hexens × FUEL

SEPT.24

SECURITY REVIEW REPORT FOR FUEL

CONTENTS

- About Hexens
- Executive summary
    - Overview
    - Scope
- Auditing details
- Severity structure
    - Severity characteristics
    - Issue symbolic codes
- Findings summary
- Weaknesses
    - Large withdrawals can be stuck because of the rate limits
    - Withdrawals from Fuel can be DoSed with the help of rate limits
    - FuelMessagePortalV3::resetRateLimitAmount() reduces currentPeriodAmount which may lead to unexpected behavior
    - Missing Zero-Value Validation for Critical Parameters in Constructor
    - It is not possible to disable the rate limits
    - The rate limit can be set retroactively in FuelERC20GatewayV4.resetRateLimitAmount()
    - Wrong comment in FuelERC20GatewayV4.sol
    - It will be possible to withdraw after the upgrade to FuelMessagePortalV3
    - proxy.sw is not following recommendations of SRC14 for the fn proxy_owner()

ABOUT HEXENS

Hexens is a cybersecurity company that strives to elevate the standards of security in Web 3.0, create a safer environment for users, and ensure mass Web 3.0 adoption.

Hexens has multiple top-notch auditing teams specialized in different fields of information security, showing extreme performance in the most challenging and technically complex tasks, including but not limited to: Infrastructure Audits, Zero Knowledge Proofs / Novel Cryptography, DeFi and NFTs. Hexens not only uses widely known methodologies and flows, but focuses on discovering and introducing new ones on a day-to-day basis.

In 2022, our team announced the closure of a $4.2 million seed round led by IOSG Ventures, the leading Web 3.0 venture capital. Other investors include Delta Blockchain Fund, Chapter One, Hash Capital, ImToken Ventures, Tenzor Capital, and angels from Polygon and other blockchain projects.

Since Hexens was founded in 2021, it has had an impressive track record and recognition in the industry: Mudit Gupta - CISO of Polygon Technology - the biggest EVM Ecosystem, joined the company advisory board after completing just a single cooperation iteration. Polygon Technology, 1inch, Lido, Hats Finance, Quickswap, Layerswap, 4K, RociFi, as well as dozens of DeFi protocols and bridges, have already become our customers and taken proactive measures towards protecting their assets.

EXECUTIVE SUMMARY

OVERVIEW

This audit covered various small updates to the bridge contracts of Fuel Network. For example, the updates introduced rate limiting.

Our security assessment was a full review of the updates made, spanning a total of 2 weeks.

During our audit, we have identified 2 high severity vulnerabilities that could have lead to stuck funds or DoS of withdrawals.

We have also identified several minor severity vulnerabilities and code optimisations.

Finally, all of our reported issues were fixed or acknowledged by the development team and consequently validated by us.

We can confidently say that the overall security and code quality have increased after completion of our audit.

SCOPE

The analyzed resources are located on:

https://github.com/FuelLabs/fuel-bridge/commit/10d0b44c4bd3e5916f20a92e7987cd5e44e3a39f

https://github.com/FuelLabs/fuel-bridge/commit/27bfdf221c8bb14d1a1d2f19786114980ef9b749

https://github.com/FuelLabs/fuel-bridge/commit/c234e48c0f02d02aac8c1f21fac31f80fde5a2b1

https://github.com/FuelLabs/fuel-bridge/commit/4e2eaa59acdece79e466ea2d8022fab9cabf1db6

https://github.com/FuelLabs/fuel-bridge/commit/c0c6390bb1949d403e38bc283f45368f63597690

https://github.com/FuelLabs/fuel-bridge/commit/127dcaf94d3b8e40923b94cc219142c420024604

https://github.com/FuelLabs/fuel-bridge/commit/849f928ffc82cc3f4e3b17fd9891d88d6daf3702

https://github.com/FuelLabs/fuel-bridge/commit/9ac659fd762f4c01f0a949f7f1c9f2fdf201eaac

https://github.com/FuelLabs/fuel-bridge/commit/394ba7d87b620a94d417c68e4a62174eef23c9b4

