Security Review - Final Report
NM-0053: BRIQ PROTOCOL

NETHERMIND

(Jun 7, 2022)

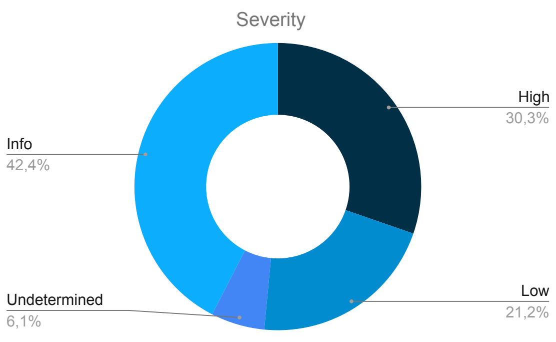
Contents

1 Executive Summary

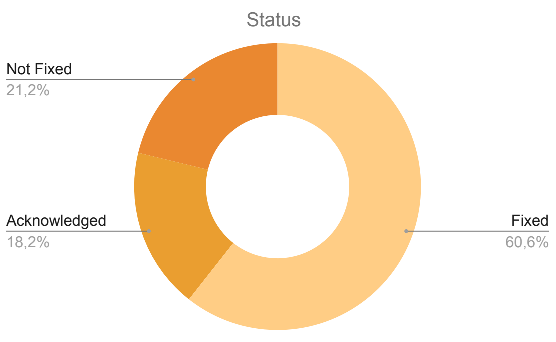
This document presents the security review performed by Nethermind on the Briq Protocol project written in the Cairo Language. The project is composed of 32 contracts (note that not all of these are in-scope for this audit). The Briq Protocol is based on the `ERC-1155 standard` with the ability to combine these tokens together into "sets" which are based on the `ERC-721 standard`. Most of the interactions with the protocol occur using materials (sometimes combined with unique ids), instead of typical token id's.

During the initial audit, we concluded that the project can benefit from better technical documentation detailing a) use cases, b) functional requirements, c) non-functional requirements, d) diagrams showing how interactions take place, and e) a more extensive test suite. The audit was carried out using manual inspection of the code base and the generation of test cases. Most of the reported issues are related to missing validations, overflow risks, and non-defensive code. The initial audit reports 33 findings (as shown in Fig. 1(a)).

During the re-audit phase, the client has **fixed 20 issues**, **acknowledged 6 issues**, and **not fixed 7 issues** (shown in Fig. 1(b)). The clarification for not fixing these issues is presented along this report. Some of these issues require more time to seek a better solution, while others are planned to be fixed during a final code cleanup.



(a)



(b)

High Severity (10), **Low Severity** (7), **Undetermined** (2), **Informational** (14).

Summary of the Audit

Audit Type	Security Review
Initial Report	May 27, 2022
Response from Client	Jun 3, 2022
Final Report	Jun 7, 2022
Methods	Manual Review, Creation of Test Cases
Repository	https://github.com/briqNFT/briq-protocol
Commit Hash (Initial Audit)	fa896be1423924fa0a535db9f53110bb7a8d8745
Commit Hash (Final Audit)	5368018c9467955afe3a83271fe2a8bd0e008274
Documentation	https://github.com/briqNFT/briq-protocol/tree/next/docs/briq

Cairo Files Reviewed (Initial Review)

```
contracts/briq_erc1155_like/*
contracts/library_erc721/*
contracts/set_erc721/*
contracts/upgrades/*
contracts/utilities/*
contracts/briq_impl.cairo
contracts/set_impl.cairo
contracts/types.cairo
```

2 Summary of Findings

Finding	Severity	Update
Potential overflows in `mutateFT_(...)`	High	Fixed
A non-fungible Briq token with `token_id` zero can be minted	High	Fixed
Potential balance reduction and/or overflow when minting fungible Briq tokens	High	Fixed
Fungible Briq tokens can be mutated to have a `material` of zero	High	Fixed
Non-fungible Briq tokens can be mutated to have a `token_id` of zero	High	Fixed
No balance check on converting Briq tokens from fungible to non-fungible	High	Fixed
Missing input validation in `convertOneToNFT_(...)`	High	Fixed
Missing input validation in `transferOneNFT_(...)`	High	Fixed
Missing input validation in `transferFrom_(...)`	High	Fixed
Not checking for overflow in `_transfer(...)`	High	Fixed
Missing events emission in `balance_enumerability.cairo`	Low	Acknowledged
Missing events emission in `minting.cairo`	Low	Acknowledged
Missing events emission in `transferability.cairo`	Low	Acknowledged
Missing input validation in `setSetAddress_(...)`	Low	Not Fixed
No event emitted when updating `RootAdmin`	Low	Fixed
Functions missing events emission in `approvals.cairo`	Low	Not Fixed
Unclear use case in `ownerOf_(...)`	Undetermined	Fixed
New `uid` can be same as existing `uid` when mutating non-fungible Briq tokens	Undetermined	Fixed
Privileged address can transfer on behalf of all users	Info	Acknowledged
Failing asserts without `with_attr`	Info	Not Fixed
Unused imports in `convert_mutate.cairo`	Info	Not Fixed
Missing event emission in `mutateFT_(...)`	Info	Acknowledged
Incorrect contract filename	Info	Fixed
The privileged contract `_mint_contract` can be set to `address(0x0)`	Info	Acknowledged
Missing overflow checks in `transferFT_(...)`	Info	Unresolved
Briq token transfers can have the same `sender` and `receiver`	Info	Fixed
Unused implicit arguments	Info	Not Fixed
Unused imports in `transferability.cairo`	Info	Not Fixed
Incorrect namespace title	Info	Fixed
Functions with `view` visibility making state changes	Info	Fixed
No event emitted and missing input validation in `proxy.constructor(...)`	Info	Fixed
Test function implemented inside core contract	Info	Not Fixed
Not using boolean type of value fixed to `0` or `1`	Info	Fixed

3 Findings

3.1 General findings

3.1.1 [Info] Failing asserts without `with_attr`

File(s): `contracts/*`

Description: Failing asserts should use `with_attr` to have an error message on failure to make end users understand what may be happening when transactions fail.

Recommendation: Use `with_attr` to add error messages to asserts that can be expected to fail during normal use of the protocol.

Status: Not Fixed.

Update from client: Aware, intend to fix some incrementally.

3.1.2 [Info] Incorrect contract filename

File(s): `contracts/utilities/UInt256_felt_conv.cairo`

Description: The name of the file `UInt256_felt_conv.cairo` is incorrect, it should be named `Uint256_felt_conv.cairo`. This filename issue causes the test `tests/set_impl_test.py` to fail.

Recommendation: Rename the file `UInt256_felt_conv.cairo` to `Uint256_felt_conv.cairo`.

Status: Fixed in commit c8b533d9bfbcb32fc9ddaaf371e2cf1852c2d888.

Update from client: Mac Os is case insensitive which is why I missed this.

