Competitive Security Assessment

SO3

Jun 5th, 2023

Secure3

secure3.io

Summary

SO3 is a decentralized Web3 social application with perpetual self-operation and negative feedback mechanism.

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	SO3
Platform & Language	Solidity
Codebase	• https://github.com/0xso3/so3-v1 • audit commit - 52815780442624d86936972fed23d38d8b540821 • final commit - b678abef20848a8c30411dc1d2d9766f37c1fa0c
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

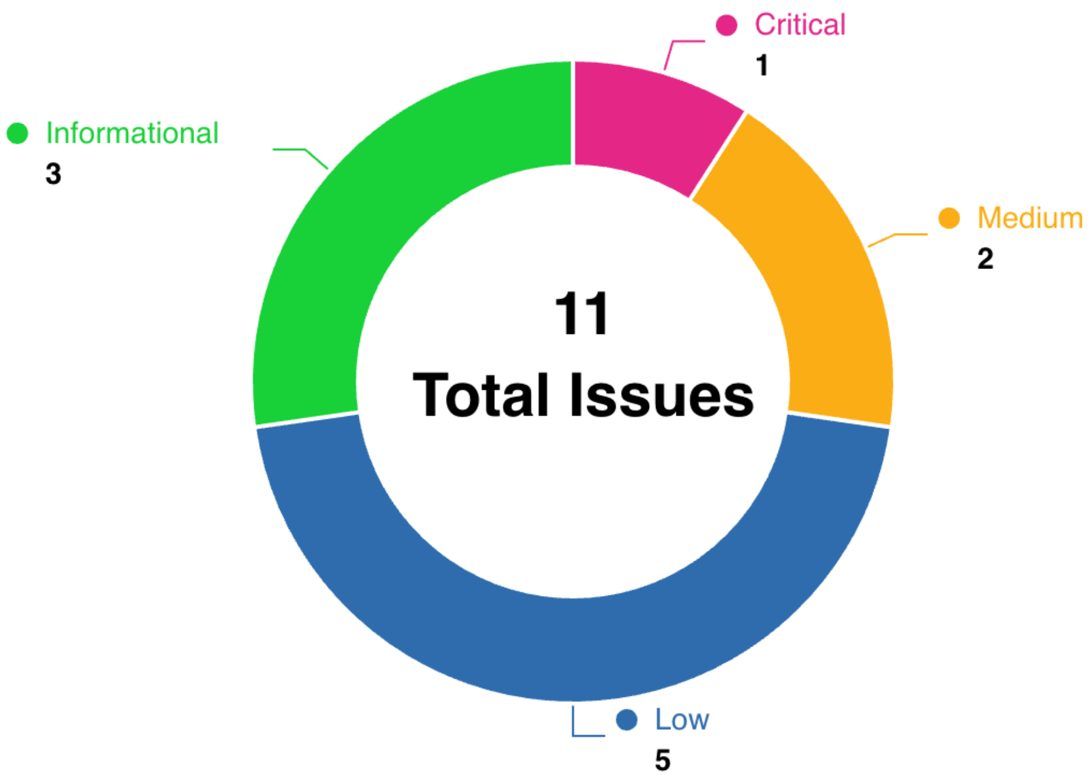
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	**1**	**0**	**0**	**0**	**1**	**0**
Medium	**2**	**0**	**0**	**2**	**0**	**0**
Low	**5**	**0**	**0**	**1**	**0**	**4**
Informational	**3**	**0**	**0**	**2**	**0**	**1**

Audit Scope

File	Commit Hash
code/src/SO3Market.sol	52815780442624d86936972fed23d38d8b540821
code/src/SO3Chef.sol	52815780442624d86936972fed23d38d8b540821
code/src/Vars.sol	52815780442624d86936972fed23d38d8b540821
code/src/SO3.sol	52815780442624d86936972fed23d38d8b540821
code/src/UUPSUpgradeableExp.sol	52815780442624d86936972fed23d38d8b540821
code/src/Counter.sol	52815780442624d86936972fed23d38d8b540821
code/src/interfaces/IChef.sol	52815780442624d86936972fed23d38d8b540821
code/src/interfaces/IMintPool.sol	52815780442624d86936972fed23d38d8b540821

