Smart Contract Audit Report

DeFiGeek-Community/yamawake
2024/03/07

INDEX

1.Introduction

Sammy Digital Security Co., Ltd. is conducting a smart contract audit for "yamawake" based on the contract with SG LLC.

We systematically detect and evaluate potential issues in the design phase, implementation flaws, and inconsistencies between the code and design documents in the smart contract implementation.

As the detected vulnerabilities may have an impact on the security of the system, we recommend implementing appropriate measures based on the risk level and impact for the identified issues.

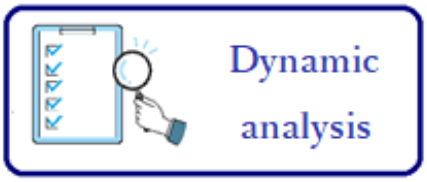
1.1 Audit Target

Project	yamawake
Contract	GitHub - DeFiGeek-Community/yamawake
Branch	bc813ed
Completion Date	2024/03/07

2. Audit Methodology

2.1 Smart Contract Audit Methodology Overview

Understands the purpose and overview of the contract and analyze design risks.



Conducts manual testing for both normal and abnormal scenarios, as well as standardized testing based on SWC (Smart Contract Weakness Classification) and Consensys Best Practices.

Performs code analysis using diagnostic tools such as Slither to identify any potential vulnerabilities or issues in the code.

Creates a report that includes explanations of discovered vulnerabilities, associated risks, and recommended countermeasures.

3. Smart Contract Audit Methodology

This audit is based on vulnerability criteria such as CWE (Common Weakness Enumeration), SWC (Smart Contract Weakness Classification and Test Cases), Ethereum Smart Contract Security Best Practices, and our own proprietary vulnerability assessment criteria. We conduct ranking based on these criteria.

重大度	説明
High	There are vulnerabilities that can lead to the functionality disruption or corruption of the contract, as well as potential loss of assets for the contract and users, loss of data, or the risk of unauthorized manipulation of assets or data. It is necessary to address these vulnerabilities before migrating to the mainnet.
Medium	There are vulnerabilities that have the potential to impact the provision of services and availability through smart contracts. These vulnerabilities at this level pose a risk of potential issues, and it is recommended to address and fix them to mitigate any potential risks.
Low	These vulnerabilities do not directly impact the provision of services or availability. However, they may pose potential issues in the future. It is recommended to address and fix these vulnerabilities at this level as much as possible to mitigate any potential risks that may arise.
Info	The items mentioned here are not vulnerabilities per se, but they include recommendations for improving code quality and enhancing security.

4.Audit Summary

4.1 Manual Review

In the manual review of the target smart contract, we are focusing on the following aspects:

・Overall code quality
・Compliance with best practices
・Ensuring alignment between expected behavior from documentation/comments and the code logic

Additionally, we conduct fuzz tests or other necessary tests as appropriate.

```
function testAddScore(uint256 x⬆) public {
    factory.addTemplate(templateName, implementionAddr, initializeSignature, transferSignature);

    uint256 value = 100;

    bytes memory args = abi.encode(value);
    address target_ = address(0x123);

    address deployAuctionAddr = factory.deployAuction(templateName,args);

    factory.auctions(deployAuctionAddr);

    vm.prank(deployAuctionAddr);

    _distributor.addScore(target_,x⬆);
```

```
[PASS] testAddScore(uint256) (runs: 256, μ: 165844, ~: 166777)
```

4.2 SWC Registry

The following is a list of vulnerabilities registered in the Smart Contract Weakness Classification and Test Cases. It serves as a reference to check if any of these vulnerabilities are present in the code.

ID	概要	実施
SWC-100	Function Default Visibility	☑
SWC-101	Integer Overflow and Underflow	☑
SWC-102	Outdated Compiler Version	☑
SWC-103	Floating Pragma	☑
SWC-104	Unchecked Call Return Value	☑
SWC-105	Unprotected Ether Withdrawal	☑
SWC-106	Unprotected SELFDESTRUCT Instruction	☑
SWC-107	Reentrancy	☑
SWC-108	State Variable Default Visibility	☑
SWC-109	Uninitialized Storage Pointer	☑
SWC-110	Assert Violation	☑
SWC-111	Use of Deprecated Solidity Functions	☑
SWC-112	Delegatecall to Untrusted Callee	☑
SWC-113	DoS with Failed Call	☑
SWC-114	Transaction Order Dependence	☑
SWC-115	Authorization through tx.origin	☑
SWC-116	Block values as a proxy for time	☑
SWC-117	Signature Malleability	☑
SWC-118	Incorrect Constructor Name	☑
SWC-119	Shadowing State Variables	☑
SWC-120	Weak Sources of Randomness from Chain Attributes	☑
SWC-121	Missing Protection against Signature Replay Attacks	☑
SWC-122	Lack of Proper Signature Verification	☑
SWC-123	Requirement Violation	☑
SWC-124	Write to Arbitrary Storage Location	☑

SWC-125	Incorrect Inheritance Order	☑
SWC-126	Insufficient Gas Griefing	☑
SWC-127	Arbitrary Jump with Function Type Variable	☑
SWC-128	DoS With Block Gas Limit	☑
SWC-129	Typographical Error	☑
SWC-130	Right-To-Left-Override control character (U+202E)	☑
SWC-131	Presence of unused variables	☑
SWC-132	Unexpected Ether balance	☑
SWC-133	Hash Collisions With Multiple Variable Length Arguments	☑
SWC-134	Message call with hardcoded gas amount	☑
SWC-135	Code With No Effects	☑
SWC-136	Unencrypted Private Data On-Chain	☑

4.3 Detected Vulnerabilities List

The following table provides a list of vulnerabilities/observations detected during the audit.

