WHITE HAT DAO

The IDOLS

Smart Contract Audit Report

theidols.io

By White Hat DAO

www.whitehatdao.com

Date: 23/02/2022

WHITE HAT DAO

Table of Contents

WHITE HAT DAO

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report.

In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and White Hat DAO and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives)

White Hat DAO owes no duty of care towards you or any other person, nor does White Hat DAOmake any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and White Hat DAOhereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report.

Except and only to the extent that it is prohibited by law, White Hat DAOhereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against White Hat DAO, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any

reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security.

No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

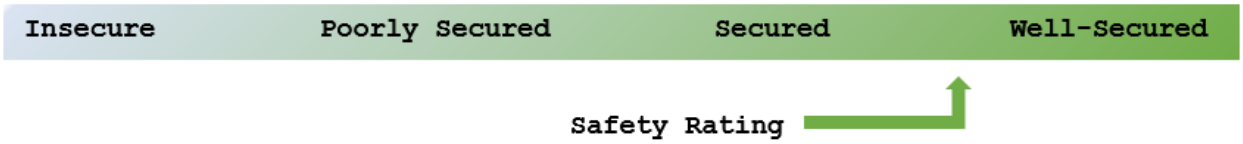
WHITE HAT DAO

Executive Summary

White Hat DAO was contracted by "The IDOLS" team to conduct a smart contract security audit. This report presents the findings of the security assessment conducted between Feb. 3, 2022 and Feb. 13, 2022. There were 10 smart contracts reviewed during this audit. The smart contracts were manually reviewed and analyzed with static analysis tools.

Based on our audit, the customers' smart contracts safety rating is shown below:

Safety Rating Bar

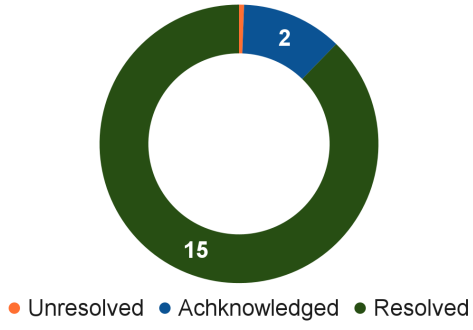


We found 4 major issues and 12 minor and informational issues. All the major issues fall under the "Centralization/Privilege" category. For a full list of these issues please refer to the Findings section of the report.

The code had good comments and documentation. The code uses the natspec standard for comments. Commenting can make the maintenance of the code much easier, as well as helping make finding bugs faster. Also, commenting is very important when writing functions that may be used in other contracts.

Here is a high-level overview of the issues found in this report:

Total Issues	16 (15 Resolved)
Critical Issues	0 (0 Resolved)
Major Issues	4 (2 Resolved)
Medium Issues	0 (0 Resolved)
Minor Issues	8 (8 Resolved)
Informational	4 (4 Resolved)



WHITE HAT DAO

Summary of Findings

The most prominent among our findings were major issues around the "Centralization/Privilege". There were also some minor/information vulnerabilities around "Gas Optimization" and "Coding Standards". Please review the recommendations around all vulnerabilities and remediate accordingly.

Issue ID	Issue Title	Category	Severity	Status
IDOL-01	Centralized unbound setter functions	Centralization/ Privilege	Major	Resolved
IDOL-02	Centralized pause function	Centralization/ Privilege	Major	Resolved
IDOL-03	Centralized swap, deposit and teardown functions	Centralization/ Privilege	Major	Acknowledged
IDOL-04	Single Point Of Failure	Centralization/ Privilege	Major	Acknowledged
IDOL-05	Variables could be declared as immutable	Gas Optimization	Minor	Resolved
IDOL-06	Shadowed Condition Check inside loop	Gas Optimization	Minor	Resolved
IDOL-07	Modifying storage value inside for loop	Gas Optimization	Minor	Resolved
IDOL-08	Modifying storage value inside for loop	Gas Optimization	Minor	Resolved

