// Private Smart Contract Security Assessment

05.09.2024 - 05.27.2024

FTSO Fast Updates protocol

Flare Network

HALBORN

FTSO Fast Updates protocol - Flare Network

Prepared by: HALBORN

Last Updated 06/21/2024

Date of Engagement by: May 9th, 2024 - May 27th, 2024

Summary

80% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
5	0	0	1	2	2

TABLE OF CONTENTS

1. Introduction

The `Flare Network` team engaged Halborn to conduct a security assessment on their smart contracts beginning on *05/09/2024* and ending on *05/27/2024*. The security assessment was scoped to the smart contracts provided in the GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

2. Assessment Summary

Halborn was provided 2 weeks for the engagement and assigned 1 full-time security engineer to review the security of the smart contracts in scope. The engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some minor security issues and recommendations, which some of them were addressed by the `Flare Network team`.

3. Test Approach And Methodology

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (`solgraph`).
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs. (`MythX`)
- Static Analysis of security for scoped contract, and imported functions (`slither`).
- Testnet deployment (`HardHat`).

Out-Of-Scope

- External libraries and financial-related attacks.
- New features/implementations after/with the **remediation commit IDs**.

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker’s control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: flare-smart-contracts-v2

(b) Assessed Commit ID: 70eaa29

(c) Items in scope:

- contracts/fastUpdates/implementation/CircularListManager.sol
- contracts/fastUpdates/implementation/FastUpdateIncentiveManager.sol
- contracts/fastUpdates/implementation/FastUpdater.sol
- contracts/fastUpdates/implementation/FastUpdatesConfiguration.sol
- contracts/fastUpdates/implementation/IncreaseManager.sol
- contracts/fastUpdates/lib/Bn256.sol
- contracts/fastUpdates/lib/FixedPointArithmetic.sol
- contracts/fastUpdates/lib/Sortition.sol

Out-of-Scope:

REMEDIATION COMMIT ID:

- b453651b453651

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	2	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
LACK OF SLASHING MECHANISM FOR MALICIOUS DATA PROVIDERS	MEDIUM	ACKNOWLEDGED

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
SEND ETHER WITH CALL INSTEAD OF TRANSFER	LOW	SOLVED - 06/19/2024
DATA PROVIDER COULD NOT SUBMIT AN UPDATE	LOW	ACKNOWLEDGED
USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS TO SAVE GAS	INFORMATIONAL	FUTURE RELEASE
LACK OF VALIDATION LEADS TO DIVISION BY ZERO	INFORMATIONAL	SOLVED - 06/17/2024

7. FINDINGS & TECH DETAILS

7.1 LACK OF SLASHING MECHANISM FOR MALICIOUS DATA PROVIDERS

// MEDIUM

Description

Slashing refers to the process of penalizing a data provider for misbehaving. If a data provider is found to be acting against the rules of the fast updates, it was not existing any penalty applied.
Moreover, it could be the possibility of sabotage attacks affecting to the honest data providers. The `submitUpdate` function allows the data providers to update feed prices according to the price's changes. It has been tested using as malicious accounts, submitting 0xffffffffffffffffffffffffffffffff deltas values as -1 (or `11111111` binary representations of 0xff) according to the delta's encoding documentation, reducing the prices feeds stored on the backlog impacting the "honest" accounts updates. Since the delta updates operations are increments/decrements from the stored feed prices, malicious actors could submit those delta updates altering the feed prices on the fast updates contract storage.

```
function submitUpdates(FastUpdates calldata _updates) external {
        require(
            block.number < _updates.sortitionBlock + submissionWindow,
            "Updates no longer accepted for the given block"
        );
        require(block.number >= _updates.sortitionBlock, "Updates not yet ava
        require((_updates.deltas.length * 4) <= feeds.length * 8, "More updat
        bytes32 msgHashed = sha256(abi.encode(_updates.sortitionBlock, _updat
        bytes32 signedMessageHash = MessageHashUtils.toEthSignedMessageHash(m
        Signature calldata signature = _updates.signature;
        address signingPolicyAddress = ECDSA.recover(signedMessageHash, signa
        require(signingPolicyAddress != address(0), "ECDSA: invalid signature

        (Bn256.G1Point memory key, uint256 weight) = _providerData(signingPol
        SortitionState memory sortitionState = SortitionState({
            baseSeed: flareSystemsManager.getSeed(flareSystemsManager.getCurr
            blockNumber: _updates.sortitionBlock,
            scoreCutoff: _currentScoreCutoff(),
            weight: weight,
            pubKey: key
        });

        SubmittedHashes storage submittedI = _getSubmitted(_updates.sortition
        bytes32 hashedRandomness =
            sha256(abi.encode(key, _updates.sortitionBlock, _updates.sortitio

        for (uint256 j = 0; j < submittedI.hashes.length; j++) {
            if (submittedI.hashes[j] == hashedRandomness) {
                revert("submission already provided");
            }
        }
        submittedI.hashes.push(hashedRandomness);

        (bool check, ) = verifySortitionCredential(sortitionState, _updates.s
        require(check, "sortition proof invalid");

        _submitDeltas(_updates.deltas);

        emit FastUpdateFeedsSubmitted(signingPolicyAddress);
    }
```