3.1.3 [Info] Privileged address can transfer on behalf of all users

File(s): `contracts/briq_erc1155_like/transferability.cairo`

Description: The privileged address `_set_backend_address` is able to control all funds on behalf of any user. Although the current protocol design relies on this feature, there are no safety features in place in the event that `_set_backend_address` is set to an incorrect or malicious address, which can lead to asset losses.

Recommendation: Clearly state in the user facing documentation the privileges, roles, and functions that can be performed by each entity of the application. In the interest of both the users and protocol owners it is recommended to have safety features in place to prevent an incorrect or dangerous address from being set.

Status: Acknowledged.

Update from client: Documentation will need to be updated. Not sure on possible mitigation.

3.2 contracts/briq_erc1155_like/balance_enumerability.cairo

3.2.1 [Low] Missing events emission in `balance_enumerability.cairo`

File(s): `contracts/briq_erc1155_like/balance_enumerability.cairo`

Description: The contract is not emitting events for important state transitions. The events are important for post-deployment monitoring. A list of functions not emitting events is presented below.

```
func _setTokenByOwner(...)
func _unsetTokenByOwner(...)
func _setMaterialByOwner(...)
func _maybeUnsetMaterialByOwner(...)
```

Recommendation: Emit events for post-deployment monitoring.

Status: Acknowledged.

Update from client: Likely a WONTFIX for now - these can be reconstructed from other events, such as Transfer.

3.2.2 [Undetermined] Unclear use case in `ownerOf_(...)`

File(s): `contracts/briq_erc1155_like/balance_enumerability.cairo`

Description: The function `balance_enumerability.ownerOf_(...)` has the following inline comment.

```
    # OZ: don't fail on res == 0, since 0 is impossible.
```

It is unclear why `res==0` is impossible. Since this is a `view` function, it can be called by anyone with any input parameter passed as `token_id`, including non-existent tokens. The code is reproduced below.

```
@view
func ownerOf_{
        syscall_ptr: felt*,
        pedersen_ptr: HashBuiltin*,
        range_check_ptr
    } (token_id: felt) -> (owner: felt):
    let (res) = _owner.read(token_id)
    # OZ : don't fail on res == 0, since 0 is impossible.
    return (res)
end
```

Recommendation: Please, clarify this use case and explain why `res` cannot be zero.

Status: Fixed in commit 963350a65badb78cc8840fa734dee83743af39c9.

Update from client: This is an unclear comment. My note means that unlike the OZ / ERC1155 standard, I don't assert 0, I just return 0. Value '0' is impossible for a briq token to legitimately take, and I don't care to assert 0. Changed the comment.

3.3 contracts/briq_erc1155_like/minting.cairo

3.3.1 [High] A non-fungible Briq token with `token_id` zero can be minted

File(s): `contracts/briq_erc1155_like/minting.cairo`

Description: The function `mintOneNFT_(...)` does not have input validation to ensure that `material` is within the valid range of `1` to `2^64 - 1`. The `token_id` to be minted is calculated with the formula `token_id = id * 2**64 + material` which means that it is possible to set `material` and `uid` to values that satisfy `material == (uid * 2**64) * -1` which will result in a `token_id` of zero, as shown in the code below:

```
@external
func minting.mintOneNFT_{
        syscall_ptr: felt*,
        pedersen_ptr: HashBuiltin*,
        range_check_ptr
    } (owner: felt, material: felt, uid: felt):
    _onlyAdminAndMintContract()

    assert_not_zero(owner)
    assert_not_zero(material)
    assert_not_zero(uid)
    assert_lt_felt(uid, 2**188)

    # Update total supply.
    let (res) = _total_supply.read(material)
    _total_supply.write(material, res + 1)

    # NFT conversion
    let briq_token_id = uid * 2**64 + material

    let (curr_owner) = _owner.read(briq_token_id)
    assert curr_owner = 0
    _owner.write(briq_token_id, owner)

    _setMaterialByOwner(owner, material, 0)
    _setTokenByOwner(owner, material, briq_token_id, 0)

    let (__addr) = get_contract_address()
    TransferSingle.emit(__addr, 0, owner, briq_token_id, 1)

    return ()
end
```

The following test(s) can be added to `tests/briq_impl_test.py` to verify this behavior:

```
@pytest.mark.asyncio
async def test_mint_token_id_zero(briq_contract):

    # The specific values that `material` and `uid` must be to create a token id of zero
    maliciousMaterial = -abs(2**64)
    maliciousUid = 1
    maliciousTokenId = maliciousUid * 2**64 + maliciousMaterial

    # Mint an NFT token
    await invoke_briq(briq_contract.mintOneNFT(owner=OTHER_ADDRESS, material=maliciousMaterial, uid=maliciousUid))

    # Get the owner of token id `0`
    zeroTokenOwner = (await briq_contract.ownerOf(0).call()).result.owner

    # Assert that the owner of the zero token is now assigned to an address
    assert zeroTokenOwner == OTHER_ADDRESS
```

It is only possible to create one of these tokens as any future mints check the owner of the "zerotoken" which will return a non-zero value. This "zerotoken" cannot be transferred or enumerated in balances as logic for checking balances relies on the `token_id` of zero to end recursion. However it permanently affects the user's `_materials_by_owner` data as the material associated with the "zerotoken" cannot be removed.

Recommendation: The function must validate that `material` in within the range of `1` to `2^64 - 1`. Check for overflows on `briq_token_id`.

Status: Fixed in commit 7ebfab22047264146b26b6bfeed3b55218c94598.

Update from client: Fixed in MintOneNFT_. A similar problem existed in MintFT_ that led to a weird state. Both fixed.

3.3.2 [High] Potential balance reduction and/or overflow when minting fungible Briq tokens

File(s): `contracts/briq_erc1155_like/minting.cairo`

Description: The function `mintFT_(...)` does not check for overflows on the variable `balance` before writing to storage. Depending on the existing balance of `owner` different behavior will be observed. It is possible to use `mintFT(...)` to reduce a users balance and it is also possible to overflow a users balance to wrap around from zero to the Starknet `P` prime value. The code is reproduced below:

```
@external
func minting.mintFT_{
        syscall_ptr: felt*,
        pedersen_ptr: HashBuiltin*,
        range_check_ptr
    } (owner: felt, material: felt, qty: felt):
    _onlyAdminAndMintContract()

    assert_not_zero(owner)
    assert_not_zero(material)
    assert_not_zero(qty)
    # Update total supply.
    let (res) = _total_supply.read(material)
    _total_supply.write(material, res + qty)

    # FT conversion
    let briq_token_id = material
    let (balance) = _balance.read(owner, briq_token_id)
    _balance.write(owner, briq_token_id, balance + qty)
    _setMaterialByOwner(owner, material, 0)
    let (__addr) = get_contract_address()
    TransferSingle.emit(__addr, 0, owner, briq_token_id, qty)
    return ()
end
```

