METATRUST

Security Assessment for

DLC.Link Jul 4

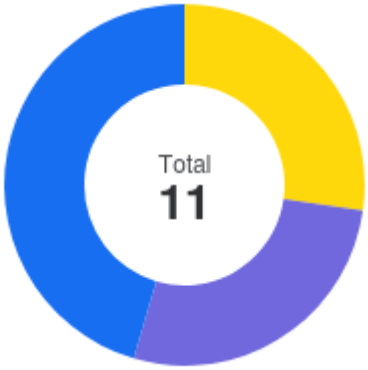
July 06, 2024

Executive Summary

Overview	
Project Name	DLC.Link Jul 4
Codebase URL	https://github.com/DLC-link/dlc-solidity
Scan Engine	Security Analyzer
Scan Time	2024/07/06 08:00:00
Commit Id	d81667bd439f0a12ed0e8f277f70ee6d6faaf0e5f340c5cae916ce987d63a3040c0f2ecf7a6cf0f0

Total	
Critical Issues	0
High risk Issues	0
Medium risk Issues	3
Low risk Issues	3
Informational Issues	5

Critical Issues	The issue can cause large economic losses, large-scale data disorder, loss of control of authority management, failure of key functions, or indirectly affect the correct operation of other smart contracts interacting with it.
High Risk Issues	The issue puts a large number of users' sensitive information at risk or is reasonably likely to lead to catastrophic impacts on clients' reputations or serious financial implications for clients and users.
Medium Risk Issues	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk Issues	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational Issue	The issue does not pose an immediate risk but is relevant to security best practices or Defence in Depth.



Critical Issues	0%	0
High risk Issues	0%	0
Medium risk Issues	27%	3
Low risk Issues	27%	3
Informational Issues	45%	5

Summary of Findings

MetaScan security assessment was performed on **July 06, 2024 08:00:00** on project **DLC.Link Jul 4** with the repository on branch **default branch**. The assessment was carried out by scanning the project's codebase using the scan engine **Security Analyzer**. There are in total **11** vulnerabilities / security risks discovered during the scanning session, among which **3** medium risk vulnerabilities, **3** low risk vulnerabilities, **5** informational issues.

ID	Description	Severity	Alleviation
MSA-001	Centralization Risk	Medium risk	Mitigated
MSA-002	Update the `dlcBTC` Incurs the Side Effect	Medium risk	Acknowledged
MSA-003	The `withdraw` Function Burns `dlcBTC`	Medium risk	Acknowledged
MSA-004	Update the `valueMinted` Incurs Side Effect	Low risk	Acknowledged
MSA-005	The `dlc` Can be Updated Repeatedly by Invoking the `setStatusFunded` and the `setStatusPending` in Order	Low risk	Fixed
MSA-006	The `msg.sender` is Not Always Same as `tx.origin`	Low risk	Fixed
MSA-007	The member `closingTxId` is deprecated	Informational	Fixed
MSA-008	Typo: Comment of the `btcTxId`	Informational	Fixed
MSA-009	Missing emit event	Informational	Fixed
MSA-010	Redundant External Call	Informational	Fixed
MSA-011	Both `valueMinted` and `valueLocked` equal to `newValueLocked`	Informational	Fixed

Findings

Medium risk (3)

1. Centralization Risk

Medium risk | Security Analyzer

In the `DLCManager` contract, the owner has the privilege of the following functions:

