Titan Industries

Hashrate Marketplace

coinspect

TITAN

Smart Contract Audit

Executive Summary

In **April 2023**, **Titan Industries** engaged **Coinspect** to perform a security-oriented source code review of the Hashrate Marketplace smart contracts. The Hashrate Marketplace allows sellers to offer their hashrate capability to potential buyers, who agree to escrow funds once they accept a hashrate rental contract.

The following issues were identified during the initial assessment:

High Risk	Medium Risk	Low Risk
Open **2**	Open **1**	Open **6**
Fixed 0	Fixed 0	Fixed 0
reported 2	reported 1	reported 6

Coinspect discovered a total of two high-risk issues, one of which (HRM-001) was already reported in a previous audit, that allowed sellers to steal escrowed funds in advance. The remaining high-risk issue allows buyers to steal the contract purchase value once it is correctly fulfilled by the seller. Also, Coinspect detected an informational issue reported in the last audit (HRM-010) still present in the source code.

The audit revealed a single medium-risk issue allowing buyers to purchase contracts with an arbitrary implementation.

Finally, the six low-risk issues are due to a lack of validation around contracts closeout, which allows a seller to inflate their selling history or a malicious buyer to leave a bad record to the contract, despite being fulfilled correctly. A third low-risk issue is due to the lack of validation around dead contracts: buyers can still purchase dead contracts, and a seller can set a contract as dead even if it's still ongoing. The rest of the problems are due to marketplace buyer fees not reimbursed, the possibility for the contract owner to steal funds by raising marketplace fees via front-running, and finally, sellers that can provide their own validator when creating a new contract.

Assessment and Scope

The audit started on **April 24th 2023** and was conducted on the main branch of the git repository at https://gitlab.com/TitanInd/proxy/smart-contracts.git as of commit **ab608bb132af944306602d68690754bfef21eeb2** of **April 5th 2023**. Titan Industries requested Coinspect to consider the contracts and files contained within the **Goerli/clonefactory** directory.

The audited files have the following sha256sum hash:

```
2b019bf24f713871a5627d283cb777c88447ad99eff308c2d979ba35e9bf2e29 Escrow.sol
209f88d51c53b8fecc6a9095f57527f083c81b223e2b3dbad8cda8725c5269d6 Implementation.sol
06f70cb649fe47332804daecbf36e52686dde11cb4960a5e89c66e7bbce9f026 CloneFactory.sol
1a7df61ad73a70443196b331862ed7ec552ea70c441c1f0aaccbbed5af00dcbb LumerinToken.sol
5d412bc7c6408ca5880284328fff37b85cf0ee4dee219153ca9b45d026a45972 Common.sol
```

The Hashrate Marketplace contracts allow sellers to publish hashrate rental offers, which can be later purchased by a given buyer. Payments are done via an escrow of `Lumerin` tokens, that sellers or a third-party validator can arbitrarily finish at any time to subsequently claim the escrowed funds back due to unexpected results. Also, the marketplace will initially be restricted to an allow-list of hashrate sellers defined by the Marketplace owner. On the other hand, there's no access restriction for hashrate buyers.

There are two main contracts, `CloneFactory` and `Implementation`. The former is mainly in charge of creating and maintaining instances of the latter, as well as providing admin functionality. Each `Implementation` instance represents a hashrate rental contract property of a given seller, which could be purchased multiple times by different buyers.

The contracts present a high degree of centralization given by the following facts:

- The `Lumerin` token contract is Pausable. Since hashrate contract rental payments are done in this token, a pause in this contract will pause all operations in the marketplace.
- The rental contract implementation can be upgraded at any time by the `CloneFactory` contract owner.
- Also, any contract can be marked as "dead" by the contract owner.
- The contract owner can modify the marketplace `sellerFeeRate` even after rental contracts are created. To make matters worse, contract purchase operations can be front-ran as described in HRM-008.

The overall code quality can be improved, as the audit revealed multiple findings regarding unused variables, potentially inadequate storage locations, unused functions, and lack of code documentation. Consider documenting each function following the NatSpec documentation format. Evaluate also adding additional validations to `address` type parameters to prevent users / the contract owner from providing an `address(0)` value.

The in-scope code repository includes tests, many of which seemed unrelated to the audited source code. However, multiple errors arouse at the time Coinspect attempted to run them, which were previously communicated to Titan Industries. Coinspect recommends considering adversarial case scenarios at the time of writing tests.

