uint256[] memory amounts =
IUniswapV2Router01(UNISWAP_ROUTER_ADDRESS).swapExactTokensForTokens(
      tokensAmount,
      0,
      path,
      msg.sender,
      block.timestamp + 5 minutes
    );
    uint256 ethAmount = amounts[1];


    (uint256 ethReserved, ) = getTokenLiquidity();
    uint256 liquidity = ERC20(LP_TOKEN_ADDRESS).totalSupply()
```

```
        * ethAmount
        * (Constants.PERCENTS_DIVIDER + PRICE_BALANCER_PERCENT)
        / Constants.PERCENTS_DIVIDER
        / ethReserved;
    ...
}
```

Recommendation: Modify the calculations of the tokens to ensure that the amount of liquidity tokens for buyback is less than the total supply.

Update: The issue was fixed in the code update.

Low severity issues

1. Interactions with an external contract out of scope of the audit (UniswapV2Pair)

Status: Open

The functions skim() and syn() make calls to the Masterchef contract which is out of the scope of the current audit. This contract may potentially create DoS attack and make these functions fail.

```
        modifier farm() {
        uint _depositedLp = masterchef.userInfo(1, address(this)).amount; //1 = pid of
DAI-WPLS farm
        masterchef.withdraw(1, _depositedLp);
        _;
        uint _avaliableLp = IERC20(daiPlsLP).balanceOf(address(this));
        IERC20(daiPlsLP).approve(address(masterchef), _avaliableLp); //@audit could be
approved once
        masterchef.deposit(1, _avaliableLp);
    }

// force balances to match reserves
    function skim(address to) external lock farm{
        address _token0 = token0; // gas savings
        address _token1 = token1; // gas savings
        _safeTransfer(_token0, to,
IERC20(_token0).balanceOf(address(this)).sub(reserve0));
```

```
        _safeTransfer(_token1, to,
IERC20(_token1).balanceOf(address(this)).sub(reserve1));
    }

    // force reserves to match balances
    function sync() external lock farm{
        _update(IERC20(token0).balanceOf(address(this)),
IERC20(token1).balanceOf(address(this)), reserve0, reserve1);
    }
```

2. Pointless deadline calculations (WARRENProtocol)
Status: Fixed

The contract in interactions with the routers calculates deadline as current block timestamp plus 5 minutes.

```
    amounts = IUniswapV2Router01(UNISWAP_ROUTER_ADDRESS).swapExactTokensForTokens(
      amountDEFAULT,
      0,
      path,
      address(this),
      block.timestamp + 5 minutes
    );
```

There is no way to set the deadline on-chain, the router always checks if `block.timestamp <= block.timestamp + 5 minutes` which is always true

Recommendation: Use just `block.timetamp` as the deadline.

Update: The deadline parameter has been changed to `block.timestamp`.

3. Dependency on the current pair reserves (WARRENProtocol)
Status: Open

The amount of rewards is calculated based on the instant pair reserves. Although there is no way to make a classic flash loan attack because token purchases are restricted only to specific addresses, large sells may affect reward calculations. For example, bond.stakeAmount will have different values

when if the stake() function is called before or after a sell.

```
  function getETHAmount(uint256 tokensAmount) public view returns(uint256) {
    (uint256 reserve0, uint256 reserve1, ) =
IUniswapV2Pair(LP_TOKEN_ADDRESS).getReserves();

    return tokensAmount * reserve0 / reserve1;
  }

  function getTokensAmount(uint256 amount) public view returns(uint256) {
    (uint256 reserve0, uint256 reserve1, ) =
IUniswapV2Pair(LP_TOKEN_ADDRESS).getReserves();

    return amount * reserve1 / reserve0;
  }
```

Recommendation: Use oracles to prevent price manipulations.

Conclusion

Warren Finance UniswapV2Pair, UniswapV2Router02, WARRENProtocol, WARRENToken contracts were audited. 3 high, 3 medium, 3 low severity issues were found.
3 high, 3 medium, 1 low severity issues have been fixed in the update.

No tests were provided with the code. The absence of tests poses a significant risk as it hinders the ability to verify the smart contract's functionality, security, and performance under various scenarios. To ensure that the code works as intended it's crucial to provide comprehensive test coverage.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability)set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without 0xGuard prior written consent.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts 0xGuard to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Static code analysis

```
Reentrancy in WARRENProtocol.buy(address,uint8,uint256) (WarrenProtocol.sol#1025-1062):
        External calls:
        - DEFAULT_TOKEN.safeTransferFrom(msg.sender,address(this),amount)
(WarrenProtocol.sol#1034)
        - refReward = distributeRefPayout(user,amount,isNewUser)
(WarrenProtocol.sol#1057)
                - returndata = address(token).functionCall(data)
(WarrenProtocol.sol#402)
                - (success,returndata) = target.call{value: value}(data)
(WarrenProtocol.sol#243)
                - DEFAULT_TOKEN.safeTransfer(upline,amount) (WarrenProtocol.sol#1111)
        - DEFAULT_TOKEN.safeTransfer(owner(),adminFee) (WarrenProtocol.sol#1059)
        - newBond(msg.sender,bondType,amount,amount - adminFee - refReward)
(WarrenProtocol.sol#1061)
                - WARRENToken(TOKEN_ADDRESS).mint(address(this),tokensAmount)
(WarrenProtocol.sol#1182)
                -
WARRENToken(TOKEN_ADDRESS).increaseAllowance(UNISWAP_ROUTER_ADDRESS,tokensAmount)
(WarrenProtocol.sol#1183)
                - (amountToken,amountETH,liquidity) = IUniswapV2Router01(UNISWAP_ROUTER_
ADDRESS).addLiquidity(TOKEN_ADDRESS,address(DEFAULT_TOKEN),tokensAmount,liquidityAmount,
0,0,address(this),block.timestamp + 300) (WarrenProtocol.sol#1185-1196)
        External calls sending eth:
        - refReward = distributeRefPayout(user,amount,isNewUser)
(WarrenProtocol.sol#1057)
                - (success,returndata) = target.call{value: value}(data)
(WarrenProtocol.sol#243)
        State variables written after the call(s):
        - newBond(msg.sender,bondType,amount,amount - adminFee - refReward)
(WarrenProtocol.sol#1061)
                - user.lastActionTime = block.timestamp (WarrenProtocol.sol#1171)
                - user.bondsNumber ++ (WarrenProtocol.sol#1174)
                - user.totalInvested += bondAmount (WarrenProtocol.sol#1175)
        WARRENProtocol.users (WarrenProtocol.sol#965) can be used in cross function
reentrancies:
        - WARRENProtocol.buy(address,uint8,uint256) (WarrenProtocol.sol#1025-1062)
        - WARRENProtocol.claim(uint256) (WarrenProtocol.sol#1293-1307)
```

