STAYSAFU
AUDIT

JANUARY 21TH, 2023

AsterFi

TABLE OF CONTENTS

AUDIT SUMMARY

This report was written for AsterFi in order to find flaws and vulnerabilities in the AsterFi 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 AsterFi 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

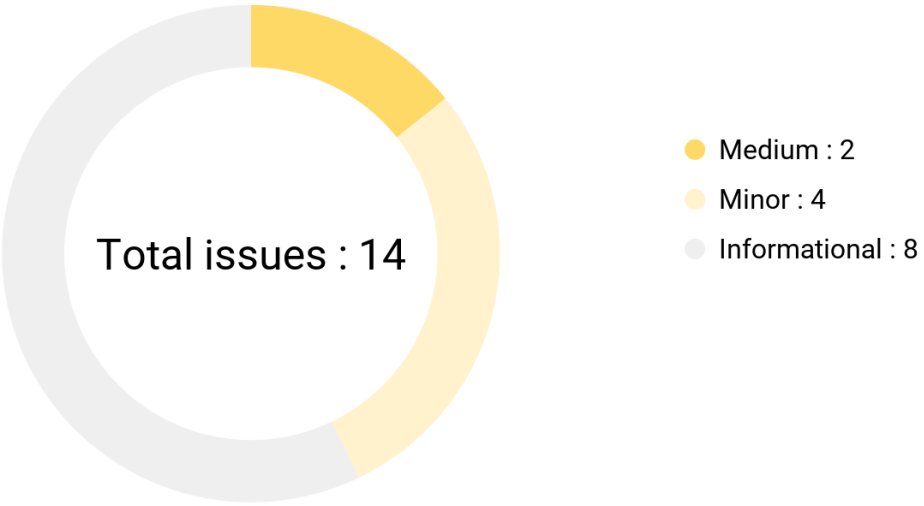
AUDIT OVERVIEW

PROJECT SUMMARY

Project name	AsterFi
Description	ASTERFI is a unique NFT collection that combines the value of cryptocurrency investments with the collectability of NFTs.
Platform	Ethereum
Language	Solidity
Codebase	https://etherscan.io/address/0xaF47Fc285A337a5B2e12ae3f2b076CF99Ce89981#code

FINDINGS SUMMARY

Vulnerability	Total
Critical	0
Major	0
Medium	2
Minor	4
Informational	8



AUDIT FINDINGS

Code	Title	Severity
OWNR-1	Transferring Ownership should be two-step process	Medium
AMNT-1	Zeroed msg.value used to update backUpAmount	Medium
MAPP-1	tokensPerWallet mapping initiated when ERC721A has a balance mapping	Minor
PASS-1	Passing msg.value across functions in existing contract context	Minor
WRAP-1	TransferERC20 after ETH wrap does nothing	Minor

SWAP-1	swapToken branch unused	Minor
STOR-1	Storage Variables can be packed to save gas	Informational
STOR-2	Storage Struct can be Packed	Informational
GAS-1	Short require strings save gas	Informational
MESS-1	Revert message may be unclear	Informational
LOOP-1	Setting iterated value to 1 in for loop	Informational
METD-1	Best practice sending ETH is to use call method	Informational
DECIM-1	Prevent external call to decimals in tokenDecimal assignment	Informational
ROOT-1	Merkle root can be updated	Informational

AMNT-1 | Zeroed msg.value used to update backUpAmount

Description

The depositNFT() function sends an external call to WETH using the msg.value from the call. However, _nftInformation.backupAmount is added to using the value from msg.value in the second branch of the if statement. In every case, the msg.value will be 0 as it has been sent to the WETH contract to wrap the ETH and update the address(this) balance.

If the value must be updated, then the msg.value will need to be cached before the call to the WETH contract.

Recommendation

Cache the value of msg.value to update backUpAmount in depositNFT().