- `pauseContract`: Allows the admin to pause the contract;
- `unpauseContract`: Allows the admin to unpause the contract;
- `getThreshold`: Allows the admin to view the current threshold;
- `setThreshold`: Allows the admin to set a new threshold;
- `getMinimumThreshold`: Allows the admin to view the minimum threshold;
- `getSignerCount`: Allows the admin to view the number of approved signers;
- `setTSSCommitment`: Allows the admin to set the TSS commitment;
- `setAttestorGroupPubKey`: Allows the admin to set the attestor group public key;
- `whitelistAddress`: Allows the admin to whitelist an address;
- `unwhitelistAddress`: Allows the admin to remove an address from the whitelist;
- `setMinimumDeposit`: Allows the admin to set a new minimum deposit;
- `setMaximumDeposit`: Allows the admin to set a new maximum deposit;
- `setBtcMintFeeRate`: Allows the admin to set a new BTC mint fee rate;
- `setBtcRedeemFeeRate`: Allows the admin to set a new BTC redeem fee rate;
- `setBtcFeeRecipient`: Allows the admin to set a new BTC fee recipient;
- `setWhitelistingEnabled`: Allows the admin to enable or disable whitelisting;
- `transferTokenContractOwnership`: Allows the admin to transfer ownership of the token contract;
- `blacklistOnTokenContract`: Allows the admin to blacklist an address on the token contract;
- `unblacklistOnTokenContract`: Allows the admin to unblacklist an address on the token contract;
- `setMinterOnTokenContract`: Allows the admin to set a minter on the token contract;
- `setBurnerOnTokenContract`: Allows the admin to set a burner on the token contract;
- `setUserVaultUUIDs`: Allows the admin to set the UUIDs for a user's vaults;
- `importData`: Allows the admin to import various data into the contract;
- `setValueMinted`: Allows the admin to set the value minted for a specific UUID;
- `setValueLocked`: Allows the admin to set the value locked for a specific UUID.

In the `DLCManager` contract, the approved signer has the privilege of the following functions:

- `setStatusFunded`: Allows the approved signer to set the status of a DLC to funded;
- `setStatusPending`: Allows the approved signer to set the status of a DLC to pending.

In the `DLCBTC` contract, the `owner` has the privilege of the following functions:

- `mint`: Allows the owner to mint new tokens to a specified address;
- `burn`: Allows the owner to burn tokens from a specified address;
- `blacklist`: Allows the owner to add an address to the blacklist, preventing it from transferring tokens;
- `unblacklist`: Allows the owner to remove an address from the blacklist, allowing it to transfer tokens again;
- `setMinter`: Allows the owner to set the minter address;
- `setBurner`: Allows the owner to set the burner address.

In the `DLCBTC` contract, the `CCIPBurner` has the privilege of the following functions:

- `burn`: Allows the CCIPBurner to burn their own tokens.

In the `DLCBTC` contract, the `CCIPMinter` has the privilege of the following functions:

- `mint`: Allows the CCIPMinter to mint new tokens to a specified address;.

File(s) Affected

DLCManager.sol #27-27

```
27  contract DLCManager is
```

DLCBTC #23-23

```
23  contract DLCBTC is
```

Recommendation

Consider implementing a decentralized governance mechanism or a multi-signature scheme that requires consensus among multiple parties before pausing or unpausing the contract. This can help mitigate the centralization risk associated with a single owner controlling critical contract functions. Alternatively, you can provide a clear justification for the centralization aspect and ensure that users are aware of the potential risks associated with a single point of control.

Alleviation Mitigated

The team responded that the `DLCAdmin` is a 4/6 multisig of the team.

2. Update the `dlcBTC` Incurs the Side Effect

Medium risk Security Analyzer

The privileged function **`importData`** can update the key state variable, **`dlcBTC`**, which incur risk for the old dlcBTC holder.

Example:

- Alice deposits 1 BTC on BitCoin;
- Alice gets 1 dlcBTC(address(0×123)), despite any fees;
- Now, the **`dlcBTC`** address is updated to a new address(address(0×456))
- Alice only holds the old **`dlcBTC`**(address(0×123)) and can not redeem **`dlbBTC`** for the $BTC on BitCoin back.

File(s) Affected

DLCManager.sol #644-644

```
644          dlcBTC = _dlcBTC;
```

Recommendation

Here are two potential solutions:

- Require all the users redeem the old **`dlcBTC`** before updating of the **`dlcBTC`**.
- Buy back the old**`dlcBTC`** after the updating of the **`dlcBTC`**.

Alleviation Acknowledged

The team responded that it is a temporary migration function from two separate contracts to one. The TokenManager is deleted, so its data has to migrate. After migration, this function will be removed.

3. The `withdraw` Function Burns `dlcBTC`

Medium risk Security Analyzer

The **`withdraw`** function can be called by the vault creator, used to update a **`dlc`** whose status is **`FUNDED`**, and burn creator's **`dlcBTC`** on EVM chains.

