Security Audit
Horizon Games
ERC1155 & ERC20MetaWrapper

Monday, 23-Dec-19

Agustín Aguilar

Introduction

The Horizon Games team requested a security audit of their ERC-1155 implementation, ERC-1155 packed balance, and ERC-20 to ERC-1155 wrapper project, the audited repositories follow.

https://github.com/arcadeum/multi-token-standard (0xea0502818746d39afb16ab834cea4f3fad5d5a9a)
https://github.com/arcadeum/erc20-meta-wrapper (0xca4b7af5b9ddc6acfcebfa54d9b932a820e14ff3)

ERC-1155 Implementation

It fully implements the multi-token standard ERC-1155, with additional implementations for meta-transactions, metadata, burning, and minting tokens.

The contracts that constitute the project follow.

- ERC1155 - Contains the minimal implementation of the ERC-1155 standard
- ERC1155Meta - Inherits from ERC1155 and adds support for meta-transactions for transferFrom, batchTransferFrom and approval
- ERC1155Metadata - Implements the ERC-1155 metadata getters and events
- ERC1155MintBurn - Inherits from ERC1155 and adds support for the minting and burning of tokens, both individually and through batches

ERC-20 to ERC-1155 wrapper

It implements a wrapper contract of ERC-20 tokens and ETH on a shared ERC-1155 token, with support for meta-transactions.

The contracts that constitute the project follow.

- MetaERC20Wrapper - Inherits from ERC1155Meta, implements the wrapping of ERC-20 and ETH, supports deposit and withdrawals; the ERC-1155 contract is shared by multiples ERC-20.

ERC-1155 packed balance implementation

It fully implements the multi-token standard ERC-1155, using a 256-bit storage slot to store up to 8 token balances. It allows for more efficient transfers of balances with certain combinations of tokens. This implementation of ERC-1155 uses an unsigned integer of 32 bits to store each balance.

The contracts that constitute the project follow.

- ERC1155PackedBalance - Contains the packed implementation of the ERC-1155 standard
- ERC1155MetaPackedBalance - Inherits from ERC1155PackedBalance and adds support for meta-transactions for transferFrom, batchTransferFrom and approval
- ERC1155MintBurnPackedBalance - Inherits from ERC1155PackedBalance and adds support for the minting and burning of tokens, both individually and using batches

Issues

Critical severity

C1 - Inflation exploit on packed balance

The method `_safeBatchTransferFrom` on the `ERC1155PackedBalance` contract handles the update of both the sender and receiver balances. This process is performed on memory to avoid unnecessary reads and writes to the contract storage.

When the sender and receiver contain the same address, both values are tracked in memory using separated variables, thus performing updates on them as if they were separated balances.

This behavior opens the possibility for an attacker to duplicate their funds by performing a batch transfer to itself. The following batch transfer should duplicate the balance of the sender:

- Assume that the `attacker` is a valid address containing 1.000 tokens of the ID 0, and execute the following call from the attacker address.

```
safeBatchTransferFrom(
      attacker,
      attacker,
      [0],
      [1000]
)
```

- After the execution, the `attacker` balance of ID 0 becomes 2000.

Proposed solution:

a) Skip the update of the balances if the sender and receiver are the same address.

Update:

- *The Horizon Games team fixed this issue on the commit `0x624814b` by not updating the balances if the sender and receiver are the same address; balance checks and emission of events are still performed.*

High severity

H1 - GasReceipt injection using `transferData`

The methods `metaSafeTransferFrom` and `metaSafeBatchTransferFrom` encode both the `gasReceipt` and `transferData` on `signedData` during a meta-transaction with a gas refund, but only encode `transferData` is encoded when a meta-transaction is not intended to have a gas refund.

A malicious application or dApp could lead a user to generate a meta-transaction that looks like a regular transfer without a gas refund but with a given `transferData` that encodes a hidden GasReceipt refund, for it to be re-interpreted as a gas refund by relaying the transaction with `_isGasFee` set to `true`.

This attacker could encode a disproportionate hidden gas refund, something in the order of "transfer all `` `_from` `` balance of X ERC20" to the attacker.

Proposed solution:

a) Make `_isGasFee` explicit by including the flag value on the signature validation.

Medium severity

M1 - Possible overflow during `_viewUpdateBinValue`

The `add` operation on `_viewUpdateBinValue` validates that a given addition doesn't exceed `2 ** 32`. This ensures that the result operation is not leaking into the balances stored at other indexes.

However, an overflow during the addition operation is possible when the result is below `2 ** 32`. Such overflow is not possible when performing a regular transfer on `ERC1155PackedBalance`, but it's conceivable during a `mint` or `batchMint` operation.

