FLR Wraps Audit Report

May 26, 2023

Table of Contents

Summary

This report has been prepared for FLR Wraps smart contract, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Overview

Project Summary

Project Name	**FLR Wraps**
Codebase	**https://github.com/flrfinance/flr-wraps-contracts**
Commit	**e6564c4e47ab69137a69619413d0b4d46f749357**
Language	**Solidity**

Audit Summary

Delivery Date	**May 26, 2023**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**10**

[WP-H1] Sophisticated validators can exploit the point system and rip off other honest validators

High

Issue Description

The current design of the validator points system for the distribution of validator fees is too simple, as it does not track the actual contribution in terms of the value of each execution.

As a result, the sophisticated validators can send many real bridge transactions to acquire more points and dilute the value of each point for all the other honest validators.

https://github.com/flrfinance/flr-wraps-contracts/blob/3e26cc142f9ff23eb5370583c7fd9b66d4d4da97/src/libraries/Multisig.sol#L243-L252

```
243 function incrementPoints(DualMultisig storage s, uint256 mask) private {
244     uint16 count = 0;
245     for (uint16 i = 0; i < maxSignersSize; i++) {
246         if ((mask & (1 << i)) != 0) {
247             s.points[i]++;
248             count++;
249         }
250     }
251     s.totalPoints += count;
252 }
```

https://github.com/flrfinance/flr-wraps-contracts/blob/3e26cc142f9ff23eb5370583c7fd9b66d4d4da97/src/Wrap.sol#L314-L324

```
314 function claimValidatorFees(address validator) public {
315     address feeRecipient = validatorFeeRecipients[validator];
316     uint64 totalPoints = multisig.totalPoints;
317     uint64 points = multisig.clearPoints(validator);
318     for (uint256 i = 0; i < tokens.length; i++) {
319         address token = tokens[i];
320         uint256 tokenValidatorFee = (accumulatedValidatorFees(token) *
321             points) / totalPoints;
```

```
            IERC20(token).safeTransfer(feeRecipient, tokenValidatorFee);
        }
    }
```

PoC

Given:

- validatorFeeBPS is `100` or `1%` ;
- total committee members: `10`
- quorum: `5`

1. Alice bridged 1M USDC and added 1k USDC of validatorFee to the `accumulatedValidatorFees` .
2. The 5 committee members who voted each get 1 point; each point is now worth 200 USDC.
3. The 3 sophisticated validators conspired to initiate 10 small bridge transactions, each with only 10 USDC and contributed 1 USDC for the accumulated fees in total, but acquired 30 points out of the total 50 new points.

As a result, the `totalPoints` is now `55` ; and `accumulatedValidatorFees` is now `1,001 USDC` . The 3 sophisticated validators now combine for a total of 30 points, and can take out `1001 * 30 / 55 == 546 USDC` .

The first 5 validators can only receive a total of `109.2 USDC` .

This is unexpected, as the first five validators would call `claimValidatorFees()` before the sophisticated validators diluted the value of their points.

Recommendation

Consider changing to a simpler real-time settlement solution:

Changes on `Multisig.sol`

1, Get rid of the points, including `incrementPoints()` ; 2, Add a new function called `getApprovers()` to get the list and total count of approvals;

```
function getApprovers(DualMultisig storage s, uint256 mask) private view returns
(uint16[] memory, uint16) {
    uint256 signersCount = s.firstCommitteeSize + s.secondCommitteeSize;
    uint16[] memory approvers = new uint16[](signersCount);
    uint16 count = 0;
    for (uint16 i = 0; i < signersCount; i++) {
        if ((mask & (1 << i)) != 0) {
            approvers[count] = i;
            count++;
        }
    }

    return (approvers, count);
}
```