```
        - WARRENProtocol.collect(address) (WarrenProtocol.sol#1420-1465)
        - WARRENProtocol.distributeRefPayout(Models.User,uint256,bool)
(WarrenProtocol.sol#1064-1140)
        - WARRENProtocol.getHoldBonusPercent(address) (WarrenProtocol.sol#1540-1552)
        - WARRENProtocol.getLiquidityBonusPercent(address)
(WarrenProtocol.sol#1554-1562)
        - WARRENProtocol.getUIData(address) (WarrenProtocol.sol#1564-1580)
        - WARRENProtocol.influencerBond(address,uint256) (WarrenProtocol.sol#1405-1418)
        - WARRENProtocol.newBond(address,uint8,uint256,uint256)
(WarrenProtocol.sol#1156-1208)
        - WARRENProtocol.rebond(uint256) (WarrenProtocol.sol#1312-1329)
        - WARRENProtocol.sell(uint256) (WarrenProtocol.sol#1331-1397)
        - WARRENProtocol.stake(uint8,uint256) (WarrenProtocol.sol#1214-1264)
        - WARRENProtocol.transfer(uint8) (WarrenProtocol.sol#1270-1290)
        - WARRENProtocol.updateReferralLevel(address,uint256)
(WarrenProtocol.sol#1142-1152)
        - WARRENProtocol.userBalance(address) (WarrenProtocol.sol#1467-1506)
        - WARRENProtocol.users (WarrenProtocol.sol#965)
Reentrancy in WARRENProtocol.stake(uint8,uint256) (WarrenProtocol.sol#1214-1264):
        External calls:
        - refReward = distributeRefPayout(user,amount,false) (WarrenProtocol.sol#1227)
                - returndata = address(token).functionCall(data)
(WarrenProtocol.sol#402)
                - (success,returndata) = target.call{value: value}(data)
(WarrenProtocol.sol#243)
                - DEFAULT_TOKEN.safeTransfer(upline,amount) (WarrenProtocol.sol#1111)
        - DEFAULT_TOKEN.safeTransfer(owner(),adminFee) (WarrenProtocol.sol#1229)
        - WARRENToken(TOKEN_ADDRESS).mint(address(this),liquidityTokensAmount)
(WarrenProtocol.sol#1236)
        - WARRENToken(TOKEN_ADDRESS).increaseAllowance(UNISWAP_ROUTER_ADDRESS,liquidityT
okensAmount) (WarrenProtocol.sol#1237)
        - (amountToken,amountETH,liquidity) = IUniswapV2Router01(UNISWAP_ROUTER_ADDRESS)
.addLiquidity(TOKEN_ADDRESS,address(DEFAULT_TOKEN),liquidityTokensAmount,defaultTokenAmo
unt,0,0,address(this),block.timestamp + 300) (WarrenProtocol.sol#1239-1248)
        External calls sending eth:
        - refReward = distributeRefPayout(user,amount,false) (WarrenProtocol.sol#1227)
                - (success,returndata) = target.call{value: value}(data)
(WarrenProtocol.sol#243)
        State variables written after the call(s):
        - bond.stakeAmount = 2 * tokensAmount (WarrenProtocol.sol#1256)
        WARRENProtocol.bonds (WarrenProtocol.sol#966) can be used in cross function
reentrancies:
```

```
        - WARRENProtocol.bonds (WarrenProtocol.sol#966)
        - WARRENProtocol.collect(address) (WarrenProtocol.sol#1420-1465)
        - WARRENProtocol.newBond(address,uint8,uint256,uint256)
(WarrenProtocol.sol#1156-1208)
        - WARRENProtocol.stake(uint8,uint256) (WarrenProtocol.sol#1214-1264)
        - WARRENProtocol.transfer(uint8) (WarrenProtocol.sol#1270-1290)
        - WARRENProtocol.userBalance(address) (WarrenProtocol.sol#1467-1506)
        - bond.stakeTime = block.timestamp (WarrenProtocol.sol#1257)
        WARRENProtocol.bonds (WarrenProtocol.sol#966) can be used in cross function
reentrancies:
        - WARRENProtocol.bonds (WarrenProtocol.sol#966)
        - WARRENProtocol.collect(address) (WarrenProtocol.sol#1420-1465)
        - WARRENProtocol.newBond(address,uint8,uint256,uint256)
(WarrenProtocol.sol#1156-1208)
        - WARRENProtocol.stake(uint8,uint256) (WarrenProtocol.sol#1214-1264)
        - WARRENProtocol.transfer(uint8) (WarrenProtocol.sol#1270-1290)
        - WARRENProtocol.userBalance(address) (WarrenProtocol.sol#1467-1506)
        - bond.collectedTime = block.timestamp (WarrenProtocol.sol#1258)
        WARRENProtocol.bonds (WarrenProtocol.sol#966) can be used in cross function
reentrancies:
        - WARRENProtocol.bonds (WarrenProtocol.sol#966)
        - WARRENProtocol.collect(address) (WarrenProtocol.sol#1420-1465)
        - WARRENProtocol.newBond(address,uint8,uint256,uint256)
(WarrenProtocol.sol#1156-1208)
        - WARRENProtocol.stake(uint8,uint256) (WarrenProtocol.sol#1214-1264)
        - WARRENProtocol.transfer(uint8) (WarrenProtocol.sol#1270-1290)
        - WARRENProtocol.userBalance(address) (WarrenProtocol.sol#1467-1506)
        - bond.stakingRewardLimit = bond.stakeAmount *
Constants.STAKING_REWARD_LIMIT_PERCENT / Constants.PERCENTS_DIVIDER
(WarrenProtocol.sol#1259)
        WARRENProtocol.bonds (WarrenProtocol.sol#966) can be used in cross function
reentrancies:
        - WARRENProtocol.bonds (WarrenProtocol.sol#966)
        - WARRENProtocol.collect(address) (WarrenProtocol.sol#1420-1465)
        - WARRENProtocol.newBond(address,uint8,uint256,uint256)
(WarrenProtocol.sol#1156-1208)
        - WARRENProtocol.stake(uint8,uint256) (WarrenProtocol.sol#1214-1264)
        - WARRENProtocol.transfer(uint8) (WarrenProtocol.sol#1270-1290)
        - WARRENProtocol.userBalance(address) (WarrenProtocol.sol#1467-1506)
        - user.liquidityCreated += amount (WarrenProtocol.sol#1250)
        WARRENProtocol.users (WarrenProtocol.sol#965) can be used in cross function
reentrancies:
```