Summary of Findings

Id	Title	Risk	Status
HRM-001	Sellers can steal all funds in escrow in advance	High	X
HRM-002	Buyer can steal funds once contract is fulfilled	High	X
HRM-003	Platform allows purchasing rental contracts with a malicious implementation	Medium	X
HRM-004	Hashrate rental contracts validators controlled by sellers	Low	⚠
HRM-005	Buyer can negatively affect contract reputation once finished	Low	⚠
HRM-006	Seller can inflate contract selling history	Low	⚠
HRM-007	Lack of validation around dead contracts	Low	⚠
HRM-008	Marketplace admin can force higher fees by front-running a contract purchase	Low	⚠
HRM-009	Sellers can grief marketplace fees to buyers	Low	⚠
HRM-010	Event logs incorrect information	Info	⚠
HRM-011	Gas Inefficiency	Info	⚠
HRM-012	Unused function	Info	⚠

Detailed Findings

HRM-001 - Sellers can steal all funds in escrow in advance

Likelihood	**High**
Impact	**High**
Risk	**High**
Resolution	**Open**
Status	**X**
Location	**Implementation.sol**

Description

A vulnerability in the contract implementation allows a malicious seller to claim almost the entire purchase price value of the contract at the very beginning of the contract. By design, sellers are allowed to claim partial earnings from hashrate contracts while the contract is still running. To collect partial earnings, sellers have to call the `setContractCloseOut` function, which executes the following lines of code:

```
else if (closeOutType == 1) {
    //this is a function call for the seller to withdraw their funds
    //at any time during the smart contracts lifecycle
    require(
        msg.sender == seller,
        "this account is not authorized to trigger a mid-contract
closeout"
    );
    getDepositContractHodlingsToSeller(price - buyerPayoutCalc());
    ...
```

The `getDepositContractHodlingsToSeller` is as follows:

```
function getDepositContractHodlingsToSeller(uint256 remaining) internal
{
    uint256 balance = myToken.balanceOf(address(this)) - remaining;
    uint256 fee = calculateFee(balance);
    uint256 transferrableBalance = balance - fee;
    myToken.transfer(marketPlaceFeeRecipient, fee);
    myToken.transfer(escrow_seller, transferrableBalance);
}
```

Which expects as parameter the portion of the value in escrow that corresponds to the buyer. However, when this function is called, the parameter passed is the value that corresponds to the seller.

Since the `setContractCloseOut` function can be triggered at any time, if triggered at time 0, the seller can withdraw the full price amount.

Aux function:

```
function buyerPayoutCalc() internal view returns (uint256) {
    uint256 durationOfContract = (block.timestamp -
startingBlockTimestamp);
    if (durationOfContract < length) {
        return
            uint256(price * uint256(length - durationOfContract)) /
            uint256(length);
    }
    return price;
}
```

Recommendation

The `getDepositContractHodlingsToSeller` function should be called with `buyerPayoutCalc()` as parameter. This is, the value in escrow that corresponds to the buyer at that point.

Status

Open

HRM-002 - Buyer can steal funds once contract is fulfilled

Likelihood	**High**
Impact	**High**
Risk	**High**
Resolution	**Open**
Status	X
Location	**contracts/Implementation.sol:200**

Description

A buyer can steal the contract purchase value once it is correctly fulfilled, due to a calculation error in the `buyerPayoutCalc` function.

By triggering a contract close out with `closeOutType == 0` once the contract has already finished, a buyer can exploit a bug in the `buyerPayoutCalc` function shown below:

```
function buyerPayoutCalc() internal view returns (uint256) {
    uint256 durationOfContract = (block.timestamp -
startingBlockTimestamp);
    if (durationOfContract < length) {
                return
                    uint256(price * uint256(length -
durationOfContract)) /
                    uint256(length);
            }
    return price;
}
```

When `durationOfContract` (the time elapsed since the contract purchase) is higher or equal than `length` (the purchased contract length), the function below returns the full price. However, it should return 0 as the contract is already fulfilled and therefore the buyer does not have to be reimbursed.

This value is then used in the `setContractCloseOut` function, e.g.:

```
uint256 buyerPayout = buyerPayoutCalc();
withdrawFunds(price - buyerPayout, buyerPayout);
```

Which sends the full `price` amount to the contract buyer.

Recommendation

The `buyerPayoutCalc` function should return `0` once the contract is fulfilled.