https://github.com/FuelLabs/fuel-bridge/commit/c1ba7a9cdc2dda3442274c5be6fc833ad5bdc2b0

https://github.com/FuelLabs/fuel-bridge/commit/d0f7dd58feb97c6db9a09273c0071154fb81081a

https://github.com/FuelLabs/fuel-bridge/commit/e3e673e31f9e72d757d68979bb6796a0b7f9c8bc

https://github.com/FuelLabs/fuel-bridge/commit/0b701715f2ce414ecd7486e18410126908bb2ce0

https://github.com/FuelLabs/fuel-bridge/commit/163e13cec64dd3043b1ed96e0683c13448ae3af3

https://github.com/FuelLabs/fuel-bridge/commit/2d972ae38791bec1024bfd6e1245c0a9019e7625

The issues described in this report were fixed. Corresponding commits are mentioned in the description.

AUDITING DETAILS

STARTED
04.09.2024

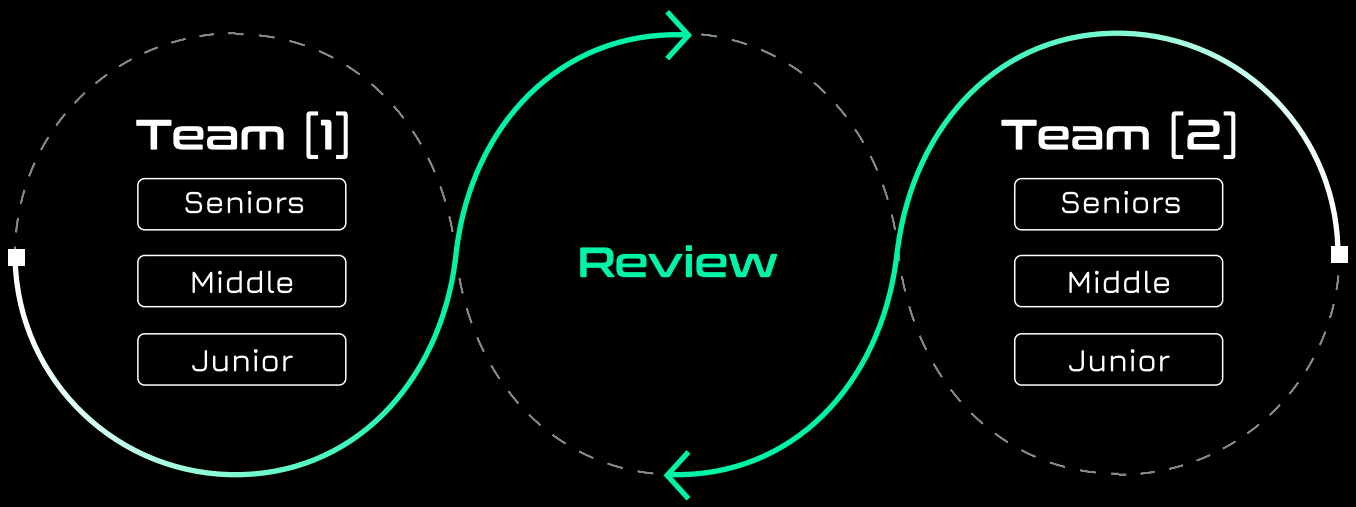
DELIVERED
17.09.2024

Review Led by

NOUREDDINE BENOMARI
Security Researcher | Hexens

HEXENS METHODOLOGY

Hexens methodology involves 2 teams, including multiple auditors of different seniority, with at least 5 security engineers. This unique cross-checking mechanism helps us provide the best quality in the market.



