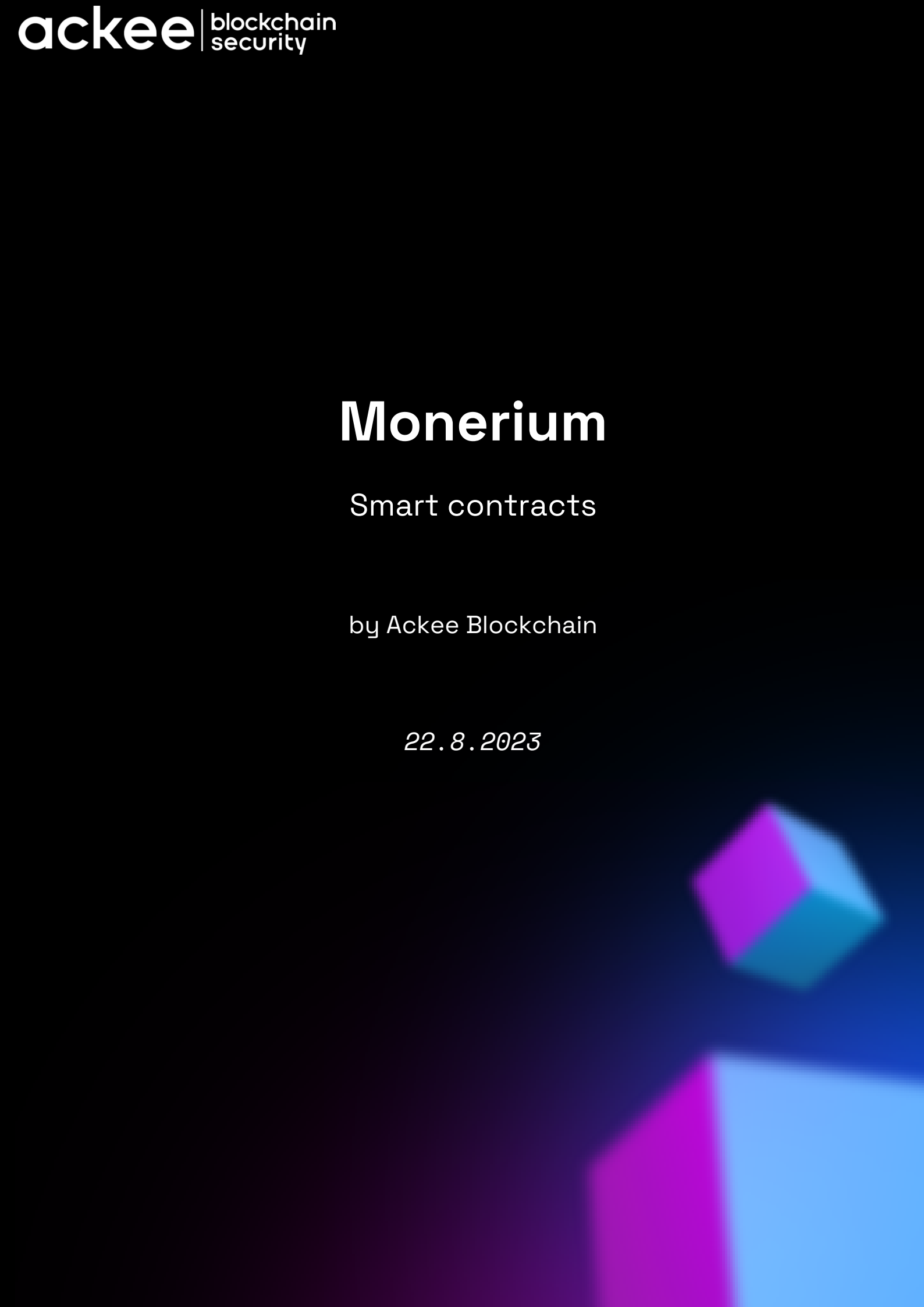
ackee | blockchain security

Monerium

Smart contracts

by Ackee Blockchain

22.8.2023



ackee | blockchain security

Contents

1. Document Revisions

1.0	Final report	4.7.2023
1.1	Fix review	17.7.2023
1.2	Updated fix review	22.8.2023

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain

Ackee Blockchain is an auditing company based in Prague, Czech Republic, specializing in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run free certification courses School of Solana, Summer School of Solidity and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, RockawayX.

