CONSENSYS
Diligence

Horizon Games Audit

Date	February 2020

- 1 Executive Summary
  - 1.1 Scope
  - 1.2 Objectives
- 2 Recommendations
  - 2.1 Document risks to meta transaction relayers
  - 2.2 Review the Code Quality recommendations in Appendix 2
- 3 System Overview
  - 3.1 Detailed design
- 4 Security Specification
  - 4.1 Actors
  - 4.2 Trust Model
  - 4.3 Important Security Properties
- 5 Issues
  - 5.1 Tokens with no decimals can be locked in Niftyswap Major Acknowledged
  - 5.2 Incorrect response from price feed if called during an `onERC1155Received` callback Medium Acknowledged
- Appendix 1 - Files in Scope
- Appendix 2 - Code Quality Recommendations
  - A.2.1 Use multiple lines for visibility, mutability and returns keywords in function declarations
  - A.2.2 Clearer names
  - A.2.3 Avoid named return values
  - A.2.4 Write buy/sell price math consistently
  - A.2.5 Use `require` instead of `getIDAddress`
- Appendix 3 - Disclosure

1 Executive Summary

This report presents the results of our engagement with Horizon Games to review the security of the Smart Contracts which make up their system, across 3 separate repositories.

The review was conducted over the course of two weeks, from February 10th to 21st by Daniel Luca and John Mardlin. Our primary point of contact at Horizon Games was Philippe Castonguay. A total of 20 person-days were spent one this work.

Although not reported on here, prior to this work, Steve Marx performed a preliminary review of the system on January 20th to confirm that would be ready for a full review.

We began the first week of the review speaking with Philippe, and reviewing the specifications in order to understand the design of the system. We simultaneously referred to the code, and asked questions in a dedicated chat channel setup for this engagement. During the first week we focused primarily on the various implementations of the ERC1155 token, which has several derived variants, with functionality such as Meta-transactions, minting and burning, wrapped ETH, and “packed balances”.

On Friday of the first week, and much of the second week, we shifted our focus towards the Niftyswap exchange, which implements an automated market maker model similar to Uniswap.

Our review identified only one potential issue (see section 7), which could potentially impact on external contracts using Niftyswap as a price oracle, and only in very specific circumstances.

1.1 Scope

Our review focused on the smart contracts in 3 repositories each at a specific commit.

Repository	Commit
https://github.com/arcadeum/multi-token-standard	83a6a9dc
https://github.com/arcadeum/erc20-meta-wrapper	6bda05ad
https://github.com/arcadeum/niftyswap	a27f3046

The list of files in scope can be found in the Appendix.

1.2 Objectives

Through discussion with the Horizon team, we identified the following priorities for our review:

1. Verifying that the behavior of the Smart Contracts was consistent with the specifications provided to us. Many of these behaviors are listed in the Security Specification section
2. Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry.

2 Recommendations

2.1 Document risks to meta transaction relayers

There are certain checks which meta transaction relayers are required to perform in order to ensure they receive sufficient payment to cover the gas costs of executing a transaction. These should be clearly communicated.

These include:

1. Estimate all the gas costs for a transfer, and ensure that _g.gasFee is sufficient to cover it. This is easily doable as the only portion of the transaction which might involve arbitrary computation has a maximum bounded by `_g.gasLimitCallback`.
2. Verify that call to `IERC1271Wallet(_signerAddress).isValidSignature(_data, _sig)` is not expected to revert.

2.2 Review the Code Quality recommendations in Appendix 2

Other comments related to readability and best practices are listed in Appendix 2.

3 System Overview

Horizon Games is building a blockchain infrastructure for onboarding game developers and provides them the tools to create their own games on top of the Ethereum ecosystem. The audited system provides a token system and a associated marketplace. Their own proof of concept is SkyWeaver a collectible card game similar to Magic: The Gathering or Hearthstone.

3.1 Detailed design

The audited system consists of the following relevant parts:

- An **ERC 1155** contract system that provides all of the needed functionality part of the official spec.
- An optimized **ERC 1155** contract system (`ERC1155PackedBalance`) that provides all of the needed functionality part of the official spec with the twist of having an optimized contract storage layout for low balance tokens. This works by packing multiple balances in the same 32 byte storage slot, saving a lot of gas because if the updated balances are part of the same storage slot, the gas cost is only paid once.
    - This should not be used for games with collectible items that are eventually rolled out of existence, because packing balances requires additional steps and eventually having a sparse balance layout can cause it to use more gas instead of less. It will use more gas because it cannot take advantage of updating the same storage slot. Instead because of the roll out of items the balances still in existence are far apart, effectively being in different storage slots. This means that it cannot take advantage of updating the same storage slot, effectively paying for the storage slot update **and** for the bit operations that pack the balances.
- A version for each **ERC 1155** and **ERC 1155 Packed Balance** that supports meta transactions. These meta transactions have incentives built-in for relayers that want to execute them on the blockchain, paying the execution cost, while receiving a predictable reward.
- A marketplace system is in place for **ERC 1155** tokens that provides an Uniswap-like decentralized exchange for users that want to trade tokens.
- A **Meta ERC 20 Wrapper** exists to provide a way for **ERC 20** tokens to be wrapped in a **ERC 1155** standard which later can be used in the marketplace.