The following test(s) can be added to `tests/briq_impl_test.py` to verify this behavior:

```
@pytest.mark.asyncio
async def test_mint_fungible_overflow_reduce_balance(briq_contract):

    # Mint 100 fungible tokens to `ADDRESS`
    await invoke_briq(briq_contract.mintFT(owner=ADDRESS, material=1, qty=100))

    # Record the balance before
    balanceBefore = (await briq_contract.balanceOfMaterial(owner=ADDRESS, material=1).call()).result.balance

    # Mint -10 fungible tokens to `ADDRESS`
    await invoke_briq(briq_contract.mintFT(owner=ADDRESS, material=1, qty=-10))

    # Record the balance after
    balanceAfter = (await briq_contract.balanceOfMaterial(owner=ADDRESS, material=1).call()).result.balance

    # Calling mint with an extremely large value causes an overflow which reduces the balance
    assert balanceAfter < balanceBefore
```

```
@pytest.mark.asyncio
async def test_mint_fungible_overflow_from_empty_balance(briq_contract):

    # Mint -10 fungible tokens to `ADDRESS`
    await invoke_briq(briq_contract.mintFT(owner=ADDRESS, material=1, qty=-10))

    # Record the balance
    balance = (await briq_contract.balanceOfMaterial(owner=ADDRESS, material=1).call()).result.balance

    # Balance is extremely large
    # The value of balance will specifically be the `P` prime value of Starknet minus 10
    assert balance > 1000000000000000000000000000000000000000000000000000000000000
```

Recommendation: Assert that the argument `qty` in the function `mintFT_(...)` is a positive number.

Status: Fixed in commit 757870a8b1db3cdd3178cf13cc0e2d60b7f4c737.

Update from client: The balance reduction issue is fixed. A similar issue existed for total supply for both FT and NFT, and has been fixed. The problem of minting negative numbers is left in but explicited, since this is just an interface issue (equivalent to minting a huge amount of FT).

3.3.3 [Low] Missing events emission in `minting.cairo`

File(s): `contracts/briq_erc1155_like/minting.cairo`

Description: The contract does not emit events for important state transitions. Events are important for post-deployment monitoring. A list of functions with missing events is presented below:

```
func setMintContract_(...)
func mintFT_(...)
func mintOneNFT_(...)
```

Recommendation: Emit events for these functions.

Status: Acknowledged.

Update from client: `SetMintContract` should, the rest is reconstructed from `Transfer()`.

3.3.4 [Info] The privileged contract `_mint_contract` can be set to `address(0x0)`

File(s): `contracts/briq_erc1155_like/minting.cairo`

Description: The function `setMintContract_(...)` allows `_mint_contract` to be set to `address(0x0)`. The code is reproduced below:

```
@external
func minting.setMintContract_{
        syscall_ptr: felt*,
        pedersen_ptr: HashBuiltin*,
        range_check_ptr
    } (address: felt):
    _onlyAdmin()
    _mint_contract.write(address)
    return ()
end
```

Recommendation: If this is intended functionality designed to disable minting from the `_mint_contract` address, it is recommended to create a separate function specifically for halting minting from `_mint_contract`. Otherwise, it is recommended to check that the argument `address` is not `address(0x0)`.

Status: Acknowledged.

Update from client: The contract will be removed in the genesis sale, so the code will be removed. As things stand, it is indeed just disabled and just looks odd.

3.4 contracts/briq_erc1155_like/convert_mutate.cairo

3.4.1 [High] Potential overflows in `mutateFT_(...)`

File(s): `contracts/briq_erc1155_like/convert_mutate.cairo`

Description: In the function `mutateFT_(...)` the storage variables `_balance` and `_total_supply` are written without any checks to ensure that no overflows can occur. The function is reproduced below:

```
@external
@external
func mutateFT_{
        syscall_ptr: felt*,
        pedersen_ptr: HashBuiltin*,
        range_check_ptr
    } (owner: felt, source_material: felt, target_material: felt, qty: felt):
    _onlyAdmin()

    assert_not_zero(qty * (source_material - target_material))

    let (balance) = _balance.read(owner, source_material)
    assert_le_felt(qty, balance)
    _balance.write(owner, source_material, balance - qty)

    let (balance) = _balance.read(owner, target_material)
    _balance.write(owner, target_material, balance + qty)

    let (res) = _total_supply.read(source_material)
    _total_supply.write(source_material, res - qty)

    let (res) = _total_supply.read(target_material)
    _total_supply.write(target_material, res + qty)

    _setMaterialByOwner(owner, target_material, 0)
    _maybeUnsetMaterialByOwner(owner, source_material)

    let (__addr) = get_contract_address()
    TransferSingle.emit(__addr, owner, 0, source_material, qty)
    TransferSingle.emit(__addr, 0, owner, target_material, qty)

    return ()
end
```

Recommendation: Check for overflows before writing to storage. In case of overflow, revert.

Status: Fixed in commit 5368018c9467955afe3a83271fe2a8bd0e008274.

3.4.2 [High] Fungible Briq tokens can be mutated to have a `material` of zero

File(s): `contracts/briq_erc1155_like/convert_mutate.cairo`

Description: The function `mutateFT_(...)` does not assert that the argument `target_material` is not zero and allows for fungible Briq tokens to be mutated to have a material of zero. The function is reproduced below:

```
@external
func mutateFT_{
        syscall_ptr: felt*,
        pedersen_ptr: HashBuiltin*,
        range_check_ptr
    } (owner: felt, source_material: felt, target_material: felt, qty: felt):
    _onlyAdmin()

    assert_not_zero(qty * (source_material - target_material))

    let (balance) = _balance.read(owner, source_material)
    assert_le_felt(qty, balance)
    _balance.write(owner, source_material, balance - qty)

    let (balance) = _balance.read(owner, target_material)
    _balance.write(owner, target_material, balance + qty)

    let (res) = _total_supply.read(source_material)
    _total_supply.write(source_material, res - qty)

    let (res) = _total_supply.read(target_material)
    _total_supply.write(target_material, res + qty)

    _setMaterialByOwner(owner, target_material, 0)
    _maybeUnsetMaterialByOwner(owner, source_material)

    let (__addr) = get_contract_address()
    TransferSingle.emit(__addr, owner, 0, source_material, qty)
    TransferSingle.emit(__addr, 0, owner, target_material, qty)

    return ()
end
```

The following test(s) can be added to `tests/briq_impl_test.py` to verify this behavior:

```
@pytest.mark.asyncio
async def test_mutate_ft_material_zero(briq_contract):

    # Mint 100 fungible tokens of material `1` to `ADDRESS`
    await invoke_briq(briq_contract.mintFT(owner=ADDRESS, material=1, qty=100))

    # Mutate 10 of the tokens to a new material with id zero
    await invoke_briq(briq_contract.mutateFT(owner=ADDRESS, source_material=1, target_material=0, qty=10))

    # Get the balance of `ADDRESS` for material with id zero
    balanceMaterialZero = (await briq_contract.balanceOfMaterial(owner=ADDRESS, material=0).call()).result.balance

    assert balanceMaterialZero == 10
```

Recommendation: Assert that the argument `target_material` in the function `mutateFT_(...)` cannot be zero.

