Titan Industries

Hashrate Marketplace

Smart Contracts

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	Open	Open
0	**0**	**0**
Fixed	Fixed	Fixed
3	1	2
Partially Fixed	Partially Fixed	Partially Fixed
0	0	1
Acknowledged	Acknowledged	Acknowledged
0	0	1
Deferred	Deferred	Deferred
0	0	2
Reported	Reported	Reported
3	1	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.

Pre-launch review

After the final review previous to the deployment on a production environment, Coinspect discovered a high-risk front-running scenario allowing sellers to steal buyers' funds, as described in `HRM-013`.

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 buyers 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.

Fix review

Titan Industries provided Coinspect with a revised version of the contracts on June 21st, 2023, that resolved the majority of the issues highlighted in this document. Additionally, Coinspect received supplementary support data from Titan Industries, including comments on the fixes and specific commit hashes for most of the issues addressed.

Pre-launch review

On September 13th 2023 Titan Industries requested Coinspect to conduct a final code review on the latest modifications included in commit with hash **9f5db4d96fe0aaf9f3449d90af919638b0f614ef** to the Hashrate Marketplace smart contracts listed below **only**, .

```
846b4d238df4726751cc0667ea688d24fa07af5bb3e09ef6fb7bfddebeb3fba5  CloneFactory.sol
5b37e6aa0057d212e5b1e575976f91095540bcfed387436a7fd3b5d462d5fd95  Escrow.sol
164f760718aa6e67199a282ba7244306ef42a7f80399c9799df717cc066e0150  Implementation.sol
9ac8640cb134c59384f10df1e04b2ffb2c47597c4f49fd2a6f3ca73719f6852f  Shared.sol
```

Note that the tests corresponding to the smart contracts were left out of scope, as these were not ready. The primary objective of the present review is to finalize a production-ready version of the contracts.

Key changes since the last analysis include a new requirement which expects sellers to pay a fee upon listing a rental contract. This is over and above the fee that is applied to both buyers and sellers upon the acquisition of hash-rate contracts. Notably, these fees will be collected in the native currency of the target chain, overriding the earlier practice of using the `Lumerin` token. Similarly, sellers are now subjected to rental contract fees every time they withdraw funds held in escrow, diverging from the earlier per-contract basis.

On the other hand, when listing a contract for rent, sellers can specify the address of the validator for such contract. Validators possess the authority to initiate a `closeOutType=0`, predominantly designed for buyers looking to terminate contracts prematurely. Consequently, by controlling these validators —and even at the risk of

accruing a negative `HistoryEntry` on their rental contract— sellers can unilaterally dissolve contracts, and cause the buyer to lose the flat marketplace fee which is non-refundable.

The `CloneFactory` and `Implementation` contracts are now upgradeable too. This grants the Hashrate Marketplace owner the ability to modify its functionality, and therefore any value within these contracts. Coinspect added a suggestion to prevent the contract implementation from being initialized or reinitialized by third-parties, further explained in `HRM-014`.

Finally, a few additional gas optimization opportunities are suggested in `HRM-015`.

Pre-launch fix review

Titan Industries has rolled out a new commit with hash **ff908e4bfac69273e0ccccb81fbb4e202b47bd27** that addresses issue `HRM-013`. This update introduces a versioning system for rental contract `Terms`. Now, Buyers must specify the `Terms` version they are agreeing to. Should this version not align with the current contract version, the purchase will be halted.

After thorough review, Coinspect has determined that the resolution is appropriate, and there are no pressing issues preventing the contract's deployment to production.

Summary of Findings

Id	Title	Risk	Status
HRM-001	Sellers can steal all funds in escrow in advance	High	✓
HRM-002	Buyer can steal funds once contract is fulfilled	High	✓
HRM-003	Platform allows purchasing rental contracts with a malicious implementation	Medium	✓
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	✓
HRM-013	[Pre-launch] Seller can steal buyer's funds by front-running contract purchase	High	✓
HRM-014	[Pre-launch] Proxy implementation contracts can be initialized in their own context	Info	⚠
HRM-015	[Pre-launch] Gas inefficiencies	Info	⚠

Detailed Findings

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

Likelihood	**High**
Impact	**High**
Risk	**High**
Resolution	**Fixed**
Status	✓
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

Fixed in commit with hash **5f822688904dacdf0a6fb765b487920f38d570ba**. The `getDepositContractHodlingsToSeller` function is now called with the `buyerPayoutCalc` function call result as parameter, as recommended previously.

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

Likelihood	**High**
Impact	**High**
Risk	**High**
Resolution	**Fixed**
Status	✓
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

Fixed in commit with hash **7fe936716817aa487f4e431d5ef1ae1e6dd82f72**. The `buyerPayoutCalc` function now returns `0` once the hashrate rental contract has elapsed.

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

Likelihood	**Low**
Impact	**High**
Risk	**Medium**
Resolution	**Fixed**
Status	✓
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

Fixed in commit with hash **12f2c3470e30cfec3dc09ec8f355618c02b50704**. The addresses of newly created hashrate rental contracts on the platform are now recorded in the `mappedContracts` mapping. Both the `setPurchaseRentalContract` and `setContractAsDead` functions have been updated to verify whether the contractAddress passed as a parameter actually belongs to the platform.

HRM-004 - Hashrate rental contracts validators controlled by sellers

Likelihood	**Medium**
Impact	**Low**
Risk	**Low**
Resolution	**Deferred**
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 should not 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

Deferred. Titan has decided to leave this issue on hold until external validators are implemented.

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

Likelihood	**Low**
Impact	**Low**
Risk	**Low**
Resolution	**Fixed**
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

Fixed in commit with hash **3566d45527710f35a00575fac84d9d145101f165**. The `setContractCloseOut` function now checks if the rental contract has elapsed. If so, it sets `goodCloseOut = true` in both the `SellerHistory` and `PurchaseInfo` structures (unified into the `HistoryEntry` structure in later commits) when selecting `closeOutType == 0`.