The overflow described happens at the whole bin and makes the affected ID balance go to zero.

Below is an example of how to replicate the issue:

- `Account[0]` has an ID 0 token balance of `4294967295`
- Execute the following internal call:

```
_mint(
      account[0],
      0,
      1157920892373161954235709850086879078532699846656405640394575840079088346726 40,
      []
)
```

- The balance of `account[0]` token ID 0 becomes zero.

Proposed solutions:

a) Use the `SafeMath` library when performing the addition of two bins.
b) Document the behavior, so `_mint` is never called with values above `4294967295`.

Low severity

L1 - Griefing by front-running transaction without fee refund

The GasReceipt scheme allows for a meta-transaction to have a fixed recipient of the fee reimbursement, working as a mechanism to avoid relayers front-running each other to obtain their fees.

An attacker can circumvent this protection by relaying those meta-transactions with the `_isGasFee` flag set to false. This would incur a cost for the attacker, but it would also deprive the original relayer of his reward. Such an attack could be used to discourage the participation of a targeted relayer.

Proposed solution:

a) Make `_isGasFee` explicit by including the flag value on the signature validation.

L2 - Noncompliant with `_operator` definitions by ERC-1155

The ERC-1155 specification defines the `_operator` figure as “the address of an account/contract that is approved to make the transfer”[1]; this requirement is not satisfied on the `ERC1155Meta` contract during a `transfer` or `transferFrom` meta-transaction.

`ERC1155Meta` inherits from `ERC1155` and makes use of the internal methods `_safeTransferFrom` and `_safeBatchTransferFrom` to perform the balance transfers, and those methods emit the events `TransferBatch` and `TransferSingle`, but both assume that `msg.sender` is always the `_operator`.

In the case of the meta-transaction, `msg.sender` takes the value of the address of the relayer, which is not approved to make the transfer, and only executes the intent of the `_from` address.

`ERC1155Meta` also performs the defined callbacks of `ERC1155`, using `_callonERC1155Received` and directly calling the `IERC1155TokenReceiver` contract; in both cases, `msg.sender` is sent to the contract as the `_operator` when that value corresponds to the relayer.

The same issue is present on the `ERC1155PackedBalance` and `ERC1155MetaPackedBalance` contracts.

[1] ERC-1155 Draft specification.
https://github.com/ethereum/eips/issues/1155

Proposed solution:

a) Replace `msg.sender` with `_from` as the `_operator` on all methods involving meta-transactions.

L3 - `ERC1155PackedBalance` supports the incorrect ERC-165 interface

The `ERC1155PackedBalance` contract declares support for the interface signature `0x97a409d2`, although this interface is defined as the one that corresponds to ERC-1155. The correct interface for ERC-1155 is `0xd9b67a26`[2], as defined on the `ERC1155` contract.

Proposed solutions:

a) Replace `0x97a409d2` by `0xd9b67a26` on `INTERFACE_SIGNATURE_ERC1155`.

b) Implement ERC-165 support for ERC-1155 in a separate contract, and inherit from the former in both `ERC1155PackedBalance` and `ERC1155`.

L4 - `ERC1155PackedBalance` points to the wrong bin on high token IDs

The `ERC1155PackedBalance` contract calculates the bin to be used for a given ID using `bin = _id * IDS_BITS_SIZE / 256`.

`IDS_BITS_SIZE` is the constant 32, meaning that any token ID above `2 ** 256 - 1 / 32` causes an overflow, making it use the wrong bin.

This issue also has the unforeseen consequence of creating mirror token IDs for each token on the contract; an example follows:

- account has a balance of 100 Token ID 0
- `balanceOf(account, 0)` returns 100
- `balanceOf(`
  `account,`
  `3618502788666131106986593281521497120414687020801267626233049500247285301248`
  `)` also returns 100

This alternative token ID can also be transferred, minted, and burned.

[2] ERC-1155 ERC-165 Interface.
https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1155.md#a-solidity-example-of-the-keccak256-generated-constants-for-the-various-magic-values-these-may-be-used-by-implementation

Proposed solutions:

a) Use `SafeMath` during `bin` and `index` calculation.
b) Replace `_id * IDS_BITS_SIZE / 256` by `_id / IDS_PER_UINT256`.

L5 - Packed batch transfer throws with empty arrays

The methods `_safeBatchTransferFrom` and `_batchMint` from the ERC1155 packed balance project `throw` if the provided `_ids` and `_amounts` arrays are empty, this behavior happens because the methods try to access the first element of the array without checking if such element exists.

Solidity throws by calling the `INVALID_OPCODE` (`0xfe`) when the program tries to access an array outside of its bounds, this opcode doesn't only halt the execution, but it also spends all the remaining gas of the call.