2.2. Audit Methodology

1. **Technical specification/documentation** - a brief overview of the system is requested from the client and the scope of the audit is defined.
2. **Tool-based analysis** - deep check with automated Solidity analysis tools and Woke is performed.
3. **Manual code review** - the code is checked line by line for common vulnerabilities, code duplication, best practices and the code architecture is reviewed.
4. **Local deployment + hacking** - the contracts are deployed locally and we try to attack the system and break it.
5. **Unit and fuzz testing** - run unit tests to ensure that the system works as expected, potentially write missing unit or fuzz tests.

2.3. Finding classification

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

Low to *High* impact issues also have a *Likelihood*, which measures the probability of exploitability during runtime.

The full definitions are as follows:

Severity

		Likelihood			
		High	**Medium**	**Low**	**-**
Impact	**High**	Critical	High	Medium	-
	Medium	High	Medium	Medium	-
	Low	Medium	Medium	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.
- **Warning** - The issue cannot be exploited given the current code and/or configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.), but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as a "Warning" or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or configuration (see above) was to change.

Likelihood

- **High** - The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** - Exploiting the issue currently requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.

2.4. Review team

Member's Name	Position
Štěpán Šonský	Lead Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.5. Disclaimer

We've put our best effort to find all vulnerabilities in the system, however our findings shouldn't be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

Revision 1.0

Monerium engaged Ackee Blockchain to perform a security review of the Monerium protocol with a total time donation of 12 engineering days in a period between June 15 and July 4, 2023 and the lead auditor was Štěpán Šonský. The scope of the audit covered all contracts in the protocol, commit `2ff1709`.

We began our review by using static analysis tools, namely Woke. We then took a deep dive into the logic of the contracts. For testing, we have involved Woke testing framework. During the review, we paid special attention to:

- ensuring the access controls are not too relaxed or too strict,
- identification of potential reentrancies in the code,
- verification of the system's arithmetic integrity,
- detection of common problems, including data validation issues,
- compliance with best practices.

Our review resulted in 18 findings, ranging from Info to Medium severity. The most severe ones are related to ownership, access control and data validations (see M1, M2, M3, M4, M5). These issues aren't a direct threat but they can create vulnerabilities due to human errors in the future. Of particular concern is the owner's multi-sig scheme of 2/6, which is severely weak. The overall code quality and architecture are not the best and contain many violations of Solidity development best practices like data validations, unused code, naming conventions, etc.

Ackee Blockchain recommends Monerium:

- increase owner's multi-sig threshold,
- review and fix the access control architecture,
- ensure return values are always validated,
- separate production contracts from testing contracts,
- remove unused code from the codebase,
- address all other reported issues.

See Revision 1.0 for the system overview of the codebase.

Revision 1.1

The review was done on the given commit: `3477259`. Monerium fixed all medium-severity issues and the multi-sig scheme has been increased to 3/6. The only acknowledged issue L1 is not addressed because of the planned redesign.

See Revision 1.1 for the review of the updated codebase and additional information we consider essential for the current scope.

Revision 1.2

The updated fix review was done on the commit `40c7c17`, which reverts the fix of M5: Missing decimals validation. The client decided to only acknowledge the issue due to the low likelihood and complicated upgrade/migration process of `TokenStorage` contract.

See Revision 1.2 for the review of the updated codebase and additional information we consider essential for the current scope.

4. Summary of Findings

The following table summarizes the findings we identified during our review. Unless overridden for purposes of readability, each finding contains:

- a *Description*,
- an *Exploit scenario*,
- a *Recommendation* and if applicable
- a *Fix*.