SEVERITY STRUCTURE

The vulnerability severity is calculated based on two components

- Impact of the vulnerability
- Probability of the vulnerability

Impact	Probability			
	rare	unlikely	likely	very likely
Low/Info	Low/Info	Low/Info	Medium	Medium
Medium	Low/Info	Medium	Medium	High
High	Medium	Medium	High	Critical
Critical	Medium	High	Critical	Critical

SEVERITY CHARACTERISTICS

Smart contract vulnerabilities can range in severity and impact, and it's important to understand their level of severity in order to prioritize their resolution. Here are the different types of severity levels of smart contract vulnerabilities:

Critical

Vulnerabilities with this level of severity can result in significant financial losses or reputational damage. They often allow an attacker to gain complete control of a contract, directly steal or freeze funds from the contract or users, or permanently block the functionality of a protocol. Examples include infinite mints and governance manipulation.

High

Vulnerabilities with this level of severity can result in some financial losses or reputational damage. They often allow an attacker to directly steal yield from the contract or users, or temporarily freeze funds. Examples include inadequate access control integer overflow/underflow, or logic bugs.

Medium

Vulnerabilities with this level of severity can result in some damage to the protocol or users, without profit for the attacker. They often allow an attacker to exploit a contract to cause harm, but the impact may be limited, such as temporarily blocking the functionality of the protocol. Examples include uninitialized storage pointers and failure to check external calls.

Low

Vulnerabilities with this level of severity may not result in financial losses or significant harm. They may, however, impact the usability or reliability of a contract. Examples include slippage and front-running, or minor logic bugs.

Informational

Vulnerabilities with this level of severity are regarding gas optimizations and code style. They often involve issues with documentation, incorrect usage of EIP standards, best practices for saving gas, or the overall design of a contract. Examples include not conforming to ERC20, or disagreement between documentation and code.

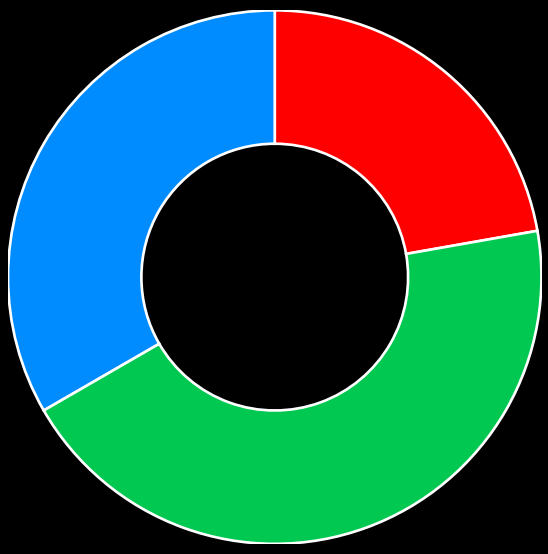
ISSUE SYMBOLIC CODES

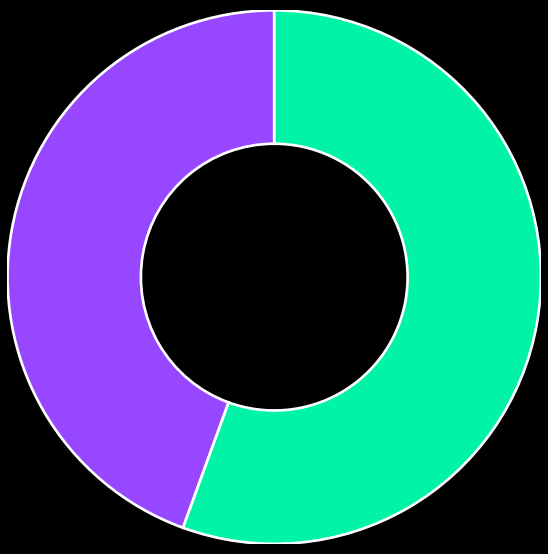
Every issue being identified and validated has its unique symbolic code assigned to the issue at the security research stage. Cause of the vulnerability reporting flow design, some of the rejected issues could be missing.

FINDINGS SUMMARY

Severity	Number of Findings
Critical	0
High	2
Medium	0
Low	4
Informational	3

Total: 9





WEAKNESSES

This section contains the list of discovered weaknesses.