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
SO3-1	Centralization Risk - Owner Has Unlimited Mint Authority, and Can Set Total Fee to close to 100%	Privilege Related	Critical	Mitigated	yosriady, 0x1337
SO3-2	Wrong power calculation in `SO3Market` contract `getPower` function	Integer Overflow and Underflow	Medium	Fixed	yosriady, Atlas
SO3-3	Potential Reentrancy from `Miner` Address	Reentrancy	Medium	Fixed	0x1337

SO3-4	**Basic computing power mismatch with the document in `Vars.sol` contract**	**Logical**	**Low**	**Declined**	**Atlas**
SO3-5	**Gift cost mismatch with the document in `SO3Market` contract `like` function**	**Logical**	**Low**	**Declined**	**Atlas**
SO3-6	**Insufficient EOA Check could Allow Miner to be a Contract Address**	**Language Specific**	**Low**	**Fixed**	**0x1337**
SO3-7	**Missing Storage Gap in Upgradeable Contracts**	**Language Specific**	**Low**	**Declined**	**0x1337**
SO3-8	**Upgradeable Contract Inherits Non Upgradeable Contract**	**Language Specific**	**Low**	**Declined**	**0x1337**
SO3-9	**Missing Event Emit**	**Code Style**	**Informational**	**Declined**	**yosriady, 0x1337**
SO3-10	**Unused code**	**Code Style**	**Informational**	**Fixed**	**yosriady, Atlas**
SO3-11	**`withrawETH()` Function should be `withdrawETH()`**	**Code Style**	**Informational**	**Fixed**	**0x1337**

SO3-1:Centralization Risk - Owner Has Unlimited Mint Authority, and Can Set Total Fee to close to 100%

Category	Severity	Status	Contributor
Privilege Related	**Critical**	**Mitigated**	**yosriady, 0x1337**

Code Reference

- code/src/SO3.sol#L12-L15
- code/src/SO3Chef.sol#L120-L142
- code/src/SO3Market.sol#L179-L182
- code/src/SO3Market.sol#L179-L210

```
    function mint(address to, uint256 amount) external {
        require(master == msg.sender, "REJ");
        _mint(to, amount);
    }

        if (newAgent == address(0)) revert ADDRESS_IS_EMPTY();
        agent = newAgent; //ignore event
    }

    function setSO3(IMintPool miner) external onlyOwner {
        if (address(miner) == address(0)) revert ADDRESS_IS_EMPTY();
        so3Miner = miner;
    }

    function setFeeBP(uint256 toMiner, uint256 toTreasury) external onlyOwner {
        require(toMiner + toTreasury < BP);
        mintFeeBP = toTreasury;
        mintFeeToMinerBP = toMiner;

        emit FeePointChanged(toTreasury, toMiner);
    }

    function setTreasury(address addr) external onlyOwner {
        if (addr == address(0)) revert ADDRESS_IS_EMPTY();

        treasury = addr;
        emit TreasuryChanged(addr);
    }

    function setTreasury(address addr) external onlyOwner {
        treasury = addr;
        emit TreasuryChanged(addr);
    }

    function setTreasury(address addr) external onlyOwner {
        treasury = addr;
        emit TreasuryChanged(addr);
    }

    function setFeeBP(uint256 toTreasuryBP, uint256 toMinerBP) external onlyOwner {
        require(toTreasuryBP + toMinerBP < BP);
        tradeFeeToTreasuryBP = toTreasuryBP;
        tradeFeeToMinerBP = toMinerBP;

        emit FeePointChanged(toTreasuryBP, toMinerBP);
    }

    function setChef(IChef chef) external onlyOwner {
        require(address(chef) != address(0));
        so3Chef = chef;
    }

    function setWhitelistStatus(bool enable) external onlyOwner {
        minerWhitelistEnabled = enable;
    }

    function setWhitelist(address[] calldata list, bool allow) external onlyOwner {
        for (uint256 i = 0; i < list.length; i++) {
            minerWhitelist[list[i]] = allow;
        }
        emit MinerWhitelistChanged(list, allow);
    }

    function setTradeStartBlock(uint256 blockNumber) external onlyOwner {
        tradeStartBlock = blockNumber;
    }
```