There might often be multiple ways to solve or alleviate the issue, with varying requirements regarding the necessary changes to the codebase. In that case, we will try to enumerate them all, clarifying which solves the underlying issue better (albeit possibly only with architectural changes) than others.

	Severity	Reported	Status
M1: Access control architecture	Medium	1.0	Fixed
M2: Renounce ownership	Medium	1.0	Fixed
M3: Weak ownership	Medium	1.0	Fixed
M4: Unchecked return values	Medium	1.0	Partially Fixed
M5: Missing decimals validation	Medium	1.0	Acknowledged
L1: Missing validations	Low	1.0	Acknowledged
W1: Impossible to remove bridgeFrontend	Warning	1.0	Fixed
W2: Unprotected functions	Warning	1.0	Fixed

	Severity	Reported	Status
W3: Missing events	Warning	1.0	Fixed
W4: Duplicated event	Warning	1.0	Invalidated
W5: Testing contracts	Warning	1.0	Fixed
W6: Multiple compiler versions	Warning	1.0	Fixed
I1: Unused library	Info	1.0	Fixed
I2: Unused variables	Info	1.0	Fixed
I3: Naming conventions	Info	1.0	Partially Fixed
I4: Unnecessary SafeMath	Info	1.0	Fixed
I5: Typos	Info	1.0	Fixed
I6: Inconsistent uint syntax	Info	1.0	Fixed

Table 2. Table of Findings

5. Report revision 1.0

5.1. System Overview

This section contains an outline of the audited contracts. Note that this is meant for understandability purposes and does not replace project documentation.

Contracts

Contracts we find important for better understanding are described in the following section.

TokenFrontend.sol

Abstract contract for user interaction with the token. It contains all functions from `IERC20` and calls implementations in the `controller`, which provides the logic and can be changed by the owner. The contract inherits from `Claimable`, `CanReclaimToken`, `NoOwner`, `IERC20`, `IPolygonPosRootToken` and `AccessControl`. The `PREDICATE_ROLE` can call `mint` function. Also, the `SYSTEM_ROLE` can call `burnFrom` and `recover` functions.

PolygonPosTokenFrontend.sol

`PolygonPosTokenFrontend` inherits all features from `TokenFrontend` and includes some additional logic. It introduces the `DEPOSITOR_ROLE`, which is set to `childChainManager_` in the constructor. The `DEPOSITOR_ROLE` can call `deposit` function, which mints `amount` of tokens to the `user`. The `withdraw` function calls `controller.burnFrom` function`.

StandardController.sol

`StandardController` inherits from `ClaimableSystemRole` and is a parent contract for other controllers (`MintableController`, `SmartController` and

`ConstantSmartController`). The controller acts as a middle layer between `TokenFrontend` and `TokenStorage` and provides basic ERC-20 operations like `transfer`, `transferFrom`, `approve` and additional ERC-677 feature `transferAndCall`.

MintableController.sol

`MintableController` extends the functionality of `StandardController` by mint and burn using the `SYSTEM_ROLE`.

SmartController.sol

`SmartController` inherits from the `MintableController` and adds a validator feature.

TokenStorage.sol

This contract provides operations on `tokenStorage` using the `TokenStorageLib` implementations.

TokenStorageLib.sol

`TokenStorageLib` is used in the `TokenStorage` contract for math operations on the `TokenStorage` struct.

MintableTokenLib.sol

Used in `MintableController` for `TokenStorage` and provides `mint` and `burn` calculations on it. Also, it uses OpenZeppelin `SignatureChecker` for authorized caller burning.

SmartTokenLib.sol

`SmartTokenLib` is a library used in `SmartController` to work with the `SmartStorage` struct. It allows to `setValidator`, `validate` and `recover` tokens in case the user loses its private key by burning them from the original address

and minting them to the new one.

ERC20Lib.sol

`ERC20Lib` provides ERC-20 standard operations (`transfer`, `transferFrom`, `approve`, `balanceOf`, `allowance`) using functions of `TokenStorage`.