Proof of Concept

The following `Hardhat` test was used in order to prove the aforementioned issue:

```
 it("halborn sabotage", async () => {
    let submissionBlockNum;
    console.log("NUM_ACCOUNTS: ",NUM_ACCOUNTS);
    for (let i = 0; i < NUM_ACCOUNTS; i++) {
      const weight = await fastUpdater.currentSortitionWeight(voters[i]);
      weights[i] = weight.toNumber();
      console.log("WEIGHTS: ",weights[i]);
      expect(weights[i]).to.equal(Math.ceil(4096 / NUM_ACCOUNTS));
    }
    // Fetch current feeds from the contract
    const startingFeeds: number[] = (await fastUpdater.fetchCurrentFeeds(indices))[0].map((x:
BN) => x.toNumber());
    console.log("startingFeeds: ",startingFeeds);
    for (let i = 0; i < NUM_FEEDS; i++) {
      expect(startingFeeds[i]).to.equal(ANCHOR_FEEDS[i]);
    }
    // Make feed updates to the contract
    // test with feeds of various length
    let feed = "+--+00--".repeat(16);
    console.log("feed: ",feed);

    let deltas = "0x" + "7d0f".repeat(16);
    const differentFeed = "--------".repeat(8) + "----"; //attacker induced -1 to feed
prices.
    console.log("differentFeed: ",differentFeed);
    let differentDeltas = "ffff".repeat(8) + "ff";
    let differentDeltasLegit = "d005".repeat(8) + "d0";
```

```
  deltas += differentDeltasLegit;
  feed += differentFeed;
  console.log("deltas: ",differentDeltas);
  console.log("feed: ",differentFeed);

  differentDeltas = "0x" + differentDeltas;
  //console.log("differentDeltas with 0x: ",differentDeltas);

  let numSubmitted = 0;
  for (;;) {
   submissionBlockNum = (await web3.eth.getBlockNumber()).toString();
   const scoreCutoff = BigInt((await fastUpdater.currentScoreCutoff()).toString());
   const baseSeed = (await flareSystemMock.getSeed(await
flareSystemMock.getCurrentRewardEpochId())).toString();
   for (let i = 0; i < NUM_ACCOUNTS; i++) {
    submissionBlockNum = (await web3.eth.getBlockNumber()).toString();
    for (let rep = 0; rep < (weights[i] ?? 0); rep++) {
     const repStr = rep.toString();
     const proof: Proof = generateVerifiableRandomnessProof(
      sortitionKeys[i] as SortitionKey,
      baseSeed,
      submissionBlockNum,
      repStr
     );

     const sortitionCredential = {
      replicate: repStr,
      gamma: {
       x: proof.gamma.x.toString(),
       y: proof.gamma.y.toString(),
      },
      c: proof.c.toString(),
      s: proof.s.toString(),
     };

     //console.log("Voters before option to sumit update: ",voters[i]);

     if (proof.gamma.x < scoreCutoff) {
      let update = deltas;
      console.log("Loop: ",i);
      //console.log("Voters within option to sumit update: ",voters[i]);

      if ((voters[i] === voters[0] )) { //attacker 1.
       // use a different update with different length for this test
       update = differentDeltas;
       console.log("Attacker[%s]: %s",i,voters[i]);
```

```
        }

        if ((voters[i] === voters[1] )) { //attacker 2.
          // use a different update with different length for this test
          update = differentDeltas;
          console.log("Attacker[%s]: %s",i,voters[i]);
        }

        if ((voters[i] === voters[2] )) { //attacker 3.
          // use a different update with different length for this test
          update = differentDeltas;
          console.log("Attacker[%s]: %s",i,voters[i]);
        }
        if ((voters[i] === voters[3] )) { //attacker 4.
          // use a different update with different length for this test
          update = differentDeltas;
          console.log("Attacker[%s]: %s",i,voters[i]);
        }

        if ((voters[i] === voters[4] )) { //attacker 5.
          // use a different update with different length for this test
          update = differentDeltas;
          console.log("Attacker[%s]: %s",i,voters[i]);
        }

        if ((voters[i] === voters[5] )) { //attacker 6.
          // use a different update with different length for this test
          update = differentDeltas;
          console.log("Attacker[%s]: %s",i,voters[i]);
        }

        if ((voters[i] === voters[6] )) { //attacker 7.
          // use a different update with different length for this test
          update = differentDeltas;
          console.log("Attacker[%s]: %s",i,voters[i]);
        }

        if ((voters[i] === voters[7] )) { //attacker 8.
          // use a different update with different length for this test
          update = differentDeltas;
          console.log("Attacker[%s]: %s",i,voters[i]);
        }

        if ((voters[i] === voters[8] )) { //attacker 9.
          // use a different update with different length for this test
          update = differentDeltas;
```

```
              console.log("Attacker[%s]: %s",i,voters[i]);
            }

            if ((voters[i] === voters[9] )) { //attacker 10.
              // use a different update with different length for this test
              update = differentDeltas;
              console.log("Attacker[%s]: %s",i,voters[i]);
            }
            if ((voters[i] === voters[10] )) { //attacker 11.
              // use a different update with different length for this test
              update = differentDeltas;
              console.log("Attacker[%s]: %s",i,voters[i]);
            }

            if ((voters[i] === voters[11] )) { //attacker 12.
              // use a different update with different length for this test
              update = differentDeltas;
              console.log("Attacker[%s]: %s",i,voters[i]);
            }

            if ((voters[i] === voters[12] )) { //attacker 13.
