Zircuit ZtakingPool

Smart Contract Security Assessment

March 11, 2024

ABSTRACT

Dedaub was commissioned to perform a security audit of the Zircuit ZtakingPool protocol. No major issues were identified and only a centralization consideration was raised concerning the migration process of the protocol. It should be noted that the migrator contract, which is one of the main parties of the protocol, had not been implemented by the time the audit was finished and only its interface had been defined.

PROTOCOL DESCRIPTION

The protocol is designed to be a multi-token staking pool. The point calculations are being handled off-chain by the events data emitted by the protocol with the final goal being the users to allow their liquidity to be migrated to Zircuit L2.

The protocol is made up of mainly 3 entities:

- The owner who is mainly responsible for enabling and disabling tokens that can be staked in the contract and also has the ability to pause the contract.
- The users who can deposit and withdraw tokens from the contract.
- The migrator contract, that will be bridging users' liquidity upon users' request.

A user should always be able to stake tokens that are enabled and should always be able to unstake any tokens that were previously staked, including tokens that have been disabled by the owner after being deposited. This is to ensure that users' funds cannot be held captive by the owner.

The user is able to initiate the migration by invoking the `migrate` function directly which requires a signature from the Zircuit signer to ensure that the user is not being phished and the `_migratorContract` that was pointed to is indeed an "official" contract that Zircuit approves. A signature expiration (`_signatureExpiry`) is present so that there is a way to invalidate previously signed `_migratorContract`, if such a need emerges. User funds migration can also be executed by the owner using the `migrateSig()` function only after the user has explicitly approved this action by signing and submitting a `Migrate` message.

The protocol does not support the staking of deflationary tokens, rebasing tokens or fee-on-transfer tokens. Furthermore, custom smart contract stakers besides multisig wallets will not be supported by the pool. The majority of multisig wallets will be compatible with the staking pool.

SETTING & CAVEATS

This audit mainly covers the contracts of the at-the-time private repository zircuit-labs/zkr-staking of the Zircuit ZtakingPool Protocol at commit `5c0f98d73a0d8e90864d7a207704a44c575d9a41`.

Two auditors worked on the codebase for 2 days on the following contracts:

```
contracts/
├── ZtakingPool.sol
└── interface/
    ├── IMigrator.sol
    ├── IWETH.sol
    └── IZtakingPool.sol
```

The audit's main target is security threats, i.e., what the community understanding would likely call "hacking", rather than the regular use of the protocol. Functional correctness (i.e. issues in "regular use") is a secondary consideration. Typically it can only be covered if we are provided with unambiguous (i.e. full-detail) specifications of what is the expected, correct behavior. In terms of functional correctness, we often trusted the code's calculations and interactions, in the absence of any other specification. Functional correctness relative to low-level calculations (including units, scaling and quantities returned from external protocols) is generally most effectively done through thorough testing rather than human auditing.

VULNERABILITIES & FUNCTIONAL ISSUES

This section details issues affecting the functionality of the contract. Dedaub generally categorizes issues according to the following severities, but may also take other considerations into account such as impact or difficulty in exploitation:

Category	Description
CRITICAL	Can be profitably exploited by any knowledgeable third-party attacker to drain a portion of the system's or users' funds OR the contract does not function as intended and severe loss of funds may result.
HIGH	Third-party attackers or faulty functionality may block the system or cause the system or users to lose funds. Important system invariants can be violated.
MEDIUM	Examples: • User or system funds can be lost when third-party systems misbehave. • DoS, under specific conditions. • Part of the functionality becomes unusable due to a programming error.
LOW	Examples: • Breaking important system invariants but without apparent consequences. • Buggy functionality for trusted users where a workaround exists. • Security issues which may manifest when the system evolves.

Issue resolution includes “dismissed” or “acknowledged” but no action taken, by the client, or “resolved”, per the auditors.