Status: Fixed in commit 5368018c9467955afe3a83271fe2a8bd0e008274.

3.4.3 [High] Non-fungible Briq tokens can be mutated to have a `token_id` of zero

File(s): `contracts/briq_erc1155_like/convert_mutate.cairo`

Description: In the function `mutateOneNFT_(...)` after `briq_token_id` has been calculated for the second time there is no check to ensure that the value is zero. The variable `briq_token_id` is calculated with the formula `briq_token_id = uid * 2**64 + target_material` which means that it is possible to set `target_material` and `uid` to values that satisfy `target_material == (uid * 2**64) * -1` which will result in a `briq_token_id` of zero. The code is reproduced below:

```
@external
func mutateOneNFT_{
        syscall_ptr: felt*,
        pedersen_ptr: HashBuiltin*,
        range_check_ptr
    } (owner: felt, source_material: felt, target_material: felt, uid: felt, new_uid: felt):
    _onlyAdmin()

    assert_lt_felt(uid, 2**188)
    assert_lt_felt(new_uid, 2**188)
    assert_not_zero(source_material - target_material)

    # NFT conversion
    let (res) = _total_supply.read(source_material)
    _total_supply.write(source_material, res - 1)

    let briq_token_id = uid * 2**64 + source_material

    let (curr_owner) = _owner.read(briq_token_id)
    assert curr_owner = owner
    _owner.write(briq_token_id, 0)

    _unsetTokenByOwner(owner, source_material, briq_token_id)
    _maybeUnsetMaterialByOwner(owner, source_material) # Keep after unset token or it won't unset

    let (res) = _total_supply.read(target_material)
    _total_supply.write(target_material, res + 1)

    # briq_token_id is not the new ID
    let briq_token_id = new_uid * 2**64 + target_material

    let (curr_owner) = _owner.read(briq_token_id)
    assert curr_owner = 0
    _owner.write(briq_token_id, owner)

    _setMaterialByOwner(owner, target_material, 0)
    _setTokenByOwner(owner, target_material, briq_token_id, 0)

    let (__addr) = get_contract_address()
    TransferSingle.emit(__addr, owner, 0, uid * 2**64 + source_material, 1)
    TransferSingle.emit(__addr, 0, owner, new_uid * 2**64 + target_material, 1)
    Mutate.emit(owner, uid * 2**64 + source_material, new_uid * 2**64 + target_material, source_material,
    ↪   target_material)

    return ()
end
```

The following test(s) can be added to `briq_impl_test.py` to verify this behavior:

```python
@pytest.mark.asyncio
async def test_mutate_one_nft_token_id_zero(briq_contract):

    # Values for the NFT before the mutation
    beforeUid = 1
    beforeMaterial = 1
    beforeTokenId = beforeUid * 2**64 + beforeMaterial

    # Mint an NFT token
    await invoke_briq(briq_contract.mintOneNFT(owner=ADDRESS, material=beforeMaterial, uid=beforeUid))

    # The specific values needed to have a token id of zero after mutate
    afterUid = 2
    afterMaterial = -abs(2 * (2**64))
    afterTokenId = afterUid * 2**64 + afterMaterial

    # Mutate the NFT token
    await invoke_briq(briq_contract.mutateOneNFT(owner=ADDRESS, source_material=beforeMaterial,
    ↪  target_material=afterMaterial, uid=beforeUid, new_uid=afterUid))

    # Get the owner of token id `0`
    zeroTokenOwner = (await briq_contract.ownerOf(0).call()).result.owner

    # Assert that the owner of the zero token is now assigned to an address
    assert zeroTokenOwner == ADDRESS
```

Recommendation: The function must validate that both `source_material` and '`target_material` are within the range of `1` to `2^64 - 1`. Add a check to `briq_token_id` before writing to storage to ensure that the value is not zero.

Status: Fixed in commit 5368018c9467955afe3a83271fe2a8bd0e008274.

3.4.4 [High] No balance check on converting Briq tokens from fungible to non-fungible

File(s): `contracts/briq_erc1155_like/convert_mutate.cairo`

Description: The function `convertOneToNFT_(...)` does not ensure that the fungible token balance of `owner` is sufficient. This allows a non-fungible Briq token to be created even if `owner` has no fungible tokens of the given material. The process of reducing the fungible token balance for the owner whose balance is zero will also cause an overflow upon writing to `_balance` which will change their fungible token balance to `STARKNET_PRIME - 1`. The code is shown below:

```
@external
func convertOneToNFT_{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*, range_check_ptr
    } (owner: felt, material: felt, uid: felt):
    _onlyAdmin()

    assert_not_zero(owner)
    assert_not_zero(material)
    assert_lt_felt(uid, 2**188)

    # NFT conversion
    let token_id = uid * 2**64 + material

    let (curr_owner) = _owner.read(token_id)
    assert curr_owner = 0
    _owner.write(token_id, owner)

    # No need to change material
    _setTokenByOwner(owner, material, token_id, 0)

    let (balance) = _balance.read(owner, material)
    _balance.write(owner, material, balance - 1)

    let (__addr) = get_contract_address()
    TransferSingle.emit(__addr, owner, 0, material, 1)
    TransferSingle.emit(__addr, 0, owner, token_id, 1)
    ConvertToNFT.emit(owner, material, token_id)

    return ()
end
```

The following test(s) can be added to `briq_impl_test.py` to verify this behavior:

```
@pytest.mark.asyncio
async def test_convert_ft_to_nft_no_balance(briq_contract):

    # Convert immediately without minting any tokens
    await invoke_briq(briq_contract.convertOneToNFT(owner=ADDRESS, material=1, uid=1))

    # Account `ADDRESS` now has a balance issue so to confirm the NFT was created transfer it to observe the balance on
    ↪  another account
    await invoke_briq(briq_contract.transferOneNFT(sender=ADDRESS, recipient=OTHER_ADDRESS, material=1 ,
    ↪  briq_token_id=1 * 2**64 + 1), ADDRESS)

    # Get the balance of the address that received the newly created NFT
    materialBalance = (await briq_contract.balanceOfMaterial(owner=OTHER_ADDRESS, material=1).call()).result.balance

    # Get the balance of fungible tokens owned by `ADDRESS`
    fungibleBalance = (await briq_contract.balanceDetailsOfMaterial_(owner=ADDRESS,
    ↪  material=1).call()).result.ft_balance

    # Balance of `ADDRESS` is now a very large value
    assert fungibleBalance == 3618502788666131213697322783095070105623107215331596699973092056135872020480

    # The NFT has successfully been transferred to another address
    assert materialBalance == 1
```

Recommendation: Ensure that the fungible token balance for `owner` is enough to convert a fungible token into a non-fungible token.