```
        - WARRENProtocol.buy(address,uint8,uint256) (WarrenProtocol.sol#1025-1062)
        - WARRENProtocol.claim(uint256) (WarrenProtocol.sol#1293-1307)
        - WARRENProtocol.collect(address) (WarrenProtocol.sol#1420-1465)
        - WARRENProtocol.distributeRefPayout(Models.User,uint256,bool)
(WarrenProtocol.sol#1064-1140)
        - WARRENProtocol.getHoldBonusPercent(address) (WarrenProtocol.sol#1540-1552)
        - WARRENProtocol.getLiquidityBonusPercent(address)
(WarrenProtocol.sol#1554-1562)
        - WARRENProtocol.getUIData(address) (WarrenProtocol.sol#1564-1580)
        - WARRENProtocol.influencerBond(address,uint256) (WarrenProtocol.sol#1405-1418)
        - WARRENProtocol.newBond(address,uint8,uint256,uint256)
(WarrenProtocol.sol#1156-1208)
        - WARRENProtocol.rebond(uint256) (WarrenProtocol.sol#1312-1329)
        - WARRENProtocol.sell(uint256) (WarrenProtocol.sol#1331-1397)
        - WARRENProtocol.stake(uint8,uint256) (WarrenProtocol.sol#1214-1264)
        - WARRENProtocol.transfer(uint8) (WarrenProtocol.sol#1270-1290)
        - WARRENProtocol.updateReferralLevel(address,uint256)
(WarrenProtocol.sol#1142-1152)
        - WARRENProtocol.userBalance(address) (WarrenProtocol.sol#1467-1506)
        - WARRENProtocol.users (WarrenProtocol.sol#965)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-
vulnerabilities
INFO:Detectors:
WARRENProtocol.getHoldBonusPercent(address) (WarrenProtocol.sol#1540-1552) performs a
multiplication on the result of a division:
        - bonusPercent = (block.timestamp - users[userAddr].lastActionTime) /
Constants.USER_HOLD_BONUS_STEP * Constants.USER_HOLD_BONUS_STEP_PERCENT
(WarrenProtocol.sol#1545-1547)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-
multiply
INFO:Detectors:
WARRENProtocol.buy(address,uint8,uint256) (WarrenProtocol.sol#1025-1062) uses a
dangerous strict equality:
        - upline == address(0) || upline == msg.sender || users[upline].bondsNumber ==
0 (WarrenProtocol.sol#1042)
WARRENProtocol.distributeRefPayout(Models.User,uint256,bool)
(WarrenProtocol.sol#1064-1140) uses a dangerous strict equality:
        - upline == address(0) (WarrenProtocol.sol#1079)
WARRENProtocol.distributeRefPayout(Models.User,uint256,bool)
(WarrenProtocol.sol#1064-1140) uses a dangerous strict equality:
        - j == 0 (WarrenProtocol.sol#1091)
```

```
WARRENProtocol.distributeRefPayout(Models.User,uint256,bool)
(WarrenProtocol.sol#1064-1140) uses a dangerous strict equality:
        - j_scope_0 == 0 (WarrenProtocol.sol#1103)
WARRENProtocol.distributeRefPayout(Models.User,uint256,bool)
(WarrenProtocol.sol#1064-1140) uses a dangerous strict equality:
        - upline == address(0) (WarrenProtocol.sol#1131)
WARRENProtocol.getHoldBonusPercent(address) (WarrenProtocol.sol#1540-1552) uses a
dangerous strict equality:
        - users[userAddr].lastActionTime == 0 (WarrenProtocol.sol#1541)
WARRENProtocol.stake(uint8,uint256) (WarrenProtocol.sol#1214-1264) uses a dangerous
strict equality:
        - require(bool,string)(bonds[msg.sender][bondIdx].stakeTime == 0,Stake: this
bond was already staked) (WarrenProtocol.sol#1217)
WARRENProtocol.userBalance(address) (WarrenProtocol.sol#1467-1506) uses a dangerous
strict equality:
        - bond.stakeTime == 0 (WarrenProtocol.sol#1479)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-
strict-equalities
INFO:Detectors:
Reentrancy in WARRENProtocol.buy(address,uint8,uint256) (WarrenProtocol.sol#1025-1062):
        External calls:
        - DEFAULT_TOKEN.safeTransferFrom(msg.sender,address(this),amount)
(WarrenProtocol.sol#1034)
        State variables written after the call(s):
        - user.upline = upline (WarrenProtocol.sol#1045)
        WARRENProtocol.users (WarrenProtocol.sol#965) can be used in cross function
reentrancies:
        - WARRENProtocol.buy(address,uint8,uint256) (WarrenProtocol.sol#1025-1062)
        - WARRENProtocol.claim(uint256) (WarrenProtocol.sol#1293-1307)
        - WARRENProtocol.collect(address) (WarrenProtocol.sol#1420-1465)
        - WARRENProtocol.distributeRefPayout(Models.User,uint256,bool)
(WarrenProtocol.sol#1064-1140)
        - WARRENProtocol.getHoldBonusPercent(address) (WarrenProtocol.sol#1540-1552)
        - WARRENProtocol.getLiquidityBonusPercent(address)
(WarrenProtocol.sol#1554-1562)
        - WARRENProtocol.getUIData(address) (WarrenProtocol.sol#1564-1580)
        - WARRENProtocol.influencerBond(address,uint256) (WarrenProtocol.sol#1405-1418)
        - WARRENProtocol.newBond(address,uint8,uint256,uint256)
(WarrenProtocol.sol#1156-1208)
        - WARRENProtocol.rebond(uint256) (WarrenProtocol.sol#1312-1329)
        - WARRENProtocol.sell(uint256) (WarrenProtocol.sol#1331-1397)
```

```
        - WARRENProtocol.stake(uint8,uint256) (WarrenProtocol.sol#1214-1264)
        - WARRENProtocol.transfer(uint8) (WarrenProtocol.sol#1270-1290)
        - WARRENProtocol.updateReferralLevel(address,uint256)
(WarrenProtocol.sol#1142-1152)
        - WARRENProtocol.userBalance(address) (WarrenProtocol.sol#1467-1506)
        - WARRENProtocol.users (WarrenProtocol.sol#965)
        - users[upline].referrals.push(msg.sender) (WarrenProtocol.sol#1048)
        WARRENProtocol.users (WarrenProtocol.sol#965) can be used in cross function
reentrancies:
        - WARRENProtocol.buy(address,uint8,uint256) (WarrenProtocol.sol#1025-1062)
        - WARRENProtocol.claim(uint256) (WarrenProtocol.sol#1293-1307)
        - WARRENProtocol.collect(address) (WarrenProtocol.sol#1420-1465)
        - WARRENProtocol.distributeRefPayout(Models.User,uint256,bool)
(WarrenProtocol.sol#1064-1140)
        - WARRENProtocol.getHoldBonusPercent(address) (WarrenProtocol.sol#1540-1552)
        - WARRENProtocol.getLiquidityBonusPercent(address)
(WarrenProtocol.sol#1554-1562)
        - WARRENProtocol.getUIData(address) (WarrenProtocol.sol#1564-1580)
        - WARRENProtocol.influencerBond(address,uint256) (WarrenProtocol.sol#1405-1418)
        - WARRENProtocol.newBond(address,uint8,uint256,uint256)
(WarrenProtocol.sol#1156-1208)
        - WARRENProtocol.rebond(uint256) (WarrenProtocol.sol#1312-1329)
        - WARRENProtocol.sell(uint256) (WarrenProtocol.sol#1331-1397)
        - WARRENProtocol.stake(uint8,uint256) (WarrenProtocol.sol#1214-1264)
        - WARRENProtocol.transfer(uint8) (WarrenProtocol.sol#1270-1290)
        - WARRENProtocol.updateReferralLevel(address,uint256)
(WarrenProtocol.sol#1142-1152)
        - WARRENProtocol.userBalance(address) (WarrenProtocol.sol#1467-1506)
        - WARRENProtocol.users (WarrenProtocol.sol#965)
Reentrancy in WARRENProtocol.distributeRefPayout(Models.User,uint256,bool)
(WarrenProtocol.sol#1064-1140):
        External calls:
        - DEFAULT_TOKEN.safeTransfer(upline,amount) (WarrenProtocol.sol#1111)
        State variables written after the call(s):
        - users[upline].totalRefReward += amount (WarrenProtocol.sol#1112)
        WARRENProtocol.users (WarrenProtocol.sol#965) can be used in cross function
reentrancies:
        - WARRENProtocol.buy(address,uint8,uint256) (WarrenProtocol.sol#1025-1062)
        - WARRENProtocol.claim(uint256) (WarrenProtocol.sol#1293-1307)
        - WARRENProtocol.collect(address) (WarrenProtocol.sol#1420-1465)
        - WARRENProtocol.distributeRefPayout(Models.User,uint256,bool)
(WarrenProtocol.sol#1064-1140)
```