3, Update `tryApprove()` to return the approvers list and count.

```
function tryApprove(
    DualMultisig storage s,
    address signer,
    bytes32 hash,
    uint256 id
) internal returns (RequestStatusTransition, uint16[] memory, uint16) {
    Request storage request = s.requests[hash];
    // If the request has already been accepted
    // then simply return.
    if (request.status == RequestStatus.Accepted) {
        return (RequestStatusTransition.Unchanged, new uint16[](0), 0);
    }

@@ 289,299 @@

    uint256 signerMask = 1 << signerInfo.index;
    // Check if the signer has already signed.
    if ((signerMask & request.approvers) != 0) {
        return (RequestStatusTransition.Unchanged, new uint16[](0), 0);
    }

```

```
@@ 307,313 @@

    // If quorum has been reached, increment the number of points.
    if (
        request.approvalsFirstCommittee >=
        s.firstCommitteeAcceptanceQuorum &&
        request.approvalsSecondCommittee >=
        s.secondCommitteeAcceptanceQuorum
    ) {
        request.status = RequestStatus.Accepted;
        s.approvedRequests[id] = hash;

        (uint16[] memory approvers, uint16 count) = getApprovers(s, request.approvers);
        return (RequestStatusTransition.UndecidedToAccepted, approvers, count);
    } else if (request.status == RequestStatus.NULL) {
        // If this is the first approval, change the request status
        // to undecided.
        request.status = RequestStatus.Undecided;
        return (RequestStatusTransition.NULLToUndecided, new uint16[](0), 0);
    }
    return (RequestStatusTransition.Unchanged, new uint16[](0), 0);
}
```

Changes on `Wrap.sol`

1, Add a new mapping for the fee balance of each token for each validator;

```
mapping(address => mapping(uint256 => uint256)) public feeBalance;
```

2, Change `approveExecute()` to add fee to validators' balance after execution;

```
function approveExecute(
    uint256 id,
    address mirrorToken,
    uint256 amount,
    address to
) public isNotPaused isValidMirrorTokenAmount(mirrorToken, amount) {
    // If the request ID is lower than the last executed ID then simply ignore the request.
```

```
    if (id < multisig.nextExecutionIndex) {
        return;
    }

    bytes32 hash = hashRequest(id, mirrorToken, amount, to);
    (Multisig.RequestStatusTransition transition, uint16[] approvers, uint16 approverCount) = multisig.tryApprove(
        msg.sender,
        hash,
        id
    );
    if (transition == Multisig.RequestStatusTransition.NULLToUndecided) {
        emit Requested(id, mirrorToken, amount, to);
    }

    if (multisig.tryExecute(hash, id)) {
        address token = mirrorTokens[mirrorToken]; // token -> address()
        uint256 fee = onExecute(token, amount, to);

        uint256 singerFee = fee / approverCount;
        for(uint16 i; i < approverCount; i++){
            feeBalance[token][approvers[i]] += fee;
        }
        emit Executed(id, token, amount - fee, to, fee);
    }
}
```

3, Change the `claimValidatorFees()` function to use `feeBalance` .

Status

✓ **Fixed**

[WP-M2] `WrapDepositRedeem` Wrong implementation of `accumulatedValidatorFees()`

Medium

Issue Description

https://github.com/flrfinance/flr-wraps-contracts/blob/89eca4b09a4a08857e96bd7f49b4941f31863386/src/WrapDepositRedeem.sol#L24-L32

```
function accumulatedValidatorFees(address token)
    public
    view
    virtual
    override(IWrap, Wrap)
    returns (uint256)
{
    return IERC20(token).balanceOf(address(this));
}
```

`accumulatedValidatorFees()` should not return all the account balance directly.

https://github.com/flrfinance/flr-wraps-contracts/blob/89eca4b09a4a08857e96bd7f49b4941f31863386/src/WrapDepositRedeem.sol#L51-L58

```
    function onExecute(
        address token,
        uint256 amount,
        address to
    ) internal virtual override returns (uint256 fee) {
        fee = calculateFee(amount, validatorFeeBPS);
        IERC20(token).safeTransfer(to, amount - fee);
    }
```

According to `onExecute()` , only part of the balance is the accumulated validator fees;

The majority of the balance should be the reserve for future redemption.

