PoolTogether - Pods

Date	March 2021

1 Executive Summary

This report presents the results of our engagement with PoolTogether to review the Pods V3 contracts.

The review was conducted by Sergii Kravchenko and Nicholas Ward over the course of ten person-days between March 29th and April 2nd, 2021.

2 Scope

Our review focused on commit hash `879dc8b911fc506dd6bead1f36eade919ccfea57` and was limited to the `Pod` and `TokenDrop` contracts along with their respective factory contracts. The list of files in scope can be found in the Appendix.

3 Findings

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be

addressed.

- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

3.1 Winning pods can be frontrun with large deposits Critical

Description

`Pod.depositTo()` grants users shares of the pod pool in exchange for `tokenAmount` of `token`.

code/pods-v3-contracts/contracts/Pod.sol:L266-L288

```
function depositTo(address to, uint256 tokenAmount)
    external
    override
    returns (uint256)
{
    require(tokenAmount > 0, "Pod:invalid-amount");

    // Allocate Shares from Deposit To Amount
    uint256 shares = _deposit(to, tokenAmount);

    // Transfer Token Transfer Message Sender
    IERC20Upgradeable(token).transferFrom(
        msg.sender,
        address(this),
        tokenAmount
    );

    // Emit Deposited
    emit Deposited(to, tokenAmount, shares);

    // Return Shares Minted
    return shares;
}
```

The winner of a prize pool is typically determined by an off-chain random number generator, which requires a request to first be made on-chain. The result of this RNG request can be seen in the mempool and frontrun. In this case, an attacker could identify a winning `Pod` contract and make a large deposit, diluting existing user shares and claiming the entire prize.

Recommendation

The modifier `pauseDepositsDuringAwarding` is included in the `Pod` contract but is unused.

code/pods-v3-contracts/contracts/Pod.sol:L142-L148

```
modifier pauseDepositsDuringAwarding() {
    require(
        !IPrizeStrategyMinimal(_prizePool.prizeStrategy()).isRngRequested(),
        "Cannot deposit while prize is being awarded"
    );
    _;
}
```

Add this modifier to the `depositTo()` function along with corresponding test cases.

3.2 Token transfers may return `false` Critical

Description

There are a lot of token transfers in the code, and most of them are just calling `transfer` or `transferFrom` without checking the return value. Ideally, due to the ERC-20 token standard, these functions should always return `True` or `False` (or revert). If a token returns `False`, the code will process the transfer as if it succeeds.

Recommendation

Use the `safeTransfer` and the `safeTransferFrom` versions of transfers from OZ.

3.3 `TokenDrop`: Unprotected `initialize()` function Critical

Description

The `TokenDrop.initialize()` function is unprotected and can be called multiple times.

code/pods-v3-contracts/contracts/TokenDrop.sol:L81-L87

```
function initialize(address _measure, address _asset) external {
    measure = IERC20Upgradeable(_measure);
    asset = IERC20Upgradeable(_asset);

    // Set Factory Deployer
    factory = msg.sender;
}
```

Among other attacks, this would allow an attacker to re-initialize any `TokenDrop` with the same `asset` and a malicious `measure` token. By manipulating the balance of a user in this malicious `measure` token, the entire `asset` token balance of the `TokenDrop` contract could be drained.

Recommendation

Add the `initializer` modifier to the `initialize()` function and include an explicit test that *every* initialization function in the system can be called once and only once.

3.4 Pod: Re-entrancy during deposit or withdrawal can lead to stealing funds Critical

Description

During the deposit, the token transfer is made after the Pod shares are minted:

code/pods-v3-contracts/contracts/Pod.sol:L274-L281

```
uint256 shares = _deposit(to, tokenAmount);

// Transfer Token Transfer Message Sender
IERC20Upgradeable(token).transferFrom(
    msg.sender,
    address(this),
    tokenAmount
);
```

That means that if the `token` allows re-entrancy, the attacker can deposit one more time inside the token transfer. If that happens, the second call will mint more tokens than it is supposed to, because the first token transfer will still not be finished. By doing so with big amounts, it's possible to drain the pod.

Recommendation

Add re-entrancy guard to the external functions.