```
        - WARRENProtocol.getHoldBonusPercent(address) (WarrenProtocol.sol#1540-1552)
        - WARRENProtocol.getLiquidityBonusPercent(address)
(WarrenProtocol.sol#1554-1562)
        - WARRENProtocol.getUIData(address) (WarrenProtocol.sol#1564-1580)
        - WARRENProtocol.influencerBond(address,uint256) (WarrenProtocol.sol#1405-1418)
        - WARRENProtocol.newBond(address,uint8,uint256,uint256)
(WarrenProtocol.sol#1156-1208)
        - WARRENProtocol.rebond(uint256) (WarrenProtocol.sol#1312-1329)
        - WARRENProtocol.sell(uint256) (WarrenProtocol.sol#1331-1397)
        - WARRENProtocol.stake(uint8,uint256) (WarrenProtocol.sol#1214-1264)
        - WARRENProtocol.transfer(uint8) (WarrenProtocol.sol#1270-1290)
        - WARRENProtocol.updateReferralLevel(address,uint256)
(WarrenProtocol.sol#1142-1152)
        - WARRENProtocol.userBalance(address) (WarrenProtocol.sol#1467-1506)
        - WARRENProtocol.users (WarrenProtocol.sol#965)
        - users[upline].refs[i] ++ (WarrenProtocol.sol#1120)
       WARRENProtocol.users (WarrenProtocol.sol#965) can be used in cross function
reentrancies:
        - WARRENProtocol.buy(address,uint8,uint256) (WarrenProtocol.sol#1025-1062)
        - WARRENProtocol.claim(uint256) (WarrenProtocol.sol#1293-1307)
        - WARRENProtocol.collect(address) (WarrenProtocol.sol#1420-1465)
        - WARRENProtocol.distributeRefPayout(Models.User,uint256,bool)
(WarrenProtocol.sol#1064-1140)
        - WARRENProtocol.getHoldBonusPercent(address) (WarrenProtocol.sol#1540-1552)
        - WARRENProtocol.getLiquidityBonusPercent(address)
(WarrenProtocol.sol#1554-1562)
        - WARRENProtocol.getUIData(address) (WarrenProtocol.sol#1564-1580)
        - WARRENProtocol.influencerBond(address,uint256) (WarrenProtocol.sol#1405-1418)
        - WARRENProtocol.newBond(address,uint8,uint256,uint256)
(WarrenProtocol.sol#1156-1208)
        - WARRENProtocol.rebond(uint256) (WarrenProtocol.sol#1312-1329)
        - WARRENProtocol.sell(uint256) (WarrenProtocol.sol#1331-1397)
        - WARRENProtocol.stake(uint8,uint256) (WarrenProtocol.sol#1214-1264)
        - WARRENProtocol.transfer(uint8) (WarrenProtocol.sol#1270-1290)
        - WARRENProtocol.updateReferralLevel(address,uint256)
(WarrenProtocol.sol#1142-1152)
        - WARRENProtocol.userBalance(address) (WarrenProtocol.sol#1467-1506)
        - WARRENProtocol.users (WarrenProtocol.sol#965)
        - users[upline].refsNumber[i] ++ (WarrenProtocol.sol#1122)
       WARRENProtocol.users (WarrenProtocol.sol#965) can be used in cross function
reentrancies:
```

```
        - WARRENProtocol.buy(address,uint8,uint256) (WarrenProtocol.sol#1025-1062)
        - WARRENProtocol.claim(uint256) (WarrenProtocol.sol#1293-1307)
        - WARRENProtocol.collect(address) (WarrenProtocol.sol#1420-1465)
        - WARRENProtocol.distributeRefPayout(Models.User,uint256,bool)
(WarrenProtocol.sol#1064-1140)
        - WARRENProtocol.getHoldBonusPercent(address) (WarrenProtocol.sol#1540-1552)
        - WARRENProtocol.getLiquidityBonusPercent(address)
(WarrenProtocol.sol#1554-1562)
        - WARRENProtocol.getUIData(address) (WarrenProtocol.sol#1564-1580)
        - WARRENProtocol.influencerBond(address,uint256) (WarrenProtocol.sol#1405-1418)
        - WARRENProtocol.newBond(address,uint8,uint256,uint256)
(WarrenProtocol.sol#1156-1208)
        - WARRENProtocol.rebond(uint256) (WarrenProtocol.sol#1312-1329)
        - WARRENProtocol.sell(uint256) (WarrenProtocol.sol#1331-1397)
        - WARRENProtocol.stake(uint8,uint256) (WarrenProtocol.sol#1214-1264)
        - WARRENProtocol.transfer(uint8) (WarrenProtocol.sol#1270-1290)
        - WARRENProtocol.updateReferralLevel(address,uint256)
(WarrenProtocol.sol#1142-1152)
        - WARRENProtocol.userBalance(address) (WarrenProtocol.sol#1467-1506)
        - WARRENProtocol.users (WarrenProtocol.sol#965)
        - updateReferralLevel(upline,ethAmount) (WarrenProtocol.sol#1135)
                - users[_userAddress].refTurnover += _amount (WarrenProtocol.sol#1143)
                - users[_userAddress].refLevel = level (WarrenProtocol.sol#1147)
        WARRENProtocol.users (WarrenProtocol.sol#965) can be used in cross function
reentrancies:
        - WARRENProtocol.buy(address,uint8,uint256) (WarrenProtocol.sol#1025-1062)
        - WARRENProtocol.claim(uint256) (WarrenProtocol.sol#1293-1307)
        - WARRENProtocol.collect(address) (WarrenProtocol.sol#1420-1465)
        - WARRENProtocol.distributeRefPayout(Models.User,uint256,bool)
(WarrenProtocol.sol#1064-1140)
        - WARRENProtocol.getHoldBonusPercent(address) (WarrenProtocol.sol#1540-1552)
        - WARRENProtocol.getLiquidityBonusPercent(address)
(WarrenProtocol.sol#1554-1562)
        - WARRENProtocol.getUIData(address) (WarrenProtocol.sol#1564-1580)
        - WARRENProtocol.influencerBond(address,uint256) (WarrenProtocol.sol#1405-1418)
        - WARRENProtocol.newBond(address,uint8,uint256,uint256)
(WarrenProtocol.sol#1156-1208)
        - WARRENProtocol.rebond(uint256) (WarrenProtocol.sol#1312-1329)
        - WARRENProtocol.sell(uint256) (WarrenProtocol.sol#1331-1397)
        - WARRENProtocol.stake(uint8,uint256) (WarrenProtocol.sol#1214-1264)
        - WARRENProtocol.transfer(uint8) (WarrenProtocol.sol#1270-1290)
```