Recommendation

Consider changing to:

```
contract WrapDepositRedeem is IWrapDepositRedeem, Wrap {
@@ 15,17 @@

    mapping(address => uint256) public accumulatedValidatorFeeAmounts;

@@ 21,24 @@

    /// @inheritdoc IWrap
    function accumulatedValidatorFees(
        address token
    ) public view virtual override(IWrap, Wrap) returns (uint256) {
        return accumulatedValidatorFeeAmounts[token];
    }

@@ 33,45 @@

    /// @inheritdoc Wrap
    function onExecute(
        address token,
        uint256 amount,
        address to
    ) internal virtual override returns (uint256 fee) {
        fee = calculateFee(amount, validatorFeeBPS);
        accumulatedValidatorFeeAmounts[token] += fee;
        IERC20(token).safeTransfer(to, amount - fee);
    }
@@ 57,65 @@

    function claimValidatorFees(address validator) public override {
        address feeRecipient = validatorFeeRecipients[validator];
        uint64 totalPoints = multisig.totalPoints;
        uint64 points = multisig.clearPoints(validator);
        for (uint256 i = 0; i < tokens.length; i++) {
            address token = tokens[i];
            uint256 tokenValidatorFee = (accumulatedValidatorFees(token) *
                points) / totalPoints;
```

```
            accumulatedValidatorFeeAmounts[token] -= tokenValidatorFee;
            IERC20(token).safeTransfer(feeRecipient, tokenValidatorFee);
        }
    }
}
```

Status

✓ Fixed

[WP-M3] Unable to redeem and `claimValidatorFees()` on `WrapDepositRedeemCustodian`

Medium

Issue Description

`WrapDepositRedeemCustodian` inherit `WrapDepositRedeem` 's `accumulatedValidatorFees()` :

https://github.com/flrfinance/flr-wraps-contracts/blob/89eca4b09a4a08857e96bd7f49b4941f31863386/src/WrapDepositRedeem.sol#L24-L32

```
function accumulatedValidatorFees(address token)
    public
    view
    virtual
    override(IWrap, Wrap)
    returns (uint256)
{
    return IERC20(token).balanceOf(address(this));
}
```

However, since all the deposits will be transferred to the `custodian` address, including the `fee` , there will always be a balance of 0.

https://github.com/flrfinance/flr-wraps-contracts/blob/3e26cc142f9ff23eb5370583c7fd9b66d4d4da97/src/WrapDepositRedeemCustodian.sol#L28-L34

```
function onDeposit(
    address token,
    uint256 amount
) internal override returns (uint256) {
    IERC20(token).safeTransferFrom(msg.sender, custodian, amount);
    return depositFees(amount);
}
```

As a result, the validators will not be able to `claimValidatorFees()` .

Furthermore, the users also cannot redeem as there is no transfer in the `onExecute()` function:

https://github.com/flrfinance/flr-wraps-contracts/blob/3e26cc142f9ff23eb5370583c7fd9b66d4d4da97/src/WrapDepositRedeemCustodian.sol#L37-L43

```
function onExecute(
    address,
    uint256 amount,
    address
) internal view override returns (uint256 fee) {
    fee = calculateFee(amount, validatorFeeBPS);
}
```

Recommendation

Consider changing to:

```
contract WrapDepositRedeemCustodian is WrapDepositRedeem {
@@ 13,25 @@

    function accumulatedValidatorFees(
        address token
    ) public view override returns (uint256) {
        return IERC20(token).balanceOf(address(this));
    }

@@ 33,40 @@

    /// @inheritdoc WrapDepositRedeem
    function onExecute(
        address,
        uint256 amount,
        address
    ) internal view override returns (uint256 fee) {
        fee = calculateFee(amount, validatorFeeBPS);
        IERC20(token).transferFrom(custodian, to, amount - fee);
        IERC20(token).transferFrom(custodian, address(this), fee);
    }
}
```