Proposed solution:

a) Skip the updating of balances if `_ids` and `_amounts` are empty.

Notes

This section includes observations and issues that don't pose a direct security risk, but they could cause an indirect risk. This includes bad practices, gas execution inefficiencies, duplicated or redundant code, misleading comments, and similar observations.

Notes are presented in no particular order.

N1 - Efficient `batchBurn` is not implemented

The method `_batchBurn` doesn't efficiently perform the burning of balances by taking advantage of the packed data structure of the balances.

N2 - `getIDBinIndex` can be simplified

The method `getIDBinIndex()` uses the hardcoded number 256 to calculate the bin index for the given ID. This calculation can also be performed using `bin = _id / IDS_PER_UINT256`, avoiding the use of hardcoded numbers.

N3 - EIP712_DOMAIN_HASH can be calculated at runtime

The method `hashEIP712Message()` makes use of the constant `EIP712_DOMAIN_HASH`, which is calculated using `keccak256` and stored on the contract state.

The same constant can be calculated at runtime using `keccak256`, given that performing a `SLOAD` costs 800 GAS[3], and a `keccak256` costs approximately 200 GAS.

N4 - `abi.encodePacked` used in place of `abi.encode`

Multiple instances have been found of using `abi.encodePacked` while casting all variables with less than 32 bytes to `uint256` or `bytes32`, the ultimate motive of such casting is to presumed to be the building of a bytes array with all values padded to 32 bytes.

The same behavior can be achieved by using `abi.encode` instead of `abi.encodePacked`; `abi.encodePacked(uint256(address(this))` is equivalent to `abi.encode(address(this))`.

N5 - Ownable doesn't follow ERC-173

The project implements its custom version of the Ownable pattern, but this implemented version is not compatible with the proposed standard Ownable interface ERC-173[4].

Compatibility can be achieved by replacing the selector of the method `getOwner()` to `owner()`.

N6 - Misuse of the term `UNDERFLOW` on SafeMath

`SafeMath` reverts with the error string `"SafeMath#sub: UNDERFLOW"` if the result of a subtraction operation with unsigned integers is below zero.

However, the term `UNDERFLOW` commonly refers to a similar phenomenon[5] that occurs with floating-point numbers. `OVERFLOW` and `WRAPAROUND` are more acceptable terms for the unsigned integer phenomenon.

[3] "Repricing for trie-size-dependent opcodes" Istanbul EIP-1884.
https://eips.ethereum.org/EIPS/eip-1884
[4] ERC-173 Ownable specification.
https://eips.ethereum.org/EIPS/eip-173
[5] Arithmetic underflow [Wikipedia Article].
https://en.wikipedia.org/wiki/Arithmetic_underflow

N7 - Lack of explicit reentrancy locks

The project makes significant use of calls to external contracts; those calls are necessary for the final goal of the project and can't be avoided.

Such a high concentration of external calls causes the project to have a high attack surface for reentrancy exploits. Although no such exploits have been found, the use of an explicit reentrancy guard modifier[6] on external methods is recommended, as is the use of the Checks-Effects-Interactions Pattern.

Proposed solution:

a) Add explicit reentrancy guard modifiers on all external methods.

N8 - Signature nonce offset is hardcoded

The contracts `ERC1155Meta` and `ERC1155MetaPackedBalance` implement a signed nonce, which is encoded on the `_sig` bytes array, on the 32 bytes following the byte 65.

The method `_signatureValidation()` hardcodes the offset when reading the encoded `nonce` as 65. To achieve better readability, consider placing this value on a constant.

N9 - `_safeTransferFrom` call doesn't depend on `_isGasFee`

The methods `_safeBatchTransferFrom` and `_safeTransferFrom` on the `ERC1155Meta` contract are being called on separated statements depending on `_isGasFee`, but those methods are not dependent on `_isGasFee` neither it's their order of execution.

Consider placing `_safeBatchTransferFrom` and `_safeTransferFrom` before `if(_isGasFee)` to improve readability.

N10 - ERC-1155 callback logic uses duplicated code

The `ERC1155Meta` contract limits the gas limit passed when calling the ERC-1155 callback; this is intended to avoid griefing attacks from the recipient.

To call `onERC1155Received()` passing a fixed `gasLimit` value the internal method `_callonERC1155Received` is avoided, and instead, the same logic is directly coded on the methods `metaSafeTransferFrom` and `metaSafeBatchTransferFrom`.

Such code is repeated, and it could be replaced with a common internal method.

[6] Reentrancy guard modifier by OpenZeppelin.
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/ReentrancyGuard.sol

Proposed solutions:

a) Add `gasLimit` as one of the parameters of `_callonERC1155Received` on the ERC-1155 contract.

b) Create an internal method on `ERC1155Meta` that allows passing a custom `gasLimit`.