3.5 TokenDrop: Re-entrancy in the `claim` function can cause to draining funds Major

Description

If the `asset` token is making a call before the transfer to the `receiver` or to any other 3-d party contract (like it's happening in the `Pod` token using the `_beforeTokenTransfer` function), the attacker can call the `drop` function inside the `transfer` call here:

code/pods-v3-contracts/contracts/TokenDrop.sol:L139-L153

```
function claim(address user) external returns (uint256) {
    drop();
    _captureNewTokensForUser(user);
    uint256 balance = userStates[user].balance;
    userStates[user].balance = 0;
    totalUnclaimed = uint256(totalUnclaimed).sub(balance).toUint112();

    // Transfer asset/reward token to user
    asset.transfer(user, balance);

    // Emit Claimed
    emit Claimed(user, balance);

    return balance;
}
```

Because the `totalUnclaimed` is already changed, but the current balance is not, the `drop` function will consider the funds from the unfinished transfer as the new tokens. These tokens will be virtually redistributed to everyone.

After that, the transfer will still happen, and further calls of the `drop()` function will fail because the following line will revert:

```
uint256 newTokens = assetTotalSupply.sub(totalUnclaimed);
```

That also means that any transfers of the `Pod` token will fail because they all are calling the `drop` function. The `TokenDrop` will "unfreeze" only if someone transfers enough tokens to the `TokenDrop` contract.

The severity of this issue is hard to evaluate because, at the moment, there's not a lot of tokens that allow this kind of re-entrancy.

Recommendation

Simply adding re-entrancy guard to the `drop` and the `claim` function won't help because the `drop` function is called from the `claim`. For that, the transfer can be moved to a separate function, and this function can have the re-entrancy guard as well as the `drop` function.

Also, it's better to make sure that `_beforeTokenTransfer` will not revert to prevent the token from being frozen.

3.6 Pod: Having multiple token drops is inconsistent Medium

Description

The `Pod` contract had the `drop` storage field and mapping of different `TokenDrop`s `(token => TokenDrop)`. When adding a new `TokenDrop` in the mapping, the `drop` field is also changed to the added `_tokenDrop`:

code/pods-v3-contracts/contracts/Pod.sol:L455-L477

```
function setTokenDrop(address _token, address _tokenDrop)
    external
    returns (bool)
{
    require(
        msg.sender == factory || msg.sender == owner(),
        "Pod:unauthorized-set-token-drop"
    );

    // Check if target<>tokenDrop mapping exists
    require(
        drops[_token] == TokenDrop(0),
        "Pod:target-tokendrop-mapping-exists"
    );

    // Set TokenDrop Referance
    drop = TokenDrop(_tokenDrop);

    // Set target<>tokenDrop mapping
    drops[_token] = drop;

    return true;
}
```

On the other hand, the `measure` token and the `asset` token of the `drop` are strictly defined by the Pod contract. They cannot be changed, so all `TokenDrop`s are supposed to have the same `asset` and `measure` tokens. So it is useless to have different `TokenDrops`.

Recommendation

The mapping seems to be unused, and only one `TokenDrop` will normally be in the system. If that code is not used, it should be deleted.

3.7 Pod: Fees are not limited by a user during the withdrawal Medium

Description

When withdrawing from the Pod, the shares are burned, and the deposit is removed from the Pod. If there are not enough deposit tokens in the contract, the remaining tokens are withdrawn from the pool contract:

code/pods-v3-contracts/contracts/Pod.sol:L523-L532

```
if (amount > currentBalance) {
    // Calculate Withdrawl Amount
    uint256 _withdraw = amount.sub(currentBalance);

    // Withdraw from Prize Pool
    uint256 exitFee = _withdrawFromPool(_withdraw);

    // Add Exit Fee to Withdrawl Amount
    amount = amount.sub(exitFee);
}
```

These tokens are withdrawn with a fee from the pool, which is not controlled or limited by the user.

Recommendation

Allow users to pass a `maxFee` parameter to control fees.

3.8 `ProxyFactory.deployMinimal()` does not check for contract creation failure Minor

Description

The function `ProxyFactory.deployMinimal()` is used by both the `PodFactory` and the `TokenDropFactory` to deploy minimal proxy contracts. This function uses inline assembly to inline a target address into the minimal proxy and deploys the resulting bytecode. It then emits an event containing the resulting address and optionally makes a low-level call to the resulting address with user-provided data.