              // use a different update with different length for this test
              update = differentDeltas;
              console.log("Attacker[%s]: %s",i,voters[i]);
            }

            if ((voters[i] === voters[13] )) { //attacker 14.
              // use a different update with different length for this test
              update = differentDeltas;
              console.log("Attacker[%s]: %s",i,voters[i]);
            }

            if ((voters[i] === voters[14] )) { //attacker 15.
              // use a different update with different length for this test
              update = differentDeltas;
              console.log("Attacker[%s]: %s",i,voters[i]);
            }

            if ((voters[i] === voters[15] )) { //attacker 16.
              // use a different update with different length for this test
              update = differentDeltas;
              console.log("Attacker[%s]: %s",i,voters[i]);
            }

            if ((voters[i] === voters[16] )) { //attacker 17.
              // use a different update with different length for this test
```

```
      update = differentDeltas;
      console.log("Attacker[%s]: %s",i,voters[i]);
    }
    if ((voters[i] === voters[17] )) { //attacker 18.
      // use a different update with different length for this test
      update = differentDeltas;
      console.log("Attacker[%s]: %s",i,voters[i]);
    }

    if ((voters[i] === voters[18] )) { //attacker 18.
      // use a different update with different length for this test
      update = differentDeltas;
      console.log("Attacker[%s]: %s",i,voters[i]);
    }

    if ((voters[i] === voters[19] )) { //attacker 20.
      // use a different update with different length for this test
      update = differentDeltas;
      console.log("Attacker[%s]: %s",i,voters[i]);
    }

    if ((voters[i] === voters[20] )) { //attacker 21.
      // use a different update with different length for this test
      update = differentDeltas;
      console.log("Attacker[%s]: %s",i,voters[i]);
    }

    if ((voters[i] === voters[21] )) { //attacker 22.
      // use a different update with different length for this test
      update = differentDeltas;
      console.log("Attacker[%s]: %s",i,voters[i]);
    }

    if ((voters[i] === voters[22] )) { //attacker 23.
      // use a different update with different length for this test
      update = differentDeltas;
      console.log("Attacker[%s]: %s",i,voters[i]);
    }

    if ((voters[i] === voters[23] )) { //attacker 24.
      // use a different update with different length for this test
      update = differentDeltas;
      console.log("Attacker[%s]: %s",i,voters[i]);
    }
    if ((voters[i] === voters[24] )) { //attacker 25.
      // use a different update with different length for this test
```

```
        update = differentDeltas;
        console.log("Attacker[%s]: %s",i,voters[i]);
      }

      if ((voters[i] === voters[25] )) { //attacker 26.
        // use a different update with different length for this test
        update = differentDeltas;
        console.log("Attacker[%s]: %s",i,voters[i]);
      }

      if ((voters[i] === voters[26] )) { //attacker 27.
        // use a different update with different length for this test
        update = differentDeltas;
        console.log("Attacker[%s]: %s",i,voters[i]);
      }

      if ((voters[i] === voters[27] )) { //attacker 28.
        // use a different update with different length for this test
        update = differentDeltas;
        console.log("Attacker[%s]: %s",i,voters[i]);
      }

      const msg = web3.eth.abi.encodeParameters(
        ["uint256", "uint256", "uint256", "uint256", "uint256", "uint256", "bytes"],
        [
          submissionBlockNum,
          repStr,
          proof.gamma.x.toString(),
          proof.gamma.y.toString(),
          proof.c.toString(),
          proof.s.toString(),
          update,
        ]
      );
      const signature = await ECDSASignature.signMessageHash(
        sha256(msg as BytesLike),
        privateKeys[i + 1].privateKey
      );

      const newFastUpdate = {
        sortitionBlock: submissionBlockNum,
        sortitionCredential: sortitionCredential,
        deltas: update,
        signature: signature,
      };
```

```
            console.log("BLOCK NUMBER: ",(await web3.eth.getBlockNumber()).toString());
            console.log("ACCOUNT: ",voters[i]);
            // Submit updates to the contract
            const tx = await fastUpdater.submitUpdates(newFastUpdate, {
              from: voters[i],
            });
            expect(tx.receipt.gasUsed).to.be.lessThan(300000);
            expectEvent(tx, "FastUpdateFeedsSubmitted", { signingPolicyAddress: voters[i] });
            let feeds: number[] = (await fastUpdater.fetchCurrentFeeds(indices))[0].map((x:
BN) => x.toNumber());
            console.log("feeds: ",feeds);

            numSubmitted++;
            console.log("NUM. SUBMITTED UPDATES: ",numSubmitted);
            if (numSubmitted >= 100) break;
          }
        }
        if (numSubmitted >= 100) break;
      }
      if (numSubmitted > 0) break;
    }

    let feeds: number[] = (await fastUpdater.fetchCurrentFeeds(indices))[0].map((x: BN) =>
x.toNumber());
    console.log("Last feeds: ",feeds);

    const tx = await fastUpdater.daemonize({
      from: flareDaemon,
    });
    expect(tx.receipt.gasUsed).to.be.lessThan(350000);

    feeds = (await fastUpdater.fetchCurrentFeeds(indices))[0].map((x: BN) => x.toNumber());
    console.log("feeds after daemon: ",feeds);

  });
```

