hexens × deq.fi

MAY.24

SECURITY REVIEW REPORT FOR DEQ.FI

CONTENTS

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

The audit focused on the Deq protocol smart contracts, which serve as a native liquid staking platform for Avail tokens on Ethereum. Users can deposit AVAIL tokens to receive stAVAIL, which accrues value over time based on staking rewards distributed on the Avail network.

Our security assessment involved a thorough review of the smart contracts over the course of one week. During this audit, we identified one high-severity issue and a few minor issues.

All of the reported issues were promptly addressed by the development team and subsequently validated by us.

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

SCOPE

The analyzed resources are located on:

https://github.com/refresh-labs/deq-contracts

The issues described in this report were fixed in the following commit:

https://github.com/refresh-labs/deq-contracts/commit/aba323b4af7f06ce6859cb6c1f4abcc5f253ad52

https://github.com/refresh-labs/deq-contracts/commit/00c28239ae821786c4a1b80962e45d22225715b2

AUDITING DETAILS

STARTED
06.05.2024

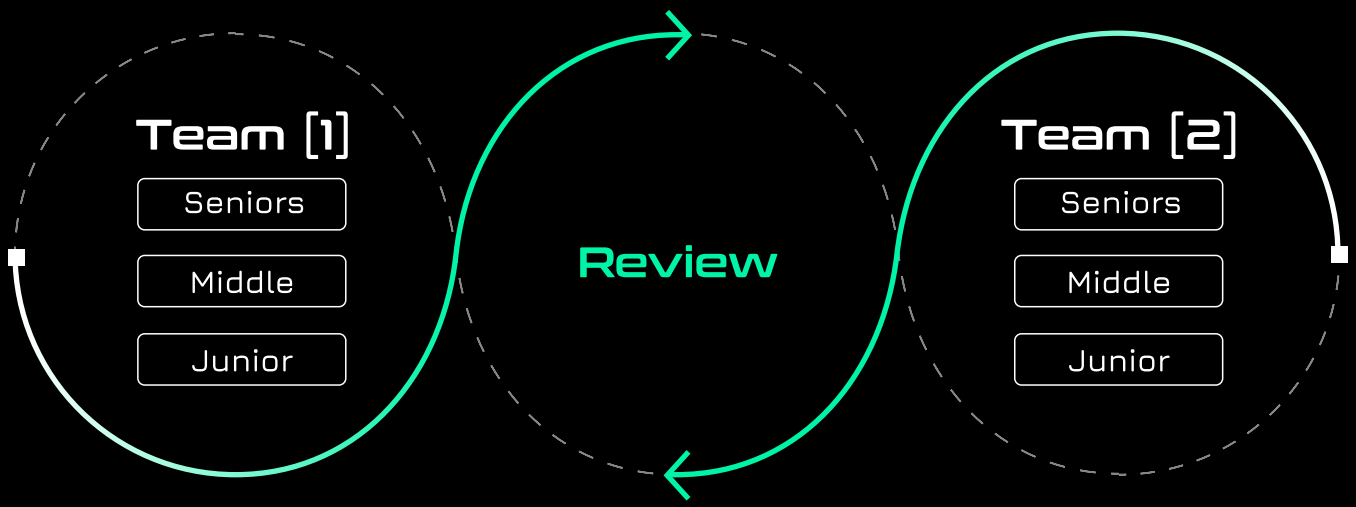
DELIVERED
10.05.2024

Review Led by

MIKHAIL EGOROV
Senior 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.