FUEL7-4

LARGE WITHDRAWALS CAN BE STUCK BECAUSE OF THE RATE LIMITS

SEVERITY: High

PATH:

contracts/messaging/gateway/FuelERC20Gateway/FuelERC20GatewayV4.sol#L414-L433

contracts/fuelchain/FuelMessagePortal/v3/FuelMessagePortalV3.sol#L230-L249

REMEDIATION:

We recommend:

- addition of the rate limits logic on the Fuel side as well
- or addition of an automated refund system in case the withdrawal is too large
- or addition of a queue system and partial withdrawals (this will be complicated in the case of ETH because of the callback)

STATUS: Acknowledged, see commentary

DESCRIPTION:

The introduced rate limits in **FuelERC20GatewayV4** and **FuelMessagePortalV3** can get a withdrawal stuck if the amount is larger than the rate limit allowance per period.

E.g., if the rate limit is **100^18** per 1 hour, and there is a withdrawal of **100^18 + 1** tokens, the call to **finalizeWithdrawal()** will always fail. The withdrawal will require manual adjustment of the rate limit for each individual case, which

can lead to other participants using more than the desired limit and also operation losses for the victims because of any execution delays and the team because of the gas fees.

```
    function _addWithdrawnAmount(address _token, uint256 _withdrawnAmount)
internal {
        uint256 currentPeriodAmountTemp;

        if (currentPeriodEnd[_token] < block.timestamp) {
            unchecked {
                currentPeriodEnd[_token] = block.timestamp +
rateLimitDuration[_token];
            }
            currentPeriodAmountTemp = _withdrawnAmount;
        } else {
            unchecked {
                currentPeriodAmountTemp = currentPeriodAmount[_token] +
_withdrawnAmount;
            }
        }

        if (currentPeriodAmountTemp > limitAmount[_token]) {
            revert RateLimitExceeded();
        }

        currentPeriodAmount[_token] = currentPeriodAmountTemp;
    }
```

```
    function _addWithdrawnAmount(uint256 _withdrawnAmount) internal {
        uint256 currentPeriodAmountTemp;

        if (currentPeriodEnd < block.timestamp) {
            unchecked {
                currentPeriodEnd = block.timestamp + rateLimitDuration;
            }
            currentPeriodAmountTemp = _withdrawnAmount;
        } else {
            unchecked {
                currentPeriodAmountTemp = currentPeriodAmount +
_withdrawnAmount;
            }
        }

        if (currentPeriodAmountTemp > limitAmount) {
            revert RateLimitExceeded();
        }

        currentPeriodAmount = currentPeriodAmountTemp;
    }
```

Commentary from the client:

“ - We are considering a fix but we have not yet clear path way on implementing them, our bandwidth is limited with the mainnet launch. But we will definitely consider a fix long term.

- We probably are going to put a withdraw limiter on the L2 contracts.”

WITHDRAWALS FROM FUEL CAN BE DOSED WITH THE HELP OF RATE LIMITS

SEVERITY: High

PATH:

contracts/messaging/gateway/FuelERC20Gateway/FuelERC20GatewayV4.sol#L414-L433

contracts/fuelchain/FuelMessagePortal/v3/FuelMessagePortalV3.sol#L230-L249

STATUS: Acknowledged, see commentary

DESCRIPTION:

The rate limits introduced in FuelERC20GatewayV4 and FuelMessagePortalV3 can be used by malicious actors to DoS the ETH and ERC20 (if enabled) withdrawals.

The attackers simply need to withdraw the amount equal to a limit and deposit it back at the start of every rate limit period.