Requirements:

- Supposing 60 accounts providers, 28 accounts will submit as delta prices variation: 0xffffffffffffffffffffffffffffffff. 32 account will submit as delta prices variation: 0x7d0f7d0f7d0f7d0f7d0f7d0f7d0f7d0f7d0f7d0f7d0f7d0f7d0f7d0f7d0fd005d005d005d005d005d005d005d0 as honest accounts.
- Starting price will be 5000.
- Same weight was used for every account.

Results:

1. First run using 28 accounts as malicious and 30 as honest account. According the sortition algorithm, the number of submissions could be differ between the samples tests:Number of submissions: 229 honest submission.13 malicious submissionResult: 4951
1. Second run using 28 accounts as malicious and 30 as honest account:
Number of submissions: 147 honest submission.7 malicious submissionResult: 4993
1. Third run using 60 honest accounts:Number of submissions: 14Result: 5126
1. Fourth run using 60 honest accounts:Number of submissions: 16Result: 5146

The feed price difference is ≈ 164. Any malicious actor could register and conduct this kind of unhonestly activities on the fast update contract.

BVSS

AO:A/AC:M/AX:M/C:N/I:C/A:N/D:N/Y:N/R:N/S:U (4.5)

Recommendation

Consider implementing a penalty mechanism to prevent data providers acting dishonestly.

Remediation Plan

ACKNOWLEDGED : The **Flare team** acknowledged this issue. The incorrect model of honest provider behavior does not concern them. Actually, some further analysis and live testing shows that the prices are quite resilient to this kind of attack. In addition, when they can identify misbehaving providers, the Flare architecture allows them to be chilled with a public vote. It has to happen manually, however.

Remediation Hash

b4536517808e8905d25f2eada955962d3ef518ef

7.2 SEND ETHER WITH CALL INSTEAD OF TRANSFER

// LOW

Description

Use call instead of transfer to send ether. And return value must be checked if sending ether is successful or not. Sending ether with the transfer is no longer recommended.

```
function offerIncentive(IncentiveOffer calldata _offer) external payable must
        (FPA.Fee dc, FPA.Range dr) = _processIncentiveOffer(_offer);
        FPA.SampleSize de = _sampleSizeIncrease(dc, dr);

        rewardManager.receiveRewards{value: FPA.Fee.unwrap(dc)} (rewardManage
        emit IncentiveOffered(dr, de, dc);
        payable(msg.sender).transfer(msg.value - FPA.Fee.unwrap(dc));
    }
```

Smart Contracts can't depend on gas costs

If gas costs are subject to change, then smart contracts can't depend on any particular gas costs. Any smart contract that uses transfer() or send() is taking a hard dependency on gas costs by forwarding a fixed amount of gas: 2300. Following consideration are described of not use transfer() function:

1. **Gas Stipend**: `.transfer()` automatically sends a gas stipend of 2300 gas along with the Ether, which is intended to be sufficient for logging events but not for executing more complex operations in the receiver contract. This was deemed safe before the EIP-1884 Ethereum upgrade, which changed the cost of certain EVM opcodes and thereby made 2300 gas insufficient for some operations like updating state variables.
2. **Error Handling**: `.transfer()` reverts the transaction if the call fails for any reason (including out-of-gas errors). This can lead to undesirable effects where a failure in one part of a transaction unwinds all other changes made by that transaction, even if those changes were valid and safe.
3. **Flexibility and Control**: Developers often need more control over the transaction, especially concerning the amount of gas forwarded along with the Ether. `.transfer()` does not allow specifying a gas amount, limiting its utility in scenarios where more complex operations need to occur in the receiver's fallback function.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:M/I:L/D:L/Y:N (4.1)

Recommendation

Ensure to use call instead of transfer. Be aware to implement checks effects interactions patterns to avoid potential reentrancy attacks.

```
(bool result, ) = payable(msg.sender).call{value: _amount}("");
require(result, "Failed to send Ether");
```

Remediation Plan

SOLVED : The **Flare team** solved this issue.

Remediation Hash

b4536517808e8905d25f2eada955962d3ef518ef

7.3 DATA PROVIDER COULD NOT SUBMIT AN UPDATE

// LOW

Description

It has been observed that when the submission window is set to 1, providers are unable to submit at least one feed update for prices. This issue arises due to the checks implemented in the `submitUpdates` function. During the transaction, the value of `_updates.sortitionBlock`is one less than the block number, which prevents users from updating the feed prices and results in the transaction being reverted with the error message: "Updates no longer accepted for the given block."

```
function submitUpdates(FastUpdates calldata _updates) external {
        require(
            block.number < _updates.sortitionBlock + submissionWindow,
            "Updates no longer accepted for the given block"
        );
        require(block.number >= _updates.sortitionBlock, "Updates not yet ava
        require((_updates.deltas.length * 4) <= feeds.length * 8, "More updat
        bytes32 msgHashed = sha256(abi.encode(_updates.sortitionBlock, _updat
        bytes32 signedMessageHash = MessageHashUtils.toEthSignedMessageHash(m
        Signature calldata signature = _updates.signature;
        address signingPolicyAddress = ECDSA.recover(signedMessageHash, signa
        require(signingPolicyAddress != address(0), "ECDSA: invalid signature

        (Bn256.G1Point memory key, uint256 weight) = _providerData(signingPol
        SortitionState memory sortitionState = SortitionState({
            baseSeed: flareSystemsManager.getSeed(flareSystemsManager.getCurr
            blockNumber: _updates.sortitionBlock,
            scoreCutoff: _currentScoreCutoff(),
            weight: weight,
            pubKey: key
        });

        SubmittedHashes storage submittedI = _getSubmitted(_updates.sortition
        bytes32 hashedRandomness =
            sha256(abi.encode(key, _updates.sortitionBlock, _updates.sortitio

        for (uint256 j = 0; j < submittedI.hashes.length; j++) {
            if (submittedI.hashes[j] == hashedRandomness) {
                revert("submission already provided");
            }
        }
        submittedI.hashes.push(hashedRandomness);

        (bool check, ) = verifySortitionCredential(sortitionState, _updates.s
        require(check, "sortition proof invalid");

        _submitDeltas(_updates.deltas);

        emit FastUpdateFeedsSubmitted(signingPolicyAddress);
    }
```

Since the submission windows are set it by the constructor, but there was a function to set again the submission window by the governor, the issue has been downgraded.