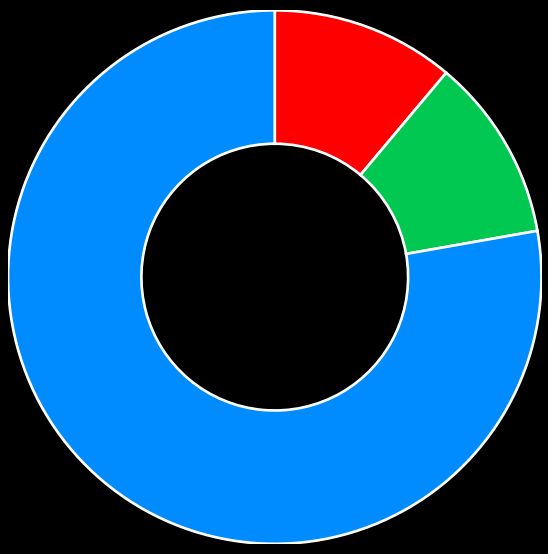
ISSUE SYMBOLIC CODES

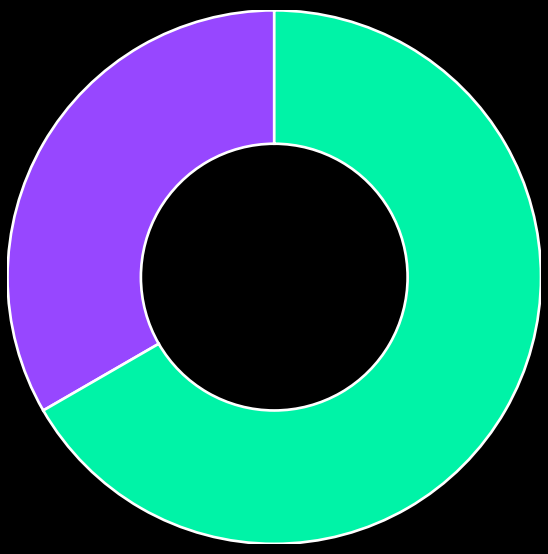
Every issue being identified and validated has its unique symbolic code assigned to the issue at the security research stage. Cause of the vulnerability reporting flow design, some of the rejected issues could be missing.

FINDINGS SUMMARY

Severity	Number of Findings
Critical	0
High	1
Medium	0
Low	1
Informational	7

Total: 9





WEAKNESSES

This section contains the list of discovered weaknesses.

RFRSH-8

BURN DOESN'T CONSIDER CURRENT SHARES-TO-ASSETS RATIO

SEVERITY: High

PATH:

src/AvailWithdrawalHelper.sol:L128-L142

REMEDIATION:

The burn() function of AvailWithdrawalHelper should utilize the current shares-to-assets ratio.

STATUS: Fixed

DESCRIPTION:

The **burn()** function of the **AvailWithdrawalHelper** contract does not consider the current ratio between shares and assets. Consequently, when a protocol incurs losses and the **updateAssets()** function of **StakedAvail** is invoked with a negative delta, the **burn()** call either reverts or distributes losses unfairly among users trying to withdraw their assets.

Consider the the following scenario where the **burn()** call reverts:

1. Alice deposits 1000 Avail tokens into the protocol (via mint()) and receives 1000 stAvail tokens.
2. Subsequently, Alice calls the **burn()** function to burn all her shares (1,000 stAvail tokens).
3. The **UPDATER_ROLE** then triggers the **updateAssets()** function with a negative delta value (e.g., -100), resulting in the asset value being updated to 900 post-execution of the function.
4. Alice proceeds to call the **burn()** function in the **AvailWithdrawalHelper** contract, subsequently invoking the **stAvail.updateAssetsFromWithdrawals(amount, shares)** function with the values amount=1000 and shares=1000.
5. In the **StakedAvail** contract, the assets value is 900, but the amount passed from the **AvailWithdrawalHelper** is 1000. This discrepancy causes the transaction to revert with an underflow error.

In another scenario, a malicious user could front-run the **updateAssets()** call by creating a withdrawal for their Avail tokens, thus avoiding paying for the losses. Less fortunate users would then have to share those additional losses of the malicious user.

```
    function _executeSwapExactAmountOutOnMakerPSM(
        address exchange,
        uint256 swapType,
        uint256 toll,
        uint256 to18ConversionFactor,
        uint256 toAmount,
        address beneficiary
    )
        internal
    {
        // If swapType is 0, then we are calling buyGem
        if (swapType == 0) {
            IMakerPSM(exchange).buyGem(beneficiary, toAmount);
        }
        // If swapType is 1, then we are calling sellGem
        else {
            // Calculate the amount to sell
            uint256 a = toAmount * WAD;
            uint256 b = (WAD - toll) * to18ConversionFactor;
            uint256 gemAmt = (a + b - 1) / b;
            // Call sellGem
            IMakerPSM(exchange).sellGem(beneficiary, gemAmt);
        }
    }
```

RFRSH-9

MISSING DEADLINE PARAMETER IN SWAPS

SEVERITY: Low

PATH:

DeqRouter.sol::swapERC20ToStAvail():L41-L55

DeqRouter.sol::swapERC20ToStAvailWithPermit():L60-L83

DeqRouter.sol::swapETHtoStAvail():L87-L101

REMEDIATION:

Allow users to set a fixed deadline as a parameter and never automatically assign the deadline as block.timestamp within the function, as that would effectively mean that the transaction has no deadline.