Description

yosriady : The owner can set the treasury address to the zero address:

```
    function setTreasury(address addr) external onlyOwner {
        treasury = addr;
        emit TreasuryChanged(addr);
    }
```

And ANYONE can call `withrawETH()` to burn the entire ETH balance in the contract:

```
    function withrawETH() public {
        require(_transferETH(treasury, address(this).balance), "F");
    }
```

The financial loss is 100% of the contract balance in the worst case.

0x1337 : The Owner role has significant privileges across the code base, the most significant of which are unlimited mint authority of `SO3` token, and no cap on fees in the `setFeeBP()` function in the `SO3Chef` contract.

If the Owner address is compromised, the malicious owner can mint a significant amount of `SO3` token to himself and dilute the value of all other `SO3` token holders. It could also use the `setFeeBP()` function to set `mintFeeBP` to close to 100%, such that when users call `claim()`, almost all tokens being minted would go to owner affiliated `treasury` address.

Other privileged setter functions present similar risks. For example, the `agent` address can be changed at will by the Owner address, which would make the legit `SO3Market` contract unable to call the `deposit()` or `withdraw() function of the `SO3Chef` contract

Recommendation

yosriady : Add a zero address check to `setTreasury`.
0x1337 : Consider adding an upper limit on the `mintFeeBP` in the `setFeeBP()` function and adequately disclose it to the community. The Owner role should be managed by a multisig with timelock, and in the long term, a community governed DAO to mitigate the risk of privileged role compromises.

Client Response

Mitigated.1. Add a zero address check to setTreasury 2. The owner role will be managed by multisig with timelock.

SO3-2: Wrong power calculation in `SO3Market` contract `getPower` function

Category	Severity	Status	Contributor
Integer Overflow and Underflow	**Medium**	**Fixed**	**yosriady, Atlas**

Code Reference

- code/src/SO3Market.sol#L158-L160

```
158:    function getPower(uint256 vol, uint256 giff) public pure returns (uint256) {
159:        return giff + MINTER_POWER_0 + (vol * MINTER_POWER_INCREASE) / 1e18;
160:    }
```

Description

yosriady : With low values of `giff` and `vol`, it's possible that an user's power gets round down to 0 when it should not.

Consider the below function:

```
function getPower(uint256 vol, uint256 giff) public pure returns (uint256) {
        return giff + MINTER_POWER_0 + (vol * MINTER_POWER_INCREASE) / 1e18;
    }
```

Due to how Solidity division works, 99 / 100 round down to 0.

In the above function you have:

```
(vol * 100) / 1e18
```

If `vol` is ever less than 1e16 and/or, a portion of the `getPower()` calculation wil round down to 0.

Even though their `giff` and `vol` are greater than 0. Users will have powers less than they should due to precision loss.

Atlas : The documents said:

Users can purchase any miner by paying the miner's value. Computing Power = 100 * Total Transaction Volume + 100

Users can give any miner a gift. Giving a heart costs 100 SO3, and the miner who receives the gift will receive a bonus of 1 computing power.

But in the `getpower` method of `SO3Market` contract, it seems inappropriate to add the `MINTER_POWER_0` variable each time.

The impact is that users can increase their power in a cheaper way, resulting they can obtain more rewards in the so3Chef contract afterwards. Because they can multiple invoke `buy` or `like` methods and then call `so3Chef.deposit` with incorrect power amount.