ERC677Lib.sol

`ERC677Lib` provides `transferAndCall` function, which calls `onTokenTransfer` callback on the `recipient` (if is compliant).

SystemRole.sol

`SystemRole` abstract contract inherits from OpenZeppelin `AccessControl` and `Ownable` implementation and it introduces management of `ADMIN_ROLE` and `SYSTEM_ROLE`. Also, it contains `mintAllowances` mapping and related setters. Functions in the `SystemRole` aren't protected by any access controls.

Ownable.sol

OpenZeppelin implementation of `Ownable` pattern.

Claimable.sol

`Claimable` inherits from the `Ownable` and adds the recommended two-step ownership transfer.

Actors

This part describes the actors of the system, their roles, and permissions.

Owner

The owner is set to `msg.sender` in the `Ownable` constructor and has the most privileged role in the system. It has the following abilities in the system.

- `BlacklistValidator`:
  - `ban`, `unban`,
  - `addSystemAccount`, `removeSystemAccount`,
  - `addAdminAccount`, `removeAdminAccount`,
- `ClaimableSystemRole`:
  - `transferOwnership`
- `MintableController`:
  - `setMaxMintAllowance`,
- `SmartController`:
  - `setValidator`,
- `StandardController`:
  - `setFrontend`, `setBridgeFrontend`,
  - `setStorage`,
  - `transferStorageOwnership`, `claimStorageOwnership`,
  - `addSystemAccount`, `removeSystemAccount`,
  - `addAdminAccount`, `removeAdminAccount`,
- `TokenFrontend`:
  - `setController`,
- `TokenStorage`:
  - `addBalance`, `subBalance`,
- `CanReclaimToken`
  - `reclaimToken`,
- `Claimable`:

    - `transferOwnership`,
- `HasNoContracts`:
    - `reclaimContract`
- `HasNoEther`:
    - `reclaimEther`
- `Ownable`:
    - `transferOwnership`, `renounceOwnership`.

According to the developers' statement, the owner account is set up using the Safe multi-sig wallet with 2/6 threshold scheme.

Admin

The `ADMIN_ROLE` can call functions with `onlyAdminAccount` modifier, namely `setMintAllowance` in the `MintableController`.

Predicate

The `PREDICATE_ROLE` can call `mint` function in the `TokenFrontend`.

System

The `SYSTEM_ROLE` is allowed to call functions protected by `onlySystemAccount` modifier. Namely `burnFrom_withCaller` and `burnFrom` in the `MintableController`, and `recover_withCaller` in the `SmartController`. The `onlyAllowedSystemAccount` modifier adds an allowance condition to `onlySystemAccount` modifier and is used in `MintableController.mintTo_withCaller` function.

Depositor

The `DEPOSITOR_ROLE` is defined and used in the `PolygonPosTokenFrontend` (and derived contracts) and allows to call `deposit` function.

User

The user role is any EOA or contract, which can interact with the protocol using unprotected `public`/`external` functions.

5.2. Trust Model

The system strongly depends on privileged roles. The owner account has significant power and is centralized to two private keys, which introduces a potential threat to the protocol. The trust of users and holders of tokens completely relies on these two owner accounts.

M1: Access control architecture

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	SystemRole.sol	Type:	Access control

Description

The `SystemRole` base contract lacks the protection of functions. Despite the presence of an `onlyOwner` modifier inherited from the `Ownable` contract, crucial functions for role management are not adequately safeguarded by it. Although child contracts may override these functions with the modifier, the dependency on manual overriding leaves room for human error.

Example of virtual function in `SystemRole.sol`:

```
function addAdminAccount(address account) public virtual {
    grantRole(ADMIN_ROLE, account);
    emit AdminAccountAdded(account);
}
```

Example of overriding function in `StandardController.sol`:

```
function addAdminAccount(address account) public override onlyOwner {
    super.addAdminAccount(account);
}
```

Vulnerability scenarios

The developer removes the overridden function from the derived contract because it looks like it only calls the super function.