STATUS: Fixed

DESCRIPTION:

Although the **DeqRouter.sol** contract's **swap** functions incorporate slippage protection, they currently lack a deadline element. This absence leaves transactions without a specified **deadline**, potentially allowing them to be included at any time by validators while they are pending in the mempool.

As a result, transactions may be executed much later than intended by the user. Consequently, users or the protocol might receive unfavorable prices, as validators could delay transaction inclusion to manipulate trade outcomes.

```
function swapERC20ToStAvail(address allowanceTarget, bytes calldata data)
external {
        (IERC20 tokenIn, IERC20 tokenOut, uint256 inAmount, uint256
minOutAmount,) =
                abi.decode(data[4:], (IERC20, IERC20, uint256, uint256,
Transformation[]));
    if (address(tokenOut) != address(avail)) revert InvalidOutputToken();
    tokenIn.safeTransferFrom(msg.sender, address(this), inAmount);
    tokenIn.forceApprove(allowanceTarget, inAmount);
    // slither-disable-next-line low-level-calls
    (bool success, bytes memory result) = swapRouter.call(data);
    if (!success) revert SwapFailed(string(result));
    uint256 outAmount = abi.decode(result, (uint256));
    if (outAmount < minOutAmount) revert ExceedsSlippage();
    // slither-disable-next-line unused-return
    avail.approve(address(stAvail), outAmount);
    stAvail.mintTo(msg.sender, outAmount);
}
```

```
function swapERC20ToStAvailWithPermit(
    address allowanceTarget,
    bytes calldata data,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) external {
        (IERC20 tokenIn, IERC20 tokenOut, uint256 inAmount, uint256
minOutAmount,) =
                abi.decode(data[4:], (IERC20, IERC20, uint256, uint256,
Transformation[]));
    if (address(tokenOut) != address(avail)) revert InvalidOutputToken();
    // if permit fails, assume executed
       try IERC20Permit(address(tokenIn)).permit(msg.sender, address(this),
inAmount, deadline, v, r, s) {} catch {}
    tokenIn.safeTransferFrom(msg.sender, address(this), inAmount);
    tokenIn.forceApprove(allowanceTarget, inAmount);
    // slither-disable-next-line low-level-calls
    (bool success, bytes memory result) = swapRouter.call(data);
    if (!success) revert SwapFailed(string(result));
    uint256 outAmount = abi.decode(result, (uint256));
    if (outAmount < minOutAmount) revert ExceedsSlippage();
    // slither-disable-next-line unused-return
    avail.approve(address(stAvail), outAmount);
    stAvail.mintTo(msg.sender, outAmount);
}
```

```
function swapETHtoStAvail(bytes calldata data) external payable {
        (address tokenIn, IERC20 tokenOut, uint256 inAmount, uint256
minOutAmount,) =
                abi.decode(data[4:], (address, IERC20, uint256, uint256,
Transformation[]));
       if (address(tokenIn) != 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE)
revert InvalidInputToken();
    if (address(tokenOut) != address(avail)) revert InvalidOutputToken();
    if (msg.value != inAmount) revert InvalidInputAmount();
    // slither-disable-next-line low-level-calls
     (bool success, bytes memory result) = swapRouter.call{value: msg.value}
(data);
    if (!success) revert SwapFailed(string(result));
    uint256 outAmount = abi.decode(result, (uint256));
    if (outAmount < minOutAmount) revert ExceedsSlippage();
    // slither-disable-next-line unused-return
    avail.approve(address(stAvail), outAmount);
    stAvail.mintTo(msg.sender, outAmount);
}
```

RFRSH-1

LACK OF EVENT EMISSION

SEVERITY: Informational

REMEDIATION:

We recommend adding appropriate event emissions to the project.

STATUS: Fixed

DESCRIPTION:

All project, especially the **AvailDepository.sol** and **DeqRouter.sol** contracts currently lack event emissions, which are essential for providing transparency and monitoring capabilities for their respective functionalities.

In the **AvailDepository.sol** contract, events serve as crucial notifications for deposit and update actions. They enable users and external systems to track deposits of Avail ERC20 tokens to the depository on Avail and updates to the depository address. Without events, stakeholders may lack visibility into these critical contract interactions.