IDOL-09	Unused and redundant fields in Bid struct	Gas Optimization	Minor	Resolved
IDOL-10	Unused and redundant fields in List struct	Gas Optimization	Minor	Resolved
IDOL-11	Missing gas refund opportunity	Gas Optimization	Minor	Resolved
IDOL-12	Missing gas refund opportunity	Gas Optimization	Minor	Resolved
IDOL-13	Hardcoded Value	Coding Standards	Informational	Resolved
IDOL-14	Copy Storage to Memory Cost More	Gas Optimization	Informational	Resolved
IDOL-15	Intermediate variable can be avoided	Gas Optimization	Informational	Resolved
IDOL-16	Intermediate variable can be avoided	Gas Optimization	Informational	Resolved

WHITE HAT DAO

Introduction

This security assessment has been prepared for "The IDOLS" to find any safety concerns, bad practices and vulnerabilities in the source code as well as any contract dependencies in scope that were not part of an officially recognized library. Comprehensive tests have been conducted, utilizing manual code review, static analysis, and 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
- Thorough line-by-line manual review of the entire codebase by industry experts
- Reviewing unit tests to ensure full coverage of the codebase

The Project Summary, Scope, Audit Details and Methodology of the audit is described in the following sections.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards. These can be found in the Findings section of the report.

WHITE HAT DAO

Project Summary

Project	The idols
Description	The Idols is a collection of 10,000 generative portraits living on the Ethereum blockchain. Each Idol is unique and is generated from over a hundred hand-drawn assets. Some Idols will have features that are rarer than others, however, since all Idols will have an equal claim on the income generated from the Idol Treasury, all Idols will have an identical intrinsic value.
Website	https://theidols.io
Platform	Ethereum
Language used	Solidity
Codebase	https://github.com/idol-labs/the-idols-nft
Commit:	13bae6d6e49f8d186e19fa5b243f85f379b99903

Project Scope

White Hat DAO was commissioned by The IDOLS to perform security assessments on smart contracts as below:

Source Code	Acknowledgement	SHA-256
idolMain.sol	Accepted	E7D0D4EC008D24CEA74D7231FBF93565C7B55F2751A1110A867E20ADC815ED75
idolmarketplace.sol	Accepted	9399F8BC9C6F39CD8A98DBC1DCA7D592C8AEEBEDB75AD27889C1976182A69C0D
idolMintContract.sol	Accepted	9551D77397C464CEFC6B78B7BD509563BFBC165C9CF024A1A02465BF7014F314

VirtueRewards.sol	Accepted	83EAA54BEB8F887EA7DA9CE85FE2AFBAAECEBC106C02B06ADA0318218FA1CE86
VirtueStaking.sol	Accepted	83F4C984E7C814F5E182AD74FB1CC364E0030C691420954F7D2E31CA2584FFA1
VirtueToken.sol	Accepted	11983C76A1F12E2CF506AA69DAE330E4E141755D33AC05D6FC8B29B0EAAFE1A5
ICurvePool.sol	Accepted	0773524AE54D214CDA30C122CA5E51A52293BB717CC38432C55A42BABE3C989F
IIdolMain.sol	Accepted	751687EEC43D183070610B2F56D5EEBA03B07707948CBF95A3993DC17D0A5F4D
IIdolMarketplace.sol	Accepted	EBF7AD81948F0E1BEF258673ED7EEE03E695507DFF50F8C1B971FDC73417F287
IRewards.sol	Accepted	678FE716462FD92E25AA8C23222E3786DD2F2EBF4F713F46D9711E9EF1441621

Audit Details

Delivery Date	02/23/2022
Received Date	02/02/2022
Key Components	idolMain.sol, idolmarketplace.sol, idolMintContract.sol, VirtueRewards.sol, VirtueStaking.sol, VirtueToken.sol

Methodology

White Hat DAO auditing team reviewed the code base of "The IDOL" from Feb. 3, 2022 and Feb. 13, 2022. The team conducted the assessment based on the repository at commit [13bae6d6e49f8d186e19fa5b243f85f379b99903](13bae6d6e49f8d186e19fa5b243f85f379b99903).