Status: Fixed in commit 5368018c9467955afe3a83271fe2a8bd0e008274.

3.4.5 [High] Missing input validation in `convertOneToNFT_(...)`

File(s): `contracts/briq_erc1155_like/convert_mutate.cairo`

Description: The parameters `material` and `uid` are not checked for range and validity. There are also no checks for overflow. The code is presented below.

```
@external
func convertOneToNFT_{
        syscall_ptr: felt*,
        pedersen_ptr: HashBuiltin*,
        range_check_ptr
    } (owner: felt, material: felt, uid: felt):
    _onlyAdmin()

    assert_not_zero(owner)
    assert_not_zero(material)
    assert_lt_felt(uid, 2**188)

    # NFT conversion
    let token_id = uid * 2**64 + material

    let (curr_owner) = _owner.read(token_id)
    assert curr_owner = 0
    _owner.write(token_id, owner)

    # No need to change material
    _setTokenByOwner(owner, material, token_id, 0)

    let (balance) = _balance.read(owner, material)
    _balance.write(owner, material, balance - 1)

    let (__addr) = get_contract_address()
    TransferSingle.emit(__addr, owner, 0, material, 1)
    TransferSingle.emit(__addr, 0, owner, token_id, 1)
    ConvertToNFT.emit(owner, material, token_id)

    return ()
end
```

Recommendation: Check `material` and `uid` for range and validity. Check for overflow. In case of error, revert.

Status: Fixed in commit 5368018c9467955afe3a83271fe2a8bd0e008274.

3.4.6 [Undetermined] New uid can be same as existing uid when mutating non-fungible Briq tokens

File(s): contracts/briq_erc1155_like/convert_mutate.cairo

Description: On L78 there is a comment that states that new_uid should be different to uid however this is not enforced in the function mutateOneNFT_(...) and it is possible to mutate a non-fungible Briq token to have a different material and the same uid. This has been set to an undetermined severity because the reason for changing the uid is that the UI can potentially conflict, however we do not know the full impact of conflicting uids on the front-end for the project. The following test(s) can be added to briq_impl_test.py to verify this behavior:

```
@pytest.mark.asyncio
async def test_mutate_one_nft_same_uid(briq_contract):

    # Values for the NFT before the mutation
    beforeUid = 1
    beforeMaterial = 1
    beforeTokenId = beforeUid * 2**64 + beforeMaterial

    # Mint an NFT token
    await invoke_briq(briq_contract.mintOneNFT(owner=ADDRESS, material=beforeMaterial, uid=beforeUid))

    # The specific values needed to have a token id of zero after mutate
    afterUid = beforeUid
    afterMaterial = 2
    afterTokenId = afterUid * 2**64 + afterMaterial

    # Mutate the NFT token
    await invoke_briq(briq_contract.mutateOneNFT(owner=ADDRESS, source_material=beforeMaterial,
    ↪  target_material=afterMaterial, uid=beforeUid, new_uid=afterUid))

    # No asserts needed, if this test passes then the NFT token has successfully been mutated while having the same uid
```

Recommendation: Add a check to ensure that uid and new_uid are not the same.

Status: Fixed in commit 5368018c9467955afe3a83271fe2a8bd0e008274.

Update from client: I think this is a broken comment, actually. The general behavior is that new_uid == uid, and that is expected to work. However, since there can be a token_id with target_material and new_uid already, there is an option to pass a new_uid different to uid. The comment is rather unclear, but this is all working. I've rephrased.

3.4.7 [Info] Unused imports in convert_mutate.cairo

File(s): contracts/briq_erc1155_like/convert_mutate.cairo

Description: The following imports are not used in the file convert_mutate.cairo:

- SignatureBuiltin
- get_caller_address
- assert_nn_le
- assert_lt
- assert_le
- get_fp_and_pc
- alloc

Recommendation: Remove the unused imports.

Status: Not Fixed.

Update from client: Will clean up at a later stage.

3.4.8 [Info] Missing event emission in mutateFT_(...)

File(s): contracts/briq_erc1155_like/convert_mutate.cairo

Description: The function mutateFT_(...) does not emit a Mutate event like other mutate function mutateOneNFT_(...).

Recommendation: Add an event emission for calls to mutateFT_(...).

Status: Acknowledged.

Update from client: This was on purpose, the tokens being fungible, mutation is equivalent to burning and minting a corresponding amount (whereas we want to track NFTs, potentially). Having a dedicated event could still be useful, I'll consider it.

3.5 contracts/briq_erc1155_like/transferability.cairo

3.5.1 [High] Missing input validation in `transferOneNFT_(...)`

File(s): `contracts/briq_erc1155_like/transferability.cairo`

Description: The function `transferOneNFT_(...)` does not validate the input parameters `recipient`, `material`, and `briq_token_id`. The input parameters `material` and `briq_token_id` are not checked to be within a valid range. The `recipient` is not checked for `address(0x0)`. The code is reproduced below.

```
@external
func transferOneNFT_{
        syscall_ptr: felt*,
        pedersen_ptr: HashBuiltin*,
        range_check_ptr
    } (sender: felt, recipient: felt, material: felt, briq_token_id: felt):
    _onlySetAnd(sender)

    assert_not_zero(sender)
    assert_not_zero(material)
    assert_not_zero(briq_token_id)

    let (curr_owner) = _owner.read(briq_token_id)
    assert sender = curr_owner
    _owner.write(briq_token_id, recipient)

    # Unset before setting, so that self-transfers work.
    _unsetTokenByOwner(sender, material, briq_token_id)
    _setTokenByOwner(recipient, material, briq_token_id, 0)

    _maybeUnsetMaterialByOwner(sender, material) # Keep after unset token or it won't unset
    _setTokenByOwner(recipient, material, briq_token_id, 0)

    let (__addr) = get_contract_address()
    TransferSingle.emit(__addr, sender, recipient, briq_token_id, 1)

    return ()
end
```

Recommendation: The function must check `recipient` for `address(0x0)`. It must also assure that `material` and `briq_token_id` are valid identifiers.

Status: Fixed in commit d3938b49f2b742a38f33de8788045478e386a455.

Update from client: Asserted recipient and material / briq_token_id accordingly.

3.5.2 [Low] Missing events emission in `transferability.cairo`

File(s): `contracts/briq_erc1155_like/transferability.cairo`

Description: The contract is not emitting events for important state transitions. The events are important for post-deployment monitoring. A list of functions that should emit events is presented below.

```
func setSetAddress_(...)
func transferFT_(...)
```

Recommendation: Consider emitting events for the functions listed above.

Status: Acknowledged.

Update from client: `SetSetAddress` should, the rest are reconstructed from `Transfer()`.

