CD SECURITY

AUDIT REPORT

The Universal Page Vault
January 2024

Introduction

A time-boxed security review of the **Universal Page Vault** protocol was done by **CD Security**, with a focus on the security aspects of the application's implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs, and on-chain monitoring are strongly recommended.

About **Universal Page Vault**

The Universal Page Vault is a staking solution for LYX holders on the LUKSO blockchain. It allows users to deposit their LYX tokens and earn rewards. This happens when enough LYX is accumulated inside the vault and a validator is registered.

1. Users can deposit `LYX` through the Vault's `deposit()` function.
2. When the vault accumulates 32 `LYX`, it is automatically deposited into the LUKSO deposit contract through `registerValidator()`.
3. After about 16 hours, a new validator is added to the LUKSO network.
4. The vault then receives staking rewards from the deposit contract and distributes them to the vault depositors.

Users can withdraw their funds but this can take some time if there are not enough LYX tokens in the Vault, and exiting a validator is required.

More docs here.

Threat Model

Privileged Roles & Actors

- Owner - the owner of the vault has the privilege to pause, unpause, and set important variables like `depositLimit` and the fee percent.
- Oracle - the oracle is responsible for rebalancing the vault balances and registering validators.
- `FeeRecipient` - a special role that can withdraw the fees collected by the vault.

Security Interview

Q: What in the protocol has value in the market?

A: The deposited LYX which is used to register validators and accrue rewards(in LYX again).

Q: In what case can the protocol/users lose money?

A: If the LYX is drained from the contract or in case there is a calculation error in `claim`, `withdraw`, `rebalance` and `claimFees` functions.

Q: What are some ways that an attacker achieves his goals?

A: To withdraw the funds from other users, to claim their rewards, or to cause a Denial-of-Service.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* - 266b2fe8020ba9f0431ae43077d2380b32a54f56**

Scope

The following smart contracts were in scope of the audit:

- `./contracts/blob/main/src/pool/Vault.sol`

The following number of issues were found, categorized by their severity:

- Critical & High: 2 issues
- Medium: 3 issues
- Low: 1 issues
- Informational: 3 issues

Findings Summary

ID	Title	Severity
[H-01]	Withdrawing funds can lead to wrong accounting	High
[H-02]	Exiting a validator will not be noticed by the Vault	High

ID	Title	Severity
[M-01]	Fee can be set to 100%	Medium
[M-02]	Users can't request to withdraw at any time like the docs say	Medium
[M-03]	Insufficient input validation	Medium
[L-01]	Use `Ownable2StepUpgradeable` instead of `OwnableUnset`	Low
[I-01]	Unnecessary `depositLimit` check inside `registerValidator`	Informational
[I-02]	NatSpec docs are missing	Informational
[I-03]	Missing event emissions in state changing methods	Informational

Detailed Findings

[H-01] Withdrawing funds can lead to wrong accounting

Severity

Impact: High, because it will mess up the internal accounting balances of the Vault

Likelihood: Medium, because it will happen when `pendingWithdrawal` is a small sum

Description

The users can withdraw their funds from the Vault and if there is not enough LYX in the contract (`totalUnstaked`), a `delayedAmount` will be deducted from `totalStaked` and be assigned to `_pendingWithdrawals[beneficiary]`:

```
    function withdraw(uint256 amount, address beneficiary) external
nonReentrant whenNotPaused {
        ...
        uint256 immediateAmount = amount > totalUnstaked ? totalUnstaked :
amount;
        uint256 delayedAmount = amount - immediateAmount;

        totalUnstaked -= immediateAmount;
        totalStaked -= delayedAmount; //@audit totalStaked is decreased
        totalPendingWithdrawal += delayedAmount;
        _pendingWithdrawals[beneficiary] += delayedAmount;
        ...
    }
```

This means that when LYX is unstaked and a validator is exited, the user will be able to `claim` the `delayedAmount` which is already deducted from the `totalStaked`. However, the LYX is still staked and the validator is running until it is exited.

However, as mentioned above, rewards are constantly sent to the Vault automatically. This can lead to a scenario when the `_pendingWithdrawals[beneficiary]` is a small amount and it is beneficial for the Vault to wait few days and to accrue rewards instead of exiting a validator. As mentioned in the docs, a withdrawal can take up to 8 days.

Let's look at an example:

1. A user wants to withdraw `10 LYX` but `totalUnstaked = 9.5 LYX`.
2. `0.5 LYX` is deducted from `totalStaked` and `_pendingWithdrawals[beneficiary] += 0.5 LYX`.
3. Instead of exiting validator, there are enough rewards so the user can `claim` the `delayedAmount` which is `0.5`.
4. Now, the `totalStaked` is `0.5` less than it should be because the funds were deducted beforehand.

In the long run, this will seriously mess up the whole internal accounting if this happens a lot of times which is quite likely for small amounts.

Recommendations

Decrease `totalStaked` only when the LYX is actually unstaked and sent to the Vault.

Client

Fixed - the `rebalance` function was redesigned

[H-02] Exiting a validator will not be noticed by the Vault

Severity

Impact: High, as the number of validators can DoS the Vault

Likelihood: Medium, as it will happen when validators are exited and re-registered

Description

The Vault has a `depositLimit` that ensures no more LYX can be deposited when a certain number is reached. Inside `deposit`, there is the following check:

```
    uint256 newTotalDeposits = Math.max(validators * DEPOSIT_AMOUNT,
totalStaked + totalUnstaked) + amount;

    if (newTotalDeposits > depositLimit) {
      revert DepositLimitExceeded(newTotalDeposits, depositLimit);
```

The `newTotalDeposits` take the higher number between `validators * DEPOSIT_AMOUNT` and `totalStaked + totalUnstaked`, adds the `amount`, and compares it to the `depositLimit`. The problem is that when a validator is exited, the `validators` variable inside the Vault is not decreased as it should be.