The team launched the audit by analyzing the specifications of the project and the key areas of interest and went through the documentation.

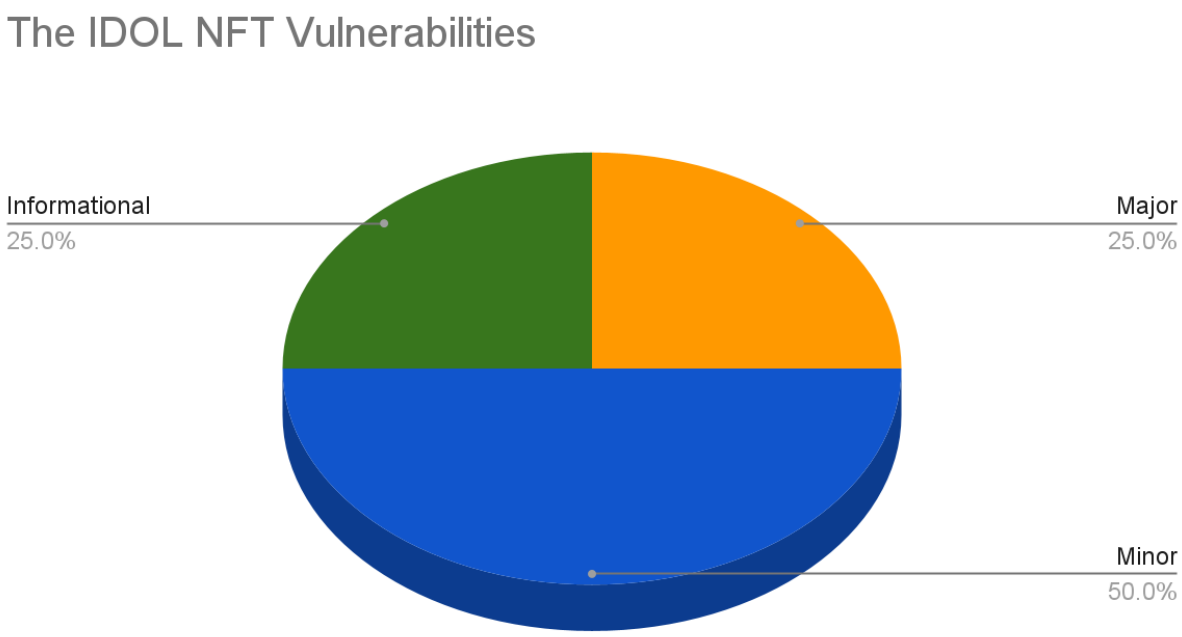
The code was manually reviewed in an attempt to identify potential vulnerabilities,The code has good unit tests coverage. We wrote some unit cases to test some edge cases. Automated analysis of the codebase was performed and results were reviewed.

The smart contracts were scanned for commonly known and more specific vulnerabilities. Following is the list of some of the vulnerabilities that were considered during the audit of the smart contract:

- Access Control
- Arbitrary token minting
- Business Logics Review
- Centralization of power
- Code clones, functionality duplication
- Conditional Completion attack
- Costly Loop
- Ownership Takeover
- Redundant fallback function
- Reentrancy
- Remote code execution
- User Balances manipulation
- Logic Flaws
- Scoping and Declarations
- Integer Overflow and Underflow attacks

Findings

We found 4 major issues and 12 minor and informational issues. All the major issues fall under the "Centralization/Privilege" category. There were also some minor/information vulnerabilities around "Gas Optimization" and "Coding Standards". Additional information on these vulnerabilities is provided in the following sections.