Consider below POC contract

```
contract MarketTest {
    function testPower() public {
        address miner = makeAddr("MINER");
        market.buy{value: MINT_PRICE}(miner);

        (, uint256 firstActivePowerAmount, , ) = chef.userInfo(miner);
        assertEq(firstActivePowerAmount, 11);

        (, uint256 beforePowerAmount, , ) = chef.userInfo(miner);
        likeMiner("alice", miner, 1, 0);
        (, uint256 afterPowerAmount, , ) = chef.userInfo(miner);

        // invoke like function with 1 but get 11 power because wrong addition of MINTER_POWER_0 eac
h time
        assertEq(afterPowerAmount - beforePowerAmount, 1);
    }
    ...
}
```

Recommendation

yosriady : Consider using fixed point math libraries such as solmate's FPML or PRBMath; or use units with more decimal places to avoid precision loss.

Atlas : Remove the addation of `MINTER_POWER_0` in the `getPower` function. only add `MINTER_POWER_0` when first active this minner.

Consider below fix in the `SO3Market.getPower()` function. Introducing a unique function called calculatePower for power calculation. or others more graceful implementations

```
    function getPower(uint256 vol, uint256 giff) public pure returns (uint256) {
        return MINTER_POWER_0 + calculatePower(vol, giff);
    }

    function calculatePower(
        uint256 vol,
        uint256 giff
    ) private pure returns (uint256) {
        return giff + (vol * MINTER_POWER_INCREASE) / 1e18;
    }
```

Client Response

Fixed

SO3-3:Potential Reentrancy from `Miner` Address

Category	Severity	Status	Contributor
Reentrancy	**Medium**	**Fixed**	**0x1337**

Code Reference

- code/src/SO3Market.sol#L105-L111
- code/src/SO3Market.sol#L221-L227

```
105:            if (!_transferETH(miner, f2)) {
106:                f1 = f1 + f2;
107:                f2 = 0;
108:            }
109:            if (!_transferETH(m.host, msg.value - f1 - f2)) {
110:                f1 = msg.value - f2;
111:            }

221:    function _transferETH(address to, uint256 amount) internal returns (bool success) {
222:        /// @solidity memory-safe-assembly
223:        assembly {
224:            // Transfer the ETH and store if it succeeded or not.
225:            success := call(gas(), to, amount, 0, 0, 0, 0)
226:        }
227:    }
```

Description

0x1337 : We have already established that the `Miner` address can actually be a smart contract address in a separate finding (code length is 0 in constructor). The implication is that the `Miner` address can reenter the `_grab()` function via the `buy()` function, and potentially receive ETH that otherwise would belong to the `host` address. The `miner` smart contract can call the `buy()` function, and if the `host` is already set, the `_grab()` function is triggered in line 65. Then in the `_grab()` function, ETH is transferred to the `miner` address, and if the `miner` address is a smart contract with a payable `receive()` or `fallback()` function, it could potentially reenter the `SOMarket` contract.

Recommendation

0x1337 : Consider using function modifiers such as `nonReentrant` from Reentrancy Guard to prevent re-entrancy at the contract level.

Client Response

Fixed.The checks in lines 16 and 18 protect against re-entry attacks by host. And can you tell me how Miner executed the attack? Even if Miner executes Buy in the Receive function, he needs to pay ETH to buy it.However, a non-accessibility check was added to the buy function in issue SO3-6.

SO3-4:Basic computing power mismatch with the document in `Vars.sol` contract

Category	Severity	Status	Contributor
Logical	Low	Declined	Atlas

Code Reference

- code/src/Vars.sol#L7

```
7:uint256 constant MINTER_POWER_0 = 10;
```

Description

Atlas : The documents said:

At the same time, each user has two attributes: value and computing power, with a basic value of 0.1 ETH and a basic computing power of 100.

But in the implementation of the Vars contract. It was a basic computing power of 10.

Recommendation

Atlas : Check that whether the basic computing power is as expected. Because it would affect the costs and profits of users throughout the whole SO3 token economy system.

Consider below fix in the sample.test() function

```
uint256 constant MINTER_POWER_0 = 100;
```

Client Response

Declined.Documentation is not up to date. At the same time, each user has two attributes: value and computing power, with a basic value of 0.01 ETH and a basic computing power of 10.

