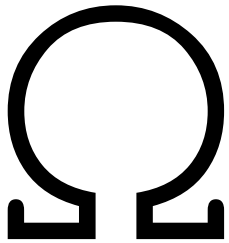
EnterDAO Landworks Audit

Final Report

December 21, 2021



Team Omega

Summary

EnterDAO has asked Team Omega to audit the contracts that define the behavior of the Landworks contracts. On the whole,

We found one high severity issue - these are issues that can lead to a loss of funds, and are essential to fix.

We classified 3 issues as “medium” - these are issues we believe you should definitely address.

In addition, 13 issues were classified as “low”, and 17 issues were classified as “info” - we believe the code would improve if these issues were addressed as well.

Resolution

EnterDAO subsequently addressed the issues in

https://github.com/EnterDAO/LandWorks-protocol/commit/2bdd45130fd2a9eb0cc571b5dd6ee9dfea5d0aa0

and

https://github.com/EnterDAO/LandWorks-YF-Contracts/commit/812a53f00a09a05b6b83fd81320c7403500dfd40

We reviewed the changes, and described the resolution of each issue below. All issues of “high” or “medium” severity were addressed, and all but one of the “low severity” issues were addressed.

Scope of the Audit

The audit concerns Solidity code from two repositories:

https://github.com/EnterDAO/LandWorks-protocol and
https://github.com/EnterDAO/LandWorks-YF-Contracts

From the LandWorks-protocol, we reviewed the contract at commit d9b3a0845e77959e5184095947b7887b908e2066, and specifically the following contracts:

./facets/decentraland/DecentralandFacet.sol
./facets/FeeFacet.sol

./facets/ERC721Facet.sol
./facets/MarketplaceFacet.sol
./libraries/LibTransfer.sol
./libraries/LibFee.sol
./libraries/marketplace/LibRent.sol
./libraries/marketplace/LibMarketplace.sol
./libraries/marketplace/LibDecentraland.sol
./libraries/LibOwnership.sol
./libraries/LibERC721.sol
./LandWorks.sol
./interfaces/decentraland/IDecentralandFacet.sol
./interfaces/decentraland/IDecentralandRegistry.sol
./interfaces/IERC721Consumable.sol
./interfaces/IFeeFacet.sol
./interfaces/IMarketplaceFacet.sol
./interfaces/IERC721Facet.sol
./interfaces/IERC173.sol

A number of contracts are adapted from one of the reference implementations of the EIP-2535 Diamond standard - https://github.com/mudgen/diamond-3-hardhat. For the following contracts, we limited our audit to the changes from the reference implementation.

./facets/DiamondCutFacet.sol
./facets/OwnershipFacet.sol
./facets/DiamondLoupeFacet.sol
./libraries/LibDiamond.sol
./interfaces/IDiamondCut.sol
./interfaces/IDiamondLoupe.sol

From the LandWorks-YF-Contracts we reviewed commit 578c6ee82fc244370e5fc600367f2badacf51fac and specifically the following contracts:

./LandWorksDecentralandStaking.sol
./interfaces/IERC721Consumable.sol
./interfaces/IDecentralandEstateRegistry.sol
./interfaces/ILandWorks.sol

We did not audit dependencies.

Methods Used

Code Review

We manually inspected the source code to identify potential security flaws.

The contracts were compiled, deployed, and tested in a test environment.

Automatic analysis

We have used automated analysis tools, including MythX and Slither to detect common potential vulnerabilities. No (true) high severity issues were identified with the automated processes. Some low severity issues, concerning mostly the solidity version setting and functions visibility, were found and we have included them below in the appropriate parts of the report.

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

Severity definitions

High	Vulnerabilities that can lead to loss of assets or data manipulations.
Medium	Vulnerabilities that are essential to fix, but that do not lead to assets loss or data manipulations
Low	Issues that do not represent direct exploit, such as poor implementations, deviations from best practice, high gas costs, etc
Info	Matters of opinion

Findings

General issues regarding Landworks-protocol

G1. Licensing violation [medium] [resolved]

The DiamondCut files are copied/adapted from the reference implementation at https://github.com/mudgen/diamond-3-hardhat and are published under the MIT license. You are free to republish the files under MIT, but you must, as the LICENSE file says, maintain the original copyright statements:

> "Copyright (c) 2021 Nick Mudge
> [...]
> The above copyright notice and this permission notice shall be included in all
> copies or substantial portions of the Software."