Or the developer creates a new contract, which inherits from the `SystemRole`

and forgets to override critical functions with `onlyOwner` modifier. Then anyone would be able to add admin account for example.

Recommendation

The `onlyOwner` modifier should be added to functions in the `SystemRole` base contract. Namely, this applies to the following functions: `addSystemAccount`, `removeSystemAccount`, `addAdminAccount`, `removeAdminAccount` and `setMaxMintAllowance`. For the `setMintAllowance` the `onlyAdminAccounts` modifier is used in the `StandardController`.

Solution (Revision 1.1)

Fixed, "All modifiers have been relocated to where the original function is declared, in `SystemRole`."

Go back to Findings Summary

M2: Renounce ownership

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	Ownable.sol	Type:	Access control

Description

The `Ownable` base contract contains the `renounceOwnership` function, which could have severe consequences for the protocol, meaning that nobody would be able to call functions protected by the `onlyOwner` modifier anymore.

```
function renounceOwnership() public onlyOwner {
    emit OwnershipRenounced(owner);
    owner = address(0);
}
```

Vulnerability scenario

The owner (multiple malicious multi-sig owners) accidentally or intentionally calls `renounceOwnership` e.g. on `TokenFrontend` and loses the ability to `setController`.

Recommendation

Remove the `renounceOwnership` function to disable this unwanted feature.

Solution (Revision 1.1)

Fixed, "The required function has been successfully incorporated."

Go back to Findings Summary

M3: Weak ownership

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	Ownable inherited	Type:	Access control

Description

The protocol strongly relies on the owner in terms of setting critical parameters like roles, validators, balances or allowances. But according to Monerium's statement, the owner account uses Safe multi-sig wallet with a threshold scheme of only 2/6.

Vulnerability scenarios

- Two owners, with malicious intent, may conspire and act against the protocol's interests.
- Private keys of two owners are compromised and an unauthorized party damages the protocol.

Recommendation

The current multi-sig scheme of 2/6 does not provide adequate security. To address this vulnerability, we recommend to increase the threshold to at least 3/6. This will ensure that no two owners can conspire or in the event of their private keys being compromised, the protocol remains secure.

Perform frequent audits of the owner keys and periodically change them to reduce the risks of keys getting compromised.

Solution (Revision 1.1)

Fixed, Monerium agreed to increase the multi-sig scheme to 3/6.

Go back to Findings Summary

M4: Unchecked return values

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	PolygonPosTokenFrontend.sol, SmartTokenLib.sol	Type:	Data validation

Description

Return values of `mint` and `burn` functions are not checked. Even though these functions return only `true` in current implementations, it remains a best practice to validate the return values to avoid future human errors.

PolygonPosTokenFrontend.sol

```
this.mintTo(user, amount);
```

```
controller.burnFrom(msg.sender, amount);
```

SmartTokenLib.sol

```
token.burn(from, amount);
token.mint(to, amount);
```

Vulnerability scenario

The developer changes the implementation of a function to return `false` under certain conditions. Transaction proceeds and the event is emitted, even when the function execution was not successful.

e.g. in PolygonPosTokenFrontend.sol:

```
function withdraw(uint256 amount) external override {
    controller.burnFrom(msg.sender, amount);
    emit Transfer(msg.sender, address(0x0), amount);
}
```

Recommendation

Implement return values checks to make the system more rigid and human error-proof.

```
require(controller.burnFrom(msg.sender, amount), "burnFrom failed");
```

Solution (Revision 1.1)

Partially Fixed, "The `SmartTokenLib` now evaluates the return value for potential future utilization. It should be noted that `PolygonPosTokenFrontend.sol` has not undergone any modifications."