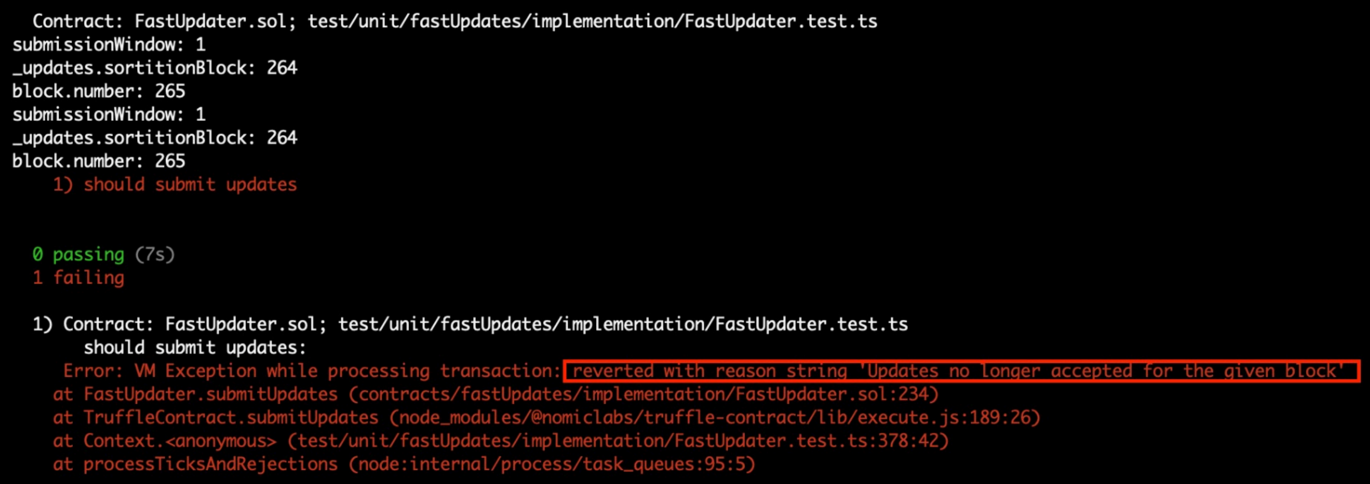
Proof of Concept

In the following test is demonstrated that if the submission window is equal to one any user could not update any feed prices.
Proof of Concept:
1. The provider got the proof and created the signature to generate the update calldata passed into the submitUpdate function on the offchain component. Since the sortition block is used for creating the proof it is needed the block number.
2. Once the transaction is performed for calling the submitUpdates the block.number is higher than the sortition block number, avoiding to update the feed prices for the provider.

```
const SUBMISSION_WINDOW = 1 as const; // only one submission window.
// Create local instance of Fast Updater contract
    fastUpdater = await FastUpdater.new(
      accounts[0],
      governance,
      addressUpdater,
      flareDaemon,
      await time.latest(),
      90,
      SUBMISSION_WINDOW
    );

it("should submit updates", async () => {
    let submissionBlockNum;
    //console.log("NUM_ACCOUNTS: ",NUM_ACCOUNTS);
    for (let i = 0; i < NUM_ACCOUNTS; i++) {
      //console.log("VOTERS: ",voters[i]);
      const weight = await fastUpdater.currentSortitionWeight(voters[i]);
      weights[i] = weight.toNumber();
      //console.log("WEIGHTS: ",weights[i]);
      expect(weights[i]).to.equal(Math.ceil(4096 / NUM_ACCOUNTS));
    }

    // Fetch current feeds from the contract
    //console.log("indices: ",indices);
    const startingFeeds: number[] = (await fastUpdater.fetchCurrentFeeds(indices))[0].map((x:
```

```
BN) => x.toNumber());
    //console.log("startingFeeds: ",startingFeeds);
    for (let i = 0; i < NUM_FEEDS; i++) {
      expect(startingFeeds[i]).to.equal(ANCHOR_FEEDS[i]);
    }

    // Make feed updates to the contract
    // test with feeds of various length
    let feed = "+--+00--".repeat(16);
    //console.log("feed: ",feed);

    let deltas = "0x" + "7d0f".repeat(16);
    //console.log("deltas: ",deltas);

    const differentFeed = "-+0000++".repeat(8) + "-+00";
    //console.log("differentFeed: ",differentFeed);

    let differentDeltas = "d005".repeat(8) + "d0";
    //console.log("differentDeltas: ",differentDeltas);

    deltas += differentDeltas;
    feed += differentFeed;
    //console.log("deltas: ",differentDeltas);
    //console.log("feed: ",differentFeed);

    differentDeltas = "0x" + differentDeltas;
    //console.log("differentDeltas with 0x: ",differentDeltas);

    let numSubmitted = 0;
    for (;;) {
      submissionBlockNum = (await web3.eth.getBlockNumber()).toString();
      //console.log("submissionBlockNum: ",submissionBlockNum);
      const scoreCutoff = BigInt((await fastUpdater.currentScoreCutoff()).toString());
      //console.log("scoreCutoff: ",scoreCutoff);
      const baseSeed = (await flareSystemMock.getSeed(await
flareSystemMock.getCurrentRewardEpochId())).toString();
      //console.log("baseSeed: ",baseSeed);
      for (let i = 0; i < NUM_ACCOUNTS; i++) {
        submissionBlockNum = (await web3.eth.getBlockNumber()).toString();
        //console.log("submissionBlockNum: ",submissionBlockNum);
        for (let rep = 0; rep < (weights[i] ?? 0); rep++) {
          //submissionBlockNum = (await web3.eth.getBlockNumber()).toString();
          const repStr = rep.toString();
          const proof: Proof = generateVerifiableRandomnessProof(
            sortitionKeys[i] as SortitionKey,
            baseSeed,
            submissionBlockNum,
            repStr
          );

          const sortitionCredential = {
            replicate: repStr,
            gamma: {
              x: proof.gamma.x.toString(),
              y: proof.gamma.y.toString(),
            },
            c: proof.c.toString(),
            s: proof.s.toString(),
          };

          if (proof.gamma.x < scoreCutoff) {
            let update = deltas;
            //console.log("DELTAS: ",update);
            if (numSubmitted == 1) {
              // use a different update with different length for this test