```
        - WARRENProtocol.updateReferralLevel(address,uint256)
(WarrenProtocol.sol#1142-1152)
        - WARRENProtocol.userBalance(address) (WarrenProtocol.sol#1467-1506)
        - WARRENProtocol.users (WarrenProtocol.sol#965)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-
vulnerabilities-1
INFO:Detectors:
WARRENProtocol.distributeRefPayout(Models.User,uint256,bool)
(WarrenProtocol.sol#1064-1140) contains a tautology or contradiction:
        - j_scope_0 >= 0 (WarrenProtocol.sol#1100)
WARRENProtocol.distributeRefPayout(Models.User,uint256,bool)
(WarrenProtocol.sol#1064-1140) contains a tautology or contradiction:
        - j >= 0 (WarrenProtocol.sol#1088)
WARRENProtocol.changePriceBalancerPercent(uint256) (WarrenProtocol.sol#1399-1403)
contains a tautology or contradiction:
        - require(bool,string)(percent >= 0 && percent <= 2500,Invalid percent amount
(0 - 2500: 0% - 25%)) (WarrenProtocol.sol#1400)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-
contradiction
INFO:Detectors:
WARRENProtocol.newBond(address,uint8,uint256,uint256) (WarrenProtocol.sol#1156-1208)
ignores return value by
WARRENToken(TOKEN_ADDRESS).increaseAllowance(UNISWAP_ROUTER_ADDRESS,tokensAmount)
(WarrenProtocol.sol#1183)
WARRENProtocol.stake(uint8,uint256) (WarrenProtocol.sol#1214-1264) ignores return value
by WARRENToken(TOKEN_ADDRESS).increaseAllowance(UNISWAP_ROUTER_ADDRESS,liquidityTokensAm
ount) (WarrenProtocol.sol#1237)
WARRENProtocol.sell(uint256) (WarrenProtocol.sol#1331-1397) ignores return value by
WARRENToken(TOKEN_ADDRESS).increaseAllowance(UNISWAP_ROUTER_ADDRESS,tokensAmount)
(WarrenProtocol.sol#1346)
WARRENProtocol.sell(uint256) (WarrenProtocol.sol#1331-1397) ignores return value by
ERC20(LP_TOKEN_ADDRESS).approve(UNISWAP_ROUTER_ADDRESS,liquidity)
(WarrenProtocol.sol#1366-1369)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
WARRENToken.constructor(address).pulsexRouterAddress (WarrenProtocol.sol#778) lacks a
zero-check on :
                - PULSEX_ROUTER_ADDRESS = pulsexRouterAddress (WarrenProtocol.sol#779)
WARRENToken.setMainContractAddress(address).contractAddress (WarrenProtocol.sol#784)
lacks a zero-check on :
                - mainContractAddress = contractAddress (WarrenProtocol.sol#787)
```

```
WARRENToken.setLPTokenAddress(address).lpTokenAddress (WarrenProtocol.sol#796) lacks a
zero-check on :
                - LP_TOKEN_ADDRESS = lpTokenAddress (WarrenProtocol.sol#799)
WARRENProtocol.constructor(address,address,address,address,address).uniswapRouterAddress
 (WarrenProtocol.sol#1005) lacks a zero-check on :
                - UNISWAP_ROUTER_ADDRESS = uniswapRouterAddress
(WarrenProtocol.sol#1011)
WARRENProtocol.constructor(address,address,address,address,address).WARRENTokenAddress
(WarrenProtocol.sol#1006) lacks a zero-check on :
                - TOKEN_ADDRESS = WARRENTokenAddress (WarrenProtocol.sol#1012)
WARRENProtocol.constructor(address,address,address,address,address).lpTokenAddress
(WarrenProtocol.sol#1007) lacks a zero-check on :
                - LP_TOKEN_ADDRESS = lpTokenAddress (WarrenProtocol.sol#1013)
WARRENProtocol.constructor(address,address,address,address,address).defaultUpline
(WarrenProtocol.sol#1008) lacks a zero-check on :
                - DEFAULT_UPLINE = defaultUpline (WarrenProtocol.sol#1015)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-
address-validation
INFO:Detectors:
WARRENProtocol.getTokensAmount(uint256) (WarrenProtocol.sol#1514-1518) has external
calls inside a loop: (reserve0,reserve1) =
IUniswapV2Pair(LP_TOKEN_ADDRESS).getReserves() (WarrenProtocol.sol#1515)
WARRENProtocol.getTokenLiquidity() (WarrenProtocol.sol#1520-1526) has external calls
inside a loop: (reserve0,reserve1) = IUniswapV2Pair(LP_TOKEN_ADDRESS).getReserves()
(WarrenProtocol.sol#1524)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-
a-loop
INFO:Detectors:
Reentrancy in WARRENProtocol.buy(address,uint8,uint256) (WarrenProtocol.sol#1025-1062):
        External calls:
        - DEFAULT_TOKEN.safeTransferFrom(msg.sender,address(this),amount)
(WarrenProtocol.sol#1034)
        Event emitted after the call(s):
        - Events.NewUser(msg.sender,upline,block.timestamp)
(WarrenProtocol.sol#1051-1053)
Reentrancy in WARRENProtocol.buy(address,uint8,uint256) (WarrenProtocol.sol#1025-1062):
        External calls:
        - DEFAULT_TOKEN.safeTransferFrom(msg.sender,address(this),amount)
(WarrenProtocol.sol#1034)
        - refReward = distributeRefPayout(user,amount,isNewUser)
(WarrenProtocol.sol#1057)
```