Critical - 0 | Major - 4 | Medium Issue - 0 | Minor - 8 | Informational -4

WHITE HAT DAO

Severity Definitions

Severity	Definitions
Critical	These vulnerabilities have a catastrophic impact on the security of the project. They can lead to loss, data manipulation, take over, etc. It is strongly recommended to fix these vulnerabilities.
Major	These vulnerabilities have a significant impact on the security of the project. They can lead to loss, data manipulation, take over, etc. It is strongly recommended to fix these vulnerabilities.
Medium	These vulnerabilities are important to fix. These vulnerabilities alone can't lead to asset loss or data manipulation. However, medium vulnerabilities can be chained to create a more severe vulnerability. It is highly recommended to review and address these vulnerabilities.
Minor	These vulnerabilities are mostly related to outdated, unused code snippets and don't have a significant impact on execution. It is suggested that the project party evaluate and consider whether these vulnerabilities need to be fixed.
Informational	These vulnerabilities don't pose an immediate risk but are relevant to security best practices. They could be code-style violations and informational statements that don't affect smart contract execution. They may be able to be ignored.

WHITE HAT DAO

Critical Vulnerabilities

No Critical severity vulnerabilities were found.

Major Vulnerabilities

IDOL-01 | Centralized unbound setter functions

Type: Centralization/Privilege
Level: Major

Description: These functions are called by the owner at any point in time with no restrictions, a misuse of these functions mid-sale or right before destruction will result in an unusable ecosystem.

Recommendation: Restrict the call to these functions to only before the sale start, once the sale has started these functions should never change.

Details:

File: `IdolMintContract.sol (130,146,319,329)`

```
function setIdolMainAddress(address _idolMainAddress)
  external
  onlyOwner
{
  idolMain = IIdolMain(_idolMainAddress);
  require(
    steth.approve(_idolMainAddress, 2**255-1),
    "Reverting because a call to steth.approve returned false."
  );
}
```

```
function setIdolMarketplaceAddress(address _idolMarketplaceAddress)
  external
  onlyOwner
```

```
  {
    idolMarketplace = IIdolMarketplace(_idolMarketplaceAddress);
  }
```

```
  function setBaseURI(string memory uri) external onlyOwner {
    idolMain.setBaseURI(uri);
  }
```

```
   function setExternalAddresses(
     address _virtueTokenAddr,
     address _idolMarketplaceAddr
    ) external onlyOwner {
      idolMain.setVirtueTokenAddr(_virtueTokenAddr);
      idolMain.setIdolMarketplaceAddr(_idolMarketplaceAddr);
      idolMarketplace.setVirtueTokenAddr(_virtueTokenAddr);
    }
```

Update: The IDOL team has acknowledged the findings and implemented WhiteHatDAO's recommendation of restricting these setter functions to before the start of the mint, see commit [b83c92264574f4c711bcc756f49de1b115df13c9](b83c92264574f4c711bcc756f49de1b115df13c9).

IDOL-02 | Centralized pause function

Type: Centralization/Privilege
Level: Major

Description: The current implementation of the pause function allows the owner to call `startPublicSale` again with new arguments that might alter the sale price and sale period.

Recommendation: A better approach to this is to setup a pause/unpause functionality that is completely independent of the `publicSaleActive` saving the current price snapshot and elapsed time to be restored on unpausing, in the case this function was intended as a stop only then a mechanism should be put in place so the owner cannot restart the sale by calling `startPublicSale` after pausing to change the parameters of the sale.

Details:

File: `IdolMintContract.sol (158)`

```
function pausePublicSale()
  external
  onlyOwner
  whenPublicSaleActive
{
  uint256 currentSalePrice = getMintPrice();
  uint256 elapsedTime = getElapsedSaleTime();
  publicSaleStartTime = 0;
  publicSaleActive = false;
  emit PublicSalePaused(currentSalePrice, elapsedTime);
}
```

Update: The IDOL team has acknowledged the findings and implemented WhiteHatDAO's recommendation of modifying the pause function that is independent of publicSaleActive, see commit 68b4f91965238b73006d0b872c1120d3ad9d028e

IDOL-03 | Centralized swap, deposit and teardown

Type: Centralization/Privilege
Level: Major

Description: The whole project relies on these two steps (swap and transfer to IdolMain) the owner has total control over calling or not colling these functions, not calling this function before `self-destruct` can lead to all funds being transferred to the owner.