CRITICAL SEVERITY:

[No critical severity issues]

HIGH SEVERITY:

[No high severity issues]

MEDIUM SEVERITY:

[No medium severity issues]

LOW SEVERITY:

ID	Description	STATUS
L1	Non-ERC20 compliant tokens are not able to be migrated	**ACKNOWLEDGED**

The `ZtakingPool` contract implements functionality that allows migration of user funds by an allowed `migratorContract`. However, there are `ERC20-like` tokens that do not completely follow the `ERC20` standards and do not return a high-level value after approving an amount of tokens to be transferred.

As of Solidity version `0.4.22` and later, the produced code checks the size of the return value after an external call and reverts the transaction in case the return data is shorter than expected. As a result, the execution of function `_migrate()` will fail for tokens that do not return a boolean value on `IERC20::approve` calls.

`ZtakingPool::_migrate():210`

```
function _migrate(...) internal {
    ...
    for (uint256 i; i < length; ++i) {
        // Dedaub: Non-ERC20 compliant tokens won't be able to be
        //         migrated. Consider using the safeApprove instead.
        IERC20(_tokens[i]).approve(_migratorContract, _amounts[i]);
    }
    ...
}
```

For that reason, we suggest using the `safeApprove` alternative wrapper which handles such tokens and ensures that they would be successfully migrated.

Comments:
According to the protocol team, no non-standard ERC20 tokens will be supported for staking.

CENTRALIZATION ISSUES:

It is often desirable for DeFi protocols to assume no trust in a central authority, including the protocol's owner. Even if the owner is reputable, users are more likely to engage with a protocol that guarantees no catastrophic failure even in the case the owner gets hacked/compromised. We list issues of this kind below. (These issues should be considered in the context of usage/deployment, as they are not uncommon. Several high-profile, high-value protocols have significant centralization threats.)

ID	Description	STATUS
N1	A compromised Zircuit signer would not serve its purpose	**ACKNOWLEDGED**

The Zircuit Signer signs valid migration messages for users who want to migrate using the `migrate()` function. The migration through the `migrate()` function cannot be completed without this signature. This is intended to protect users from using unauthorized or malicious migrator contracts, as they could be the victims of a phishing or similar attack. The protocol owners and operators would have to ensure that the Zircuit signer remains uncompromised (while also operating, most probably as part of the backend infrastructure), as otherwise it could be used to sign validation messages for malicious migrator contracts.

OTHER / ADVISORY ISSUES:

This section details issues that are not thought to directly affect the functionality of the project, but we recommend considering them.

ID	Description	STATUS
A1	Migration's destination address could be `0`	**INFO**
	There is no check in the `_migrate()` function to ensure that the `destination` is not the `0` address. The migrator contract could implement this check, still failing early would not hurt.	
A2	Compiler version considerations	**INFO**
	As of solidity version `0.8.20` and later the `PUSH0` opcode was introduced. It is already supported by most EVM chains, but there may be some that have not added support for it yet. We raise this consideration to warn the developers that any chain besides Ethereum, on which the protocol may be deployed in the future, has to support this opcode for the protocol to function properly as it is compiled with version `0.8.24`.	
A3	Compiler bugs	**INFO**
	The code is compiled with Solidity `0.8.24`. Version `0.8.24`, in particular, has no known bugs at the time of writing.	

DISCLAIMER

The audited contracts have been analyzed using automated techniques and extensive human inspection in accordance with state-of-the-art practices as of the date of this report. The audit makes no statements or warranties on the security of the code. On its own, it cannot be considered a sufficient assessment of the correctness of the contract. While we have conducted an analysis to the best of our ability, it is our recommendation for high-value contracts to commission several independent audits, a public bug bounty program, as well as continuous security auditing and monitoring through Dedaub Security Suite.

ABOUT DEDAUB

Dedaub offers significant security expertise combined with cutting-edge program analysis technology to secure some of the most prominent protocols in DeFi. The founders, as well as many of Dedaub's auditors, have a strong academic research background together with a real-world hacker mentality to secure code. Protocol blockchain developers hire us for our foundational analysis tools and deep expertise in program analysis, reverse engineering, DeFi exploits, cryptography and financial mathematics.