The design to the **`withdraw`** function probably tries to burn **`dlcBTC`** on the EVM chains, then the other component of the DLC.Link protocol releases the locked BTC on the BitCoin.

The concern is how a user is guaranteed to receive the BTC on the BitCoin after a call to the **`withdraw`** function.

File(s) Affected

DLCManager.sol #397-418

```
397     function withdraw(
398         bytes32 uuid,
399         uint256 amount
400     ) external onlyVaultCreator(uuid) whenNotPaused {
401         DLCLink.DLC storage dlc = dlcs[dlcIDsByUUID[uuid]];
402 
403         // Validation checks
404         if (dlc.uuid == bytes32(0)) revert DLCNotFound();
405         if (dlc.status != DLCLink.DLCStatus.FUNDED) revert DLCNotFunded();
406         if (amount > dlcBTC.balanceOf(dlc.creator))
407             revert InsufficientTokenBalance(
408                 dlcBTC.balanceOf(dlc.creator),
409                 amount
410             );
411         if (amount > dlc.valueMinted) {
412             revert InsufficientMintedBalance(dlc.valueMinted, amount);
413         }
414 
415         dlc.valueMinted -= amount;
416         _burnTokens(dlc.creator, amount);
417         emit Withdraw(uuid, amount, msg.sender);
418     }
```

Alleviation Acknowledged

The team responded that BTC is only released after the user has burned their dlcBTC tokens. This is a security feature so no BTC can be released without the appropriate burn.

Low risk (3)

1. Update the `valueMinted` Incurs Side Effect

Low risk Security Analyzer

The privileged function `setValueMinted` can update `valueMinted` for a `dlc`. It lacks check if the `dlc` exists or not, and lacks check if the status of the `dlc` is ok or not for updating.

More importantly, this function will block users from withdrawing if users `dlc`'s valueMinted being decreased, due to there is substraction between `dlc.valueMinted` and the amount to be withdrew, `amount`:

```
function withdraw(
    bytes32 uuid,
    uint256 amount
) external onlyVaultCreator(uuid) whenNotPaused {
    ...
    dlc.valueMinted -= amount;
    _burnTokens(dlc.creator, amount);
    ...
}
```

File(s) Affected

DLCManager.sol #654-661

```
654     function setValueMinted(
655         bytes32 uuid,
656         uint256 valueMinted
657     ) external onlyAdmin {
658         DLCLink.DLC storage dlc = dlcs[dlcIDsByUUID[uuid]];
659         dlc.valueMinted = valueMinted;
660     }
661
```

Recommendation

Checking if it is an intended design, and removing it if it is unnecessary.

Alleviation Acknowledged

The team responded that it is only a temporary function, like the other migration functions. When the team introduces withdraw/deposit functionality, to keep existing vaults backwards compatible, the team needs to manually update them to reflect the new state machine transitions. After migration, these functions will be deleted in a second upgrade.

2. The `dlc` Can be Updated Repeatedly by Invoking the `setStatusFunded` and the `setStatusPending` in Order

Low risk Security Analyzer

The status of a `dlc` can be updated by the following functions:

- `setupVault()`: Null → READY
- `setStatusFunded()`: **READY or AUX_STATE_1(Pending)** → **FUNDED**
- `setStatusPending()`: **FUNDED** → **AUX_STATE_1(Pending)**
- `withdraw()`: FUNDED → FUNDED

The concern is that the `dlc` can be updated repeatedly by invoking the `setStatusFunded` and the `setStatusPending` in order, with the status transitions between `AUX_STATE_1(Pending)` and `FUNDED`.

As a result, the members of the `dlc` only record the latest values after a call to `setStatusFunded` or `setStatusPending`:

```
function setStatusFunded(
    bytes32 uuid,
    string calldata btcTxId,
    bytes[] calldata signatures,
    string calldata taprootPubKey,
    uint256 newValueLocked
) external whenNotPaused onlyApprovedSigners {
    ...
    dlc.fundingTxId = btcTxId;
    dlc.wdTxId = "";
    dlc.status = DLCLink.DLCStatus.FUNDED;
    dlc.taprootPubKey = taprootPubKey;

    dlc.valueLocked = newValueLocked;
    dlc.valueMinted += amountToMint;
    ...
}

function setStatusPending(
    bytes32 uuid,
    string calldata btcTxId,
    bytes[] calldata signatures,
    uint256 newValueLocked
) external whenNotPaused onlyApprovedSigners {
    _attestorMultisigIsValid(
        abi.encode(
```

```
                uuid,
                btcTxId,
                "set-status-redeem-pending",
                newValueLocked
            ),
            signatures
        );
        ...
        dlc.status = DLCLink.DLCStatus.AUX_STATE_1;
        dlc.wdTxId = btcTxId;
        ...
    }
```

Then the old values of a `dlc` will be overwritten.