N11 - `ERC1155` don't inherit from ERC-1155 Interface

The implementations of ERC-1155 (`ERC1155.sol` and `ERC1155PackedBalance.sol`) don't inherit from the defined interface `IERC1155.sol`.

Ignoring the interface increases the likelihood of one of such implementations not being compliant with the ERC-1155 specification and also causes the implementations to redefine the events to emit.

N12 - Paying fees with ERC-20 requires fully compliant tokens

The `ERC1155Meta` contract provides a mechanism to reimburse the transaction fee to the relayer. This reimbursement can be performed using an ERC-20 or ERC-1155 token.

In the case of ERC-20, the reimbursement mechanism requires the ERC-20 token to return `true` if the `transferFrom` operation is successful.

Giving the fact that not all ERC-20 tokens are fully compliant with the ERC-20 specification, consider using an alternative mechanism to validate if the ERC-20 transfer was successful.

N13 - `msg.sender` is defined as operator during `_mint` and `_burn`

The `ERC1155MintBurn` and `ERC1155PackedMintBurn` contracts define the `msg.sender` of the call as the `_operator` when performing any `_mint` or `_burn` operation.

Although the ERC-1155 specification doesn't specify what value the `_operator` should take during such operations, it can't be assumed that `msg.sender` is always a close-enough choice for any contract using those methods.

Proposed solution:

a) Add `_operator` as one of the input parameters of `_mint`, `_burn`, `_batchMint` and, `_batchBurn`.

N14 - Commented code enforces wrong requirement

The method `safeTransferFrom` on the `ERC1155` contract contains a commented out require statement validating if the balance of the sender is enough to perform the transfer. This line is commented out stating that such validation is performed using `SafeMath`.

The commented line contains the code `require(_amount >= balances[_from][_id])` which should be `require(_amount <= balances[_from][_id])`.

N15 - SignatureValidator signature types don't match the ERC-2126 draft

The `SignatureValidator` contract utilizes the ERC-2126 signature type recognition standard to identify and validate the provided signatures.

Although the mechanism used is similar, the signature bytes identifier don't match the defined identifiers on the ERC-2126 draft[7].

Additionally, the `SignatureValidator` contract defines `WalletBytes32` as having the signature type `0x04`, but this signature type is not part of the current draft of ERC-2126.

N16 - The projects contain TODO comments

Some `TODO` comments have been found on `MetaERC20Wrapper.sol` and `ERC1155Meta.sol`.

N17 - Hardcoded tokenId 1 as ETH on `MetaERC20Wrapper.sol`

The `MetaERC20Wrapper` contract uses the token ID 1 for wrapping ETH.

ID 1 is not defined as a constant; it is hardcoded each time the contract checks whether given token ID corresponds to ETH; consider placing this value on a constant.

N18 - Avoid using `transfer()` to transfer ETH

The `MetaERC20Wrapper` contract uses `transfer()` to withdraw wrapped ETH, and this Solidity method provides a fixed `gasLimit` of 2300 to the recipient of the ETH.

[7] ERC-2126 Signature types.
https://github.com/ethereum/EIPs/blob/202d578acb76bb4b8d0f46630eff4965ca61c092/EIPS/eip-2126.md#signature-types

Giving the fact that some operations on the EVM may be repriced in the future, this could lead to some external contracts not being able to withdraw the ETH from the `MetaERC20Wrapper` contract; consider using an implicit reentrancy lock instead of `transfer()`[8].

N19 - Unused variable `_data` on `onERC1155Received()`

The `onERC1155Received` method on the `MetaERC20Wrapper` contract ignores the received parameter `_data`; consider removing the variable name to avoid copying such data into memory.

N20 - URI event is declared but never used

The `ERC1155` and `ERC1155PackedBalance` contracts declare the `URI(string, uint256)` event corresponding to the metadata specification of ERC-1155, this event is never used on the contracts, and instead is implemented on `ERC1155Metadata`.

[8] Reentrancy After Istanbul [OpenZeppelin]
https://blog.openzeppelin.com/reentrancy-after-istanbul/

Final thoughts

The contracts composing the audited projects are well written and efficient. The developers have put considerable thought into following existing standards or drafts for meta-transactions, signature validation, ERC-20, and ERC-1155 tokens; such effort should make this set of contracts more compatible with existing and future dApps.

Some critical and high vulnerabilities have been found and must be addressed before deploying the projects in a production environment.

Giving the high complexity of the audited contracts, it is also recommended to evaluate adding some preemptive controls and double-checks, like reentrancy guards and extra asserts.

- December 2019 - Agustín Aguilar