              update = differentDeltas;
              //console.log("DELTAS if numSubmitted == 1: ",update);
            }

            const msg = web3.eth.abi.encodeParameters(
              ["uint256", "uint256", "uint256", "uint256", "uint256", "uint256", "bytes"],
              [
                submissionBlockNum,
                repStr,
                proof.gamma.x.toString(),
                proof.gamma.y.toString(),
                proof.c.toString(),
                proof.s.toString(),
                update,
              ]
            );
            //console.log("msg: ",msg);
            const signature = await ECDSASignature.signMessageHash(
              sha256(msg as BytesLike),
              privateKeys[i + 1].privateKey
            );
            //console.log("signature: ",signature);

            const newFastUpdate = {
              sortitionBlock: submissionBlockNum,
              sortitionCredential: sortitionCredential,
              deltas: update,
              signature: signature,
            };

            //console.log("newFastUpdate: ",newFastUpdate);
            // Submit updates to the contract
            const tx = await fastUpdater.submitUpdates(newFastUpdate, {
              from: accounts[0],
            });
            expect(tx.receipt.gasUsed).to.be.lessThan(300000);
            expectEvent(tx, "FastUpdateFeedsSubmitted", { signingPolicyAddress: voters[i] });



            let caughtError = false;
            try {
              // test if submitting again gives error
              await fastUpdater.submitUpdates(newFastUpdate, {
                from: voters[i],
              });
            } catch (e) {
              expect(e).to.be.not.empty;
              caughtError = true;
            }
            expect(caughtError).to.equal(true);

            numSubmitted++;
            if (numSubmitted >= 2) break;
          }
        }
        if (numSubmitted >= 20) break;
      }
      if (numSubmitted > 0) break;
    }

  });
```

Evidence:



```
  Contract: FastUpdater.sol; test/unit/fastUpdates/implementation/FastUpdater.test.ts
submissionWindow: 1
_updates.sortitionBlock: 264
block.number: 265
submissionWindow: 1
_updates.sortitionBlock: 264
block.number: 265
    1) should submit updates


  0 passing (7s)
  1 failing

  1) Contract: FastUpdater.sol; test/unit/fastUpdates/implementation/FastUpdater.test.ts
       should submit updates:
     Error: VM Exception while processing transaction: reverted with reason string 'Updates no longer accepted for the given block'
    at FastUpdater.submitUpdates (contracts/fastUpdates/implementation/FastUpdater.sol:234)
    at TruffleContract.submitUpdates (node_modules/@nomiclabs/truffle-contract/lib/execute.js:189:26)
    at Context.<anonymous> (test/unit/fastUpdates/implementation/FastUpdater.test.ts:378:42)
    at processTicksAndRejections (node:internal/process/task_queues:95:5)
```

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:C/D:N/Y:N/R:F/S:U (2.5)

Recommendation

Consider checking that the submission windows is a number higher than 1.

Remediation Plan

ACKNOWLEDGED : The **Flare team** acknowledged this issue. It's an "unreachable state" because the submission window is controlled only by governance and is explicitly intended to be set large enough to allow submissions to reach the chain in time. They won't be changing anything here.

Remediation Hash

b4536517808e8905d25f2eada955962d3ef518ef

7.4 USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS TO SAVE GAS

// INFORMATIONAL

Description

Custom errors from Solidity 0.8.4 are cheaper than revert strings (cheaper deployment cost and runtime cost when the revert condition is met). Custom errors are defined using the `error` statement, which can be used inside and outside of contracts (including interfaces and libraries).

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

Ensure to use custom errors in order to save gas.