Go back to Findings Summary

M5: Missing decimals validation

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	TokenStorageLib.sol	Type:	Data validation

Description

The `TokenStorage` struct holds `balances`, `totalSupply` and `allowed` values, but does not determine decimals of these values. Decimals are hardcoded to 18 in the controller but lack any validation.

```
struct TokenStorage {
    mapping(address => uint) balances;
    mapping(address => mapping(address => uint)) allowed;
    uint totalSupply;
}
```

Vulnerability scenario

A controller with different decimals is deployed with the attached `TokenStorage` struct nominated in 18 decimals. Due to missing validations, the transaction does not revert and lets the controller with e.g. 4 decimals operate on the storage nominated in 18 decimals.

Recommendation

Move the decimals specification to the `TokenStorage` struct to reduce the risk of decimal mismatch.

```
struct TokenStorage {
    mapping(address => uint256) balances;
    mapping(address => mapping(address => uint)) allowed;
    uint256 totalSupply;
```

```
    uint8 decimals;
}
```

If it's needed to keep the decimals in the controller, then introduce validation checks during the deployment of the controller to verify the compatibility of decimal values between the controller and the `TokenStorage`.

Solution (Revision 1.1)

Fixed, "The `decimals` value has been relocated from the controller to the `TokenStorage`."

Solution (Revision 1.2)

Acknowledged. The client reverted the fix from revision 1.1 and decided not to deploy the fix on the mainnet due to the low likelihood and risks of migration process of `TokenStorage`.

Go back to Findings Summary

L1: Missing validations

Low severity issue

Impact:	Low	Likelihood:	Low
Target:	PolygonPosTokenFrontend.sol, TokenFrontend.sol	Type:	Data validation

Description

The constructor of `PolygonPosTokenFrontend` lacks essential validations for zero-address and zero-length.

```
constructor(
        string memory name_,
        string memory symbol_,
        bytes3 ticker_,
        address childChainManager_
    ) TokenFrontend(name_, symbol_, ticker_) {
        _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
        _setupRole(DEPOSITOR_ROLE, childChainManager_);
    }
```

Vulnerability scenario

The empty parameter is passed during the deployment, the contract become unusable and it would require re-deployment.

Recommendation

Add zero-address check for `childChainManager_` in the `PolygonPosTokenFrontend` constructor, and zero-length validation for `name_` and `symbol_` in the `TokenFrontend` constructor.

Solution (Revision 1.1)

Acknowledged, "In the event that a redeployment of our frontend contracts is required, we have plans in place to enhance their design, making them more flexible and upgradeable. Therefore, this issue has been acknowledged but not immediately addressed."

Go back to Findings Summary

W1: Impossible to remove bridgeFrontend

Impact:	Warning	Likelihood:	N/A
Target:	StandardController.sol	Type:	Access control

Description

There is no way to remove bridge frontend addresses from the `bridgeFrontends` mapping in `StandardController`.

Vulnerability scenario

One of the `bridgeFrontends` becomes malicious and misuses `onlyFrontend` functions.

Recommendation

Implement the function `removeBridgeFrontend` to have more control over `bridgeFrontends` collection.

```
function removeBridgeFrontend(address frontend_) public onlyOwner {
    bridgeFrontends[frontend_] = false;
    emit BridgeFrontendRemove(frontend_);
}
```

Solution (Revision 1.1)

Fixed, `The required function and corresponding event have been successfully added.`

Go back to Findings Summary

W2: Unprotected functions

Impact:	Warning	Likelihood:	N/A
Target:	BlacklistValidator.sol	Type:	Access control

Description

The `BlacklistValidator` contract inherits from `SystemRole` contract and does not override functions `setMaxMintAllowance` and `setMintAllowance` with `onlyOwner` modifier. Therefore anyone can call `setMaxMintAllowance` and `setMintAllowance`. However, these state variables aren't used in the `BlacklistValidator` and rather point to bad inheritance architecture.

Recommendation

Refactor the `BlacklistValidator` inheritance to remove unused/unnecessary state variables and functions from it.

Solution (Revision 1.1)

Fixed, "The allowance functions have been moved from `SystemRole` to `MintableController`, resulting in the `BlacklistValidator` no longer possessing any unprotected inherited functions."