The attack is relatively cheap to perform, as the costs would only be the bridging fees to and from Fuel (a couple of dollars per period).

```
    function _addWithdrawnAmount(address _token, uint256 _withdrawnAmount)
internal {
        uint256 currentPeriodAmountTemp;

        if (currentPeriodEnd[_token] < block.timestamp) {
            unchecked {
                currentPeriodEnd[_token] = block.timestamp +
rateLimitDuration[_token];
            }
            currentPeriodAmountTemp = _withdrawnAmount;
        } else {
            unchecked {
                currentPeriodAmountTemp = currentPeriodAmount[_token] +
_withdrawnAmount;
            }
        }

        if (currentPeriodAmountTemp > limitAmount[_token]) {
            revert RateLimitExceeded();
        }

        currentPeriodAmount[_token] = currentPeriodAmountTemp;
    }
```

```
    function _addWithdrawnAmount(uint256 _withdrawnAmount) internal {
        uint256 currentPeriodAmountTemp;

        if (currentPeriodEnd < block.timestamp) {
            unchecked {
                currentPeriodEnd = block.timestamp + rateLimitDuration;
            }
            currentPeriodAmountTemp = _withdrawnAmount;
        } else {
            unchecked {
                currentPeriodAmountTemp = currentPeriodAmount +
_withdrawnAmount;
            }
        }

        if (currentPeriodAmountTemp > limitAmount) {
            revert RateLimitExceeded();
        }

        currentPeriodAmount = currentPeriodAmountTemp;
    }
```

Commentary from the client:

“ - We are considering a fix but we have not yet clear path way on implementing them, our bandwidth is limited with the mainnet launch. But we will definitely consider a fix long term.

- Is really a trade-off, we cannot deal with it without removing the feature completely. We prefer to deal with DoS issues.”

FUELMESSAGEPORTALV3::RESETRATELIMITAMOUNT() REDUCES CURRENTPERIODAMOUNT WHICH MAY LEAD TO UNEXPECTED BEHAVIOR

SEVERITY: Low

PATH:

FuelMessagePortalV3.sol#L78-L108

FuelERC20GatewayV4.sol#L156-L188

REMEDIATION:

Do not adjust currentPeriodAmount inside FuelMessagePortalV3::resetRateLimitAmount() when the period has not expired and the new limit amount is less than the withdrawn amount in this period.

STATUS: Fixed

DESCRIPTION:

Inside the function **FuelMessagePortalV3::resetRateLimitAmount()**, when the period has not expired and the new limit amount is less than the withdrawn amount in this period, the state var **currentPeriodAmount** is assigned to the new limitAmount (line 104 FuelMessagePortalV3.sol).

Normally this won't cause any problem, but in the following scenario, it may behave unexpectedly: the **SET_RATE_LIMITER_ROLE** sets the limit amount to a lower value and then they set it to a higher value.

For example, let's say the initial limit amount was **10e18** and in this period the amount withdrawn **currentPeriodAmount** is also **10e18**.

Then the **SET_RATE_LIMITER_ROLE** sets a new limit amount to **3e18**. The protocol will adjust the **currentPeriodAmount** to **3e18**.

However the **SET_RATE_LIMITER_ROLE** realizes that their intention was actually setting it to **9e18** and they call **resetRateLimitAmount()** (within the same period) again with **9e18**.

This will be a problem, since in this period additional **6e18** can be withdrawn, because **currentPeriodAmount** is not updated when the limitAmount is then set to **9e18**.

Additionally, the **SET_RATE_LIMITER_ROLE** might not expect that setting the limitAmount from **10e18** to **3e18** and then to **9e18** will behave differently from setting the limitAmount directly to **9e18**.

FuelERC20GatewayV4::resetRateLimitAmount()
(FuelERC20GatewayV4.sol#L156-L188) has almost identical code, so the same issue also exists there.

```
/**
     * @notice Resets the rate limit amount.
     * @param _amount The amount to reset the limit to.
     */
    function resetRateLimitAmount(uint256 _amount) external
onlyRole(SET_RATE_LIMITER_ROLE) {
        uint256 withdrawalLimitAmountToSet;
        bool amountWithdrawnLoweredToLimit;
        bool withdrawalAmountResetToZero;

        // if period has expired then currentPeriodAmount is zero
        if (currentPeriodEnd < block.timestamp) {
            unchecked {
                currentPeriodEnd = block.timestamp + rateLimitDuration;
            }
            withdrawalAmountResetToZero = true;
        } else {
            // If the withdrawn amount is higher, it is set to the new limit
amount
            if (_amount < currentPeriodAmount) {
                withdrawalLimitAmountToSet = _amount;
                amountWithdrawnLoweredToLimit = true;
            }
        }

        limitAmount = _amount;

        if (withdrawalAmountResetToZero || amountWithdrawnLoweredToLimit) {
            currentPeriodAmount = withdrawalLimitAmountToSet;
        }

        emit ResetRateLimit(_amount);
    }
```