Recommendation: We suggest adding more strict checks on these functions to ensure the sale revenues were used for the purpose stated in the Idols whitepaper, and to be in line with the project's promises of permissionless and trustless.

Details:

File: `IdolMintContract.sol(291,302)`

```
  function swap(uint _slippageBps)
    onlyOwner
    external
    returns(uint result)
  {
    return stethPool.exchange{ value: address(this).balance }(0, 1, address(this).balance,
address(this).balance * _slippageBps / 10000);
  }
```

```
  function depositStethIdolMain()
    onlyOwner
    external
  {
    idolMain.depositSteth(steth.balanceOf(address(this)));
  }
```

File: `IdolMintContract.sol (344)`

```
  function tearDown() external onlyOwner {
    selfdestruct(payable(owner()));
  }
```

Update: The IDOL team has acknowledged WhiteHatDAO's findings around the tearDown function. The team believes that retaining the functionality as-is is important in case an unforeseen emergency were to happen during the mint event. Please note that once the mint is completed and the funds transferred to IdolMain, this centralization aspect goes away.

IDOL-04 | Single Point Of Failure

Type: Centralization/Privilege
Level: Major

Description: The whole project is designed around one level of access control (owner), the owner has all the privileged calls if the owner's wallet gets compromised an attacker can take over the whole project.

Recommendation: We strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets

Here are some feasible:

1. Assignment of privileged roles (owner) to multi-signature wallets to prevent a single point of failure due to the private key
2. Time-lock with reasonable latency, e.g., 48 hours, for awareness of privileged operations.
3. Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Details:

File: `All files`

Update: The IDOL team has acknowledged this finding. The team would like to indicate that it was always our intention to use a Gnosis Safe multisig to govern the contracts.

Medium Vulnerabilities

No Medium severity vulnerabilities were found.

Minor Vulnerabilities

IDOL-05 | Variables could be declared as immutable

Type: Gas Optimization
Level: Minor

Description: Variables are declared as storage values while they are never modified after constructor execution, future reads of this value in the contract code will load the value from storage costing extra gas.

Recommendation: Declare as immutable [ref].

Details:

```
IdolMain.sol: (20 & 26)
  ERC20 public steth;

  address public mintContractAddress;

IdolMintContract.sol: (65, 72 & 73)
  uint256 public godIdOffset;

  ICurvePool public stethPool;
  ERC20 public steth;

IdolMarketplace.sol: (60)
  IdolMain public idolMain;

VirtueToken.sol: (18,19)
  address public idolMainAddress;
  address public idolMarketAddress;

VirtueStaking.sol: (18)
  address public mintContractAddress;

VirtueRewards.sol: (21,25,30)
  IERC20 public stakingToken;

  IERC20 public rewardsToken;

  VirtueStaking public virtueStakingContract;
```

Update: The IDOL team has acknowledged the findings and implemented WhiteHatDAO's recommendation in commit a05b65b2195981c520c6107fd2011a22ecca0105

IDOL-06 | Shadowed Condition Check inside loop

Type: Gas Optimization
Level: Minor

Description: The condition `publicGodsMinted < MAX_GODS_TO_MINT` will always be true because the `require` statement shadows it.

Recommendation: Remove the redundant condition check.

Details:

File: `IdolMintContract.sol (248)`

```
    require(
      publicGodsMinted + _numGodsToMint <= MAX_GODS_TO_MINT -
NUM_RESERVED_NFTS,
      "Minting would exceed max supply"
    );

// then later inside the loop
    for (uint256 i = 0; i < _numGodsToMint; i++) {
      if (publicGodsMinted < MAX_GODS_TO_MINT) {
```

Update: The IDOL team has acknowledged the findings and implemented WhiteHatDAO's recommendation in commit a05b65b2195981c520c6107fd2011a22ecca0105

IDOL-07 | Modifying storage value inside for loop

Type: Gas Optimization
Level: Minor

Description: storage is an expensive memory and incrementing `publicGodsMinted` on each iteration will result in high gas fees (`SLOAD` + `SSTORE` per iteration) and serving no purpose (*see IDOL-08 above for the shadowed condition check*), and the index `i` can be used to generate the `idToMint`.