4 Security Specification

This section describes, **from a security perspective**, the expected behavior of the system under audit. It is not a substitute for documentation. The purpose of this section is to identify specific security properties that were validated by the audit team.

4.1 Actors

The relevant actors are listed below with their respective abilities:

- `ERC1155` specific actors
  - `anyone` can
    - check balances of any address
    - check transfer approvals
    - check supported interfaces (**ERC 165** and **ERC 1155**)
  - `token owner` can
    - do whatever `anyone` can do
    - transfer, approve transfers, revoke transfers of ERC 1155 tokens
    - trade those said tokens on the `NiftyswapExchange`
  - if there is an `owner` they can
    - do whatever the `token owners` can do (if they have tokens)
    - mint and burn new tokens that can be used as fungible or non fungible tokens within `NiftyswapExchange`
- `NiftyswapFactory` specific actors
  - `anyone` can
    - create new exchanges
    - check what exchange exists for any pair of **ERC 1155 token** and **ERC 1155 (base token, id)**
- `NiftyswapExchange` specific actors
  - `anyone` can
    - check buy or sell prices for a pair of token ids and base token
    - find out how many base tokens are in reserve for the specified token ids
    - check the traded token address
    - check the base traded token address and id
    - check total supply
    - check the factory address
    - check supported interfaces (**ERC 165** and **ERC 1155**)
  - `ERC 1155 tokens owner` can
    - trade the defined **ERC 1155 tokens** for the specified **ERC 1155 (base token, id)** and viceversa
    - add or remove liquidity to and from the liquidity pool

    - `Niftyswap liquidity token owners` can
        - transfer, approve transfers, revoke transfers for their tokens
        - exchange their liquidity tokens for the defined **ERC 1155 tokens** and **ERC 1155 (base token, id)** effectively removing liquidity

- `MetaERC20Wrapper` specific actors
    - `anyone` can
        - retrieve the internal id corresponding to the wrapped **ERC 20 token**
        - retrieve the **ERC 20 token** address corresponding to the internal id
        - check how many types of **ERC 20 tokens** are registered in the contract
        - check supported interfaces (**ERC 165** and **ERC 1155**)
    - `ERC 20 token owners` can
        - do whatever `anyone` can do
        - deposit **ether** or any **ERC 20 tokens** for wapping in a **ERC 1155 meta token**
        - withdraw their **ERC 1155 meta tokens** to retrieve the initial **ERC 20 tokens** or **ether**

4.2 Trust Model

The information in this section is similar to the previous, but is framed in such a way as to help users understand the model.

The contracts in scope for this review were completely absent of any permissioning scheme which would require trust in a centralized administrator type of role. All the actor roles listed above may be performed by anyone with access to the ethereum network, and sufficient resources to participate.

As a caveat, the contracts we reviewed can be considered as configurable via inheritance, and the parameters chosen upon deployment, thus we are unable to comment on the final deployed system at this time. We would also expect some ability for an administrater to mint and burn tokens to be commonly deployed in the ERC1155 contract

4.3 Important Security Properties

The following is a non-exhaustive list of security properties that were verified in this audit:

- `ERC1155` Implements the spec defined in the official
    - EIP 1155
        - handles balances correctly
        - transfers tokens between users
        - approves tokens for users to transfer
        - returns the number of tokens approved, balances
        - returns whether a user is allowed to transfer a number of tokens on behalf of another user
    - EIP 165
        - returns interface signature for **EIP 165** and **EIP 1155**
- `ERC1155Packed`
    - has the same correctness specified above for `ERC1155`

    - balances are tightly packed together in the same storage slot without the possibility of over/under flows
- `ERC1155Meta`
    - has the same correctness specified above for `ERC1155`
    - is able to handle meta transactions correctly
    - provides relayers a way to be compensated for executing transactions on behalf of other users
    - protects the meta transaction relayers from gas griefing attacks
    - protects the meta transaction signers from relayers changing the initial intended action
- `MetaERC20Wrapper`
    - handles deposits and withdrawals correctly for any users owning an ERC20 token and provides a meta token that represents the initial ERC20 token wrapped as an **ERC 1155**
    - handles transfers and approvals of the created meta tokens between addresses
    - checking the successful transfer (when deposits and withdrawals are done) also handles **ERC 20** tokens that do not return `true` on success
- `NiftyswapExchange`
    - correctly handles the liquidity pool (increasing or decreasing) based on the Uniswap formulas
    - correctly computes the buy or sell price based on the Uniswap formulas
    - handles buys / sells and liquidity addition / removal making sure the caller of the `onERC1155BatchReceived` is the correct one in each case
    - returns correct token and base token reserves
    - applies a fee on buys and sells
    - returns interface signature for **EIP 165** and **EIP 1155**
- `NiftyswapFactory`
    - creates a new exchange without doing any checks, leaving this responsibility for `NiftyswapExchange`

5 Issues

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.