Go back to Findings Summary

W3: Missing events

Impact:	Warning	Likelihood:	N/A
Target:	Claimable.sol, SystemRole.sol, ClaimableSystemRole.sol	Type:	Best practices

Description

The function `transferOwnership` in `Claimable` and `ClaimableSystemRole` contracts changes the contract state (`pendingOwner`) but does not emit the event. Also, the function `setMaxMintAllowance` in the `SystemRole` lacks event emit.

```
function transferOwnership(address newOwner) public virtual override
onlyOwner {
    pendingOwner = newOwner;
}
```

Recommendation

Emit the event in `transferOwnership` function. It's generally good practice to emit events after every contract state change.

```
function transferOwnership(address newOwner) public virtual override
onlyOwner {
    pendingOwner = newOwner;
    emit PendingOwner(pendingOwner);
}
```

Solution (Revision 1.1)

Fixed, "The necessary events have been successfully incorporated."

Go back to Findings Summary

W4: Duplicated event

Impact:	Warning	Likelihood:	N/A
Target:	TokenFrontend.sol	Type:	Events

Description

The `TokenFrontend` contract contains a duplicated event emit in the `transferAndCall` function.

```
emit Transfer(msg.sender, to, amount);
emit Transfer(msg.sender, to, amount, data);
```

Recommendation

Remove the first emit and use only the second one, which contains the same parameters.

Solution (Revision 1.1)

Invalidated, "The presence of this duplication is to ensure compliance with both ERC667 and ERC20 standards."

Go back to Findings Summary

W5: Testing contracts

Impact:	Warning	Likelihood:	N/A
Target:	`**/*`	Type:	Best practices

Description

The project's structure currently mixes testing contracts with production contracts within the same directory, which reduces code clarity.

Recommendation

Relocate all contracts that are not intended for deployment on the mainnet into distinct directories, such as `test` and `mock`. This approach will improve organization and enhance the readability of the codebase.

Solution (Revision 1.1)

Fixed, "Contracts not intended for deployment have been appropriately relocated into a separate directory, such as scripts, tests, and so on."

Go back to Findings Summary

W6: Multiple compiler versions

Impact:	Warning	Likelihood:	N/A
Target:	`**/*`	Type:	Compiler

Description

The project uses inconsistent `pragma solidity` syntax and versions. Mixing compiler versions might lead to unpredictability and potential issues during the compilation and deployment of contracts.

```
pragma solidity ^0.8.0;
```

```
pragma solidity ^0.8.11;
```

```
pragma solidity 0.8.11;
```

Recommendation

Always use the same compiler version for all contracts in the project.

Solution (Revision 1.1)

Fixed, "All contracts in the project now utilize the same compiler version. The selected version is the most recently deployed one, 0.8.11."

Go back to Findings Summary

I1: Unused library

Impact:	Info	Likelihood:	N/A
Target:	Roles.sol	Type:	Best practices

Description

The `Roles` library is not used in the project.

Recommendation

Remove the unused library and keep the codebase clean of any unused code.

Solution (Revision 1.1)

Fixed, "The "Roles" library has been successfully eliminated from the project."

Go back to Findings Summary

I2: Unused variables

Impact:	Info	Likelihood:	N/A
Target:	StandardController.sol	Type:	Best practices

Description

The `StandardController` contract contains two unused state variables `name` and `symbol`. Unused code decreases code readability and it does not look professional.

```
string public name;
string public symbol;
```

Recommendation

Remove these unused variables from the `StandardController` contract, as well as any other unused code across the project. This will enhance code readability and maintainability.

Solution (Revision 1.1)

Fixed, "The unused `name` and `symbol` variables have been removed from `StandardController.sol`."

Go back to Findings Summary

I3: Naming conventions

Impact:	Info	Likelihood:	N/A
Target:	newControllerAndBridge.sol, StandardController.sol	Type:	Best practices

Description

This informational issue summarizes naming convention violations.