Recommendation: Only increment the `publicGodsMinted` by `_numGodsToMint` after the for loop, and calculating the `idToMint` using `i` (see below)

```
    for (uint256 i = 0; i < _numGodsToMint; i++) {
        uint idToMint = (publicGodsMinted + i + godIdOffset) % MAX_GODS_TO_MINT;
        idolMain.mint(msg.sender, idToMint, false);
    }
    publicGodsMinted += _numGodsToMint;
```

Details:

File: `IdolMintContract.sol (251)`

```
    for (uint256 i = 0; i < _numGodsToMint; i++) {
      if (publicGodsMinted < MAX_GODS_TO_MINT) {
        uint idToMint = (publicGodsMinted + godIdOffset) % MAX_GODS_TO_MINT;
        idolMain.mint(msg.sender, idToMint, false);
        publicGodsMinted++; // modifying storage value on each iteration
      }
    }
```

Update: The IDOL team has acknowledged the findings and implemented WhiteHatDAO's recommendation in commit [a05b65b2195981c520c6107fd2011a22ecca0105]

IDOL-08 | Modifying storage value inside for loop

Type: Gas Optimization
Level: Minor

Description: storage is an expensive memory and incrementing `reservedGodsMinted` on each iteration will result in high gas fees (1 `SLOAD` +1 `SSTORE` per iteration).

Recommendation: Only increment the `reservedGodsMinted` by `_numGodsToMint` after the for a loop, also consider using the index `i` to calculate the `idToMint` (see bellow)

```
    for (uint256 i = 0; i < _numGodsToMint; i++) {
        uint idToMint = (publicGodsMinted + i + godIdOffset) % MAX_GODS_TO_MINT;
        idolMain.mint(msg.sender, idToMint, false);
    }
    publicGodsMinted += _numGodsToMint;
```

Details:

```
for (uint256 i = 0; i < _numGodsToMint; i++) {
  uint idToMint = (reservedGodsMinted + MAX_GODS_TO_MINT - NUM_RESERVED_NFTS +
                  godIdOffset) %  MAX_GODS_TO_MINT;

  idolMain.mint(_mintAddress, idToMint, _lock);
  reservedGodsMinted++;
}
```

Update: The IDOL team has acknowledged the findings and implemented WhiteHatDAO's recommendation in commit [a05b65b2195981c520c6107fd2011a22ecca0105]

IDOL-09 | Unused and redundant fields in Bid struct

Type: Gas Optimization
Level: Minor

Description:

1. `hasBid` is redundant. Its use can be avoided altogether by checking if the bid value is greater than zero.
2. `godIndex` is redundant as the Bid is stored in a `godId => Bid mapping` the id is always known, also nowhere in the contract code is this field accessed.

Recommendation: We suggest removing these two fields.

Details:

File: `IdolMarketplace.sol (36)`

```
  // Bid is a struct holding metadata for when a user bids on a God they are
interested in buying.
  struct Bid {
    // hasBid specifies whether there is an active Bid for the God.
    bool hasBid;

    // godIndex specifies the god ID that is being bid on.
    uint godIndex;

    // bidder indicates the address of who posted the God Bid.
```

```
    address bidder;

    // value is the amount (in wei) that the Bid is offering for the God.
    uint value;
  }
```

Update: The IDOL team has acknowledged the findings and implemented WhiteHatDAO's recommendation in commit [a05b65b2195981c520c6107fd2011a22ecca0105]

IDOL-10 - Unused and redundant fields in Listing struct

Type: Gas Optimization
Level: Minor

Description:

1. `isForSale` can be removed and replaced by a simple check `minValue > 0`. This will also fix the potential accidental `minValue=0` by user or software mistake that could lead to users giving away their NFT for free.
2. `godIndex` is redundant as the Listings are stored in a `godId => Listing mapping` the id is always known, also nowhere in the contract code is this field accessed.