OWNR-1 | Transferring Ownership should be two-step process

Description

If the transferOwnership transfer uses the wrong address or a zero address, then the owner role would be lost forever for the contract. The transfer ownership function should check that the input address is not a zero address, and the process should be two steps. The first step would be nominating an address to take ownership, and the second would allow the nominated accounts to call the acceptOwnership() function for the transfer to be a success. This will ensure that the account is active and valid that is accepting ownership of the contract.

Recommendation

Change the transferOwnership() function into a two step process with a transfer function and a separate accept function.

MAPP-1 | tokensPerWallet mapping initiated when ERC721A has a balance mapping

Description

The mapping tokensPerWallet is used to iterate upon when NFTs are minted via the contract. However, there is a mapping in the ERC721A standard which returns the number of tokens that a specified address holds. The added logic would cost extra gas when NFTs are being minted and is not required if the standard already accounts for token balances.

It is also worth noting that the tokenPerWallet mapping is not decremented when an NFT is burnt in the withdrawNFTBackUp() function. This should not however be an issue if the tokensPerWallet mapping is removed and the logic from ERC721A is used instead.

Recommendation

Remove the tokensPerWallet mapping and use the ERC721A balanceOf() function instead to query balances. Replace the logic throughout the contract where tokenPerWallet is used with balanceOf().

PASS-1 | Passing msg.value across functions in existing contract context

Description

The mintTokens() function takes a field for the msg.value when it's called from other functions within the contract. However, the msg.value will remain constant when calling functions within the context of the existing contract. This should allow the msg.value input field to be removed from mintTokens() and the msg.value global variable to be directly accessed for calculations.

Recommendation

Remove the _msgValue input field from _mintTokens() and use the global variable of msg.value for calculations within this function.

WRAP-1 | TransferERC20 after ETH wrap does nothing

Description

When ETH is wrapped via the WETH contract, the balance of the msg.sender is updated in the WETH contract. However, in revealNFT() an ERC20 transfer is called after ETH is wrapped, sending the backUpAmount to address(this). This function would effectively do nothing, as the WETH balance of address(this) is updated to backUpAmount when IERC20(WETH).deposit is called, as the msg.sender in an external contract call is changed to the contract rather than the original msg.sender.

Recommendation

Remove the transferERC20() logic from revealNFT() as the WETH balance for address(this) will be updated to the correct value when WETH.deposit() is called.

This logic also occurs in DepositNFT().

SWAP-1 | swapToken branch unused

Description

The second branch of logic would never be used in the swapToken() function. In this function, the tokenIn variable is set to WETH and the if statement checks if the tokenIn or tokenOut equal WETH in the first branch. This would mean this branch is always initiated and the logic never routes to the second branch.

Recommendation

Adjust the logic to either have a single branch or pass the tokenIn value to the swapToken function to ensure the second branch of logic is used.

STOR-1 | Storage Variables can be packed to save gas

Description

The storage variables for tokenPerWL, supplyCounter, and owner could be packed into a single slot. This would reduce the gas cost when storage is read from when a combination of these variables are read from cold storage. There is no risk of overflow for tokenperWL when using uint8, the MAX_SUPPLY cap at 2,000 allows uint16 for supplyCounter, and by default, the owner variable occupies 20 bytes meaning it would fit into the same slot.

Recommendation

Pack the three storage variables highlighted into a single slot by using different uint values.

```
uint8 public tokenPerWL = 3;
uint16 public supplyCounter;
```

STOR-2 | Storage Struct can be Packed

Description

The struct being used for NFTInformation can be packed into a smaller number of slots, which will save gas when reading and writing. The max value that is assigned to the backupAmount and backUpLimit will be 0.2 ether or 2e17 meaning a smaller uint assignment can be used such as uint64 (max value of 1.8e19). This would allow four variables to be packed into the same slot as revealed, tokenDecimal, and backup units can all fit into one 32 byte slot.