Audit Target	Risk Level and Number of Findings			
	High	Medium	Low	Info
Vulnerabilities across the entire	0	0	0	2
BaseTemplate.sol	0	0	0	0
Distributor.sol	0	0	0	0
Factory.sol	0	0	0	1
FeePool.sol	0	0	0	0
SampleToken.sol	0	0	0	0
TemplateV1.sol	0	0	0	0
TemplateV1WithCreationFee.sol	0	0	0	0
YMWK.sol	0	0	0	0
Total	0	0	0	3

No	details	Risk	Status
A.1	Floating Pragma	Info	acknowledged
A.2	Insufficient appropriate comments in the code	Info	fixed
A.3	Concerns about memory reference offset	Info	acknowledged

4.4 Vulnerability details

A.1 Floating Pragma

▼Summary

It is recommended to deploy contract code with the same compiler version as the most thoroughly tested one. By fixing the pragma, unknown bug risks can be mitigated. Although pragma solidity ^0.8.18; is specified, it is recommended to fix the pragma to the latest version.

A.2 Insufficient appropriate comments in the code

▼Summary

The following points are lacking NatSpec (Natural Specification) in this smart contract:

・Contract description: Explanation of the purpose and functionality of the contract.
・Function description: Description of the purpose, arguments, and return values of each function.
・Event description: Explanation of the purpose and occurrence conditions of events.
・Modifier description: Explanation of the conditions checked by modifiers and the role they play. For example, the "onlyAuction" modifier.

By adding NatSpec and enhancing documentation, you can improve code readability, understanding, and maintainability.

A.3 Concerns about memory reference offset

▼Summary

There is a concern regarding the possibility of a mismatch between the expected address and the cloned address due to memory reference offset in inline assembly memory operations. While the actual risk of this occurring is low, if security is a priority, it is recommended to pay attention to the generation method of the salt and the accuracy of the implemented address.

▼Relevant code (Factory.sol > Line: 103)

```
/// @dev Deploy implemention's minimal proxy by create2
ftrace | funcSig
function _createClone(    Factory._createClone(address) (src/Factory.sol#103-124) uses assembly INLINE ASM (src/Factory.sol#109-122)
    address implementation_
) internal returns (address result) {
    nonce += 1;
    bytes32 salt = keccak256(abi.encodePacked(implementation_, nonce));
    // OpenZeppelin Contracts (last updated v4.8.0) (proxy/Clones.sol)
    assembly {
        mstore(
            0x00,
            or(
                shr(0xe8, shl(0x60, implementation_)),
                0x3d602d80600a3d3981f3363d3d373d3d3d363d73000000
            )
        )
        mstore(
            0x20,
            or(shl(0x78, implementation_), 0x5af43d82803e903d91602b57fd5bf3)
        )
        result := create2(0, 0x09, 0x37, salt)
    }
    require(result != address(0), "ERC1167: create2 failed");
}
```

4.5 Static Analysis Details

Details The following is the result of code analysis conducted by a code analysis tool.

① Slither

▼List of High-Risk Findings (10 items)

```
TemplateV1.withdrawRaisedETH() (src/TemplateV1.sol#164-190) sends eth to arbitrary user
        Dangerous calls:
        - address(owner).transfer(address(this).balance) (src/TemplateV1.sol#189)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbi
trary-destinations

SampleToken is re-used:
        - SampleToken (src/SampleToken.sol#11-29)
        - SampleToken (src/sampleToken.sol#11-29)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#name-reused

StdCheats.vm (lib/forge-std/src/StdCheats.sol#480) shadows:
        - StdCheatsSafe.vm (lib/forge-std/src/StdCheats.sol#10)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
```

▼Analysis Details

1. The function TemplateV1.withdrawRaisedETH() has the potential to send Ether to any user. This poses a risk of attackers exploiting the contract to illegitimately withdraw Ether. It is necessary to review the implementation of this function and strictly control the destination of Ether transfers.

2. In StdCheats.vm, it is overwriting StdCheatsSafe.vm. This can potentially cause unexpected bugs due to variables having the same name and wider scope of influence. It is important to make variable names unique to avoid such confusion.

3. The following functions are ignoring return values. This can lead to incorrect processing of transaction results. It is recommended to

modify them to properly handle the return values as needed.