MISSING ZERO-VALUE VALIDATION FOR CRITICAL PARAMETERS IN CONSTRUCTOR

SEVERITY: Low

PATH:

FuelChainState.sol#L76-96

REMEDIATION:

Add a check specifically for timeToFinalize, as ensuring it is non-zero will indirectly handle issues with other values like blocksPerCommitInterval and commitCooldown, as they will naturally revert through the existing logic if they are zero:

require(timeToFinalize > 0, "timeToFinalize cannot be zero");

STATUS: Fixed

DESCRIPTION:

In **FuelChainState.sol**, the constructor does not validate that **timeToFinalize**, **blocksPerCommitInterval**, and **commitCooldown** are non-zero. This could result in misconfigurations or unintended behavior if any of these values are set to **0**.

```
    constructor(uint256 timeToFinalize, uint256 blocksPerCommitInterval,
uint32 commitCooldown) {
        if (timeToFinalize > commitCooldown) {
            revert FinalizationIsGtCooldown();
        }uint256 circularBufferSizeInSeconds = NUM_COMMIT_SLOTS *
blocksPerCommitInterval;

    if (timeToFinalize > circularBufferSizeInSeconds) {
        revert TimeToFinalizeTooLarge();
    }

    if (commitCooldown > circularBufferSizeInSeconds) {
        revert CommitCooldownTooLarge();
    }

    TIME_TO_FINALIZE = timeToFinalize;
    COMMIT_COOLDOWN = commitCooldown;
    BLOCKS_PER_COMMIT_INTERVAL = blocksPerCommitInterval;

    _disableInitializers();
}
```

IT IS NOT POSSIBLE TO DISABLE THE RATE LIMITS

SEVERITY: Low

PATH:

FuelMessagePortalV3.sol#L190

FuelERC20GatewayV4.sol#L303

REMEDIATION:

We recommend adding a function that will disable the rate limit or adding a special case when it'll be skipped.

STATUS: Fixed, commits: 1, 2

DESCRIPTION:

The function responsible for the rate limit updates in **FuelMessagePortalV3** and **FuelERC20GatewayV4**, **_addWithdrawnAmount()**, will always be called once activated on every withdrawal. Even if the limit is raised to uint256.max, users would still pay about 3000 gas for each transaction.

```
            _addWithdrawnAmount(withdrawnAmount);
```

```
        if (currentPeriodEnd[tokenAddress] != 0)
_addWithdrawnAmount(tokenAddress, amount);
```

THE RATE LIMIT CAN BE SET RETROACTIVELY IN FUELERC20GATEWAYV4.RESETRATELIMITAMOUNT()

SEVERITY: Low

PATH:

FuelERC20GatewayV4.sol#L156-L188

REMEDIATION:

We recommend the addition of a temporary variable to store the new limit until the end of the current period or start the new period immediately during the call to the reset function.

STATUS: Fixed, commits: 1, 2, 3

DESCRIPTION:

The **resetRateLimitAmount()** function in both **FuelMessagePortalV3** and **FuelERC20GatewayV4** will set the new limit immediately to the current period if it hasn't yet ended.

The function in **FuelERC20GatewayV4** can also update the limit duration. A new higher limit amount and duration (the limit is updated less frequently but with larger amounts) will lead to the possibility of withdrawing a large amount of ERC20 tokens in a short period of time.

For example, suppose the old period is 1 token per 120 seconds, and the new one is 1000 tokens per day. At the moment of change, users can withdraw 1000 tokens in the remaining 120-second period as if it was updated retroactively and lasted a day.