Recommendation

Change the type for backupLimit and backupAmount to uint64 and pack them into a single slot with revealed and tokenDecimal for the NFTInformation struct.

GAS-1 | Short require strings save gas

Description

Strings in solidity are handled in 32 byte chunks. If the strings used in require statements are longer than 32 bytes this will use more gas, which is why it would be recommended to use shortened strings that are less than 32 bytes. This can issue can be found on the following strings:

“you need to reveal your NFT first”

“you need to mint more than 0 tokens”

Recommendation

Shorten the strings referenced above to ensure the gas cost for these functions are reduced.

MESS-1 | Revert message may be unclear

Description

The revert message used in whitelistMint() when the totalSupply plus amount is more than MAX_SUPPLY may confuse the user. It currently suggests “all nfts are minted” whereas there are still NFTs that could be minted in theory but the current number is more than MAX_SUPPLY. It would be recommended to create a message that states the amount of NFTs being minted is more than the number available.

Recommendation

Adjust the message in the require statement to clearly state the error.

LOOP-1 | Setting iterated value to 1 in for loop

Description

By setting i to 1 in the for loop in _mintTokens(), extra gas is used to change the value of i from zero to a non-zero value without completing a loop. To save gas, initialize i as default (0) and iterate from 0 whilst i < _amount rather than <= amount.

In addition, as there is no chance of i overflowing in this transaction gas can be saved by iterating i in an unchecked block at the end of the loop.

Recommendation

Initialize i as 0 in the for loop and iterate whilst i is less than _amount.

Iterate on i an unchecked block at the end/ final step of the loop to save gas.

```
for (uint256 i; i < _amount; ) {
// Existing logic within loop

unchecked {
        i++;
}
}
```

METD-1 | Best practice sending ETH is to use call method

Description

The best practice approach to send native tokens ETH as per the Solidity docs is to use the call method. It would be recommended to adopt the suggested approach to follow best practices in the _mintTokens() function when sending ETH.

Recommendation

Use the best practice approach to send ETH in the _mintTokens() function:

```
(bool sent, bytes memory data) = to.call{value: msg.value}("");
require(sent, "TX_FAILED");
```

This logic should also be used when sending ETH in withdraw().

MAPP-1 | Prevent external call to decimals in tokenDecimal assignment

Description

All of the tokens being used for rewardCoins have 18 decimals excluding Bitcoin which has 8 decimals. The external call using the IERC20 interface to decimals() could be avoided by using an if statement checking the most common branch of logic first, which would be if the tokenBackUp address equals WBTC address (comments on most probable tokenBackup can be found in the additional information).

By checking if the address equals WBTC, the decimals can then be set as 6 in the case of WBTC or 18 in the else statement.

Recommendation

Remove the IERC20() call to decimals() in revealNFT() and replace it with an if/else statement checking if the tokenBackUp address equals the most probable outcome first.

```
if(_nftInformation.tokenBackUp == WBTCAddress) {
        _nftInformation.tokenDecimal = 8;
} else {
        _nftInformation.tokenDecimal = 18;
}
```

ROOT-1 | Merkle root can be updated

Description

If the merkle root is likely to stay the same then it would be recommended to set it either as an immutable variable on deployment or add a bool to check whether its value has been set when using setMerkleRoot. As the owner would be able to adjust the value of the merkle root an unlimited amount of times with the current logic, which undermines the use of a merkle root which is normally used as an immutable hash used for whitelisted.

Recommendation

Set the merkle root on deployment and remove setter logic if the root is known. Otherwise, add a bool state variable and require statement that ensures the merkle root has not been set for it to be changed.

Global security warnings

WeightRarity will assign Bitcoin to tokenBackup most frequently

The logic used in weightRarity adds the weight to the lowerBound of zero if the pseudoRandomNumber is neither more than lowerBound or less than weight. It's worth noting that as iterations start from 0 Bitcoin will be the first index used and this will mean that it is also the highest probable index returned from this function.

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.