- Distributor.claim
- FeePool.withdrawToken
- TemplateV1.claim
- TemplateV1.withdrawERC20Onsale
- TemplateV1.initializeTransfer
- TemplateV1WithCreationFee.claim
- TemplateV1WithCreationFee.withdrawERC20Onsale
- TemplateV1WithCreationFee.initializeTransfer

▼ List of Medium-Risk Findings (24 items)

```
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-131) per
forms a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#98)
        - inverse = (3 * denominator) ^ 2 (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#113)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-131) per
forms a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#98)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#1
17)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-131) per
forms a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#98)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#1
18)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-131) per
forms a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#98)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#1
19)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-131) per
forms a multiplication on the result of a division:
20)
        - current_rate = (current_rate * RATE_REDUCTION_COEFFICIENT) / RATE_DENOMINATOR (src/YMWK.sol#171-
173)
YMWK.mintable_in_timeframe(uint256,uint256) (src/YMWK.sol#131-182) performs a multiplication on the result
 of a division:
        - to_mint += current_rate * (current_end - current_start) (src/YMWK.sol#164)
        - current_rate = (current_rate * RATE_REDUCTION_COEFFICIENT) / RATE_DENOMINATOR (src/YMWK.sol#171-
173)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

StdCheatsSafe.rawToConvertedEIPTx1559s(StdCheatsSafe.RawTx1559[]).i (lib/forge-std/src/StdCheats.sol#245)
is a local variable never initialized
StdCheatsSafe.readEIP1559ScriptArtifact(string).artifact (lib/forge-std/src/StdCheats.sol#232) is a local
variable never initialized
StdCheatsSafe.rawToConvertedReceipts(StdCheatsSafe.RawReceipt[]).i (lib/forge-std/src/StdCheats.sol#313) i
s a local variable never initialized
StdCheatsSafe.rawToConvertedEIPTx1559(StdCheatsSafe.RawTx1559).transaction (lib/forge-std/src/StdCheats.so
l#252) is a local variable never initialized
StdCheatsSafe.rawToConvertedEIP1559Detail(StdCheatsSafe.RawTx1559Detail).txDetail (lib/forge-std/src/StdCh
eats.sol#268) is a local variable never initialized
YMWK.mintable_in_timeframe(uint256,uint256).i (src/YMWK.sol#151) is a local variable never initialized
StdCheatsSafe.rawToConvertedReceipt(StdCheatsSafe.RawReceipt).receipt (lib/forge-std/src/StdCheats.sol#320
) is a local variable never initialized
StdCheatsSafe.rawToConvertedReceiptLogs(StdCheatsSafe.RawReceiptLog[]).i (lib/forge-std/src/StdCheats.sol#
344) is a local variable never initialized
```

```
FeePool.withdrawToken(address,address[]).i (src/FeePool.sol#30) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

StdChains.getChainWithUpdatedRpcUrl(string,StdChains.Chain) (lib/forge-std/src/StdChains.sol#153-172) igno
res return value by vm.rpcUrl(chainAlias) (lib/forge-std/src/StdChains.sol#155-169)
StdCheatsSafe.isFork() (lib/forge-std/src/StdCheats.sol#428-432) ignores return value by vm.activeFork() (
lib/forge-std/src/StdCheats.sol#429-431)
ERC721._checkOnERC721Received(address,address,uint256,bytes) (lib/openzeppelin-contracts/contracts/token/E
RC721/ERC721.sol#399-421) ignores return value by IERC721Receiver(to).onERC721Received(_msgSender(),from,t
okenId,data) (lib/openzeppelin-contracts/contracts/token/ERC721/ERC721.sol#406-417)
FactoryTest.testAddScore(uint256) (test/Factory.t.sol#33-50) ignores return value by factory.auctions(depl
oyAuctionAddr) (test/Factory.t.sol#43)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

▼Analysis Details

1. The function Math.mulDiv(uint256,uint256,uint256) performs multiplication on the result of division. This operation can lead to inaccurate calculation results.

2. ThefunctionsTemplateV1._calculateAllocation(uint256,uint256,uint256)andTemplateV1WithCreationFee._calculateAllocation(uint256,uint256,uint256) also perform multiplication on the result of division. This can result in inaccurate calculation results.

3. The function YMWK.mintable_in_timeframe(uint256,uint256) also performs multiplication on the result of division. Similarly, this can lead to inaccurate calculation results.

4. Several uninitialized local variables have been detected. These variables are not being used and can potentially impact code readability and maintainability. Consider properly initializing these variables or removing them if they are unnecessary.

■Summary of Analysis Results

Upon analyzing the results of the static analysis, several vulnerabilities were detected, but no vulnerabilities were found that would significantly impact the service's provision or availability in reality. The majority of the results pertain to libraries, and no major vulnerabilities have been identified.

For the detected HIGH-risk and MEDIUM-risk vulnerabilities (excluding LOW and below), please refer to the information provided above. We recommend reviewing them and taking further testing or mitigation measures as necessary.

5. Disclaimer

This smart contract audit is designed to detect vulnerabilities based on our auditing methodology, but it does not guarantee the detection of all security issues related to this smart contract.

To provide further security assurance, we recommend engaging multiple auditing firms and implementing bug bounty programs, among other measures.