We would like to confirm with client if it is an intended design, if so, at least it is recommended to record all the parameter of each call by emitting an event.

Example:

```
event SetStatusFunded(...);
emit SetStatusFunded(uuid, btcTxId, wdTxId, dlc.creator, taprootPubKey, valueLocked, valueMinted);
```

File(s) Affected

DLCManager.sol #310-355

```
310     function setStatusFunded(
311         bytes32 uuid,
312         string calldata btcTxId,
313         bytes[] calldata signatures,
314         string calldata taprootPubKey,
315         uint256 newValueLocked
316     ) external whenNotPaused onlyApprovedSigners {
317         _attestorMultisigIsValid(
318             abi.encode(uuid, btcTxId, "set-status-funded", newValueLocked),
319             signatures
320         );
321         DLCLink.DLC storage dlc = dlcs[dlcIDsByUUID[uuid]];
322 
323         if (dlc.uuid == bytes32(0)) revert DLCNotFound();
324         if (
325             dlc.status != DLCLink.DLCStatus.READY &&
326             dlc.status != DLCLink.DLCStatus.AUX_STATE_1
327         ) revert DLCNotReadyOrPending();
328 
329         if (newValueLocked < dlc.valueMinted) {
330             // During a withdrawal, a burn should have already happened
331             revert UnderCollateralized(newValueLocked, dlc.valueMinted);
332         }
333 
334         uint256 amountToMint = newValueLocked - dlc.valueMinted;
335 
336         if (amountToMint > maximumDeposit) {
337             revert DepositTooLarge(amountToMint, maximumDeposit);
338         }
339         // Add this back later when we want a minimum
340         if (amountToMint > 0 && amountToMint < minimumDeposit) {
341             revert DepositTooSmall(amountToMint, minimumDeposit);
342         }
343 
344         dlc.fundingTxId = btcTxId;
345         dlc.wdTxId = "";
346         dlc.status = DLCLink.DLCStatus.FUNDED;
347         dlc.taprootPubKey = taprootPubKey;
348 
349         dlc.valueLocked = newValueLocked;
350         dlc.valueMinted += amountToMint;
351 
352         _mintTokens(dlc.creator, amountToMint);
353 
354         emit SetStatusFunded(uuid, btcTxId, dlc.creator);
355     }
```

DLCManager.sol #365-389

```
365     function setStatusPending(
366         bytes32 uuid,
367         string calldata btcTxId,
368         bytes[] calldata signatures,
369         uint256 newValueLocked
370     ) external whenNotPaused onlyApprovedSigners {
371         _attestorMultisigIsValid(
372             abi.encode(
373                 uuid,
374                 btcTxId,
375                 "set-status-redeem-pending",
376                 newValueLocked
377             ),
378             signatures
379         );
380         DLCLink.DLC storage dlc = dlcs[dlcIDsByUUID[uuid]];
381 
382         if (dlc.uuid == bytes32(0)) revert DLCNotFound();
383         if (dlc.status != DLCLink.DLCStatus.FUNDED) revert DLCNotFunded();
384 
385         dlc.status = DLCLink.DLCStatus.AUX_STATE_1;
386         dlc.wdTxId = btcTxId;
387 
388         emit SetStatusPending(uuid, btcTxId, dlc.creator);
389     }
```

Recommendation

The findings is addressed by extending the event signatures, in the commit 9f9cfad. \

The team also responded that it is expected behaviour in the new State Machine.

Alleviation Fixed

-

3. The `msg.sender` is Not Always Same as `tx.origin`

Low risk Security Analyzer

In the `setupVault`, the `tx.origin` is assigned to `dlcs[index].creator`, but the `msg.sender` is assigned to the creator field of the event `CreateDLC`.

