STAYSAFU AUDIT

MARCH 12TH, 2022

THE TAVERN

TABLE OF CONTENTS

AUDIT SUMMARY

This report was written for The Tavern in order to find flaws and vulnerabilities in the The Tavern project's source code, as well as any contract dependencies that weren't part of an officially recognized library.

A comprehensive examination has been performed, utilizing Static Analysis, Manual Review, and The Tavern Deployment techniques. The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors
- Assessing the codebase to ensure compliance with current best practices and industry standards
- Ensuring contract logic meets the specifications and intentions of the client
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders
- Through line-by-line manual review of the entire codebase by industry expert

AUDIT OVERVIEW

PROJECT SUMMARY

Project name	THE TAVERN
Description	The Tavern is a passive income protocol that improves upon current node projects, by combining gamification, NFTs and a treasury-backed token to reward participants sustainably over the long term.
Platform	Avalanche
Language	Solidity
Codebase	https://github.com/TavernInnkeeper/tavern-smart-contracts/tree/update/contracts

FINDINGS SUMMARY

Vulnerability	Total
Critical	0
Major	0
Medium	1
Minor	4
Informational	3

EXECUTIVE SUMMARY

The Tavern is a passive income protocol based on gamification, NFTs and passive income. The core token of the project is the Mead (MEAD) (ERC20 token), which will be launched with a total supply of 2500000. The initial supply will be divided as follows :

- 54% for rewards reserve (including 10% for LP Rewards)
- 10% for treasury reserve
- 16% for whitelist presale
- 20% for liquidity

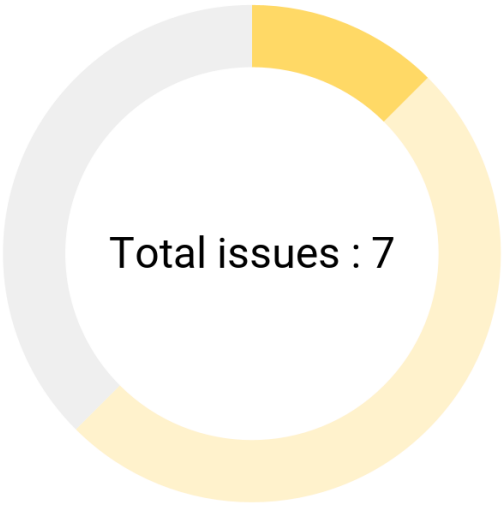
The project is also powered by the Brewery (BREWERY) NFT. Holding Brewery rewards you with a certain amount of Mead. Brewery is an upgradeable NFT since you can change its name, gain experience with it (which increases the number of Mead it earns via a tier system), and trade it on the open market. When purchasing a BREWERY, the fees will be split as follows :

- 70% goes to the rewards pool
- 30% goes to the treasury

When claiming Mead from Brewery, the holder will be taxed at a fixed rate (between 12% and 18%) based on its reputation. Brewers reputation is a type of experience system that rewards brewers for continued beneficial participation in the protocol.

There have been no major or critical issues related to the codebase and all findings listed here are informational, minor or medium security issues. The major security problem is the centralization of privileges.

AUDIT FINDINGS



- Medium : 1
- Minor : 4
- Informational : 3

Code	Title	Severity
CENT-1	Centralization of major privileges	Medium
BLOC-1	Usage of block.timestamp	Minor
THRE-2a	Missing threshold for major function	Minor
THRE-2b	Missing threshold for major function	Minor
COMP-1	Unfixed version of compiler	Minor
LOG-1	Logical programming error	Informational
MSG-3	Error message too long	Informational
MSG-1	Unclear error message	Informational

MSG-1 | Unclear error message

Description

Some of the error messages in the smart contract are unclear or badly formulated. In order to optimize error handling and project maintenance, we recommend using the clearest possible error messages.

1 error of this type has been found in xMEAD.sol.

1 error of this type has been found in Mead.sol.

Recommendation

We recommend replacing these ambiguous messages with clearer ones :

```
//Edited code with clearer error messages in xMEAD.sol
//l71
require(account != address(0), "Cannot redeem from zero
address");

//Edited code with clearer error messages in Mead.sol
//l156 (we recommend splitting this require statement in
order to have clearer error handling)

require(!blacklist[from] , "Sender address blacklisted");
require(!blacklist[to] , "receiver address blacklisted");
```

MSG-3 | Too long error message

Description

Audited smart contracts contain some error messages that are too long. The industry standards specify error messages must have a maximal length of 32 bytes. We recommend having the shortest possible error messages to optimize gas costs (see github.com/ethereum/solidity/issues/4588) and improve error handling.
5 issues of this type have been found in Mead.sol.
3 issues of this type have been found in Brewery.sol.
1 issue of this type has been found in TavernSettings.sol.

Recommendation

We recommend shortening these error messages :

```
//Edited code containing missing error message in
Mead.sol
//l262
require(currentAllowance >= subtractedValue, "ERC20:
allowance under zero");
//l283
require(account != address(0), "ERC20: burn from the 0
address");
//l288
require(accountBalance >= amount, "ERC20: burn amount
over balance");
//l304
```

```
require(owner != address(0), "ERC20: approve from 0
address");
//l305
require(spender != address(0), "ERC20: approve to 0
address");

//Edited code containing missing error message in
Brewery.sol
//l150
require(breweryCount > 0, "Not enough pending MEAD");
//l170
require(totalRewards >= cost, "Not enough pending MEAD");
//l372
require(getApproved(_tokenId) == address(0), "BREWERY is
approved for spend/list");

//Edited code containing missing error message in
TavernSettings.sol
//l81
require(_classTaxes.length == classCount, "Class tax
array length != count");
```

LOG-1 | Logic programming error

Description

Some parts of the audited code contain logical errors. This is not a security problem but we still would like to inform you and bring you an explanation and a solution.