HRM-006 - Seller can inflate contract selling history

Likelihood	**Low**
Impact	**Low**
Risk	**Low**
Resolution	**Fixed**
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

Fixed in commit with hash **1962c5483c8f829c4da67fb073d601bd9eb74b2a**. When called with `closeOutType` 2 or 3, the `setContractCloseOut` function now verifies that the rental contract is in Running state before updating the `History` entry.

HRM-007 - Lack of validation around dead contracts

Likelihood	**Low**
Impact	**Low**
Risk	**Low**
Resolution	**Partially Fixed**
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

Partially fixed, commit hash reviewed **da82b0cb840805dc575f8ab762b7a5f16b787e23**. The `CloneFactory` contract now implements a function `setContractDeleted`, which allows pausing and unpausing rental contracts. Buyers are not able to purchase dead contracts anymore. However, sellers can still set contracts as deleted while in Running state.

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

Likelihood	**Low**
Impact	**Low**
Risk	**Low**
Resolution	**Acknowledged**
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

Titan has Acknowledged this issue.

HRM-009 - Sellers can grief marketplace fees to buyers

Likelihood	**Low**
Impact	**Low**
Risk	**Low**
Resolution	**Deferred**
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

Deferred. Titan has decided to address this issue in the future for a larger mainnet release.

HRM-010 - Event logs incorrect information

Likelihood	_
Impact	**Recommendation**
Risk	**Info**
Resolution	**Acknowledged**
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

Acknowledged. Titan has decided to keep the event for troubleshooting purposes.

HRM-011 - Gas inefficiency

Likelihood	_
Impact	Recommendation
Risk	Info
Resolution	Deferred
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

Deferred. Titan has decided to add this change in a later release.

HRM-012 - Unused function

Likelihood	_
Impact	Recommendation
Risk	Info
Resolution	Fixed
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

Fixed in commit with hash **19bb2496e71ebe5cf65a02e2c7373f16bdd9696a**. The `dueAmount` function was removed from the contract.

HRM-013 - [Pre-launch] Seller can steal buyer's funds by front-running contract purchase

Likelihood	**High**
Impact	**High**
Risk	**High**
Resolution	**Fixed**
Status	✓
Location	`CloneFactory.sol`

Description

Sellers can alter the terms of a rental contract just prior to its purchase by a buyer. As a result, seller can steal the buyer's `Lumerin` funds —specifically, up to the allowance granted to the `CloneFactory` contract— by setting a very short contract duration.

Once the buyer purchased a contract by calling the `setPurchaseRentalContract` function, the seller can front-run this transaction and update the rental contract terms via the `setUpdateContractInformation` function.

For instance, a malicious seller could select a very short contract duration and set the contract price to the maximum `Lumerin` allowance granted to the `CloneFactory` contract. As soon as this ill-modified contract duration expires, the seller can claim the `Lumerin` funds held in escrow.

Recommendation

Integrate a time-lock mechanism in the `setUpdateContractInformation` function when the contract is in the `Available` state. This time-lock will delay the application of any modified rental contract terms. Once the time-lock is up, any subsequent purchase should reflect the updated terms.

Status

Fixed. In the commit with hash **ff908e4bfac69273e0ccccb81fbb4e202b47bd27**, the rental contract `Terms` have been versioned. Buyers are now required to specify which `Terms` version they are agreeing to. If there's a discrepancy between the buyer's stated `Terms` version and the current contract version, the rental contract purchase will be rejected.

HRM-014 - [Pre-launch] Proxy implementation contracts can be initialized in their own context

Likelihood	_
Impact	**Recommendation**
Risk	**Info**
Resolution	**Deferred**
Status	⚠

Location	`CloneFactory.sol`
	`Implementation.sol`
	`Escrow.sol`

Description

Any individual can initialize the proxy implementation contract in its own context once it's deployed. Nonetheless, this does not represent a direct risk as proxies are not deployed using the `UUPS` mechanism. Using `UUPS` proxies with the current version of the contracts would allow adversaries to override the implementation contract.

Recommendation

Consider invoking the `_disableInitializers` function within constructors. This action ensures that implementations are not initialized in their own context. Otherwise, refrain from deploying the present implementation using a `UUPS` proxy.

Status

Deferred. Titan has decided to add this change at a later stage.

HRM-015 - [Pre-launch] Gas inefficiencies

Likelihood	_
Impact	Recommendation
Risk	Info
Resolution	Deferred
Status	⚠
Location	`CloneFactory.sol` `Implementation.sol`

Description

Coinspect detected a few gas improvement opportunities across the code.

For instance, there is no need to define the variables in the snippet below as its value could be used straight into the `require` statement.

```
uint256 requiredAllowance = _price;

uint256 actualAllowance = lumerin.allowance(msg.sender, address(this));

require(
    actualAllowance >= requiredAllowance,
    "not authorized to spend required funds"
);
```

Plus, the `require` statement in the `payMarketplaceFee` function is executed once again, right after calling the `payMarketplaceFee` function. For instance, in the `setPurchaseRentalContract` and `setCreateNewRentalContract` functions from the `CloneFactory` contract, and the `setContractCloseOut` function from the `Implementation` contract, the `require` is executed twice.

```
/* ETH buyer marketplace purchase fee */
bool sent = payMarketplaceFee(); <-- require() executed 1st time
require(sent, "Failed to pay marketplace purchase fee"); <-- require()
executed 2nd time
```

Recommendation

Consider improving the code highlighted above to reduce gas costs.

Status

Deferred. Titan has decided to add this change at a later stage.

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.