Remediation Plan

PENDING: The **Flare team** will solve this issue in a future release

Remediation Hash

b4536517808e8905d25f2eada955962d3ef518ef

7.5 LACK OF VALIDATION LEADS TO DIVISION BY ZERO

// INFORMATIONAL

Description

It has been observed a potential division by zero issue.

```
function _processIncentiveOffer(
        IncentiveOffer calldata _offer
    )
        internal
        returns (FPA.Fee _contribution, FPA.Range _rangeIncrease)
    {
        require(msg.value >> 120 == 0, "Incentive offer value capped at 120 bits");
        _contribution = FPA.Fee.wrap(uint240(msg.value));
        _rangeIncrease = _offer.rangeIncrease;

        FPA.Range finalRange = FPA.add(range, _rangeIncrease);
        if (FPA.lessThan(_offer.rangeLimit, finalRange)) {
            finalRange = _offer.rangeLimit;
            FPA.Range newRangeIncrease = FPA.lessThan(finalRange, range) ? FPA.zeroR :
FPA.sub(finalRange, range);
            _contribution = FPA.mul(FPA.frac(newRangeIncrease, _rangeIncrease),
_contribution);
            _rangeIncrease = newRangeIncrease;
        }
        require(FPA.lessThan(finalRange, sampleSize), "Offer would make the precision greater
than 100%");
    }
```

This line attempts to scale the `_contribution` by the fraction of the new range increase (`newRangeIncrease`) over the originally proposed range increase (`_rangeIncrease`). The function `FPA.frac(newRangeIncrease, _rangeIncrease)` computes this fraction. However, if `_rangeIncrease` is zero, this calculation will attempt to divide by zero, which is not allowed in Solidity and will cause the transaction to revert.

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

To fix this issue, it should be added a check to ensure that `_rangeIncrease` is not zero before executing the division. Here's a modification that includes such a safeguard:

```
if (_rangeIncrease != FPA.zeroR) {
    FPA.Range finalRange = FPA.add(range, _rangeIncrease);
    if (FPA.lessThan(_offer.rangeLimit, finalRange)) {
        finalRange = _offer.rangeLimit;
        FPA.Range newRangeIncrease = FPA.lessThan(finalRange, range) ? FPA.zeroR :
FPA.sub(finalRange, range);
        _contribution = FPA.mul(FPA.frac(newRangeIncrease, _rangeIncrease), _contribution);
        _rangeIncrease = newRangeIncrease;
    }
} else {
    // Handle the case where _rangeIncrease is zero
    // You might revert the transaction, set a default value, or handle it in another
appropriate way
    revert("Range increase cannot be zero.");
}
require(FPA.lessThan(finalRange, sampleSize), "Offer would make the precision greater than
100%");
```

Remediation Plan

SOLVED : The **Flare team** solved this issue.

Remediation Hash

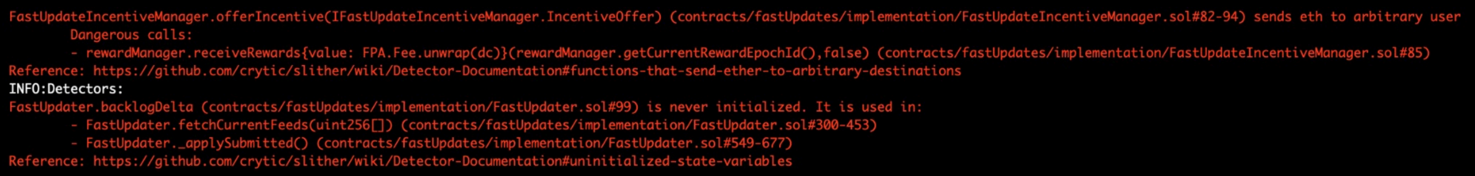
b4536517808e8905d25f2eada955962d3ef518ef

STATIC ANALYSIS REPORT

Description

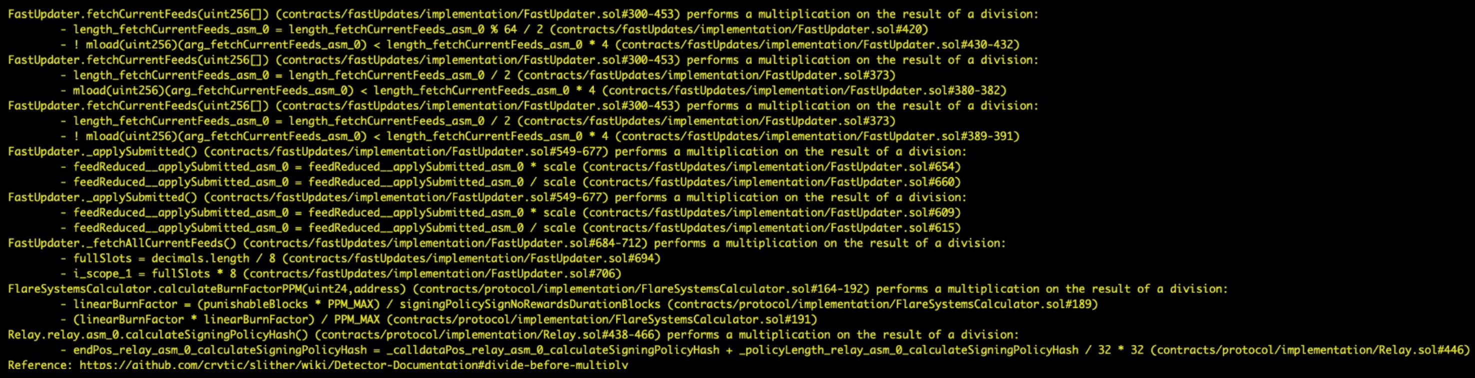
Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their abis and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.
All issues identified by Slither were proved to be false positives or have been added to the issue list in this report.

