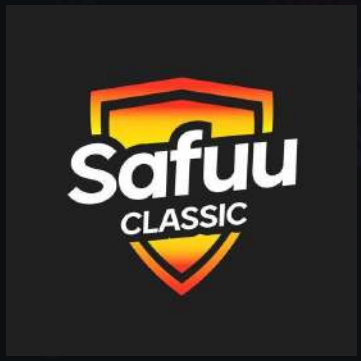
Coinsult

Advanced Manual Smart Contract Audit



Project: Safuu Classic

Website: https://safuuclassic.com/

Low-Risk	Medium-Risk	High-Risk
4 low-risk code issues found	0 medium-risk code issues found	0 high-risk code issues found

Contract Address

0xD724Bbe5B419394E370200e7D2370F731678cB48

Disclaimer

Coinsult is not responsible if a project turns out to be a scam, rug-pull or honeypot. We only provide a detailed analysis for your own research.

Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

The information provided in this audit is for informational purposes only and should not be considered investment advice. Coinsult does not endorse, recommend, support or suggest to invest in any project.

Coinsult can not be held responsible for when a project turns out to be a rug-pull, honeypot or scam.

Tokenomics

Rank	Address	Quantity (Token)	Percentage
1	0x65e949d2958083a2469923fc34f801cc8b47c4a0	325,000	100.0000%

Source Code

Coinsult was comissioned by Safuu Classic to perform an audit based on the following smart contract:

https://bscscan.com/address/0xD724Bbe5B419394E370200e7D2370F731678cB48#code

Manual Code Review

In this audit report we will highlight all these issues:

🟢 Low-Risk

4 low-risk code
issues found

🟡 Medium-Risk

0 medium-risk code
issues found

🔴 High-Risk

0 high-risk code
issues found

The detailed report continues on the next page...

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Contract contains Reentrancy vulnerabilities

Additional information: This combination increases risk of malicious intent. While it may be justified by some complex mechanics (e.g. rebase, reflections, buyback).

More information: Slither

```
function _transferFrom(
    address sender,
    address recipient,
    uint256 amount
) internal returns (bool) {

    require(!blacklist[sender] &amp;amp;&amp;amp; !blacklist[recipient], "in_blacklist");

    if (inSwap) {
        return _basicTransfer(sender, recipient, amount);
    }
    if (shouldRebase()) {
       rebase();
    }

    if (shouldAddLiquidity()) {
        addLiquidity();
    }

    if (shouldSwapBack()) {
        swapBack();
    }
```

Recommendation

Apply the check-effects-interactions pattern.

Exploit scenario

```
function withdrawBalance(){
    // send userBalance[msg.sender] Ether to msg.sender
    // if mgs.sender is a contract, it will call its fallback function
    if( ! (msg.sender.call.value(userBalance[msg.sender])() ) ){
        throw;
    }
    userBalance[msg.sender] = 0;
}
```

Bob uses the re-entrancy bug to call withdrawBalance two times, and withdraw more than its initial deposit to the contract.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Avoid relying on block.timestamp

block.timestamp can be manipulated by miners.

```
function rebase() internal {

    if ( inSwap ) return;
    uint256 rebaseRate;
    uint256 deltaTimeFromInit = block.timestamp - _initRebaseStartTime;
    uint256 deltaTime = block.timestamp - _lastRebasedTime;
    uint256 times = deltaTime.div(15 minutes);
    uint256 epoch = times.mul(15);

    if (deltaTimeFromInit = (7 * 365 days)) {
        rebaseRate = 2;
    } else if (deltaTimeFromInit &gt;= ((15 * 365 days) / 10)) {
        rebaseRate = 14;
    }else if (deltaTimeFromInit &gt;= (365 days)) {
        rebaseRate = 211;
    }

    for (uint256 i = 0; i &lt; times; i++) {
        _totalSupply = _totalSupply
            .mul((10**RATE_DECIMALS).add(rebaseRate))
            .div(10**RATE_DECIMALS);
    }
```

Recommendation

Do not use `block.timestamp`, now or `blockhash` as a source of randomness

Exploit scenario

```
contract Game {

    uint reward_determining_number;

    function guessing() external{
      reward_determining_number = uint256(block.blockhash(10000)) % 10;
    }
}
```

Eve is a miner. Eve calls `guessing` and re-orders the block containing the transaction. As a result, Eve wins the game.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Functions that send Ether to arbitrary destinations

Unprotected call to a function sending Ether to an arbitrary address.

```
function swapBack() internal swapping {

    uint256 amountToSwap = _gonBalances[address(this)].div(_gonsPerFragment);

    if( amountToSwap == 0) {
        return;
    }

    uint256 balanceBefore = address(this).balance;
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = router.WETH();


    router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        amountToSwap,
        0,
        path,
        address(this),
        block.timestamp
    );
```

Recommendation

Ensure that an arbitrary user cannot withdraw unauthorized funds.