Recommendation: We suggest removing these two fields, and making the necessary changes required when removing `isForSale` on line 118.

Details:

File: `IdolMarketplace.sol (36)`

```
  struct Listing {
    // isForSale specifies whether there is an active listing for the God.
    bool isForSale;

    // godIndex specifies the god ID that is for sale.
    uint godIndex;

    // seller indicates the address of who posted the God Listing.
    address seller;
```

```
    // minValue specifies the minimum value that the owner will accept when buying
through the
    // buyGod function.
    uint minValue;

    // onlySellTo is an optional field that specifies that the God can only be sold
to a specific
    // address. Uses the 0x0 address if no specific address is specified.
    address onlySellTo;
  }
```

Update: The IDOL team has acknowledged the findings and implemented WhiteHatDAO's recommendation in commit [a05b65b2195981c520c6107fd2011a22ecca0105]

IDOL-11 | Missing gas refund opportunity

Type: Gas Optimization
Level: Minor

Description: Zeroing the `godListings[_godId]` will be more beneficial in this case and will result in a gas refund for zeroing a non-zero field.

Recommendation: We suggest zeroing the field combined with the suggestions from (IDOL-10) see (solc docs).

```
  function _removeGodListing(uint _godId) private {
    delete godListings[_godId];
    emit GodUnlisted(_godId);
  }
```

Details:

File: `IdolMarketplace.sol (106)`

```
  function _removeGodListing(uint _godId) private {
    godListings[_godId] = Listing(false, _godId, msg.sender, 0, address(0x0));
    emit GodUnlisted(_godId);
```

```
}
```

Update: The IDOL team has acknowledged the findings and implemented WhiteHatDAO's recommendation in commit a05b65b2195981c520c6107fd2011a22ecca0105

IDOL-12 | Missing gas refund opportunity

Type: Gas Optimization
Level: Minor

Description: Zeroing the `godListings[_godId]` and `godBids[_godId]` will be more beneficial in this case and will result in a gas refund for zeroing a non-zero field.

Recommendation: We suggest zeroing the field using `delete` combined with the suggestions from (IDOL-09) see (solc docs).

Details:

File: `IdolMarketplace.sol (192,196,215)`

```
    godListings[_godId] = Listing(false, _godId, existingBid.bidder, 0,
address(0x0));

    godBids[_godId] = Bid(false, _godId, address(0x0), 0);

    godBids[_godId] = Bid(false, _godId, address(0x0), 0);
```

Update: The IDOL team has acknowledged the findings and implemented WhiteHatDAO's recommendation in commit a05b65b2195981c520c6107fd2011a22ecca0105

Informational Vulnerabilities

IDOL-13 | Hardcoded Value

Type: Coding Standards
Level: Informational

Description: The value 10000 is hardcoded in this statement replacing it with the already declared constant `MAX_GODS_TO_MINT` is more appropriate.

Recommendation: Replacing the hardcoded value with the already declared constant `MAX_GODS_TO_MINT`.

Details:

```
IdolMintContract.sol: (line 97)
godIdOffset = uint256(keccak256(abi.encodePacked(_metadataHash,
_discordGeneratedHash))) % 10000;
```

Update: The IDOL team has acknowledged the findings and implemented WhiteHatDAO's recommendation in commit a05b65b2195981c520c6107fd2011a22ecca0105

IDOL-14 - Copy Storage to Memory Cost More

Type: Gas Optimization
Level: Informational

Description: Loading `bid` into memory will load 4 fields into memory paying gas for reading from storage and for allocating memory, but `bid` is only read twice so in this case reading from storage would have cost 2 `SLOAD` instructions while copying to memory cost 4 `SLOAD`, 1 `MSTORE`, and 2 `MLOAD` instructions, as a rule of thumb copying to memory is only beneficial if the reads are more than the required reads to copy storage to memory.