Status

✓ Fixed

[WP-M4] Potential DoS attack by initiating a bridge transaction that cannot be fulfilled and block the entire queue

Medium

Issue Description

https://github.com/flrfinance/flr-wraps-contracts/blob/3e26cc142f9ff23eb5370583c7fd9b66d4d4da97/src/libraries/Multisig.sol#L327-L337

```
function tryExecute(
    DualMultisig storage s,
    bytes32 hash,
    uint256 id
) internal returns (bool) {
    if (id == s.nextExecutionIndex && s.approvedRequests[id] == hash) {
        s.nextExecutionIndex++;
        return true;
    }
    return false;
}
```

In the current implementation, requests must be executed in order and cannot be skipped. Otherwise, subsequent requests will be blocked.

https://github.com/flrfinance/flr-wraps-contracts/blob/3e26cc142f9ff23eb5370583c7fd9b66d4d4da97/src/WrapDepositRedeem.sol#L46-L53

```
function onExecute(
    address token,
    uint256 amount,
    address to
) internal virtual override returns (uint256 fee) {
    fee = calculateFee(amount, validatorFeeBPS);
    IERC20(token).safeTransfer(to, amount - fee);
}
```

However, `onExecute()` depends on the successful call to the external contract (`token`).

For example, if the recipient is blacklisted in USDC, the whole bridge will be blocked.

Also, one can intentionally bridge some USDC to a blacklisted address in order to initiate a DoS attack on the system.

Recommendation

Consider changing `IERC20(token).safeTransfer(to, amount - fee)` to require `to` to collect it themselves (from PUSH to PULL), or remove the sequential execution limitation, or allow the admin to bump `nextExecutionIndex` .

Status

✓ **Fixed**

[WP-L5] `Multisig` Both committees cannot have the `maxCommitteeSize` (128) of members at the same time

Low

Issue Description

Per the comments:

> a multisig with two committees ... Each committee cannot have more than 128 members

In `Multisig.sol: L81-83, L89` , `approvers` is a bitmask typed `uint256` , repersenting a bijection between bits and signer indices.

More specifically, approver.bitAt$(i) \leftrightarrow$ index$_i$, where `index` is defined at L97.

As `index` is declared as `uint8` , its value ranges from 0 to 255.

However, the index is actually asigned values from 1 to 256:

https://github.com/flrfinance/flr-wraps-contracts/blob/3e26cc142f9ff23eb5370583c7fd9b66d4d4da97/src/libraries/Multisig.sol#L199-L225

```
199 function addSigner(
200     DualMultisig storage s,
201     address signer,
202     bool isFirstCommittee
203 ) internal {
204     uint8 committeeSize = (
205         isFirstCommittee ? s.firstCommitteeSize : s.secondCommitteeSize
206     );
207     if (committeeSize == maxCommitteeSize) {
208         revert MaxCommitteeSizeReached();
209     }
210
211     SignerInfo storage signerInfo = s.signers[signer];
212     if (signerInfo.status != SignerStatus.Uninitialized) {
213         revert SignerAlreadyExists(signer);
214     }
215
```

```
    if (isFirstCommittee) {
        s.firstCommitteeSize++;
        signerInfo.status = SignerStatus.FirstCommittee;
    } else {
        s.secondCommitteeSize++;
        signerInfo.status = SignerStatus.SecondCommittee;
    }

    signerInfo.index = s.firstCommitteeSize + s.secondCommitteeSize;
}
```

That is, when one of the committees already have the `maxCommitteeSize` (128) of members, then the other committee can not have the `maxCommitteeSize` (128) number of members, as when adding the 128th member, `signerInfo.index` will overflow.