You violate this license in the files DiamondCutFacet.sol, DiamondLoupeFacet.sol and to a lesser degree in Landworks.sol which takes the fallback function from mudgen's Diamond.sol code, because you use "substantial portions of the software" but have not included the copyright notice.
Recommendation: Leave the original copyright attribution intact in those files that were copied and only slightly modified. Acknowledge the re-use of the code in the main LICENSE file and make clear that the copyright claimed there applies only to the code written by you.
Severity: Medium
Resolution: The issue was resolved

G2. Documentation is outdated [info] [resolved]

The description in the whitepaper of the protocol functions is outdated - for example, "add" should be "list" and "remove" should be "delist," etc.
Recommendation: Update the whitepaper.md file (or the code).
Severity: Info
Resolution: The documentation was updated.

G3. Vulnerabilities in package dependencies [low] [resolved]

Npm audit reports:

> 123 vulnerabilities (25 low, 52 moderate, 41 high, 5 critical)

One of the vulnerabilities refers to a security advisory that regards version 4.3.2 of the OpenZeppelin contracts (https://github.com/advisories/GHSA-wmpv-c2jp-j2xg). This vulnerability applies to the ERC1155Supply extension, which is not used in the contracts, and so does not pose a direct threat.
The other vulnerabilities reported by npm audit do not concern the solidity code or the solidity compiler directly, and so pose no direct threat either.
Recommendation: As a best practice, we recommend to run “npm audit fix” to automatically fix those vulnerabilities that can be fixed automatically, and specifically to upgrade the OpenZeppelin dependency to the latest version - 4.4.0 - or in any case to the patched version 4.3.3
Severity: Low
Resolution: This issue was resolved. Open Zeppelin has been upgraded to version 4.4.0.

G4. Pin smart contract dependencies to a specific version [low] [resolved]

Currently the OpenZeppelin dependency specifies a range of releases rather than a specific release

```
"@openzeppelin/contracts": "^4.3.2"
```

Recommendation: It is recommended that smart contract package dependencies refer to a specific version of the code in package.json instead of depending on package-lock.json. In this way, it will be unambiguous to later developers or third parties that will want to build on or verify your contract code which code version was compiled and deployed.
Severity: Low
Resolution: The issue was resolved.

G5. Solidity version is not up to date [low] [resolved]

The current version of Solidity used in the contracts is 0.8.9. At the moment of writing this report, a new version, 0.8.10, is available, which contains various bug fixes and optimizations.
Recommendation: Upgrade Solidity to the latest version
Severity: Low
Resolution: The issue was resolved.

G6. Coverage is not part of the CI action [info] [not resolved]

The coverage script is not included in the continuous integration actions.
Recommendation: Run the coverage script as part of the continuous integration process. This would have also helped to catch the following issues G7 and G8.

Severity: Info
Resolution: The issue was not addressed.

G7. Some tests fail under coverage [low] [not resolved]

When running npx hardhat coverage, some tests fail - and the coverage report is incomplete

```
> Istanbul reports written to ./coverage/ and ./coverage.json
Error in plugin solidity-coverage: ❌ 2 test(s) failed under coverage.

For more info run Hardhat with --show-stack-traces
```

Recommendation: Fix these errors.
Severity: low
Resolution: The issue was not resolved.

G8. Coverage is not properly configured [info] [resolved]

Mocks and test files are included in the coverage report as well, contaminating the results and reduces the total coverage score.
Recommendation: Configure the coverage settings to exclude mock and test files.
Severity: Info
Resolution: The issue was resolved.

G9. deploy.ts script is not tested [info] [not resolved]

The correct functioning of the deployment script “deploy.ts” is essential for the security and functioning of the system. The deployment script contains a bug (issue E1 described below) which would probably have been caught if the script were tested.
Recommendation: Add tests for the deploy script. Or even better, use the same code for deploying contracts in the test environment that you will use in production.
Severity: Info
Resolution: This issue was not resolved.