Recommendation: Do not copy storage to memory.

Details:

File: `IdolMarketplace.sol (138)`

```
    Bid memory bid = godBids[_godId];
    if (bid.bidder == msg.sender) {
```

```
    // Kill bid and refund value
    godBids[_godId] = Bid(false, _godId, address(0x0), 0);
    pendingWithdrawals[msg.sender] += bid.value;
  }
```

Update: The IDOL team accepted and applied our recommendations in commit [a05b65b2195981c520c6107fd2011a22ecca0105]

IDOL-15 | Intermediate variable can be avoided

Type: Gas Optimization
Level: Informational

Description: Intermediate variable `amount` can be avoided by reordering the statements, this is safe because of the function being `nonReentrant`.

Recommendation: Simplify the function. See below:

```
function withdrawPendingFunds() external nonReentrant {
    Address.sendValue(payable(msg.sender), pendingWithdrawals[msg.sender]);
    delete pendingWithdrawals[msg.sender];
}
```

Details:

File: `IdolMarketplace.sol (138)`

```
function withdrawPendingFunds() external nonReentrant {
    uint amount = pendingWithdrawals[msg.sender];
    pendingWithdrawals[msg.sender] = 0;
    Address.sendValue(payable(msg.sender), amount);
}
```

Update: The IDOL team has acknowledged the findings and implemented WhiteHatDAO's recommendation in commit [a05b65b2195981c520c6107fd2011a22ecca0105]

IDOL-16 | Intermediate variable can be avoided

Type: Gas Optimization
Level: Informational

Description: Intermediate variable `amount` can be avoided by reordering the statements, this is safe because of the function being `nonReentrant`.

Recommendation: Simplify the function. See below:

```
  function withdrawBidForGod(uint _godId) external nonReentrant {
    Bid memory existingBid = godBids[_godId];
    require(existingBid.bidder == msg.sender, "Cannot withdraw a bid not made by
the sender.");

    emit GodBidWithdrawn(_godId, existingBid.value, msg.sender);

    Address.sendValue(payable(msg.sender), existingBid.value);

    delete godBids[_godId];
  }
```

Details:

File: `IdolMarketplace.sol (214)`

```
  function withdrawBidForGod(uint _godId) external nonReentrant {
    Bid memory existingBid = godBids[_godId];
    require(existingBid.bidder == msg.sender, "Cannot withdraw a bid not made by
the sender.");

    emit GodBidWithdrawn(_godId, existingBid.value, msg.sender);
    uint amount = existingBid.value;
    godBids[_godId] = Bid(false, _godId, address(0x0), 0);

    Address.sendValue(payable(msg.sender), amount);
  }
```

Update: The IDOL team has acknowledged the findings and implemented WhiteHatDAO's recommendation in commit [a05b65b2195981c520c6107fd2011a22ecca0105]

Conclusion

White Hat DAO has worked with "The IDOLS" team to perform this audit. There were 10 smart contracts reviewed during this audit. The smart contracts were manually reviewed and analyzed with static analysis tools. The findings of these reviews were provided in this report.

The Code had good unit tests coverage for all functionalities. We constructed some unit tests to test edge cases. We were unable to fully determine the integrity of the code and the logic flow.

The code was commented well. Comments are helpful in understanding the overall architecture and the logic flow of the contracts.

This audit has found 4 major, 8 minor, and 4 informational vulnerabilities.

Update: 02/22/22 - All issues have been resolved except for issue IDOL-03 and IDOL-04. The Idols Team has Acknowledged IDOL-03 and IDOL-04. Please refer to the findings section to see IDOL Team's justification.

WHITE HAT DAO

Change Log

- 2021-02-11 - Initial report
- 2021-02-13 - Draft v0.1 completed
- 2021-02-14 - Draft updated, v0.2
- 2021-02-22 - Final Report updated with client's response and updates.