SO3-5:Gift cost mismatch with the document in `SO3Market` contract `like` function

Category	Severity	Status	Contributor
Logical	Low	Declined	Atlas

Code Reference

- code/src/SO3Market.sol#L126

```
126:        uint256 amount = giff * 1000 * 1e18;
```

Description

Atlas : The documents said:

Users can give any miner a gift. Giving a heart costs 100 SO3, and the miner who receives the gift will receive a bonus of 1 computing power. The SO3 consumed for giving gifts will be burned.

But in the implementation of the `like` function. It costs 1000 SO3.

Recommendation

Atlas : Check that whether the cost are as expected.

Consider below fix in the `sample.test()` function

```
//burn so3
uint256 amount = giff * 100 * 1e18;
```

Client Response

Declined.Documentation is not up to date. Users can give any miner a gift. Giving a heart costs 1000 SO3, and the miner who receives the gift will receive a bonus of 1 computing power. The SO3 consumed for giving gifts will be burned. Note: We will reset to 200 SO3.

SO3-6:Insufficient EOA Check could Allow Miner to be a Contract Address

Category	Severity	Status	Contributor
Language Specific	Low	Fixed	0x1337

Code Reference

- code/src/SO3Market.sol#L71

```
71:        if (miner.code.length > 0) revert MINNER_MUSTBE_EOA();
```

Description

0x1337 : In line 71 of the `SO3Market` contract, the check intends to enforce that the `miner` can only be an EOA address. However, the code length check is insufficient, because "this method relies on extcodesize/address.code.length, which returns 0 for contracts in construction, since the code is only stored at the end of the constructor execution". Refer to the commentary in the OpenZeppelin implementation of the `isContract()` check below.

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/09329f8a18f08df65863a5060f6e776bf7fccacf/contracts/utils/Address.sol#L40-L45

The implication is that the miner address can be a smart contract address, with potential reentrancy issue described in a separate finding.

Recommendation

0x1337 : Add a check in line 71 to enforce that `msg.sender == tx.origin`

Client Response

Fixed.Allow miner is smart contract ,and add nonReentrant check for buy method.

SO3-7:Missing Storage Gap in Upgradeable Contracts

Category	Severity	Status	Contributor
Language Specific	Low	Declined	0x1337

Code Reference

- code/src/UUPSUpgradeableExp.sol#L19-L21

```
19:     function _init() internal {
20:         _transferOwnership(msg.sender);
21:     }
```

Description

0x1337 : There is no storage gap in the implementation contracts.

Best practice is to include storage gap in logic contracts to allow for inheritance and the inclusion of additional variables in contract storage in future upgrades. Please refer to the bottom part of the following OZ guide.

https://docs.openzeppelin.com/contracts/3.x/upgradeable#storage_gaps

Recommendation

0x1337 : Add a storage gap of a reasonable size in the implementation contracts

Client Response

Declined.It is safe for current contract

SO3-8:Upgradeable Contract Inherits Non Upgradeable Contract

Category	Severity	Status	Contributor
Language Specific	**Low**	**Declined**	**0x1337**

Code Reference

- code/src/UUPSUpgradeableExp.sol#L6-L8

```
6:import "@openzeppelin/contracts/access/Ownable.sol";
7:
8:abstract contract UUPSUpgradeableExp is Ownable, Initializable, UUPSUpgradeable {
```

Description

0x1337 : Upgradeable contract should inherit other upgradeable contracts in order to maintain upgradeability. Currently, the `UUPSUpgradeableExp` contract inherits OpenZeppelin's `Ownable` contract, which is not upgradeable. It should inherit the `OwnableUpgradeable` contract instead.