G10. Consider adding an interface for the “entire Landworks Diamond” [info] [resolved]

Integrators may find it useful to have available the interface to interact with the Landworks Diamond “core functionality” - i.e. an ILandWorks interface that inherits from ERC721Facet, FeeFacet, etc. etc.
Recommendation: Consider adding an interface definition for the LandWorks Diamond so that integrators can easily interact with the contract.
Severity: Info
Resolution: This issue was resolved by adding ILandWorks.sol

G11. Duplicated definitions of events [info] [partially resolved]

Some events are defined in different places in the code, like the “Rent” event that is defined in IMarketPlace.sol, IDecentraland.sol and LibRent.sol, the event “ClaimRentFee” that is defined in both IFeeFacet.sol and IMarketPlaceFacet.sol, and the “SetTokenPayment” and “setFee” events that are defined both in IFeeFacet.sol and LibFee.sol.
Recommendation: Code duplication is to be avoided, and it is better to have a single definition for each event to avoid errors.
Severity: Info
Resolution: This issue was partially resolved

G12. Add indexes to events [low] [resolved]

Very few event arguments are indexed, which will make the Events less useful than they could be. For example, in the ClaimProtocolFee event in IFeeFacet.sol:

```
event ClaimProtocolFee(address _token, address _recipient, uint256 _amount);
```

The intended use here is probably that these events will be searchable by token and recipient, and to enable such a search on ethereum nodes, these arguments need to be indexed:

```
event ClaimProtocolFee(address indexed _token, address indexed _recipient, uint256 _amount);
```

Recommendation: Evaluate, for each defined event, which arguments need to be indexed and change the code accordingly.
Severity: Low
Resolution: This issue was resolved.

DecentralandFacet.sol

D1. updateOperator does not set the actual operator [info] [not resolved]

The “updateOperator” function associates an operator address to a rentId. Even if the rent is currently active, the call does not change the operator on the decentraland contract - for that, the user has to call a separate function, “updateState”.
Recommendation: This behavior can be confusing, and the user experience can be somewhat improved by calling “updateState” as part of the updateOperator function in case the rent is currently active.
Severity: Info
Resolution: This issue was not resolved.

D2. Possible re-entry in rentDecentraLand [info] [not resolved]

The “rentDecentraland” function calls the “rent” function, which will call the transfer function of the payment token, which is an external contract. This means that if the payment token contract is malicious, it could re-enter.
Recommendation: To exclude re-entry, consider using OpenZeppelin’s ReentrancyGuard pattern
Severity: Info (we do not see a viable attack here, and if the payment token is actually corrupted it can do damage in much more straightforward ways).
Resolution: This issue was not resolved.

ERC721Facet.sol

E1. initERC721() is not called in the deployment procedure [low] [resolved]

The contract defines a function called “initERC721()” which is called in the tests but not in the deploy.ts script.
Recommendation: Call this function as part of the deployment procedure. As there is a theoretical risk that this function is called by an attacker after the Landworks diamond is deployed, consider calling the function in the constructor of the LandWorks diamond contract.
Severity: Low
Resolution: This issue was resolved as recommended.

FeeFacet.sol

F1. Do not use NULL value as a significant value [low] [resolved]

The value address(0) for the token address is used to signify that Ether instead of an ERC20 token should be transferred. This is also the default value if a user does not provide any value at all, which can potentially lead to mistakes.
Recommendation: Define a constant ETHEREUM_TOKEN, set it to a significant value (i.e. address(1)), and use that throughout, in FeeFacet as well as in LibTransfer.sol and MarketplaceFacet.sol.
Severity: Low
Resolution: This issue was resolved as recommended.

F2. Two different accounts can claim rent fee [low] [resolved]

In FeeFacet.claimRentfee, as well as in MarketplaceFacet.updateConditions, where similar logic is duplicated, it depends on the caller of the function who will receive the rent: if the sender is also the consumer of the asset, then she will receive the rent fee, if the caller is instead the owner, or an account that was approved by the owner, the rent is transferred to the owner of the asset.
This is confusing, and creates ambiguity about who has the right to claim the rent fee.
Recommendation: Make the receiver of the rent independent of the caller of the function - for example, implement logic such as sending the rent to the consumer, if a consumer exists, and sending it to the owner otherwise.
Severity: Low
Resolution: The issue was resolved: if a consumer exists, the rent goes to the consumer, and to the owner otherwise.