5.1 Tokens with no decimals can be locked in Niftyswap Major Acknowledged

Resolution
This will be addressed by only listing tokens with at least 2 decimals. This should be well documented in the Niftyswap repository and code comments.

Description

Assume the Niftyswap exchange has: 1. wrapped DAI as the base currency, and 2. it's ERC1155 contract has a token called "Blue Dragons", which are a "low fungibility" token, with zero decimals, and a total supply of 100.

Consider the following scenario on the Niftyswap exchange:

1. 10 people each add 1,000 DAI, and 1 BlueDragon. They get 1,000 pool tokens each.
2. Someone buys 1 BlueDragon, at a price of 1,117 base Tokens (per the constant product pricing model).
3. Niftyswap's balances are now 11,117 baseTokens, 9 Blue Dragons.
4. Someone removes liquidity by burning 1,000 pool tokens:
    1. They would get 1111 base tokens (`1000 * 11,117/ 10000`).
    2. They would get 0 Blue Dragons due to the rounding on integer math.

Recommendation

Through conversation with the developers, we agreed the right approach is for tokens to have at least 2 decimals to minimize the negative effects of rounding down.

5.2 Incorrect response from price feed if called during an `onERC1155Received` callback Medium Acknowledged

Resolution
The design will not be modified. Horizon Games should clearly document this risk for 3rd parties seeking to use Niftyswap as a price feed.

Description

The ERC 1155 standard requires that smart contracts must implement `onERC1155Received` and `onERC1155BatchReceived` to accept transfers.

This means that on any token received, code run on the receiving smart contract.

In `NiftyswapExchange` when adding / removing liquidity or buying tokens, the methods mentioned above are called when the tokens are sent. When this happens, the state of the contract is changed but not completed, the tokens are sent to the receiving smart contract but the state is not completely updated.

This happens in these cases

`_baseToToken` (when buying tokens)

code/niftyswap/contracts/exchange/NiftyswapExchange.sol:L163-L169

```
// // Refund Base Token if any
if (totalRefundBaseTokens > 0) {
  baseToken.safeTransferFrom(address(this), _recipient, baseTokenID,
totalRefundBaseTokens, "");
}

// Send Tokens all tokens purchased
token.safeBatchTransferFrom(address(this), _recipient, _tokenIds,
_tokensBoughtAmounts, "");
```

`_removeLiquidity`

code/niftyswap/contracts/exchange/NiftyswapExchange.sol:L485-L487

```
// Transfer total Base Tokens and all Tokens ids
baseToken.safeTransferFrom(address(this), _provider, baseTokenID, totalBaseTokens,
"");
token.safeBatchTransferFrom(address(this), _provider, _tokenIds, tokenAmounts, "");
```

`_addLiquidity`

code/niftyswap/contracts/exchange/NiftyswapExchange.sol:L403-L407

```
// Mint liquidity pool tokens
_batchMint(_provider, _tokenIds, liquiditiesToMint, "");

// Transfer all Base Tokens to this contract
baseToken.safeTransferFrom(_provider, address(this), baseTokenID, totalBaseTokens,
abi.encode(DEPOSIT_SIG));
```

Each of these examples send some tokens to the smart contract, which triggers calling some code on the receiving smart contract.

While these methods have the `nonReentrant` modifier which protects them from re-netrancy, the result of the methods `getPrice_baseToToken` and `getPrice_tokenToBase` is affected. These 2 methods do not have the `nonReentrant` modifier.

The price reported by the `getPrice_baseToToken` and `getPrice_tokenToBase` methods is incorrect (until after the end of the transaction) because they rely on the number of tokens owned by the `NiftyswapExchange`; which between the calls is not finalized. Hence the price reported will be incorrect.

This gives the smart contract which receives the tokens, the opportunity to use other systems (if they exist) that rely on the result of `getPrice_baseToToken` and `getPrice_tokenToBase` to use the returned price to its advantage.

It's important to note that this is a bug only if other systems rely on the price reported by this `NiftyswapExchange`. Also the current contract is not affected, nor its balances or internal ledger, only other systems relying on its reported price will be fooled.

Recommendation

Because there is no way to enforce how other systems work, a restriction can be added on `NiftyswapExchange` to protect other systems (if any) that rely on `NiftyswapExchange` for price discovery.