Recommendation

Consider changing to:

```
function addSigner(
    DualMultisig storage s,
    address signer,
    bool isFirstCommittee
) internal {
    uint8 committeeSize = (
        isFirstCommittee ? s.firstCommitteeSize : s.secondCommitteeSize
    );
    if (committeeSize == maxCommitteeSize) {
        revert MaxCommitteeSizeReached();
    }

    SignerInfo storage signerInfo = s.signers[signer];
    if (signerInfo.status != SignerStatus.Uninitialized) {
        revert SignerAlreadyExists(signer);
    }

    signerInfo.index = s.firstCommitteeSize + s.secondCommitteeSize;

    if (isFirstCommittee) {
        s.firstCommitteeSize++;
```

```
        signerInfo.status = SignerStatus.FirstCommittee;
    } else {
        s.secondCommitteeSize++;
        signerInfo.status = SignerStatus.SecondCommittee;
    }
}
```

So that the `signerInfo.index` starts with `0` instead `1` .

Status

ⓘ Acknowledged

[WP-L6] `configureValidatorFeeRecipient()` should be able to be called by the validator themselves

Low

Issue Description

https://github.com/flrfinance/flr-wraps-contracts/blob/3e26cc142f9ff23eb5370583c7fd9b66d4d4da97/src/Wrap.sol#L306-L311

```
function configureValidatorFeeRecipient(
    address validator,
    address feeRecipient
) external onlyRole(DEFAULT_ADMIN_ROLE) {
    validatorFeeRecipients[validator] = feeRecipient;
}
```

The `feeRecipient` should be configured by the validator themselves, rather than the admin.

Conventionally, if a specific recipient address is not set, it should fall back to the original validator address.

Recommendation

Consider changing to:

```
function configureValidatorFeeRecipient(
    address validator,
    address feeRecipient
) external {
    if (msg.sender != validator) revert Auth();
    validatorFeeRecipients[validator] = feeRecipient;
}
```

Also, consider disallowing setting the `feeRecipient` as `address(0)` to avoid potential misapplication.

Or, consider supporting fallingback to `validator` address when the `feeRecipient` is set as `address(0)` .

Status

ⓘ Acknowledged

[WP-I7] Committee seats cannot be recycled

Informational

Issue Description

The total number of both committees combined is 256:

https://github.com/flrfinance/flr-wraps-contracts/blob/3e26cc142f9ff23eb5370583c7fd9b66d4d4da97/src/libraries/Multisig.sol#L73-L90

```
    /// @dev Request info.
    /// @param approvalsFirstCommittee Number of approvals
    /// by the first committee.
    /// @param approvalsSecondCommittee Number of approvals
    /// by the second committee.
    /// @param status Status of the request.
    /// @param approvers Bitmask for signers from the two
    /// committees who have accepted the request.
    /// @notice Approvers is a bitmask. For example, a set bit at
    /// position 2 in the approvers bitmask indicates that the
    /// signer with index 2 has approved the request.
    struct Request {
        uint8 approvalsFirstCommittee; // slot 1 (0 - 7 bits)
        uint8 approvalsSecondCommittee; // slot 1 (8 - 15 bits)
        RequestStatus status; // slot 1 (16 - 23 bits)
        // slot 1 (23 - 255 spare bits)
        uint256 approvers; // slot 2
    }
```

Every new signer will take a new seat (`index`) instead of reusing a previously removed signer's seat (`removeSigner()`):

https://github.com/flrfinance/flr-wraps-contracts/blob/3e26cc142f9ff23eb5370583c7fd9b66d4d4da97/src/libraries/Multisig.sol#L199-L225

```
function addSigner(
    DualMultisig storage s,
    address signer,
    bool isFirstCommittee
```

```
) internal {
    uint8 committeeSize = (
        isFirstCommittee ? s.firstCommitteeSize : s.secondCommitteeSize
    );
    if (committeeSize == maxCommitteeSize) {
        revert MaxCommitteeSizeReached();
    }

    SignerInfo storage signerInfo = s.signers[signer];
    if (signerInfo.status != SignerStatus.Uninitialized) {
        revert SignerAlreadyExists(signer);
    }

    if (isFirstCommittee) {
        s.firstCommitteeSize++;
        signerInfo.status = SignerStatus.FirstCommittee;
    } else {
        s.secondCommitteeSize++;
        signerInfo.status = SignerStatus.SecondCommittee;
    }

    signerInfo.index = s.firstCommitteeSize + s.secondCommitteeSize;
}
```

Over a sufficient period of time, there will be more and more seats that become unusable, and the valid committee members will get fewer.