LandWorks.sol

W1. Do not allow for the contract to receive ETH [low] [resolved]

The Landworks contract explicitly defines functionality to receive ETH (the constructor is payable, and it defines a default “receive” function). However, the contract has no functionality to manage its ether balance, so if any ETH is sent to the contract, it will be effectively stuck there (unless the contract is upgraded)

Recommendation: Make it clear from the contract code that no ETH is meant to be sent to the contract by removing the “payable” modifier from the constructor and removing the “receive” function (or throwing an error if the “receive” function is called).
Severity: Low
Resolution: The issue was resolved.

W2. Supported interfaces list should be configurable [info] [not resolved]

From line 24ff, a number of supported interfaces are declared - such as IERC165, IDiamondCut, IERC721Consumable, etc. These interface definitions are hardcoded, and so can not be changed.
In contrast, the DiamondCut interface allows the owner of the contract to change the supported APIs. This potentially can lead to a result in which the LandWorks diamond declares supporting an interface that it does not support - or vice versa.
Recommendation: Add functions to add and remove interfaces from the list supportedInterfaces.
Severity: Info
Resolution: This issue was not resolved.

LibERC721.sol

L1. Transfer to claim rent can be front-run [medium] [resolved]

A LandWorks NFT also represents the right to claim any outstanding (paid, but as yet unclaimed) rent. A malicious buyer could negotiate a price with a seller to pay for the NFT, together with the claim on outstanding rent; subsequently, the seller can front-run the token sale transaction, and claim the outstanding rent fees before the NFT is transferred.
Recommendation: Assign the rent fee to the owner/ consumer (if exists) instead of the asset, or automatically claim the rent fee before a transfer occurs.
Severity: Medium
Resolution: This issue was resolved - in ERC721Facet.sol, the functions “transferFrom” and “safeTransferFrom” now pay out the rent before transferring the token.

LibRent.sol

R1. Do state changes before external calls [low] [resolved]

On line 64, safeTransfer is called on the payment token, which is an external contract. This call happens before a number of state changes - in particular before the Rent object is actually added to the rents set. If such a contract is compromised or malicious, it could re-enter.
Recommendation. Although we do not see an immediate attack, it is best practice to execute state changes before external calls - i.e. to move the transfer call to the end of the rent function.
Severity: Low
Resolution: This issue was resolved as recommended.

LibTransfer.sol

T1. Do not use transfer [low] [resolved]

In line 28, ether is transferred using a “transfer()” call. The transfer call reserves a fixed amount of 2300 gas for the transfer. However, in the future, gas costs may change - and if the new gas cost for transferring ether will be set to a value higher than 2300, this call will fail.
Please see https://consensys.net/diligence/blog/2019/09/stop-using-soliditys-transfer-now/ for a detailed discussion.
Recommendation: Instead, use this pattern:

```
// This forwards all available gas. Be sure to check the return value!
(bool success, ) = msg.sender.call.value(amount)("");
require(success, "Transfer failed.");
```

Severity: Low
Resolution: This issue was resolved as recommended.

MarketplaceFacet.sol

M1. Do state changes before external calls [low] [resolved]

On line 57 in “list”, the ERC721 token is transferred from the registry to the LandWorks contract before the asset is added to the listed assets, and so re-entrancy is possible if somehow the registry is corrupted.
Recommendation: Although we do not see a vulnerability here, it is best practice to do external calls after state changes, and we recommend you follow the practice.
Severity: Low
Resolution: The issue was resolved as recommended.

M2. Typo in comments [info] [resolved]

One line 96, and again on line 184, the comment states that the asset is transferred to the caller. This is not the case: the asset is transferred to its owner.
Recommendation: Fix both comments.
Severity: Info
Resolution: This issue was resolved

M3. Consider returning rentID in rent function [info] [resolved]

Calling “rent()” currently returns no meaningful value - it makes sense to return the rentId, or the return value of LibRent.rent, to make integration by third-party contracts easier
Recommendation: Pass on the return value of LibRent.rent
Severity: Info
Resolution: The issue was resolved as recommended.