Adding a `nonReentrant` modifier on the view methods `getPrice_baseToToken` and `getPrice_tokenToBase` will add a bit of protection for the ecosystem.

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash (truncated)
erc20-meta-wrapper/…/IMetaERC20Wrapper.sol	4918ea1
erc20-meta-wrapper/…/MetaERC20Wrapper.sol	11307a8
multi-tokens-standard/…/IERC1155.sol	4d15ff7
multi-tokens-standard/…/IERC1155Meta.sol	4daf7e4
multi-tokens-standard/…/IERC1155Metadata.sol	5c7a378
multi-tokens-standard/…/IERC1155MintBurn.sol	166ef0d
multi-tokens-standard/…/IERC1155TokenReceiver.sol	af0036c
multi-tokens-standard/…/IERC1271Wallet.sol	9f00d40
multi-tokens-standard/…/IERC165.sol	bf172c4
multi-tokens-standard/…/IERC20.sol	62ff22e
multi-tokens-standard/…/ERC1155.sol	9174634
multi-tokens-standard/…/ERC1155Meta.sol	3f2f5ee
multi-tokens-standard/…/ERC1155Metadata.sol	7f9027f
multi-tokens-standard/…/ERC1155MintBurn.sol	7f9f510
multi-tokens-standard/…/ERC1155MetaPackedBalance.sol	f7cc692
multi-tokens-standard/…/ERC1155MintBurnPackedBalance.sol	a5e07c7
multi-tokens-standard/…/ERC1155PackedBalance.sol	24493f9
multi-tokens-standard/…/Address.sol	51ad620
multi-tokens-standard/…/LibBytes.sol	01404ab
multi-tokens-standard/…/LibEIP712.sol	29bb0a6
multi-tokens-standard/…/Ownable.sol	a90ad66
multi-tokens-standard/…/SafeMath.sol	669746d
multi-tokens-standard/…/SignatureValidator.sol	a431481
niftyswap/…/NiftyswapExchange.sol	0437cc3

File	SHA-1 hash (truncated)
niftyswap/…/NiftyswapFactory.sol	0e5600d
niftyswap/…/IERC165.sol	95157b4
niftyswap/…/IERC20.sol	823eb59
niftyswap/…/INiftyswapExchange.sol	125f529
niftyswap/…/INiftyswapFactory.sol	57cbb20
niftyswap/…/ERC20.sol	6e0049a
niftyswap/…/ReentrancyGuard.sol	2dc8a11
niftyswap/…/SafeMath.sol	288882f

Appendix 2 - Code Quality Recommendations

A.2.1 Use multiple lines for visibility, mutability and returns keywords in function declarations

This style convention is sometimes used:

```
function _getTokenReserves(
    uint256[] memory _tokenIds)
    internal view returns (uint256[] memory)
```

The following is more common, and in our opinion more readable:

```
function _getTokenReserves(
  uint256[] memory _tokenIds
)
  internal
  view
  returns (uint256[] memory)
{
```

A.2.2 Clearer names

Because of the number of variables in the system, token related ones in particular, a review of variable names would help to improve clarity for other reviewers or developers building on the contracts

The following are some suggestions for consideration.

In ERC1155Meta_*:

- Prefer `_gasReceipt` over `_g`
- Prefer `_sigData` over `_data` in call to `_signatureValidation()`
- Prefer `signature` over `sig` in `(sig, signedData) = abi.decode(_sigData, (bytes, bytes))`

In NiftySwap:

- Prefer `multiToken` over `token`
- Prefer `currency` over `baseToken`

Also try to clarify when a variable is a mapping by using the covention `mapping(uint=>uint) fooToBar`, and when it is an array by using the word `list` in the name.

A.2.3 Avoid named return values

For example, these functions use named return values, but don't refer to them:

```
  function getBaseTokenInfo() external view returns (address _baseTokenAddress,
uint256 _baseTokenID) {
    return (address(baseToken), baseTokenID);
  }
```

```
 function divRound(uint256 a, uint256 b) internal pure returns (uint256 val, bool
rounded) {
    return a % b == 0 ? (a/b, false) : ((a/b).add(1), true);
  }
```

A.2.4 Write buy/sell price math consistently

The math in `getBuyPrice()` and `getSellPrice()` both match the spec, but look quite different.

A.2.5 Use `require` instead of `getIDAddress`

In ERC1155, `getIdAddress()` is used to verify if a token's id is registered:

```
      // Checks if id is registered
     getIdAddress(_ids[i]);
```

A cheaper and more explicit way of achieving this would be to just take the check from that function:

```
require(IDtoAddress[_id] != address(0x0), "MetaERC20Wrapper#getIdAddress:
UNREGISTERED_TOKEN")
```

Appendix 3 - 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.