3.5.3 [Low] Missing input validation in `setSetAddress_(...)`

File(s): `contracts/briq_erc1155_like/transferability.cairo`

Description: The function `setSetAddress_(...)` does not validate the input parameter `address`. Although the function can only be called by the admin, the function can be more robust. Users also need protection from admin actions. Moreover, the function also does not emit an event. The code is reproduced below:

```
@external
func setSetAddress_{
        syscall_ptr: felt*,
        pedersen_ptr: HashBuiltin*,
        range_check_ptr
    } (address: felt):
    _onlyAdmin()
    _set_backend_address.write(address)
    return ()
end
```

Recommendation: The function must assert that the input parameter `address` is different from `address(0x0)`. The function should also emit an event.

Status: Not Fixed.

Update from client: Similar class of problem to other privileged functions. Not entirely sure how to proceed at the moment.

3.5.4 [Info] Missing overflow checks in `transferFT_(...)`

File(s): `contracts/briq_erc1155_like/transferability.cairo`

Description: The function `transferFT_(...)` does not explicitly check for overflows when updating the senders and receivers balance. The code is reproduced below:

```
@external
func transferFT_{
        syscall_ptr: felt*,
        pedersen_ptr: HashBuiltin*,
        range_check_ptr
    } (sender: felt, recipient: felt, material: felt, qty: felt):
    _onlySetAnd(sender)

    assert_not_zero(sender)
    assert_not_zero(material)
    assert_not_zero(qty)

    # FT conversion
    let briq_token_id = material

    let (balance_sender) = _balance.read(sender, briq_token_id)
    assert_le_felt(qty, balance_sender)
    _balance.write(sender, briq_token_id, balance_sender - qty)

    let (balance) = _balance.read(recipient, briq_token_id)
    _balance.write(recipient, briq_token_id, balance + qty)

    _setMaterialByOwner(recipient, material, 0)
    _maybeUnsetMaterialByOwner(sender, material)

    let (__addr) = get_contract_address()
    TransferSingle.emit(__addr, sender, recipient, briq_token_id, qty)

    return ()
end
```

Recommendation: Check for overflows before writing to `_balance` for both the sender and receiver.

Status: Fixed in commit 4052be9793f5937ba7cb0bf7f440b2e885584bad.

Update from client: Added check. Interestingly, `balanceOf_`'s result can overflow from too many FT and NFT at once. I suppose I can't really fix that too easily here.

3.5.5 [Info] Briq token transfers can have the same sender and receiver

File(s): contracts/briq_erc1155_like/transferability.cairo

Description: The functions transferFT_(...) and transferOneNFT_(...) allow the sender and receiver address to be the same. Currently this doesn't introduce any bugs, however given that this protocol is still under development any changes may create bugs in the future.

Recommendation: During the discussion between the Nethermind and Briq teams it was mentioned that self transfers are not necessary for the protocol and do not benefit the user in any way. Since self transfers are not necessary for the protocol it is recommended to remove this feature as any changes to the protocol may introduce bugs related to self transfers in the future.

Status: Fixed in commit 1c5499117057dadd1e2b33fe74c8ec19e5e003f2.

Update from client: Asserted against.

3.5.6 [Info] Unused imports in transferability.cairo

File(s): contracts/briq_erc1155_like/transferability.cairo

Description: The following imports are not used in the file transferability.cairo:

– assert_nn_le
– assert_lt
– assert_lt_felt
– get_fp_and_pc
– alloc

Recommendation: Remove the unused imports.

Status: Not Fixed.

Update from client: Planning to do a cleanup pass.

3.6 contracts/library_erc721/transferability_enum.cairo

3.6.1 [High] Missing input validation in transferFrom_(...)

File(s): contracts/library_erc721/transferability_enum.cairo

Description: The function transferFrom_(...) does not check if the token_id is within a valid range, the sender and the recipient for address(0x0). It also does not check if sender is different from recipient. The function is reproduced below.

```
@external
func transferFrom_{
        syscall_ptr: felt*,
        pedersen_ptr: HashBuiltin*,
        range_check_ptr
    } (sender: felt, recipient: felt, token_id: felt):
    ERC721_approvals._onlyApproved(sender, token_id)

    ERC721_lib_transfer._transfer(sender, recipient, token_id)
    # Unset before setting, so that self-transfers work.
    ERC721_enumerability._unsetTokenByOwner(sender, token_id)
    ERC721_enumerability._setTokenByOwner(recipient, token_id, 0)

    return ()
end
```

Recommendation: Validate the token_id, sender, and recipient.

Status: Fixed in commit ab7111e8e8a7bda09f88c49575ac112a2856e196.

Update from client: Added checks in the called _transfer function.

3.6.2 [Info] Incorrect namespace title

File(s): contracts/library_erc721/transferability_enum.cairo

Description: The namespace title ERC271_transferability appears to be a spelled incorrectly, it should be ERC721_transferability. This issue extends into contracts/set_impl.cairo on L27 and L55 where the import also uses the same spelling error.

Recommendation: Rename the namespace to ERC721_transferability and reflect these changes in contracts/set_impl.cairo

Status: Fixed in commit d1f75e56e5f16cd5f6d3535d05f233deacd615c6.

3.7 contracts/set_impl.cairo

3.7.1 [Info] Functions with `view` visibility making changes to state

File(s): `contracts/set_impl.cairo`

Description: The functions `assemble(...)`, `disassemble(...)` and `transferOneNFT(...)` use the `@view` decorator. However, these functions make state changes.

Recommendation: Change the decorators for `assemble(...)`, `disassemble(...)` and `transferOneNFT(...)` to use the `@external` decorator.

Status: Fixed in commit 89fdd2db048985549c3c4af4e999dc47a6914726.

Update from client: Changed to `external`.

3.8 contracts/briq_impl.cairo

3.8.1 [Info] Unused implicit arguments

File(s): `contracts/briq_impl.cairo`

Description: The functions `balanceOf_(...)`, `balanceDetailsOf_(...)`, `multiBalanceOf_(...)`, `totalSupply_(...)` use the implicit argument `bitwise_ptr` which is unused by all functions. The relevant functions are shown below: disa

Recommendation: Remove the implicit argument `bitwise_ptr` from the relevant functions.

Status: Not Fixed.

Update from client: I think I've added them to some of these because called functions use it and it avoids having to set local pointers. I also think that cairo-lang itself might evolve to make these redundant. Will consider in the future a cleanup here.

3.9 contracts/upgrades/proxy.cairo

3.9.1 [Info] No event emitted and missing input validation in `proxy.constructor(...)`

File(s): `contracts/upgrades/proxy.cairo`

Description: The `proxy.constructor(...)` does not emit any event. However, it performs important state changes: a) define the `admin` of the application; b) set the implementation address. The `admin` and the `implementation_address` are not checked for `address(0x0)`. The code is shown below.

```
func constructor{
        syscall_ptr: felt*,
        pedersen_ptr: HashBuiltin*,
        range_check_ptr
    }(admin: felt, implementation_address: felt):
    Proxy_initializer(admin)
    Proxy_set_implementation(implementation_address)
    return ()
end
```

Recommendation: Emit an event in the `proxy.constructor(...)` containing `admin` and the `implementation_address`. This event will be important to check that addresses have been set accordingly. Check `admin` and `implementation_address` for `address(0x0)`.