```
                - returndata = address(token).functionCall(data)
(WarrenProtocol.sol#402)
                - (success,returndata) = target.call{value: value}(data)
(WarrenProtocol.sol#243)
                - DEFAULT_TOKEN.safeTransfer(upline,amount) (WarrenProtocol.sol#1111)
        External calls sending eth:
        - refReward = distributeRefPayout(user,amount,isNewUser)
(WarrenProtocol.sol#1057)
                - (success,returndata) = target.call{value: value}(data)
(WarrenProtocol.sol#243)
        Event emitted after the call(s):
        - Events.RefPayout(msg.sender,upline,i,amount,block.timestamp)
(WarrenProtocol.sol#1115-1117)
                - refReward = distributeRefPayout(user,amount,isNewUser)
(WarrenProtocol.sol#1057)
Reentrancy in WARRENProtocol.buy(address,uint8,uint256) (WarrenProtocol.sol#1025-1062):
        External calls:
        - DEFAULT_TOKEN.safeTransferFrom(msg.sender,address(this),amount)
(WarrenProtocol.sol#1034)
        - refReward = distributeRefPayout(user,amount,isNewUser)
(WarrenProtocol.sol#1057)
                - returndata = address(token).functionCall(data)
(WarrenProtocol.sol#402)
                - (success,returndata) = target.call{value: value}(data)
(WarrenProtocol.sol#243)
                - DEFAULT_TOKEN.safeTransfer(upline,amount) (WarrenProtocol.sol#1111)
        - DEFAULT_TOKEN.safeTransfer(owner(),adminFee) (WarrenProtocol.sol#1059)
        - newBond(msg.sender,bondType,amount,amount - adminFee - refReward)
(WarrenProtocol.sol#1061)
                - WARRENToken(TOKEN_ADDRESS).mint(address(this),tokensAmount)
(WarrenProtocol.sol#1182)
                -
WARRENToken(TOKEN_ADDRESS).increaseAllowance(UNISWAP_ROUTER_ADDRESS,tokensAmount)
(WarrenProtocol.sol#1183)
                - (amountToken,amountETH,liquidity) = IUniswapV2Router01(UNISWAP_ROUTER_
ADDRESS).addLiquidity(TOKEN_ADDRESS,address(DEFAULT_TOKEN),tokensAmount,liquidityAmount,
0,0,address(this),block.timestamp + 300) (WarrenProtocol.sol#1185-1196)
        External calls sending eth:
        - refReward = distributeRefPayout(user,amount,isNewUser)
(WarrenProtocol.sol#1057)
                - (success,returndata) = target.call{value: value}(data)
(WarrenProtocol.sol#243)
```

```
        Event emitted after the call(s):
        - Events.LiquidityAdded(amountToken,amountETH,liquidity,block.timestamp)
(WarrenProtocol.sol#1198-1200)
                - newBond(msg.sender,bondType,amount,amount - adminFee - refReward)
(WarrenProtocol.sol#1061)
        - Events.NewBond(userAddr,bondType,user.bondsNumber -
1,bondAmount,tokensAmount,liquidityAmount == 0,block.timestamp)
(WarrenProtocol.sol#1203-1205)
                - newBond(msg.sender,bondType,amount,amount - adminFee - refReward)
(WarrenProtocol.sol#1061)
Reentrancy in WARRENProtocol.claim(uint256) (WarrenProtocol.sol#1293-1307):
        External calls:
        - WARRENToken(TOKEN_ADDRESS).mint(msg.sender,tokensAmount)
(WarrenProtocol.sol#1302)
        Event emitted after the call(s):
        - Events.Claim(msg.sender,tokensAmount,block.timestamp)
(WarrenProtocol.sol#1304-1306)
Reentrancy in WARRENProtocol.distributeRefPayout(Models.User,uint256,bool)
(WarrenProtocol.sol#1064-1140):
        External calls:
        - DEFAULT_TOKEN.safeTransfer(upline,amount) (WarrenProtocol.sol#1111)
        Event emitted after the call(s):
        - Events.RefPayout(msg.sender,upline,i,amount,block.timestamp)
(WarrenProtocol.sol#1115-1117)
Reentrancy in WARRENProtocol.influencerBond(address,uint256)
(WarrenProtocol.sol#1405-1418):
        External calls:
        - bondIdx = newBond(userAddr,4,ethAmount,0) (WarrenProtocol.sol#1411)
                - WARRENToken(TOKEN_ADDRESS).mint(address(this),tokensAmount)
(WarrenProtocol.sol#1182)
                -
WARRENToken(TOKEN_ADDRESS).increaseAllowance(UNISWAP_ROUTER_ADDRESS,tokensAmount)
(WarrenProtocol.sol#1183)
                - (amountToken,amountETH,liquidity) = IUniswapV2Router01(UNISWAP_ROUTER_
ADDRESS).addLiquidity(TOKEN_ADDRESS,address(DEFAULT_TOKEN),tokensAmount,liquidityAmount,
0,0,address(this),block.timestamp + 300) (WarrenProtocol.sol#1185-1196)
        - WARRENToken(TOKEN_ADDRESS).burn(tokensAmount) (WarrenProtocol.sol#1413)
        Event emitted after the call(s):
        - Events.NewBond(userAddr,4,bondIdx,ethAmount,tokensAmount * 95 /
100,false,block.timestamp) (WarrenProtocol.sol#1415-1417)
Reentrancy in WARRENProtocol.newBond(address,uint8,uint256,uint256)
(WarrenProtocol.sol#1156-1208):
```

```
        External calls:
        - WARRENToken(TOKEN_ADDRESS).mint(address(this),tokensAmount)
(WarrenProtocol.sol#1182)
        -
WARRENToken(TOKEN_ADDRESS).increaseAllowance(UNISWAP_ROUTER_ADDRESS,tokensAmount)
(WarrenProtocol.sol#1183)
        - (amountToken,amountETH,liquidity) = IUniswapV2Router01(UNISWAP_ROUTER_ADDRESS)
.addLiquidity(TOKEN_ADDRESS,address(DEFAULT_TOKEN),tokensAmount,liquidityAmount,0,0,addr
ess(this),block.timestamp + 300) (WarrenProtocol.sol#1185-1196)
        Event emitted after the call(s):
        - Events.LiquidityAdded(amountToken,amountETH,liquidity,block.timestamp)
(WarrenProtocol.sol#1198-1200)
        - Events.NewBond(userAddr,bondType,user.bondsNumber -
1,bondAmount,tokensAmount,liquidityAmount == 0,block.timestamp)
(WarrenProtocol.sol#1203-1205)
Reentrancy in WARRENProtocol.rebond(uint256) (WarrenProtocol.sol#1312-1329):
        External calls:
        - bondIdx = newBond(msg.sender,0,ethAmount,0) (WarrenProtocol.sol#1324)
                - WARRENToken(TOKEN_ADDRESS).mint(address(this),tokensAmount)
(WarrenProtocol.sol#1182)
                -
WARRENToken(TOKEN_ADDRESS).increaseAllowance(UNISWAP_ROUTER_ADDRESS,tokensAmount)
(WarrenProtocol.sol#1183)
                - (amountToken,amountETH,liquidity) = IUniswapV2Router01(UNISWAP_ROUTER_
ADDRESS).addLiquidity(TOKEN_ADDRESS,address(DEFAULT_TOKEN),tokensAmount,liquidityAmount,
0,0,address(this),block.timestamp + 300) (WarrenProtocol.sol#1185-1196)
        Event emitted after the call(s):
        - Events.ReBond(msg.sender,bondIdx,ethAmount,tokensAmount,block.timestamp)
(WarrenProtocol.sol#1326-1328)
Reentrancy in WARRENProtocol.sell(uint256) (WarrenProtocol.sol#1331-1397):
        External calls:
        - WARRENToken(TOKEN_ADDRESS).mint(address(this),tokensAmount)
(WarrenProtocol.sol#1345)
        -
WARRENToken(TOKEN_ADDRESS).increaseAllowance(UNISWAP_ROUTER_ADDRESS,tokensAmount)
(WarrenProtocol.sol#1346)
        - amounts = IUniswapV2Router01(UNISWAP_ROUTER_ADDRESS).swapExactTokensForTokens(
tokensAmount,0,path,msg.sender,block.timestamp + 300) (WarrenProtocol.sol#1348-1354)
        - ERC20(LP_TOKEN_ADDRESS).approve(UNISWAP_ROUTER_ADDRESS,liquidity)
(WarrenProtocol.sol#1366-1369)
        - (amountDEFAULT) = IUniswapV2Router01(UNISWAP_ROUTER_ADDRESS).removeLiquidity(T
```