Status

Open

HRM-003 - Platform allows purchasing rental contracts with a malicious implementation

Likelihood	**Low**
Impact	**High**
Risk	**Medium**
Resolution	**Open**
Status	X
Location	**CloneFactory.sol:90**

Description

The `CloneFactory` contract allows buyers to purchase rental contracts deployed outside the `CloneFactory` contract. Therefore, an adversary can deploy a malicious rental `Implementation` contract and trick victims into purchasing this contract to steal `Lumerin` funds.

The `setPurchaseRentalContract` fails to validate whether the `contractAddress` received as a parameter belongs to the `rentalContracts` array before instantiating an `Implementation` using this address.

```
function setPurchaseRentalContract(
    address contractAddress,
    string memory _cipherText
) external {
    Implementation targetContract = Implementation(contractAddress);
    uint256 _price = targetContract.price();
    ...
}
```

A similar situation occurs with the `setContractAsDead`:

```
function setContractAsDead(address _contract, bool closeout) public {
    Implementation _tempContract = Implementation(_contract);
    ...
}
```

Recommendation

Expect the and `setPurchaseRentalContract` and `setContractAsDead` functions to receive the rental contract address index in the `rentalContracts` array and retrieve the address in the given index, instead of receiving the contract address.

Otherwise validate the contract address received belongs to the `rentalContracts` array, although it may not be the most gas-efficient solution.

Status

Open

HRM-004 - Hashrate rental contracts validators controlled by sellers

Likelihood	**Medium**
Impact	**Low**
Risk	**Low**
Resolution	**Open**
Status	⚠
Location	**CloneFactory.sol**

Description

Rental contract sellers can configure a validator of their choice when creating a new rental contract with the `setCreateNewRentalContract` function below, instead of using the validator address already set in the `CloneFactory` contract. There's currently no direct impact or risk as validators do not have additional privileges than buyers. However, since the rental contracts implementation can be upgraded, an eventual upgrade could concede validators (and therefore sellers) additional privileges.

```
function setCreateNewRentalContract(
    uint256 _price,
    uint256 _limit,
    uint256 _speed,
    uint256 _length,
    address _validator,
    string memory _pubKey
) external onlyInWhitelist returns (address) {
```

Plus, the `CloneFactory` contract constructor receives a validator address - which could be address(0). However, this validator address is not read/used anywhere else.

Recommendation

Sellers shouldn't be able to choose the validator for their rental contract. Instead, contracts should use the validator configured in the `CloneFactory` contract. Validators should be impartial to both selling and buying parties.

Status

Open

HRM-005 - Buyer can negatively affect contract reputation once finished

Likelihood	Low
Impact	Low
Risk	Low
Resolution	Open
Status	⚠
Location	contracts/Implementation.sol:204

Description

A buyer can force a "bad" closeout of a contract despite being successfully fulfilled by the seller.

The `setContractCloseOut` function allows buyers to close the contract with `closeOutType == 0` even after the contract finished. This close out type generates a bad reputation record (`goodCloseout = false`) for both the seller and the buyer, as displayed below.

```
buyerHistory[buyer].push(PurchaseInfo(false,startingBlockTimestamp,
block.timestamp, price, speed, length));
sellerHistory.push(SellerHistory(false,startingBlockTimestamp,
block.timestamp, price, speed, length, buyer));
```

Being `PurchaseInfo` and `SellerHistory`:

```
struct PurchaseInfo {
    bool goodCloseout;
    uint256 _purchaseTime;
    uint256 endingTime;
    uint256 _price;
    uint256 _speed;
    uint256 _length;
}
```

```
struct SellerHistory {
    bool goodCloseout;
    uint256 _purchaseTime;
    uint256 endingTime;
    uint256 _price;
    uint256 _speed;
    uint256 _length;
    address _buyer;
}
```

An adversary determined to damaging the seller's image can take advantage of this closeout type once the contract finished correctly.

Recommendation

Do not allow buyers to use `closeOutType == 0` once the contract has finished.

Plus, the buyer can also negatively affect the seller's reputation intentionally by cancelling the contract at the very last minute. Consider implementing a mechanism to deter this kind of behavior.

Status

Open

HRM-006 - Seller can inflate contract selling history

Likelihood	Low
Impact	Low
Risk	Low
Resolution	Open
Status	⚠
Location	Implementation.sol

Description

A seller can manipulate the contract to generate numerous `SellerHistory` instances, artificially boosting their sales record.