In **FuelMessagePortalV3**, this can also lead to a momentary increase in withdrawals, but it does not pose an issue as the time periods will stay the same.

```
    function resetRateLimitAmount(address _token, uint256 _amount, uint256
_rateLimitDuration) external onlyRole(SET_RATE_LIMITER_ROLE) {
        uint256 withdrawalLimitAmountToSet;
        bool amountWithdrawnLoweredToLimit;
        bool withdrawalAmountResetToZero;

        // avoid multiple SLOADS
        uint256 rateLimitDurationEndTimestamp = currentPeriodEnd[_token];

        // set new rate limit duration
        rateLimitDuration[_token] = _rateLimitDuration;

        // if period has expired then currentPeriodAmount is zero
        if (rateLimitDurationEndTimestamp < block.timestamp) {
            unchecked {
                currentPeriodEnd[_token] = block.timestamp +
_rateLimitDuration;
            }
            withdrawalAmountResetToZero = true;
        } else {
            // If the withdrawn amount is higher, it is set to the new limit
amount
            if (_amount < currentPeriodAmount[_token]) {
                withdrawalLimitAmountToSet = _amount;
                amountWithdrawnLoweredToLimit = true;
            }
        }

        limitAmount[_token] = _amount;

        if (withdrawalAmountResetToZero || amountWithdrawnLoweredToLimit) {
            currentPeriodAmount[_token] = withdrawalLimitAmountToSet;
        }

        emit RateLimitUpdated(_token, _amount);
    }
```

WRONG COMMENT IN FUELERC20GATEWAYV4.SOL

SEVERITY: Informational

PATH:

FuelERC20GatewayV4.sol#L80-L81

REMEDIATION:

Consider fixing the comment for the rateLimitDuration.

STATUS: Fixed

DESCRIPTION:

The comment in FuelERC20GatewayV4.sol on line 80 for the **rateLimitDuration** is incorrect.

```
    /// @notice Amounts already withdrawn this period for each token.
    mapping(address => uint256) public rateLimitDuration;
```

IT WILL BE POSSIBLE TO WITHDRAW AFTER THE UPGRADE TO FUELMESSAGEPORTALV3

SEVERITY: Informational

PATH:

packages/solidity-contracts/deploy/beta5/008.fuel_message_portal_v3.ts#L36-L46

STATUS: Acknowledged

DESCRIPTION:

The scripts of the previous testnets include a call to pause withdrawals right after the upgrade to **FuelMessagePortal**.

Given that the upgrade and the pausing will be performed as separate transactions, it will still be possible to withdraw in between.

```
const portal = FuelMessagePortalV3.connect(address, deployer);

await portal
  .pauseWithdrawals()
  .then((tx) => {
    console.log('Pausing withdrawals...', tx.hash);
    return tx.wait();
  })
  .then(() => console.log('Withdrawals paused'));

return true;
```

PROXY.SW IS NOT FOLLOWING RECOMMENDATIONS OF SRC14 FOR THE FN PROXY_OWNER()

SEVERITY: Informational

PATH:

packages/fungible-token/bridge-fungible-token/proxy/src/proxy.sw

REMEDIATION:

Consider following the docs by updating the function name _proxy_owner() to proxy_owner().

STATUS: Acknowledged

DESCRIPTION:

The SRC14 docs (https://docs.fuel.network/docs/sway-standards/src-14-simple-upgradeable-proxies/#optional-public-functions) are recommending:

The following functions are RECOMMENDED to be implemented by a proxy contract to follow the SRC-14 standard:

fn proxy_owner() -> State;

However, **packages/fungible-token/bridge-fungible-token/proxy/src/proxy.sw** implements the function with a leading underscore as **fn _proxy_owner()**.

```
impl Proxy for Contract {
    #[storage(read)]
    fn _proxy_owner() -> State {
        let owner = storage::SRC14.owner.read();

        match owner {
            State::Uninitialized => INITIAL_OWNER,
            _ => owner,
        }
    }
```


hexens
×
FUEL