README.md

R1. Explain that you need a config file before giving the npx hardhat compile instruction [info] [resolved]

In the README it says in the deployment instructions that you need to create the config.ts before deploying, but you also need it even before compiling, which comes first in the instructions.

Recommendation: Explain before the compiling instructions that you need to first create the config.ts file.
Severity: Info
Resolution: The issue was resolved as recommended.

General issues regarding Landworks-YF

Y1. Licensing violations [medium] [resolved]

The Staking Rewards contract is adapted from
https://github.com/Synthetixio/synthetix/blob/develop/contracts/StakingRewards.sol
This file was published under the MIT license. You can republish the files under MIT, but you must, as the LICENSE file in that repository says, maintain the original copyright statements:

> "Copyright (c) 2019 Synthetix
> [...]
> The above copyright notice and this permission notice shall be included in all
> copies or substantial portions of the Software."

What you definitely can *not* do is claim copyright yourself.
Recommendation: Leave the original copyright attribution intact in those files that were copied and only slightly modified.
Severity: Medium

Y2. Solidity version is not up to date [info] [resolved]

The current version of Solidity used in the contracts is 0.8.9. A new version, 0.8.10, is available.
Recommendation: Upgrade Solidity to the latest version.
Severity: Info
Resolution: The issue was resolved.

Y3. Coverage is not part of the CI action [info] [resolved]

The coverage script is not included in the continuous integration actions.
Recommendation: Run the coverage script as part of the continuous integration process.
Severity: Info
Resolution: The issue was resolved.

Y4. Coverage is configured improperly [info] [resolved]

Mocks and test files are included in the coverage report as well, contaminating the results and reduces the total coverage score.
Recommendation: Configure the coverage settings to exclude mock and test files.
Severity: Info
Resolution: The issue was resolved.

Y5. Compilation gives a warning [info] [resolved]

Compiling the contracts gives a warning from the EstateRegistryMock.sol file.
Recommendation: Since this warning is from a mock file, it is safe to simply silence it so that the compiler will not warn about it.
Severity: Info
Resolution: The issue was resolved.

LandWorksDecentralandStaking.sol

S1. Estate can get stuck in the contract if its size changes [high] [resolved]

The getAmount function is used both on stake and withdraw to get the amount of land to add or remove from the contract. In case the deposited NFT is an estate, the getAmount function will call the external Decentraland estate registry contract to get the amount worth of the estate.
An issue may arise if this external getEstateSize function returns a different value on withdraw than it has returned on stake. In case the value is greater, the withdraw call will revert on line 172 due to the balance of the caller being now lower than the size of the estate on withdraw, and the estate will be stuck in the contract.
This could be exploited as a griefing attack by someone adding land to an estate after it has been staked using the safeTransferFrom function in the Decentraland LANDRegistry contract, or by any potential exploit or vulnerability in the Decentraland contracts.
Recommendation: During staking, save the size of each deposited token, and use that saved size in the withdraw function instead of calling the getAmount function there.
Severity: High - owners may lose access to their estates.
Resolution: The issue was resolved. The size of each deposited asset is now saved on staking and used in the withdraw function instead of making an external call to read the size of the asset.

S2. Misleading comment in getAmount function [info] [resolved]

The getAmount function description says “Gets the represented amount to be staked”, but this is not accurate as it is also used to get the amount to be withdrawn.
Recommendation: Leave the comment as it is but use the getAmount function only in the stake function. See recommendation on issue S1 for more info.
Severity: Info
Resolution: The issue was resolved. The function was renamed to “computeAmount” and now is only called when staking.

S3. Misleading comment in stake function [low] [resolved]

On line 120 in “stake”, the comment says “Save who is the owner of the token”, but the saved address can also be someone who is not the owner but is approved to move the token. Which makes the comment misleading and means that, on withdrawal, the token will go to the depositor and not necessarily the original owner.
Recommendation: Update the comment to say the “staker” or “depositor” instead of the “owner” or update the code to save the original owner instead of the msg.sender.
Severity: Low