The `setContractCloseOut` function does not check whether a contract is already closed. Therefore, once a contract finished, the seller calling this function multiple times with `closeOutType == 2` can generate and save multiple `SellerHistory` objects into the `sellerHistory` array.

```
else if (closeOutType == 2 || closeOutType == 3) {
    require(
        block.timestamp - startingBlockTimestamp >= length,
        "the contract has yet to be carried to term"
    );
    if (closeOutType == 3) {
        withdrawFunds(myToken.balanceOf(address(this)), 0);
    }
    buyerHistory[buyer].push(PurchaseInfo(true,startingBlockTimestamp,
block.timestamp, price, speed, length));
    sellerHistory.push(SellerHistory(true,startingBlockTimestamp,
block.timestamp, price, speed, length, buyer));
    setContractVariableUpdate();
    emit contractClosed(buyer);
}
```

Recommendation

Once a contract has been closed, do not permit any further closure.

Status

Open

HRM-007 - Lack of validation around dead contracts

Likelihood	**Low**
Impact	**Low**
Risk	**Low**
Resolution	**Open**
Status	⚠
Location	**CloneFactory.sol**

Description

Poor validation allows the marketplace contract owner and contract sellers to mark ongoing contracts as dead. On the other hand, buyers can purchase dead contracts. While there may not be an immediate effect, a potential adversary could abuse this to create confusion among systems or users who rely on this information.

The `CloneFactory` contract owner or hashrate rental contract sellers can set a contract of their own as dead, by calling the `setContractAsDead` function below. However, this function does not check whether a contract is still running before adding the contract to the `isContractDead` mapping.

```
function setContractAsDead(address _contract, bool closeout) public {
    Implementation _tempContract = Implementation(_contract);
    require(
        msg.sender == owner || msg.sender == _tempContract.seller(),
        "you arent approved to mark this contract as dead"
    );
    isContractDead[_contract] = true;
    if (closeout) {
        _tempContract.setContractCloseOut(4);
    }
}
```

On the other hand, buyers can still purchase a dead contract since the `isContractDead` mapping is not utilized elsewhere within the source code.

Recommendation

Check whether a contract is still ongoing before adding it to the `isContractDead` mapping.

Consider validating whether a contract is dead in the `setPurchaseRentalContract` function to prevent the purchase of obsolete contracts.

Status

Open

HRM-008 - Marketplace admin can force higher fees by front-running a contract purchase

Likelihood	Low
Impact	Low
Risk	Low
Resolution	Open
Status	⚠
Location	CloneFactory.sol

Description

The `CloneFactory` contract owner can front-run a contract purchase transaction to raise Hashrate Marketplace fees. This would considerably increase the rental contract cost for the buyer and/or the seller, leading to the unexpected loss of funds.

The `CloneFactory` contract owner is allowed to modify the seller and buyer fees by the functions below:

```
function setChangeSellerFeeRate(uint256 _newFee) external onlyOwner {
    sellerFeeRate = _newFee;
}

function setChangeBuyerFeeRate(uint256 _newFee) external onlyOwner {
    buyerFeeRate = _newFee;
}
```

Also, note that the `setPurchaseRentalContract` function below uses the `buyerFeeRate` variable to calculate the marketplace fee:

```
function setPurchaseRentalContract(
    address contractAddress,
    string memory _cipherText
) external {
    Implementation targetContract = Implementation(contractAddress);
    uint256 _price = targetContract.price();
    uint256 _marketplaceFee = _price / buyerFeeRate;
...
}
```

A prerequisite for this attack to work is for the contract to posses enough `Lumerin` allowance for the victim's funds.

Recommendation

Allow the buyer to provide a maximum `buyerFeeRate` accepted for the purchase to succeed.

Status

Open

HRM-009 - Sellers can grief marketplace fees to buyers

Likelihood	**Low**
Impact	**Low**
Risk	**Low**
Resolution	**Open**
Status	⚠
Location	**CloneFactory.sol** **Implementation.sol**

Description

In the event of a contract's early termination, the marketplace does not refund buyer fees. However, seller fees are reimbursed in such instances. This discrepancy, combined with sellers' ability to control the rental contract validator (refer to HRM-004), enables them to launch a griefing attack targeting buyer marketplace fees. Consequently, buyers will lose their marketplace fees in this scenario.