Status: Fixed in commit 4f03d9c2592427fb81ba6ca15e6fb42649357511.

Update from client: Done on my end of the contracts. Ideally, Open Zeppelin will change their code on that side.

3.10 contracts/upgrades/upgradable_mixin.cairo

3.10.1 [Low] No event emitted when updating RootAdmin

File(s): `contracts/upgrades/upgradable_mixin.cairo`

Description: The function `upgradable_mixin.setRootAdmin_(...)` does not emit any event. It calls the function `library.Proxy_set_admin(...)`, which also does emit any event. Both functions do not check the `new_admin` for `address(0x0)`. The code of both functions are presented below.

```
func upgradable_mixin.setRootAdmin_{
        syscall_ptr: felt*,
        pedersen_ptr: HashBuiltin*,
        range_check_ptr
    } (new_admin: felt):
    _onlyAdmin()
    Proxy_set_admin(new_admin)
    return ()
end
```

```
func library.Proxy_set_admin{
        syscall_ptr: felt*,
        pedersen_ptr: HashBuiltin*,
        range_check_ptr
    }(new_admin: felt):
    Proxy_admin.write(new_admin)
    return ()
end
```

Recommendation: Emit an event in `library.Proxy_set_admin(...)` containing the `new_admin`. Check the `new_admin` of both functions for `address(0x0)`. If the check is `true`, revert.

Status: Fixed in commit 4f03d9c2592427fb81ba6ca15e6fb42649357511.

Update from client: Done on my end of the contracts. Ideally, Open Zeppelin will change their code on that side.

3.11 contracts/library_erc721/approvals.cairo

3.11.1 [Low] Functions missing events emission in approvals.cairo

File(s): `contracts/library_erc721/approvals.cairo`

Description: The contract is not emitting events for important state transitions. The events are important for post-deployment monitoring. A list of functions not emitting events is presented below.

```
func approve_(...)
func setApprovalForAll_(...)
```

Recommendation: Emit events for the functions listed above.

Status: Not Fixed.

Update from client: Need to be added, as those are defined in ERC721 too.

3.12 contracts/library_erc721/transferability_library.cairo

3.12.1 [High] Not checking for overflow in `_transfer(...)`

File(s): `contracts/library_erc721/transferability_library.cairo`

Description: The function `_transfer(...)` does not check if `balance` has overflowed before writing to storage variable `_balance` for both the sender and recipient. The code is reproduced below:

```
func _transfer{
        syscall_ptr: felt*,
        pedersen_ptr: HashBuiltin*,
        range_check_ptr
    } (sender: felt, recipient: felt, token_id: felt):
    # Reset approval (0 cost if was 0 before on starknet I believe)
    ERC721_approvals.approve_nocheck_(0, token_id)

    let (curr_owner) = _owner.read(token_id)
    assert sender = curr_owner
    _owner.write(token_id, recipient)

    let (balance) = _balance.read(sender)
    _balance.write(sender, balance - 1)
    let (balance) = _balance.read(recipient)
    _balance.write(recipient, balance + 1)

    _onTransfer(sender, recipient, token_id)

    return ()
end
```

Recommendation: Check for overflows on `balance` before writing to storage variable `_balance`.

Status: Fixed in commit 4477d5d0c4a2dc0781b2647e530103e42aabe247.

Update from client: Overflow check added.

3.13 contracts/set_erc721/token_uri.cairo

3.13.1 [Low] Test function implemented inside core contract

File(s): `contracts/set_erc721/token_uri.cairo`

Description: The function `setTokenURI_(...)` has a comment stating that the function should be used for testing only. It is generally recommended to separate smart contract code used exclusively in testing from core protocol smart contract code. The function is reproduced below:

```
## Testing only
@external
func setTokenURI_{
        syscall_ptr: felt*,
        pedersen_ptr: HashBuiltin*,
        bitwise_ptr: BitwiseBuiltin*,
        range_check_ptr
    } (token_id: felt, uri_len: felt, uri: felt*):
    alloc_locals
    _onlyAdmin()

    # TODO: is this useless?
    let (owner) = _owner.read(token_id)
    assert_not_zero(owner)
    _setTokenURI(0, token_id, uri_len, uri)
    URI.emit(uri_len, uri, token_id)
    return()
end
```

Recommendation: Move `setTokenURI_` to a separate contract which imports `set_impl.cairo` to create a separation of concerns between testing code and protocol code.

Status: Not Fixed.

Update from client: Planned but not fixed yet. Only admins can use it also mitigates the problem.

3.13.2 [Info] Not using boolean type of value fixed to `0` or `1`

File(s): `contracts/set_erc721/token_uri.cairo`

Description: The function `_setTokenURI` is an internal function and is only called with `may_use_special_token_mode` set to `0` or `1`. However this is not asserted inside `_setTokenURI`. If `may_use_special_token_mode` is at some point in the future determined non-deterministically this would allow `uri_len * may_use_special_token_mode == 2` condition to pass with an invalid `uri` array. The function is reproduced below:

```
func _setTokenURI{
        syscall_ptr: felt*,
        pedersen_ptr: HashBuiltin*,
        bitwise_ptr: BitwiseBuiltin*,
        range_check_ptr
    } (may_use_special_token_mode: felt, token_id: felt, uri_len: felt, uri: felt*):
    assert_not_zero(uri_len)
    assert_lt_felt(uri[0], 2**249)

    # This is 0 or 1
    if uri_len * may_use_special_token_mode == 2:
        let (rem) = bitwise_and(uri[1], 2**59 - 1)
        if uri[1] == rem:
            # The rest has already been written in the token-id
            # Flag it with both special bits for continuation and 'part of token_id'.
            _token_uri.write(token_id, uri[0] * 4 + 3)
            return ()
        end
        # Write the first URI with the special continuation LSB
        _token_uri.write(token_id, uri[0] * 4 + 1)
        _setExtraTokenURI(token_id, uri_len - 2, 0, uri + 1)
        # event_uri.emit(token_id, uri_len, uri)
    else:
        if uri_len == 1:
            # Just write the URI normally.
            _token_uri.write(token_id, uri[0] * 4)
            return ()
        end
        # Write the first URI with the special continuation LSB
        _token_uri.write(token_id, uri[0] * 4 + 1)
        _setExtraTokenURI(token_id, uri_len - 2, 0, uri + 1)
        # event_uri.emit(token_id, uri_len, uri)
    end
    return()
end
```

Recommendation: To improve maintainability, there should be a check to assert that `may_use_special_token_mode` is a boolean value (either `0` or `1`). Using the Cairo `bool` library is also recommended to improve readability.

Status: Fixed in commit 402976a7018db658cf1b2c91937b8c8a69631210.