The result of a `create()` operation in assembly will be the zero address in the event that a revert or an exceptional halting state is encountered during contract creation. If execution of the contract initialization code succeeds but returns no runtime bytecode, it

is also possible for the `create()` operation to return a nonzero address that contains no code.

code/pods-v3-contracts/contracts/external/ProxyFactory.sol:L9-L35

```
function deployMinimal(address _logic, bytes memory _data)
    public
    returns (address proxy)
{
    // Adapted from https://github.com/optionality/clone-factory/blob/32782f82dfc5a00d103a
    bytes20 targetBytes = bytes20(_logic);
    assembly {
        let clone := mload(0x40)
        mstore(
            clone,
            0x3d602d80600a3d3981f3363d3d373d3d3d363d730000000000000000000000000000
        )
        mstore(add(clone, 0x14), targetBytes)
        mstore(
            add(clone, 0x28),
            0x5af43d82803e903d91602b57fd5bf3000000000000000000000000000000000000
        )
        proxy := create(0, clone, 0x37)
    }

    emit ProxyCreated(address(proxy));

    if (_data.length > 0) {
        (bool success, ) = proxy.call(_data);
        require(success, "ProxyFactory/constructor-call-failed");
    }
}
```

Recommendation

At a minimum, add a check that the resulting proxy address is nonzero before emitting the `ProxyCreated` event and performing the low-level call. Consider also checking the `extcodesize` of the proxy address is greater than zero.

Also note that the bytecode in the deployed “Clone” contract was not reviewed due to time constraints.

3.9 `Pod.setManager()` checks validity of wrong address Minor

Description

The function `Pod.setManager()` allows the `owner` of the Pod contract to change the Pod's `manager`. It checks that the value of the existing `manager` in storage is nonzero. This is presumably intended to ensure that the `owner` has provided a valid `newManager` parameter in calldata.

The current check will always pass once the contract is initialized with a nonzero `manager`. But, the contract can currently be initialized with a manager of `IPodManager(address(0))`. In this case, the check would prevent the `manager` from ever being updated.

code/pods-v3-contracts/contracts/Pod.sol:L233-L240

```
function setManager(IPodManager newManager)
    public
    virtual
    onlyOwner
    returns (bool)
{
    // Require Valid Address
    require(address(manager) != address(0), "Pod:invalid-manager-address");
```

Recommendation

Change the check to:

```
require(address(newManager) != address(0), "Pod:invalid-manager-address");
```

More generally, attempt to define validity criteria for all input values that are as strict as possible. Consider preventing zero inputs or inputs that might conflict with other addresses in the smart contract system altogether, including in contract initialization functions.

4 Recommendations

4.1 Rename `Withdrawl` event to `Withdrawal`

Description

The `Pod` contract contains an event `Withdrawl(address, uint256, uint256)`:

code/pods-v3-contracts/contracts/Pod.sol:L76-L79

```
/**
 * @dev Emitted when user withdraws
 */
event Withdrawl(address user, uint256 amount, uint256 shares);
```

This appears to be a misspelling of the word `Withdrawal`. This is of course not a problem given it's consistent use, but could cause confusion for users or issues in future contract updates.

Appendix 1 - Files in Scope

File	SHA-1 hash
Pod.sol	641689b5f218fca0efdb5bbdd341188b28330d06
PodFactory.sol	222481a98d4e43cb7ecea718c9c128fac1e0ac57
TokenDrop.sol	ab9713b77031662e16ce9e4b6b7766b1d2f6ff44
TokenDropFactory.sol	eab23cdc4b779bb062de96a2a4dba0973556a895

Appendix 2 - Disclosure

ConsenSys Diligence ("CD") typically receives compensation from one or more clients (the "Clients") for performing the analysis contained in these reports (the "Reports"). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of

this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself 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 Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

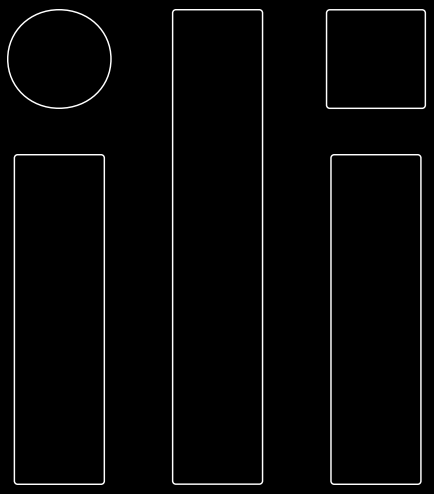
LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.

Request a Security Review Today

Get in touch with our team to request a quote for a smart contract audit.

CONTACT US



AUDITS

FUZZING

SCRIBBLE

BLOG

TOOLS

RESEARCH

ABOUT

CONTACT

CAREERS

PRIVACY POLICY

Subscribe to Our Newsletter

Stay up-to-date on our latest offerings, tools, and the world of blockchain security.

Email*

e-mail address

→