```
OKEN_ADDRESS,address(DEFAULT_TOKEN),liquidity,0,0,address(this),block.timestamp + 300)
(WarrenProtocol.sol#1371-1379)
        - amounts = IUniswapV2Router01(UNISWAP_ROUTER_ADDRESS).swapExactTokensForTokens(
amountDEFAULT,0,path,address(this),block.timestamp + 300)
(WarrenProtocol.sol#1386-1392)
        Event emitted after the call(s):
        - Events.Sell(msg.sender,tokensAmount,ethAmount,block.timestamp)
(WarrenProtocol.sol#1394-1396)
Reentrancy in WARRENProtocol.stake(uint8,uint256) (WarrenProtocol.sol#1214-1264):
        External calls:
        - refReward = distributeRefPayout(user,amount,false) (WarrenProtocol.sol#1227)
                - returndata = address(token).functionCall(data)
(WarrenProtocol.sol#402)
                - (success,returndata) = target.call{value: value}(data)
(WarrenProtocol.sol#243)
                - DEFAULT_TOKEN.safeTransfer(upline,amount) (WarrenProtocol.sol#1111)
        - DEFAULT_TOKEN.safeTransfer(owner(),adminFee) (WarrenProtocol.sol#1229)
        - WARRENToken(TOKEN_ADDRESS).mint(address(this),liquidityTokensAmount)
(WarrenProtocol.sol#1236)
        - WARRENToken(TOKEN_ADDRESS).increaseAllowance(UNISWAP_ROUTER_ADDRESS,liquidityT
okensAmount) (WarrenProtocol.sol#1237)
        - (amountToken,amountETH,liquidity) = IUniswapV2Router01(UNISWAP_ROUTER_ADDRESS)
.addLiquidity(TOKEN_ADDRESS,address(DEFAULT_TOKEN),liquidityTokensAmount,defaultTokenAmo
unt,0,0,address(this),block.timestamp + 300) (WarrenProtocol.sol#1239-1248)
        External calls sending eth:
        - refReward = distributeRefPayout(user,amount,false) (WarrenProtocol.sol#1227)
                - (success,returndata) = target.call{value: value}(data)
(WarrenProtocol.sol#243)
        Event emitted after the call(s):
        - Events.LiquidityAdded(amountToken,amountETH,liquidity,block.timestamp)
(WarrenProtocol.sol#1252-1254)
        - Events.StakeBond(msg.sender,bondIdx,tokensAmount,amount,block.timestamp)
(WarrenProtocol.sol#1261-1263)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-
vulnerabilities-3
INFO:Detectors:
WARRENProtocol.buy(address,uint8,uint256) (WarrenProtocol.sol#1025-1062) uses timestamp
for comparisons
        Dangerous comparisons:
        - require(bool,string)(users[msg.sender].bondsNumber <
Constants.BONDS_LIMIT,Buy: you have reached bonds limit) (WarrenProtocol.sol#1030)
```

```
        - upline == address(0) || upline == msg.sender || users[upline].bondsNumber ==
0 (WarrenProtocol.sol#1042)
WARRENProtocol.distributeRefPayout(Models.User,uint256,bool)
(WarrenProtocol.sol#1064-1140) uses timestamp for comparisons
        Dangerous comparisons:
        - upline == address(0) (WarrenProtocol.sol#1079)
        - j >= 0 (WarrenProtocol.sol#1088)
        - j == 0 (WarrenProtocol.sol#1091)
        - users[upline].refLevel > maxRefLevel && !
distributedLevels[users[upline].refLevel] (WarrenProtocol.sol#1095)
        - j_scope_0 >= 0 (WarrenProtocol.sol#1100)
        - j_scope_0 == 0 (WarrenProtocol.sol#1103)
        - upline == address(0) (WarrenProtocol.sol#1131)
WARRENProtocol.updateReferralLevel(address,uint256) (WarrenProtocol.sol#1142-1152) uses
timestamp for comparisons
        Dangerous comparisons:
        - users[_userAddress].refTurnover >= REFERRAL_LEVELS_MILESTONES[level]
(WarrenProtocol.sol#1146)
WARRENProtocol.stake(uint8,uint256) (WarrenProtocol.sol#1214-1264) uses timestamp for
comparisons
        Dangerous comparisons:
        - require(bool,string)(bondIdx < users[msg.sender].bondsNumber,Stake: invalid
bond index) (WarrenProtocol.sol#1215)
        - require(bool,string)(! bonds[msg.sender][bondIdx].isClosed,Stake: this bond
already closed) (WarrenProtocol.sol#1216)
        - require(bool,string)(bonds[msg.sender][bondIdx].stakeTime == 0,Stake: this
bond was already staked) (WarrenProtocol.sol#1217)
WARRENProtocol.transfer(uint8) (WarrenProtocol.sol#1270-1290) uses timestamp for
comparisons
        Dangerous comparisons:
        - require(bool,string)(bondIdx < users[msg.sender].bondsNumber,Transfer:
invalid bond index) (WarrenProtocol.sol#1273)
        - require(bool,string)(block.timestamp >= bond.creationTime +
bond.freezePeriod,Transfer: this bond is still freeze) (WarrenProtocol.sol#1276-1279)
WARRENProtocol.claim(uint256) (WarrenProtocol.sol#1293-1307) uses timestamp for
comparisons
        Dangerous comparisons:
        - require(bool,string)(user.balance >= tokensAmount,Claim: insufficient balance)
(WarrenProtocol.sol#1298)
WARRENProtocol.rebond(uint256) (WarrenProtocol.sol#1312-1329) uses timestamp for
comparisons
```