Update from client: Fixed as recommended.

4 Documentation Evaluation

Documenting the code is adding enough information to explain what it does so that it is easy to understand the purpose and the underlying functionality of each file/function/line. Documentation can come not only in the form of a read-me but through comments, websites and even videos. Besides being a good programming practice, providing proper documentation improves the efficiency of audits. Less audit hours are required and the time spent auditing can be used more effectively improving the overall output of the audit. Inline documentation allows programmers, testers, auditors and application users to better understand the code. **We recommend the following improvements be made to the documentation of this project**:

– Inline comments to describe each function (what it does, inputs, outputs).

– Inline comments for lines of code that could benefit from an added description for further context.

– Formalize the functional requirements of the project, and continue to update these requirements as the project is developed.

5 Test Suite Evaluation

The test suite is reduced and validates the behavior of the application if users behave as expected. Several edge cases are not handled. Unit Testing is the testing phase where each unit in the system is individually tested. The goal is to isolate each part of the system to ensure they are working as specified. The developer must consider the requirements of each function and make sure that the input/output parameters are in range. In this sense, the system's technical specification plays an important role in creating good test cases, delimiting the scenarios for using a function, and which situations should be avoided by each function. **It is critical that the test suite be created very carefully to reduce the risk of exploits, handling edge cases, and making sure that inputs and outputs of functions are in valid ranges and that overflow has not happened**.

On the other hand, systemic tests validate the integration between the various modules of the system in an production-like environment. The system must be robust enough to deal with users interacting with it in unplanned ways, such as calling functions in a different order than expected, passing wrong input values, and even trying to hack the system.

Our analysis of the code indicates that many parameters are not properly validated. It is essential to assure that each input parameter has a valid range, the user calling the function has the proper privilege, inside the function check for overflows, and making sure that the output is also in valid range. **In view of this, our recommendation is to considerably expand the test suite, as well as improve the technical documentation of the project, with special attention to edge cases**.

During the re-audit phase Cairo `0.9.0` had been released, which introduced breaking changes to smart contracts written in Cairo `0.8.0`. Because there is no upper limit for the `cairo-lang` version in `requirements.txt` the new version of `cairo-lang` is used to compile the contracts which causes them to fail compilation and testing. In order to run the tests correctly, line 3 was changed to `cairo-lang>=0.8.1,<0.9` and requirements had to be reinstalled.

Test Suite Provided (Initial Review and Final Review)

`tests/*`

5.1 Tests Output for the Initial Report

```
pytest
==== test session starts ====
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0
rootdir: /briq-protocol
plugins: asyncio-0.18.3, web3-5.29.0, typeguard-2.13.3
asyncio: mode=legacy
collected 52 items

tests/box_factory_perf_test.py ...[  5%]
tests/box_test.py ................[ 25%]
tests/briq_impl_test.py ..........[ 38%]
tests/proxy_test.py ..............[ 50%]
tests/set_impl_test.py ...........[ 67%]
tests/shape_test.py ..............[100%]

===== warnings summary =====
../../../cairo_venv/lib/python3.8/site-packages/pytest_asyncio/plugin.py:191
  /home/cris/cairo_venv/lib/python3.8/site-packages/pytest_asyncio/plugin.py:191: DeprecationWarning: The
  ↪  'asyncio_mode' default value will change to 'strict' in future, please explicitly use 'asyncio_mode=strict' or
  ↪  'asyncio_mode=auto' in pytest configuration file.
    config.issue_config_time_warning(LEGACY_MODE, stacklevel=2)

../../../cairo_venv/lib/python3.8/site-packages/frozendict/__init__.py:16
  /home/cris/cairo_venv/lib/python3.8/site-packages/frozendict/__init__.py:16: DeprecationWarning: Using or importing
  ↪  the ABCs from 'collections' instead of from 'collections.abc' is deprecated since Python 3.3, and in 3.10 it will
  ↪  stop working
    class frozendict(collections.Mapping):

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
==== 52 passed, 2 warnings in 1587.91s (0:26:27) ====
```

5.2 Tests Output for the Final Report

```
pytest
===== test session starts =====
platform linux -- Python 3.8.10, pytest-7.1.2, pluggy-1.0.0
rootdir: /briq-protocol
plugins: asyncio-0.18.3, typeguard-2.13.3, web3-5.29.2
asyncio: mode=legacy
collected 58 items

tests/box_factory_perf_test.py ...    [  5%]
tests/box_test.py ..........          [ 22%]
tests/briq_impl_test.py .............[ 44%]
tests/proxy_test.py ......            [ 55%]
tests/set_impl_test.py .........      [ 70%]
tests/shape_test.py .................[100%]

===== warnings summary =====
venv/lib/python3.8/site-packages/pytest_asyncio/plugin.py:191
  /briq-protocol/venv/lib/python3.8/site-packages/pytest_asyncio/plugin.py:191: DeprecationWarning: The 'asyncio_mode'
  ↪  default value will change to 'strict' in future, please explicitly use 'asyncio_mode=strict' or
  ↪  'asyncio_mode=auto' in pytest configuration file.
    config.issue_config_time_warning(LEGACY_MODE, stacklevel=2)

venv/lib/python3.8/site-packages/frozendict/__init__.py:16
  /briq-protocol/venv/lib/python3.8/site-packages/frozendict/__init__.py:16: DeprecationWarning: Using or importing the
  ↪  ABCs from 'collections' instead of from 'collections.abc' is deprecated since Python 3.3, and in 3.10 it will stop
  ↪  working
    class frozendict(collections.Mapping):

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
===== 58 passed, 2 warnings in 1999.84s (0:33:19) =====
```

6 About Nethermind

Founded in 2017 by a small team of world-class technologists, Nethermind builds Ethereum solutions for developers and enterprises. Boosted by a grant from the Ethereum Foundation in August 2018, our team has worked tirelessly to deliver the fastest Ethereum client in the market. Our flagship Ethereum client is all about performance and flexibility. Built on .NET core, a widespread, enterprise-friendly platform, Nethermind makes integration with existing infrastructures simple, without losing sight of stability, reliability, data integrity, and security

Nethermind is made up of several engineering teams across various disciplines, all collaborating to realize the Ethereum roadmap, by conducting research and building high-quality tools. Teams focus on specific areas of the Ethereum problem space. Each consists of specialists and experienced developers working alongside interns, learning the ropes in the Nethermind Internship Program.

Our mission is to gather passionate talent from around the world, and to tackle some of the blockchain's most complex problems. Nethermind provides software solutions and services for developers and enterprises building the Ethereum ecosystem. We offer security reviews to projects built on EVM compatible chains and StarkNet. We have expertise in multiple areas of the Ethereum ecosystem, including protocol design, smart contracts (written in Solidity and Cairo), MEV, etc. We develop some of the most used tools on Starknet and one of the most used Ethereum clients. Learn more about us at `https://nethermind.io`.