The filename of contract `NewControllerAndBridgeFrontend` starts with the lowercase character `newControllerAndBridge.sol`.

The function `avoidBlackholes` in `StandardController` contarct is `internal`, but does not contain the underscore prefix.

`StandardController` contains the following functions with mixed case naming:

- `transfer_withCaller`
- `transferFrom_withCaller`
- `approve_withCaller`
- `transferAndCall_withCaller`

The duplicated name `TokenStorage` is used for contract and struct.

Recommendation

- Use CapitalCamelCase for Solidity filenames.
- Use camelCase for function names.
- Use underscore prefix for `private`/`internal` functions and variables. This is not a strict Solidity language naming convention, but it's widely adopted because it increases code readability.

- Use unique naming for `TokenStorage` contract and struct.

Solution (Revision 1.1)

Partially Fixed, ”Your proposed naming convention has been adopted. However, this issue remains partially resolved as our static frontend continues to use our controller’s `transfer_withCaller`, among others.”

Go back to Findings Summary

I4: Unnecessary SafeMath

Impact:	Info	Likelihood:	N/A
Target:	BasicToken.sol	Type:	Best practices

Description

The `BasicToken` contract uses the `SafeMath` library with Solidity ^0.8.0, which already includes implicit overflow/underflow safety.

Recommendation

Remove the `SafeMath` library from the project.

Solution (Revision 1.1)

Fixed, "The `SafeMath` component has been successfully removed from the project."

Go back to Findings Summary

I5: Typos

Impact:	Info	Likelihood:	N/A
Target:	TokenFrontend.sol	Type:	Best practices

Description

- Documentation of `burnFrom` in `TokenFrontend` contract contains typo "removfes".
- `setAllowed` documentation in `TokenStorage` contains "Qunatity".

Recommendation

Correct typographical errors in the documentation.

Solution (Revision 1.1)

Fixed, "All identified typographical errors in the comments have been corrected."

Go back to Findings Summary

I6: Inconsistent uint syntax

Impact:	Info	Likelihood:	N/A
Target:	`**/*`	Type:	Best practices

Description

The project uses inconsistent `uint` syntax. In some contracts, there is `uint`, and in others `uint256`. Although `uint` is an alias for `uint256`, consistent syntax improves code quality and readability.

Recommendation

Use the preferred `uint256` syntax in all places. Explicit declaration of size improves the readability of the code.

Solution (Revision 1.1)

Fixed, `uint` has been replaced by `uint256` in whole project.

Go back to Findings Summary

6. Report revision 1.1

6.1. System Overview

Aside from issues fixes, no significant changes have been made to the codebase and trust model.

Contracts

Updates in contracts we found important to mention in the fix review.

SystemRole.sol

Functions in the `SystemRole` contract are now protected by `onlyOwner` and `onlyAdminAccounts` modifiers.

7. Report revision 1.2

7.1. System Overview

The updated codebase contains only the revert commit of M5: Missing decimals validation.

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, Monerium: Smart contracts, 22.8.2023.

Appendix B: Glossary of terms

The following terms might be used throughout the document:

Superclass/Ancestor of C

A contract that C inherits/derives from.

Subclass/Child of C

A contract that inherits/derives from C.

Syntactic contract

A Solidity contract. May have an inheritance chain, and may be deployed.

Deployed contract

An EVM account with non-zero code. If its source was written in Solidity, it was created through at least one syntactic contract. If that contract had superclasses (parents), it would be composed of multiple syntactic contracts.

Init/initialization function

A non-constructor function that serves as an initializer. Often used in upgradeable contracts.

External entrypoint

A `public` or `external` function.

Public/Publicly-accessible function/entrypoint

An `external` or `public` function that can be successfully executed by any network account.

Mutating function

A non-`view` and non-`pure` function.

ackee | blockchain security

Thank You

Ackee Blockchain a.s.


Prague, Czech Republic


hello@ackeeblockchain.com


https://twitter.com/AckeeBlockchain