ackee | blockchain security

Squid

Deposit contracts

by Ackee Blockchain

8.6.2023

ackee | blockchain security

Contents

1. Document Revisions

1.0	Final report	8.6.2023

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

Squid engaged Ackee Blockchain to perform a security review of the Squid with a total time donation of 5 engineering days in a period between May 30 and June 8, 2023 and the lead auditor was Štěpán Šonský.

The audit has been performed on the commit `abb9534` and during the audit was updated to `55e2b2b`. Scope was the following:

- DepositReceiver.sol
- ReceiverImplementation.sol
- SquidDepositService.sol
- SquidRouter.sol
- SquidMulticall.sol

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

- detecting possible reentrancies in the code,
- ensuring functions are called with correct params,
- ensuring access controls are not too weak or too strict,
- looking for common issues such as data validation.

Our review resulted in 7 findings, ranging from Info to Medium severity. The most severe one (see M1: One-step ownership transfer) is not likely but can cause permanent loss of the ability to upgrade the contracts. Overall code quality is solid, and the project is well structured, however, it lacks detailed

in-code documentation.

Ackee Blockchain recommends Squid to:

- implement two-step ownership transfer,
- be careful while using deprecated `selfdestruct`,
- use more strict data validations,
- create NatSpec documentation.

See Revision 1.0 for the system overview of the codebase.

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: One-step ownership transfer	Medium	1.0	Reported
M2: Missing destinationAddress validation	Medium	1.0	Reported
W1: Contract ID validations	Warning	1.0	Reported
W2: Selfdestruct deprecation	Warning	1.0	Reported
W3: Usage of `solc` optimizer	Warning	1.0	Reported
I1: Duplicated validation	Info	1.0	Reported
I2: Missing documentation	Info	1.0	Reported

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.

DepositReceiver.sol

Contract for one-time use. It contains only constructor with parameters `delegateData` and `refundRecipient`. The constructor calls `delegateData` on `DepositService.receiverImplementation()` using `delegatecall`. If the `refundRecipient` is zero-address then the `msg.sender` is used. And finally, the contract calls `selfdestruct` with `refundRecipient` as a parameter.

ReceiverImplementation.sol

`ReceiverImplementation` is a logic contract for `DepositReceiver` and it provides communication between `SquidDepositService` and `SquidRouter`. It contains `external` functions `receiveAndBridgeCall`, `receiveAndCallBridge`, `receiveAndCallBridgeCall`, and `receiveAndFundAndRunMulticall`.

SquidDepositService.sol

Contract for interaction, which generates deposit addresses using the `CREATE2` algorithm and deploys `DepositReceiver` for the following operations:

- `bridgeCallDeposit`

- `callBridgeDeposit`
- `callBridgeCallDeposit`
- `fundAndRunMulticallDeposit`
- `refundBridgeCallDeposit`
- `refundCallBridgeDeposit`
- `refundCallBridgeCallDeposit`
- `refundFundAndRunMulticallDeposit`

Refund functions can be called only by `refundRecipient`.

SquidRouter.sol

`SquidRouter` is `AxelarForecallable`, which is `AxelarExecutable` extended with express features. `SquidRouter` provides interaction with `AxelarGateway` for sending and receiving cross-chain messages using the Axelar network.

SquidMulticall

This contract is used by `SquidRouter` and performs calls defined by `Call` struct. The function `run(Call[] calls)` sequentially executes these tasks.

Actors

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

Owner

The owner role is a part of Axelar's `Upgradable` implementation. The owner can upgrade the contracts and also, can transfer ownership to another address using a one-step process.

Pauser

Pauser can pause the `SquidRouter` and disallow users calling `external`

functions `bridgeCall`, `callBridge`, `callBridgeCall` and `fundAndRunMulticall`. Pauser can also transfer the rote to another address and the process using a two-step process.

User

User role is any EOA or contract, which can interact with the protocol, therefore calling all `external`/`public` functions.

Axekar Network

Axelar Network is a cross-chain messaging protocol used by Squid. From the current audit perspective, it is considered as a separate black box.