When buyers purchase rental contracts by calling the `setPurchaseRentalContract` function, the `CloneFactory` contract deducts the buyer fees from the amount transferred.

```
function setPurchaseRentalContract(
    address contractAddress,
    string memory _cipherText
) external {
    Implementation targetContract = Implementation(contractAddress);
    uint256 _price = targetContract.price();
    uint256 _marketplaceFee = _price / buyerFeeRate;

    uint256 requiredAllowance = _price + _marketplaceFee;
    uint256 actualAllowance = lumerin.allowance(msg.sender,
address(this));

    require(actualAllowance >= requiredAllowance, "not authorized to
spend required funds");
    bool tokensTransfered = lumerin.transferFrom(
        msg.sender,
        contractAddress,
        _price
    );

    require(tokensTransfered, "lumerin transfer failed");

    bool feeTransfer = lumerin.transferFrom(
        msg.sender,
        marketPlaceFeeRecipient,
        _marketplaceFee
    );
    ...
}
```

However, when calling the `setContractCloseOut` with `closeOutType == 0`, these fees are not reimbursed:

```
function setContractCloseOut(uint256 closeOutType) public {
    if (closeOutType == 0) {
        //this is a function call to be triggered by the buyer or
validator
        //in the event that a contract needs to be canceled early for
any reason
        require(
            msg.sender == buyer || msg.sender == validator,
            "this account is not authorized to trigger an early
closeout"
        );

        uint256 buyerPayout = buyerPayoutCalc();

        withdrawFunds(price - buyerPayout, buyerPayout);

buyerHistory[buyer].push(PurchaseInfo(false,startingBlockTimestamp,
block.timestamp, price, speed, length));

        sellerHistory.push(SellerHistory(false,startingBlockTimestamp,
block.timestamp, price, speed, length, buyer));
        setContractVariableUpdate();
        emit contractClosed(buyer);
    }
    ...
}
```

Recommendation

Return the buyer fees upon a `closeOutType == 0` closeout, in proportion to the rental contract duration.

Status

Open

HRM-010 - Event logs incorrect information

Likelihood	_
Impact	_
Risk	Info
Resolution	**Open**
Status	⚠
Location	**Implementation.sol:132**

Description

When the `setPurchaseContract` function is called, the `contractPurchased` event is emitted. However, it receives `msg.sender` as a parameter which is the address of the `CloneFactory` contract. Therefore, all the events will log the `CloneFactory` contract address upon every purchase, which does not provide any value.

```
emit contractPurchased(msg.sender);
```

Recommendation

Consider replacing the CloneFactory contract address by the buyer address (`_buyer`) instead.

Status

Open

HRM-011 - Gas Inefficiency

Likelihood	_
Impact	_
Risk	**Info**
Resolution	**Open**
Status	⚠
Location	**CloneFactory.sol**

Description

The source code can be refined to decrease gas consumption, consequently lowering transaction expenses.

The following code can be improved by using `whitelist[msg.sender] || noMoreWhitelist` instead.

```
require(
    whitelist[msg.sender] == true || noMoreWhitelist == true,
    "you are not an approved seller on this marketplace"
);
```

On the other hand, the code contains multiple functions with `string memory` parameters. Consider switching its storage location to `calldata`.

Also, the `CloneFactory.sol` constructor stores both the `Lumerin` token object as well as the Lumerin contract deploy address. Consider storing just one.

```
constructor(address _lmn, address _validator) {
    Implementation _imp = new Implementation();
    baseImplementation = address(_imp);
    lmnDeploy = _lmn; //deployed address of lumeirn token
    validator = _validator;
    lumerin = Lumerin(_lmn);
    owner = msg.sender;
    marketPlaceFeeRecipient = msg.sender;
    buyerFeeRate = 100;
    sellerFeeRate = 100;
}
```

Finally, the `address webfacingAddress` variable declared in the `CloneFactory` contract is not set or accessed. Consider deleting it.

Recommendation

Below is provided a quick recap of suggestions made in the previous section:

- If `variable` is boolean, consider evaluating this value instead of `variable == true`
- Choose `calldata` storage type over `memory` when possible
- Avoid storing duplicate information
- Remove unused variables

Status

Open

HRM-012 - Unused function

Likelihood	_
Impact	_
Risk	**Info**
Resolution	**Open**
Status	⚠
Location	**Escrow.sol**

Description

The internal `dueAmount` function from the `Escrow` contract is not used throughout the in-scope source code.

```
function dueAmount() internal returns (uint256) {
    if (myToken.balanceOf(address(this)) > contractTotal) {
        myToken.transfer(
            escrow_purchaser,
            myToken.balanceOf(address(this)) - contractTotal
        );
        return 0;
    }
    return contractTotal - myToken.balanceOf(address(this));
}
```

Recommendation

Consider deleting the `dueAmount` function.

Status

Open

Disclaimer

The information presented in this document is provided "as is" and without warranty. The present security audit does not cover any off-chain systems or frontends that communicate with the contracts, nor the general operational security of the organization that developed the code.