Refer to the OpenZeppelin guideline here: https://docs.openzeppelin.com/contracts/3.x/upgradeable

Recommendation

0x1337 : Inherit `OwnableUpgradeable` from this directory instead and initialize the `Owner` address accordingly: https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/access/OwnableUpgradeable.sol

Client Response

Declined.It is safe for current contract

SO3-9:Missing Event Emit

Category	Severity	Status	Contributor
Code Style	**Informational**	**Declined**	**yosriady, 0x1337**

Code Reference

- code/src/SO3.sol#L17-L24
- code/src/SO3Chef.sol#L119
- code/src/SO3Chef.sol#L119-L127
- code/src/SO3Market.sol#L192-L199
- code/src/SO3Market.sol#L208

```
    function setMaster(address acct) external onlyOwner {
        require(acct != address(0), "EMPTY");
        master = acct;
    }

    function resign() external {
        require(master == msg.sender, "REJ");
        master = address(0);

    function setAgent(address newAgent) external onlyOwner {

    function setAgent(address newAgent) external onlyOwner {
        if (newAgent == address(0)) revert ADDRESS_IS_EMPTY();
        agent = newAgent; //ignore event
    }

    function setSO3(IMintPool miner) external onlyOwner {
        if (address(miner) == address(0)) revert ADDRESS_IS_EMPTY();
        so3Miner = miner;
    }

    function setChef(IChef chef) external onlyOwner {
        require(address(chef) != address(0));
        so3Chef = chef;
    }

    function setWhitelistStatus(bool enable) external onlyOwner {
        minerWhitelistEnabled = enable;
    }

    function setTradeStartBlock(uint256 blockNumber) external onlyOwner {
```

Description

yosriady : The following functions does not emit any events. All critical state changing operations should emit events. Especially admin functions.

```
    function setWhitelistStatus(bool enable) external onlyOwner {
        minerWhitelistEnabled = enable;
    }

    function setTradeStartBlock(uint256 blockNumber) external onlyOwner {
        tradeStartBlock = blockNumber;
    }


    function setAgent(address newAgent) external onlyOwner {
        if (newAgent == address(0)) revert ADDRESS_IS_EMPTY();
        agent = newAgent; //ignore event
    }

    function setSO3(IMintPool miner) external onlyOwner {
        if (address(miner) == address(0)) revert ADDRESS_IS_EMPTY();
        so3Miner = miner;
    }
```

0x1337 : Privileged setter functions should emit event. While some setter functions do, some are missing event emit.

Recommendation

yosriady : Emit an event.

0x1337 : Include event emit in all privileged setter functions

Client Response

Declined.We don’t need use it on off-chain.

SO3-10:Unused code

Category	Severity	Status	Contributor
Code Style	**Informational**	**Fixed**	**yosriady, Atlas**

Code Reference

- code/src/Vars.sol#L22
- code/src/SO3Chef.sol#L71-L85

```
22:error GIFF_OVERFLOW();

71:    function deposit(address acct, address host, uint256 amount) external override {
72:        if (msg.sender != agent) revert INVALID_CHEF_AGENT();
73:        _updateUser(acct);
74:
75:        address currHost = userInfo[acct].host;
76:        if (currHost == address(0)) {
77:            userInfo[acct].host = host;
78:        } else if (currHost != host) {
79:            revert HOST_MISMATCH();
80:        }
81:        unchecked {
82:            userInfo[acct].amount += amount;
83:        }
84:        totalDeposits += amount;
85:    }
```

Description

yosriady : The following errors are unused anywhere in the system and can be removed:

```
error GIFF_OVERFLOW();
```

Atlas : The `Deposit` event in SO3Chef is declared but was never used.

Recommendation

yosriady : Remove unused code.
Atlas : Emit `Deposit` event in `deposit` function.

Consider below fix in the `SO3Chef.deposit()` function

```
function deposit(address acct, address host, uint256 amount) external override {
    ...
    unchecked {
        userInfo[acct].amount += amount;
    }
    totalDeposits += amount;
    emit Deposit(acct, amount);
}
```

Client Response

Fixed

SO3-11: `withrawETH()` Function should be `withdrawETH()`

Category	Severity	Status	Contributor
Code Style	Informational	Fixed	0x1337

Code Reference

- code/src/SO3Market.sol#L149

```
149:    function withrawETH() public {
```

Description

0x1337 : Typo in function name `withrawETH()`

Recommendation

0x1337 : Fix the typo and change function name to `withdrawETH()`

Client Response

Fixed

Disclaimer