1 error of this type has been found in Brewery.sol.

Recommandation

We recommend updating this part of the smart contract to fit the function's logic :

```
//Edited code containing corrected logic from Brewery.sol
//l496
function editTier(uint256 _tier, uint256 _xp, uint256
_yield) external onlyRole(DEFAULT_ADMIN_ROLE) {
        require(tiers.length > _tier, "Tier doesnt
exist");
//Let's imagine here you set the 'require' condition as
//<tiers.length >= _tier>. Here is an example of what
//this bad condition can cause :
//If the 'tiers' array has a length of 1, the _tier
//argument can be set at 1. But calling tiers[1] is not
//valid, since tiers array has a length of 1.
        tiers[_tier] = _xp;
        yields[_tier] = _yield;
    }
```

BLOC-1 | Using block.timestamp

Description

The use of block.timestamp can be problematic. The timestamp can be partially manipulated by the miner (see https://cryptomarketpool.com/block-timestamp-manipulation-attack/). In this smart contract this is not critical as in the worst case an attacker could force the automatic liquify to run faster.

1 error of this type has been found in Brewery.sol.

Recommendation

We fully understand the smart contract's logic of the BREWERY NFT. The use of block.timestamp is required to power the reward mechanism and we cannot replace it. Nevertheless, it is still useful to point out this kind of potential security problem.

COMP-1 | Unfixed version of compiler

Description

Audited contracts do not have locked compiler versions, meaning a range of compiler versions can be used. This can lead to differing bytecodes being produced depending on the compiler version, which can create confusion when debugging as bugs may be specific to a specific compiler version(s).

To rectify this, we recommend setting the compiler to a single version, the lowest version tested to be compatible with the code, an example of this change can be seen below.

Recommendation

We recommend fixing the compiler version to the most recent one :

```
//Edited code containing fixed compiler version (usable
for every audited smart contract)
pragma solidity 0.8.4;
```

THRE-2a | Missing threshold in minting function

Description

Minting function (mint from Mead.sol) does not implement any threshold. The owner can then mint a potentially infinite amount of tokens to the treasury address.

1 error of this type has been found in Mead.sol.

Recommendation

We recommend adding a threshold (as example : set a maximum minting threshold of 1% of the total supply) to the function to avoid this problem. You can also implement a cooldown to fight this problem :

```
//Edited code containing threshold for minting function
//l131
    function mint(uint256 _amount) public onlyOwner {
require(_amount > _totalSupply/100, "cannot mint that
much");
//1% is only an example, we recommend you find the right
//threshold to fit the project's logic
        _mint(msg.sender, _amount * 10**DECIMALS);
    }
```

THRE-2b | Missing threshold in burning function

Description

Burning function (burn from Mead.sol) does not implement any threshold. The owner can then mint a potentially infinite amount of tokens to the treasury address.

1 error of this type has been found in Mead.sol.

Recommendation

We recommend adding a threshold (as example : set a maximum burning threshold of 1% of the total supply) to the function to avoid this problem. You can also implement a cooldown to fight this problem :

```
//Edited code containing threshold for minting function
//l131
    function burn(uint256 _amount) public onlyOwner {
require(_amount > _totalSupply/100, "cannot burn that
much");
//1% is only an example, we recommend you find the right
//threshold to fit the project's logic
        _mint(msg.sender, _amount * 10**DECIMALS);
    }
```

CENT-1 | Centralization of major privileges

Description

The onlyOwner modifier of the Mead.sol smart contract gives major privileges over it (owner can handle the white/blacklist, burn without limit or withdraw tokens)*. This can be a problem, in the case of a hack, an attacker who has taken possession of these privileged accounts could damage the project and the investors.

*This list is not exhaustive but presents the most sensitive points.

The Admin modifier of the Brewery.sol smart contract gives major privileges over it (Admin can change parameters for every single NFT)*. This can be a problem, in the case of a hack, an attacker who has taken possession of these privileged accounts could damage the project and the investors.

*This list is not exhaustive but presents the most sensitive points.

The DEFAULT_ADMIN_ROLE modifier of the ClassManager.sol smart contract gives major privileges over it (Admin can notably clear class threshold)*. This can be a problem, in the case of a hack, an attacker who has taken possession of these privileged accounts could damage the project and the investors.

*This list is not exhaustive but presents the most sensitive point

The onlyOwner modifier of the TavernSettings.sol smart contract gives major privileges over it (owner can notably change other Tavern's project smart contracts' addresses)*. This can be a problem, in the case of a

hack, an attacker who has taken possession of these privileged accounts could damage the project and the investors.

*This list is not exhaustive but presents the most sensitive points.

Recommendation

We recommend at least to use a multi-sig wallet for admin/ owner/manager address, and at best to establish a community governance protocol to avoid such centralization. For more information, see https://solidity-by-example.org /app/multi-sig-wallet/

Global security warnings

These are safety issues for the whole project. They are not necessarily critical problems but they are inherent in the structure of the project itself. Potential attack vectors for these security problems should be monitored.

CENT-1 | Global SPOF (Single Point Of Failure)

The project's smart contracts often have a problem of centralized privileges. The owner and Admin system in particular can be subject to attack. To address this security issue we recommend using a multi-sig wallet, establishing secure project administration protocols and strengthening the security of project administrators.

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 StaySAFU's 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 StaySAFU 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.

Blockchain technology and cryptographic assets present a high level of ongoing risk.

StaySAFU's position is that each company and individual are responsible for their own due diligence and continuous security. StaySAFU's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or fun.