Exploit scenario

```
contract ArbitrarySend{
    address destination;
    function setDestination(){
        destination = msg.sender;
    }

    function withdraw() public{
        destination.transfer(this.balance);
    }
}
```

Bob calls `setDestination` and `withdraw`. As a result he withdraws the contract's balance.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Write after write

Variables that are written but never read and written again.

```
function swapBack() internal swapping {

    uint256 amountToSwap = _gonBalances[address(this)].div(_gonsPerFragment);

    if( amountToSwap == 0) {
        return;
    }

    uint256 balanceBefore = address(this).balance;
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = router.WETH();


    router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        amountToSwap,
        0,
        path,
        address(this),
        block.timestamp
    );
```

Recommendation

Fix or remove the writes.

Exploit scenario

```
```solidity
contract Buggy{
    function my_func() external initializer{
        // ...
        a = b;
        a = c;
        // ..
    }
}
```

`a` is first asigned to `b`, and then to `c`. As a result the first write does nothing.
```

Owner privileges

- Owner cannot set fees higher than 25%
- Owner cannot pause trading
- Owner cannot change max transaction amount
- Owner can exclude from fees
- ⚠ Owner can blacklist contract addresses

Extra notes by the team

No notes

Contract Snapshot

```
contract SafuuClassic is ERC20Detailed, Ownable {

using SafeMath for uint256;
using SafeMathInt for int256;

event LogRebase(uint256 indexed epoch, uint256 totalSupply);

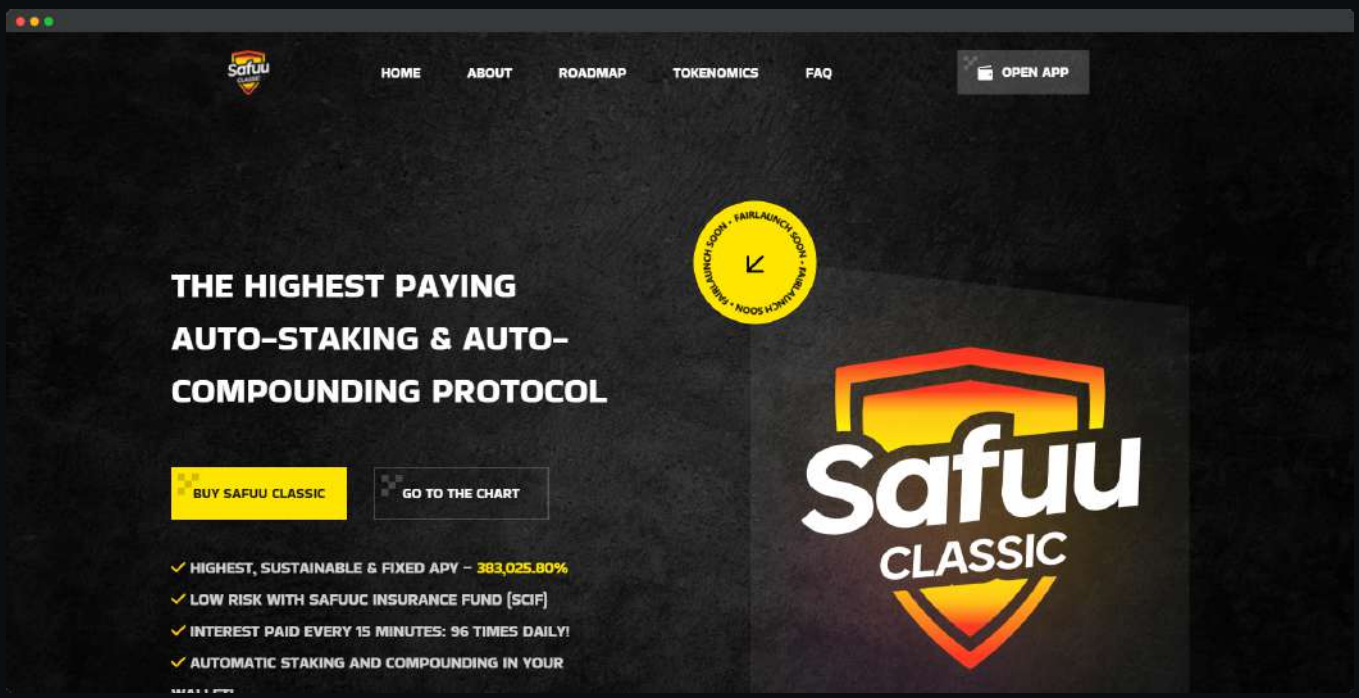
string public _name = "Safuu Classic";
string public _symbol = "SAFUUC";
uint8 public _decimals = 5;

IPancakeSwapPair public pairContract;
mapping(address =&gt; bool) _isFeeExempt;

modifier validRecipient(address to) {
    require(to != address(0x0));
    _;
}
```

Website Review

Coinsult checks the website completely manually and looks for visual, technical and textual errors. We also look at the security, speed and accessibility of the website. In short, a complete check to see if the website meets the current standard of the web development industry.



- Mobile Friendly
- Does not contain jQuery errors
- SSL Secured
- No major spelling errors

Project Overview

Not KYC verified by Coinsult

Safuu Classic

Audited by Coinsult.net

Date: 1 July 2022

Advanced Manual Smart Contract Audit