Note that the tx.origin is not always same as `msg.sender`, especially if a contract invoke the `setupVault` function to create a vault.

The same as the address used in the `userVaults` mapping.

The `tx.origin` is used to be the creator, in the `setupVault` function that has a modifier `onlyWhitelisted` to validate the caller, but, the implementation of the `onlyWhitelisted` checks the `msg.sender` rather than the `tx.orign`.

Or, is it an intended design for the `onlyWhitelisted` modifier to check on `msg.sender` to allow contract to create a vault.

File(s) Affected

DLCManager.sol #114-118

```
114     modifier onlyWhitelisted() {
115         if (whitelistingEnabled && !_whitelistedAddresses[msg.sender])
116             revert NotWhitelisted();
117         _;
118     }
```

DLCManager.sol #275-295

```
275         dlcs[_index] = DLCLink.DLC({
276             uuid: _uuid,
277             protocolContract: msg.sender,
278             valueLocked: 0,
279             valueMinted: 0,
280             timestamp: block.timestamp,
281             creator: tx.origin,
282             status: DLCLink.DLCStatus.READY,
283             fundingTxId: "",
284             closingTxId: "",
285             wdTxId: "",
286             btcFeeRecipient: btcFeeRecipient,
287             btcMintFeeBasisPoints: btcMintFeeRate,
288             btcRedeemFeeBasisPoints: btcRedeemFeeRate,
289             taprootPubKey: ""
290         });
291
292         emit CreateDLC(_uuid, msg.sender, block.timestamp);
293
294         dlcIDsByUUID[_uuid] = _index;
295         userVaults[msg.sender].push(_uuid);
```

Recommendation

Updating the paramter passed to the **`CreateDLC`** event, and the **`userVaults`** mapping to be **`tx.origin`**.

Checking if the **`onlyWhitelisted`** modifier tries to check on **`msg.sender`** to allow contract to create a vault.

Alleviation Fixed

The team addressed this finding by changing the `msg.sender` to the `tx.origin`, in the commit 74837e.
The team keeps the option for contracts open.

Informational (5)

1. The member `closingTxId` is deprecated

Informational Security Analyzer

Due to the delete of the **`postCloseDLC`** function, the member **`closingTxId`** turns deprecated and unused.

File(s) Affected

DLCManager.sol #284-284

```
284             closingTxId: "",
```

Alleviation Fixed

The team responded that due to upgrade-compatibility, the team cannot delete it from storage. Marked as deprecated.

2. Typo: Comment of the `btcTxId`

Informational Security Analyzer

The `setStatusPending` updates status of a `dlc` rather than funding a `dlc`, so the comment below is not right:

```
 * @param   btcTxId  DLC Funding Transaction ID on the Bitcoin blockchain.//@audit not a DLC Funding Transaction ID
...
function setStatusPending(
    bytes32 uuid,
    string calldata btcTxId,
    bytes[] calldata signatures,
    uint256 newValueLocked
) external whenNotPaused onlyApprovedSigners {
    _attestorMultisigIsValid(
        abi.encode(
            uuid,
            btcTxId,
            "set-status-redeem-pending",
            newValueLocked
        ),
        signatures
    );
...
    dlc.wdTxId = btcTxId;

    emit SetStatusPending(uuid, btcTxId, dlc.creator);
...
```

Meanwhile, the parameter `btcTxId` is assigned to the state variable `dlc.wdTxId`, so its name intended to be `wdTxId`.

File(s) Affected

DLCManager.sol #361-389

```
361       * @param   btcTxId  DLC Funding Transaction ID on the Bitcoin blockchain.
362       * @param   signatures  Signatures of the Attestors
363       * @param   newValueLocked  New value locked in the DLC. For this function this will always be 0
364       */
365      function setStatusPending(
366          bytes32 uuid,
367          string calldata btcTxId,
368          bytes[] calldata signatures,
369          uint256 newValueLocked
370      ) external whenNotPaused onlyApprovedSigners {
371          _attestorMultisigIsValid(
372              abi.encode(
373                  uuid,
374                  btcTxId,
375                  "set-status-redeem-pending",
376                  newValueLocked
377              ),
378              signatures
379          );
380          DLCLink.DLC storage dlc = dlcs[dlcIDsByUUID[uuid]];
381
382          if (dlc.uuid == bytes32(0)) revert DLCNotFound();
383          if (dlc.status != DLCLink.DLCStatus.FUNDED) revert DLCNotFunded();
384
385          dlc.status = DLCLink.DLCStatus.AUX_STATE_1;
386          dlc.wdTxId = btcTxId;
387
388          emit SetStatusPending(uuid, btcTxId, dlc.creator);
389      }
```

Recommendation

Updating the above typos.