For example, if 10 validators are registered and after some time 5 of them are exited because of user withdrawals the `validators * DEPOSIT_AMOUNT` will be still equal to 320 while the `totalStaked + totalUnstaked` will be much smaller because of funds exiting the Vault. For simplicity, if the `depositLimit` is set to 320, no more users will be able to deposit LYX even though the funds in the Vault are much less than the limit.

Recommendations

Consider comparing only `totalStaked + totalUnstaked` to the `depositLimit` as this should represent the actual balance related to the Vault. Additionally, consider implementing a mechanism to decrease the `validators` variable when a validator is exited.

Client

Acknowledged. We refer to validators to track indexes and prevent deposits exceeding number of validators we can run

[M-01] Fee can be set to 100%

Impact: Medium, because the protocol can be broken and the code could give false calculations

Likelihood: Medium, as it can be gamed but it needs a compromised / malicious owner

Description

In the Vault docs we can see the following:

`The vault fee is 8% of the rewards earned by the vault.`

Now let's look at the code:

```
function setFee(uint32 newFee) external onlyOwner {
        if (newFee > _FEE_BASIS) {
            revert InvalidAmount(newFee);
        }
        ...
```

`_FEE_BASIS` is equal to 100% meaning that the `newFee` can be set to be up to 100%. Plus there is not a lower constraint, meaning the fee can be set to 0% as well.

Recommendations

Either mention in the docs that the fee can be changed but it is 8% for now or set a reasonable upper and lower constraint for `fee`. For example, 15% as an upper limit and 2% as a lower limit, if the `newFee` is set out of these boundaries - revert.

Client

Fixed

[M-02] Users can't request to withdraw at any time like the docs say

Impact: Medium, because there is a scenario where the users won't be able to withdraw their funds

Likelihood: Medium, as it requires compromised / malicious owner to pause the `withdraw` functionality

Description

In the Vault documentation page, we can see the following answer to the FAQ "How do I withdraw?":

`You can withdraw your LYX and rewards at any time.`

But that is not 100% true. There is a pause functionality controlled by the owner present in the contract. Meaning if there is a malicious or compromised owner, he can pause the contract which will affect exactly the `withdraw` function (other ones as well) and it will become not callable due to the `whenNotPaused` modifier in this function.

Recommendations

While this is a scenario that requires a special factor (malicious/compromised owner or an emergency situation), there is still a chance for that to occur. For that reason, we suggest not to guarantee that users will be able to withdraw their rewards at any time.

Client

Acknowledged. Ownership will be renounced later on.

[M-03] Insufficient input validation

Severity

Impact: Medium, because setting an inappropriate value here can lead to reverting of important functions

Likelihood: Medium, as requires compromised / malicious owner

Description

The `newDepositLimit` parameter in `setDepositLimit()` is missing any constraints and if we have a malicious or compromised owner or one that does a "fat-finger", can input a huge number or a really small number. This method's argument will result in a big problem in certain functions, where `depositLimit` is used.

```
function setDepositLimit(uint256 newDepositLimit) external onlyOwner {
        //@audit not constrained in any way
        uint256 previousDepositLimit = depositLimit;
        depositLimit = newDepositLimit;
        emit DepositLimitChanged(previousDepositLimit, newDepositLimit);
    }
```

Recommendations

Set a reasonable upper and lower constraint for `newDepositLimit`.

Client

Fixed

[L-01] Use `Ownable2StepUpgradeable` instead of `OwnableUnset`

`Ownable2Step` and `Ownable2StepUpgradeable` prevent the contract ownership from mistakenly being transferred to an address that cannot handle it (e.g. due to a typo in the address), by requiring that the recipient of the owner permissions actively accept via a contract call of its own. When we have an upgradeable contract like this one (`Vault.sol`), we should use the upgradable version of `Ownable2Step.sol`.

Recommendations

Use @openzeppelin/contracts-upgradeable/access/Ownable2StepUpgradeable.sol.

Client

Acknowledged

[I-01] Unnecessary `depositLimit` check inside `registerValidator`

The `registerValidator` checks if `(validators + 1) * DEPOSIT_AMOUNT > depositLimit` to make sure that if 1 more validator is registered, the sum is not more than the `depositLimit`. However, this is unnecessary because this check is already present in the `deposit` method and there is no way to register a validator if there are less than 32 LYX already in the contract.

```
        if (totalUnstaked < DEPOSIT_AMOUNT) {
            revert InsufficientBalance(totalUnstaked, DEPOSIT_AMOUNT);
        }
```

Client

Fixed

[I-02] NatSpec docs are missing

All of the external methods are missing NatSpec. NatSpec documentation is essential for a better understanding of the code by developers and auditors and is strongly recommended. Please refer to the NatSpec format and follow the guidelines outlined there.

Client

Acknowledged

[I-03] Missing event emissions in state changing methods

It's a best practice to emit events on every state changing method for off-chain monitoring. The following methods are missing event emissions, which should be added:

- `allowlist()`
- `setRestricted()`
- `registerValidator()`

Client

Acknowledged