```
        Dangerous comparisons:
        - require(bool,string)(users[msg.sender].bondsNumber <
Constants.BONDS_LIMIT,Rebond: you have reached bonds limit) (WarrenProtocol.sol#1313)
WARRENProtocol.sell(uint256) (WarrenProtocol.sol#1331-1397) uses timestamp for
comparisons
        Dangerous comparisons:
        - require(bool,string)(user.balance >= tokensAmount,Sell: insufficient balance)
(WarrenProtocol.sol#1336)
WARRENProtocol.influencerBond(address,uint256) (WarrenProtocol.sol#1405-1418) uses
timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(users[userAddr].bondsNumber < Constants.BONDS_LIMIT,User
have reached bonds limit) (WarrenProtocol.sol#1406)
WARRENProtocol.collect(address) (WarrenProtocol.sol#1420-1465) uses timestamp for
comparisons
        Dangerous comparisons:
        - block.timestamp >= bond.creationTime + bond.freezePeriod
(WarrenProtocol.sol#1434)
        - bond.collectedReward + tokensAmount >= bond.stakingRewardLimit
(WarrenProtocol.sol#1453)
WARRENProtocol.userBalance(address) (WarrenProtocol.sol#1467-1506) uses timestamp for
comparisons
        Dangerous comparisons:
        - i < bondsNumber (WarrenProtocol.sol#1471)
        - bond.stakeTime == 0 (WarrenProtocol.sol#1479)
        - block.timestamp >= bond.creationTime + bond.freezePeriod
(WarrenProtocol.sol#1480)
        - bond.collectedReward + tokensAmount >= bond.stakingRewardLimit
(WarrenProtocol.sol#1497)
WARRENProtocol.getHoldBonusPercent(address) (WarrenProtocol.sol#1540-1552) uses
timestamp for comparisons
        Dangerous comparisons:
        - users[userAddr].lastActionTime == 0 (WarrenProtocol.sol#1541)
        - bonusPercent > Constants.USER_HOLD_BONUS_LIMIT_PERCENT
(WarrenProtocol.sol#1549)
WARRENProtocol.getLiquidityBonusPercent(address) (WarrenProtocol.sol#1554-1562) uses
timestamp for comparisons
        Dangerous comparisons:
        - bonusPercent > Constants.LIQUIDITY_BONUS_LIMIT_PERCENT
(WarrenProtocol.sol#1559)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-
timestamp
```

```
INFO:Detectors:
Address._revert(bytes) (WarrenProtocol.sol#302-314) uses assembly
        - INLINE ASM (WarrenProtocol.sol#307-310)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
WARRENProtocol.distributeRefPayout(Models.User,uint256,bool)
(WarrenProtocol.sol#1064-1140) has a high cyclomatic complexity (13).
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#cyclomatic-
complexity
INFO:Detectors:
Address.functionDelegateCall(address,bytes) (WarrenProtocol.sol#260-263) is never used
and should be removed
Address.functionStaticCall(address,bytes) (WarrenProtocol.sol#251-254) is never used
and should be removed
Address.sendValue(address,uint256) (WarrenProtocol.sol#197-206) is never used and
should be removed
Address.verifyCallResult(bool,bytes) (WarrenProtocol.sol#291-297) is never used and
should be removed
Context._msgData() (WarrenProtocol.sol#538-540) is never used and should be removed
ERC20._beforeTokenTransfer(address,address,uint256) (WarrenProtocol.sol#707) is never
used and should be removed
SafeERC20._callOptionalReturnBool(IERC20,bytes) (WarrenProtocol.sol#416-423) is never
used and should be removed
SafeERC20.forceApprove(IERC20,address,uint256) (WarrenProtocol.sol#382-389) is never
used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (WarrenProtocol.sol#367-375) is
never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (WarrenProtocol.sol#358-361) is
never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.20 (WarrenProtocol.sol#1) necessitates a version too recent to be
trusted. Consider deploying with 0.8.18.
solc-0.8.20 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-
versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (WarrenProtocol.sol#197-206):
        - (success) = recipient.call{value: amount}() (WarrenProtocol.sol#202)
Low level call in Address.functionCallWithValue(address,bytes,uint256)
(WarrenProtocol.sol#239-245):
```

```
        - (success,returndata) = target.call{value: value}(data)
(WarrenProtocol.sol#243)
Low level call in Address.functionStaticCall(address,bytes)
(WarrenProtocol.sol#251-254):
        - (success,returndata) = target.staticcall(data) (WarrenProtocol.sol#252)
Low level call in Address.functionDelegateCall(address,bytes)
(WarrenProtocol.sol#260-263):
        - (success,returndata) = target.delegatecall(data) (WarrenProtocol.sol#261)
Low level call in SafeERC20._callOptionalReturnBool(IERC20,bytes)
(WarrenProtocol.sol#416-423):
        - (success,returndata) = address(token).call(data) (WarrenProtocol.sol#421)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-
calls
INFO:Detectors:
Function IERC20Permit.DOMAIN_SEPARATOR() (WarrenProtocol.sol#160) is not in mixedCase
Function IUniswapV2Pair.DOMAIN_SEPARATOR() (WarrenProtocol.sol#441) is not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (WarrenProtocol.sol#442) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (WarrenProtocol.sol#459) is not in
mixedCase
Variable WARRENToken.PULSEX_ROUTER_ADDRESS (WarrenProtocol.sol#773) is not in mixedCase
Variable WARRENToken.LP_TOKEN_ADDRESS (WarrenProtocol.sol#774) is not in mixedCase
Parameter WARRENProtocol.updateReferralLevel(address,uint256)._userAddress
(WarrenProtocol.sol#1142) is not in mixedCase
Parameter WARRENProtocol.updateReferralLevel(address,uint256)._amount
(WarrenProtocol.sol#1142) is not in mixedCase
Variable WARRENProtocol.TOKEN_ADDRESS (WarrenProtocol.sol#969) is not in mixedCase
Variable WARRENProtocol.LP_TOKEN_ADDRESS (WarrenProtocol.sol#970) is not in mixedCase
Variable WARRENProtocol.UNISWAP_ROUTER_ADDRESS (WarrenProtocol.sol#971) is not in
mixedCase
Variable WARRENProtocol.DEFAULT_UPLINE (WarrenProtocol.sol#972) is not in mixedCase
Variable WARRENProtocol.DEFAULT_TOKEN (WarrenProtocol.sol#973) is not in mixedCase
Variable WARRENProtocol.REFERRAL_LEVELS_PERCENTS (WarrenProtocol.sol#975) is not in
mixedCase
Variable WARRENProtocol.REFERRAL_LEVELS_MILESTONES (WarrenProtocol.sol#976) is not in
mixedCase
Variable WARRENProtocol.PRICE_BALANCER_PERCENT (WarrenProtocol.sol#980) is not in
mixedCase
Variable WARRENProtocol.BOND_FREEZE_PERIODS (WarrenProtocol.sol#982-988) is not in
mixedCase
Variable WARRENProtocol.BOND_FREEZE_PERCENTS (WarrenProtocol.sol#989-995) is not in
mixedCase
```

```
Variable WARRENProtocol.BOND_ACTIVATIONS (WarrenProtocol.sol#996-1002) is not in
mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-
solidity-naming-conventions
INFO:Detectors:
Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256
,address,uint256).amountADesired (WarrenProtocol.sol#480) is too similar to IUniswapV2Ro
uter01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amo
untBDesired (WarrenProtocol.sol#481)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-
too-similar
INFO:Slither:WarrenProtocol.sol analyzed (16 contracts with 85 detectors), 91 result(s)
found
```

0x Guard