Alleviation Fixed

The finding is addressed by the team, in the commit 150d29.

3. Missing emit event

Informational Security Analyzer

Functions update key state variables are recommended to emit event to record their historical values, for better tracking.

File(s) Affected

DLCManager.sol #629-669

```
629             address user,
630             bytes32[] calldata uuids
631         ) external onlyAdmin {
632             userVaults[user] = uuids;
633         }
634
635         function importData(
636             DLCBTC _dlcBTC,
637             string calldata _btcFeeRecipient,
638             uint256 _minimumDeposit,
639             uint256 _maximumDeposit,
640             uint256 _btcMintFeeRate,
641             uint256 _btcRedeemFeeRate,
642             bool _whitelistingEnabled
643         ) external onlyAdmin {
644             dlcBTC = _dlcBTC;
645             btcFeeRecipient = _btcFeeRecipient;
646             minimumDeposit = _minimumDeposit;
647             maximumDeposit = _maximumDeposit;
648             btcMintFeeRate = _btcMintFeeRate;
649             btcRedeemFeeRate = _btcRedeemFeeRate;
650             whitelistingEnabled = _whitelistingEnabled;
651         }
652
653         // Temporary migration functions to bring old vaults up to speed with withdraw PR
654         function setValueMinted(
655             bytes32 uuid,
656             uint256 valueMinted
657         ) external onlyAdmin {
658             DLCLink.DLC storage dlc = dlcs[dlcIDsByUUID[uuid]];
659             dlc.valueMinted = valueMinted;
660         }
661
662         function setValueLocked(
663             bytes32 uuid,
664             uint256 valueLocked
665         ) external onlyAdmin {
666             DLCLink.DLC storage dlc = dlcs[dlcIDsByUUID[uuid]];
667             dlc.valueLocked = valueLocked;
668         }
669     }
```

Recommendation

Emitting events for state variables update functions.

Alleviation Fixed

This finding is addressed by the team by emitting events, in commit 1ad8fb0

4. Redundant External Call

Informational Security Analyzer

The `getVault` function call the `getDLC` function of the same contract with the keyword `this`, which will perform an external call and use more gas.

```
    function getVault(bytes32 uuid) public view returns (DLCLink.DLC memory) {
        return this.getDLC(uuid);
    }

    function getDLC(bytes32 uuid) external view returns (DLCLink.DLC memory) {
        DLCLink.DLC memory _dlc = dlcs[dlcIDsByUUID[uuid]];
        if (_dlc.uuid == bytes32(0)) revert DLCNotFound();
        if (_dlc.uuid != uuid) revert DLCNotFound();
        return _dlc;
    }
```

File(s) Affected

DLCManager.sol #461-463

```
461     function getVault(bytes32 uuid) public view returns (DLCLink.DLC memory) {
462         return this.getDLC(uuid);
463     }
```

Recommendation

Updating the visibility of the `getDLC` from external to public and invoking it from the `getVault` function without the `this` keyword.

Alleviation Fixed

This finding is addressed, in the commit b36cba

5. Both `valueMinted` and `valueLocked` equal to `newValueLocked`

Informational Security Analyzer

In the `setStatusFunded` function, the `valueLocked` is assigned with `newValueLocked`:

```
dlc.valueLocked = newValueLocked;
```

the `valueMinted` is increased by `amountToMint`:

```
uint256 amountToMint = newValueLocked - dlc.valueMinted;
dlc.valueMinted += amountToMint;
```

Note that, the `amountToMint` comes from `newValueLocked - dlc.valueMinted`, thus, after the increment of the `valueMinted` by `amountToMint`, the `valueMinted` finally equals to the `newValueLocked`.