By the time that the total number of valid seats is lower than the quorum, the system will be forced to deprecate the current set of contracts and deploy new ones, or lower the quorum which will also lower the security level at the same time.

Status

ⓘ Acknowledged

[WP-I8] Consider providing a method to revoke the votes by the signer in `removeSigner()`

Informational

Issue Description

https://github.com/flrfinance/flr-wraps-contracts/blob/3e26cc142f9ff23eb5370583c7fd9b66d4d4da97/src/Wrap.sol#L298-L303

```
function removeValidator(
    address validator
) external onlyRole(DEFAULT_ADMIN_ROLE) {
    claimValidatorFees(validator);
    multisig.removeSigner(validator);
}
```

The current implementation of `removeValidator()` will trigger `claimValidatorFees()` and change the `status` of the `signer` to `Removed` .

However, the votes cast by the signer on the pending executions will remain valid.

This can be a problem if the number of signers got compromised (so that needed to be removed) is near the quorum.

Recommendation

Consider introducing a new method that allows the admin to also decrease the votes:

```
function removeValidator(
    address signer,
    bytes32[] memory hashs,
) external onlyRole(DEFAULT_ADMIN_ROLE) {
    claimValidatorFees(signer);
    SignerInfo memory signerInfo = s.signers[signer];
    // decode and loop hashs and remove the vote from each one
    for (uint256 i = 0; i < hashs.length; ++i) {
        Request storage request = s.requests[hash];
```

```
        if (request.status != RequestStatus.Undecided) continue;
        uint256 signerMask = 1 << signerInfo.index;
        // make sure the signer has already signed.
        if ((signerMask & request.approvers) != 0) {
            request.approvers &= ~signerMask;
            if (signerInfo.status == SignerStatus.FirstCommittee) {
                --request.approvalsFirstCommittee;
            } else {
                --request.approvalsSecondCommittee;
            }
        }
    }
    multisig.removeSigner(signer);
}
```

Status

ⓘ Acknowledged

[WP-G9] The total number of signers is known, therefore it is unnecessary to loop 256 times

Gas

Issue Description

https://github.com/flrfinance/flr-wraps-contracts/blob/89eca4b09a4a08857e96bd7f49b4941f31863386/src/libraries/Multisig.sol#L246-L255

```
function incrementPoints(DualMultisig storage s, uint256 mask) private {
    uint16 count = 0;
    for (uint16 i = 0; i < maxSignersSize; i++) {
        if ((mask & (1 << i)) != 0) {
            s.points[i]++;
            count++;
        }
    }
    s.totalPoints += count;
}
```

Recommendation

```
function incrementPoints(DualMultisig storage s, uint256 mask) private {
    uint16 count = 0;
    uint256 signersCount = s.firstCommitteeSize + s.secondCommitteeSize;
    for (uint16 i = 1; i <= signersCount; i++) {
        if ((mask & (1 << i)) != 0) {
            s.points[i]++;
            count++;
        }
    }
    s.totalPoints += count;
}
```

Status

✓ Fixed

[WP-N10] Fee-on-transfer tokens and rebasing tokens are not supported

Issue Description

It is worth noting that the system is not designed to support fee-on-transfer tokens and rebasing tokens.

The admin should not add these tokens.

Status

Acknowledged

Appendix

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by WatchPug; however, WatchPug does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Smart Contract technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.