High



```
FastUpdateIncentiveManager.offerIncentive(IFastUpdateIncentiveManager.IncentiveOffer) (contracts/fastUpdates/implementation/FastUpdateIncentiveManager.sol#82-94) sends eth to arbitrary user
        Dangerous calls:
        - rewardManager.receiveRewards{value: FPA.Fee.unwrap(dc)}(rewardManager.getCurrentRewardEpochId(),false) (contracts/fastUpdates/implementation/FastUpdateIncentiveManager.sol#85)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
FastUpdater.backlogDelta (contracts/fastUpdates/implementation/FastUpdater.sol#99) is never initialized. It is used in:
        - FastUpdater.fetchCurrentFeeds(uint256[]) (contracts/fastUpdates/implementation/FastUpdater.sol#300-453)
        - FastUpdater._applySubmitted() (contracts/fastUpdates/implementation/FastUpdater.sol#549-677)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables
```

Medium



```
FastUpdater.fetchCurrentFeeds(uint256[]) (contracts/fastUpdates/implementation/FastUpdater.sol#300-453) performs a multiplication on the result of a division:
        - length_fetchCurrentFeeds_asm_0 = length_fetchCurrentFeeds_asm_0 % 64 / 2 (contracts/fastUpdates/implementation/FastUpdater.sol#420)
        - ! mload(uint256)(arg_fetchCurrentFeeds_asm_0) < length_fetchCurrentFeeds_asm_0 * 4 (contracts/fastUpdates/implementation/FastUpdater.sol#430-432)
FastUpdater.fetchCurrentFeeds(uint256[]) (contracts/fastUpdates/implementation/FastUpdater.sol#300-453) performs a multiplication on the result of a division:
        - length_fetchCurrentFeeds_asm_0 = length_fetchCurrentFeeds_asm_0 / 2 (contracts/fastUpdates/implementation/FastUpdater.sol#373)
        - mload(uint256)(arg_fetchCurrentFeeds_asm_0) < length_fetchCurrentFeeds_asm_0 * 4 (contracts/fastUpdates/implementation/FastUpdater.sol#380-382)
FastUpdater.fetchCurrentFeeds(uint256[]) (contracts/fastUpdates/implementation/FastUpdater.sol#300-453) performs a multiplication on the result of a division:
        - length_fetchCurrentFeeds_asm_0 = length_fetchCurrentFeeds_asm_0 / 2 (contracts/fastUpdates/implementation/FastUpdater.sol#373)
        - ! mload(uint256)(arg_fetchCurrentFeeds_asm_0) < length_fetchCurrentFeeds_asm_0 * 4 (contracts/fastUpdates/implementation/FastUpdater.sol#389-391)
FastUpdater._applySubmitted() (contracts/fastUpdates/implementation/FastUpdater.sol#549-677) performs a multiplication on the result of a division:
        - feedReduced__applySubmitted_asm_0 = feedReduced__applySubmitted_asm_0 * scale (contracts/fastUpdates/implementation/FastUpdater.sol#654)
        - feedReduced__applySubmitted_asm_0 = feedReduced__applySubmitted_asm_0 / scale (contracts/fastUpdates/implementation/FastUpdater.sol#660)
FastUpdater._applySubmitted() (contracts/fastUpdates/implementation/FastUpdater.sol#549-677) performs a multiplication on the result of a division:
        - feedReduced__applySubmitted_asm_0 = feedReduced__applySubmitted_asm_0 * scale (contracts/fastUpdates/implementation/FastUpdater.sol#609)
        - feedReduced__applySubmitted_asm_0 = feedReduced__applySubmitted_asm_0 / scale (contracts/fastUpdates/implementation/FastUpdater.sol#615)
FastUpdater._fetchAllCurrentFeeds() (contracts/fastUpdates/implementation/FastUpdater.sol#684-712) performs a multiplication on the result of a division:
        - fullSlots = decimals.length / 8 (contracts/fastUpdates/implementation/FastUpdater.sol#694)
        - i_scope_1 = fullSlots * 8 (contracts/fastUpdates/implementation/FastUpdater.sol#706)
FlareSystemsCalculator.calculateBurnFactorPPM(uint24,address) (contracts/protocol/implementation/FlareSystemsCalculator.sol#164-192) performs a multiplication on the result of a division:
        - linearBurnFactor = (punishableBlocks * PPM_MAX) / signingPolicySignNoRewardsDurationBlocks (contracts/protocol/implementation/FlareSystemsCalculator.sol#189)
        - (linearBurnFactor * linearBurnFactor) / PPM_MAX (contracts/protocol/implementation/FlareSystemsCalculator.sol#191)
Relay.relay.asm_0.calculateSigningPolicyHash() (contracts/protocol/implementation/Relay.sol#438-466) performs a multiplication on the result of a division:
        - endPos_relay_asm_0_calculateSigningPolicyHash = _calldataPos_relay_asm_0_calculateSigningPolicyHash + _policyLength_relay_asm_0_calculateSigningPolicyHash / 32 * 32 (contracts/protocol/implementation/Relay.sol#446)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
```

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project’s integrity and addressing potential vulnerabilities introduced by code modifications.