Consider if it is an intended design, if so the operation `+=` could be replace by an assignment as below to save gas:

```
dlc.valueMinted = newValueLocked;
```

File(s) Affected

DLCManager.sol #334-350

```
334         uint256 amountToMint = newValueLocked - dlc.valueMinted;
335
336         if (amountToMint > maximumDeposit) {
337             revert DepositTooLarge(amountToMint, maximumDeposit);
338         }
339         // Add this back later when we want a minimum
340         if (amountToMint > 0 && amountToMint < minimumDeposit) {
341             revert DepositTooSmall(amountToMint, minimumDeposit);
342         }
343
344         dlc.fundingTxId = btcTxId;
345         dlc.wdTxId = "";
346         dlc.status = DLCLink.DLCStatus.FUNDED;
347         dlc.taprootPubKey = taprootPubKey;
348
349         dlc.valueLocked = newValueLocked;
350         dlc.valueMinted += amountToMint;
```

Alleviation Fixed

This finding is addressed by the team by replacing the `+=` operation, in the commit f340c5

Audit Scope

File	SHA256	File Path
DLCManager.sol	1cd9efe5f7e2ee65680030e5768d63e8	/DLCManager.sol
DLCBTC	b3d4ef9687517def4c7a525c15a9c237	/DLCBTC

Disclaimer

This report is governed by the stipulations (including but not limited to service descriptions, confidentiality, disclaimers, and liability limitations) outlined in the Services Agreement, or as detailed in the scope of services and terms provided to you, the Customer or Company, within the context of the Agreement. The Company is permitted to use this report only as allowed under the terms of the Agreement. Without explicit written permission from MetaTrust, this report must not be shared, disclosed, referenced, or depended upon by any third parties, nor should copies be distributed to anyone other than the Company.

It is important to clarify that this report neither endorses nor disapproves any specific project or team. It should not be viewed as a reflection of the economic value or potential of any product or asset developed by teams or projects engaging MetaTrust for security evaluations. This report does not guarantee that the technology assessed is completely free of bugs, nor does it comment on the business practices, models, or legal compliance of the technology's creators.

This report is not intended to serve as investment advice or a tool for investment decisions related to any project. It represents a thorough assessment process aimed at enhancing code quality and mitigating risks inherent in cryptographic tokens and blockchain technology. Blockchain and cryptographic assets inherently carry ongoing risks. MetaTrust's role is to support companies and individuals in their security diligence and to reduce risks associated with the use of emerging and evolving technologies. However, MetaTrust does not guarantee the security or functionality of the technologies it evaluates.

MetaTrust's assessment services are contingent on various dependencies and are continuously evolving. Accessing or using these services, including reports and materials, is at your own risk, on an as-is and as-available basis. Cryptographic tokens are novel technologies with inherent technical risks and uncertainties. The assessment reports may contain inaccuracies, such as false positives or negatives, and unpredictable outcomes. The services may rely on multiple third-party layers.

All services, labels, assessment reports, work products, and other materials, or any results from their use, are provided "as is" and "as available," with all faults and defects, without any warranty. MetaTrust expressly disclaims all warranties, whether express, implied, statutory, or otherwise, including but not limited to warranties of merchantability, fitness for a particular purpose, title, non-infringement, and any warranties arising from course of dealing, usage, or trade practice. MetaTrust does not guarantee that the services, reports, or materials will meet specific requirements, be error-free, or be compatible with other software, systems, or services.

Neither MetaTrust nor its agents make any representations or warranties regarding the accuracy, reliability, or currency of any content provided through the services. MetaTrust is not liable for any content inaccuracies, personal injuries, property damages, or any loss resulting from the use of the services, reports, or materials.

Third-party materials are provided "as is," and any warranty concerning them is strictly between the Customer and the third-party owner or distributor. The services, reports, and materials are intended solely for the Customer and should not be relied upon by others or shared without MetaTrust's consent. No third party or representative thereof shall have any rights or claims against MetaTrust regarding these services, reports, or materials.

The provisions and warranties of MetaTrust in this agreement are exclusively for the Customer's benefit. No third party has any rights or claims against MetaTrust regarding these provisions or warranties. For clarity, the services, including any assessment reports or materials, should not be used as financial, tax, legal, regulatory, or other forms of advice.