5.2. Trust Model

The Squid protocol is built on top of Axelar network, therefore it inherits security properties from Axelar GMP. Squid protocol availability depends on the pauser role. Users also have to trust the owner in terms of upgradeability.

M1: One-step ownership transfer

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	SquidDepositService.sol, SquidRouter.sol	Type:	Access controls

Description

`SquidDepositService` and `SquidRouter` inherits from Axelar's `Upgradable` abstract contract, which uses a one-step ownership transfer.

Vulnerability scenario

Ownership is accidentally transferred to an incorrect address and the ability to upgrade the contracts is permanently lost.

Recommendation

Implement a two-step ownership transfer, where the newly proposed address has to accept the ownership.

Go back to Findings Summary

M2: Missing destinationAddress validation

Medium severity issue

Impact:	Medium	Likelihood:	Low
Target:	SquidRouter.sol	Type:	Data validation

Description

`destinationAddress` is not checked for an empty value in `SquidRouter`.

Recommendation

Add a `require` statement for `destinationAddress`.

```
require(destinationAddress.length > 0, 'Empty destinationAddress');
```

Go back to Findings Summary

W1: Contract ID validations

Impact:	Warning	Likelihood:	N/A
Target:	ReceiverImplementation.sol, SquidDepositService.sol, SquidRouter.sol	Type:	Data validation

Description

Many components implement `contractId()`, which could be used for more strict, robust contract validations in constructors.

Recommendation

Use `contractId()` validations for all components, which support it.

```
require(IAxelarGasService(_gasService).contractId() == keccak256('axelar-gas-service'), 'Invalid gas service contract');
```

Go back to Findings Summary

W2: Selfdestruct deprecation

Impact:	Warning	Likelihood:	N/A
Target:	DepositReceiver.sol	Type:	Best practices

Description

Constructor of `DepositReceiver` calls `selfdestruct` function, which is deprecated since Solidity 0.8.18. There is a proposal EIP-4758 to deactivate `SELFDESTRUCT` opcode by changing it to `SENDALL`, which does recover all funds to the caller but does not delete any code or storage.

Recommendation

Be aware of these upcoming breaking changes and use `selfdestruct` with caution.

Go back to Findings Summary

W3: Usage of `solc` optimizer

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

Description

The project uses `solc` optimizer. Enabling `solc` optimizer may lead to unexpected bugs.

The Solidity compiler was audited in November 2018, and the audit concluded that the optimizer may not be safe.

Vulnerability scenario

A few months after deployment, a vulnerability is discovered in the optimizer. As a result, it is possible to attack the protocol.

Recommendation

Until the `solc` optimizer undergoes more stringent security analysis, opt-out using it. This will ensure the protocol is resilient to any existing bugs in the optimizer.

Go back to Findings Summary

I1: Duplicated validation

Impact:	Info	Likelihood:	N/A
Target:	SquidDepositService.sol	Type:	Data validation

Description

The gateway address is validated to zero-address in `SquidDepositService` and then also in `ReceiverImplementation` constructor, which is called from `SquidDepositService` constructor:

```
constructor(address _router, address _gateway, address _refundIssuer) {
    if (_gateway == address(0) || _refundIssuer == address(0)) revert
ZeroAddressProvided();

    gateway = _gateway;
    refundIssuer = _refundIssuer;
    receiverImplementation = address(new ReceiverImplementation(_router,
_gateway));
}
```

`ReceiverImplementation` constructor:

```
constructor(address _router, address _gateway) {
    if (_router == address(0) || _gateway == address(0)) revert
ZeroAddressProvided();

    router = _router;
    gateway = _gateway;
}
```

Recommendation

Remove the duplicated `_gateway` zero-address validation from `SquidDepositService` constructor.

Go back to Findings Summary

I2: Missing documentation

Impact:	Info	Likelihood:	N/A
Target:	All files	Type:	Best practices

Description

Although the code is relatively simple and easy to read, the project is missing detailed documentation.

Recommendation

We strongly recommend covering the code using NatSpec standard documentation. High-quality documentation has to be an essential part of any professional project.

Go back to Findings Summary

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, Squid: Deposit contracts, 8.6.2023.

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