Similarly, in the **DeqRouter.sol** contract, events are necessary for transparent notification of token swap transactions to staked Avail. Events would allow users and external systems to monitor contract activities, ensuring that swaps are executed correctly and under user expectations.

```
function deposit() external onlyRole(DEPOSITOR_ROLE) {
    uint256 amount = avail.balanceOf(address(this));
     // keep 1 wei so slot stays warm, intentionally leave return unused,
since OZ impl does not return false
    // slither-disable-next-line unused-return
    avail.approve(address(bridge), amount - 1);
    bridge.sendAVAIL(depository, amount - 1);
}
```

RFRSH-2

MISSING ERROR HANDLING IN CATCH BLOCK

SEVERITY: Informational

PATH:

src/StakedAvail.sol:L178

src/DeqRouter.sol:L108

REMEDIATION:

It is recommended to add a check for verifying the allowance amount after the permit function call inside the catch block.

STATUS: Acknowledged, see commentary

DESCRIPTION:

The catch blocks of the **mintWithPermit()** and **swapERC20ToStAvailWithPermit()** functions could benefit from including error-handling code to address potential front-running attack scenarios. Specifically, the code mentioned should verify if the allowance amount is sufficient.

```
/// @notice Mints LST based on the amount of Avail staked with permit
/// @dev Reverts if amount is 0
/// @param amount Amount of Avail to stake
/// @param deadline Deadline for the permit
/// @param v Signature v
/// @param r Signature r
/// @param s Signature s
function mintWithPermit(uint256 amount, uint256 deadline, uint8 v, bytes32 r, bytes32 s) external {
    if (amount == 0) revert ZeroAmount();
    uint256 shares = previewMint(amount);
    // slither-disable-next-line events-maths
    assets += amount;
    _mint(msg.sender, shares);
        try  IERC20Permit(address(avail)).permit(msg.sender,  address(this), amount, deadline, v, r, s) {} catch {}
    avail.safeTransferFrom(msg.sender, depository, amount);
}
```

```
/// @notice Swaps an ERC20 token to staked Avail with permit
/// @param allowanceTarget Address of the allowance target from 0x API
/// @param data Data for the swap from 0x API
function swapERC20ToStAvailWithPermit(
    address allowanceTarget,
    bytes calldata data,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) external whenNotPaused {
        (IERC20 tokenIn, IERC20 tokenOut, uint256 inAmount, uint256
minOutAmount,) =
                abi.decode(data[4:], (IERC20, IERC20, uint256, uint256,
Transformation[]));
    if (address(tokenOut) != address(avail)) revert InvalidOutputToken();
    // if permit fails, assume executed
       try IERC20Permit(address(tokenIn)).permit(msg.sender, address(this),
inAmount, deadline, v, r, s) {} catch {}
    tokenIn.safeTransferFrom(msg.sender, address(this), inAmount);
    tokenIn.forceApprove(allowanceTarget, inAmount);
    // slither-disable-next-line low-level-calls
    (bool success, bytes memory result) = swapRouter.call(data);
    if (!success) revert SwapFailed(string(result));
    uint256 outAmount = abi.decode(result, (uint256));
    if (outAmount < minOutAmount) revert ExceedsSlippage();
    // slither-disable-next-line unused-return
    avail.approve(address(stAvail), outAmount);
    stAvail.mintTo(msg.sender, outAmount);
}
```

Commentary from the client:

“ - The error handling wastes gas without any added usefulness. Any added allowance checks would already happen in the consequent transfers.”

RFRSH-3

LACK OF HANDLING FOR FEE-ON-TRANSFER TOKENS IN DEQROUTER CONTRACT

SEVERITY: Informational

PATH:

DeqRouter.sol::swapERC20ToStAvail():L41-L55

DeqRouter.sol::swapERC20ToStAvailWithPermit():L60-L83

REMEDIATION:

We recommend explicitly documenting the protocol's stance regarding fees-on-transfer tokens.

STATUS: Acknowledged, see commentary

DESCRIPTION:

The **DeqRouter.sol** contract facilitates the swapping of ERC20 tokens to staked Avail tokens. However, it does not incorporate a mechanism to handle fee-on-transfer tokens effectively. Fee-on-transfer tokens are ERC20 tokens that implement a fee mechanism on transfers, deducting a percentage of tokens from each transfer and redistributing them to holders or a designated address.

When swapping fee-on-transfer tokens using the **DeqRouter.sol** contract, users may experience discrepancies between the requested transfer amount and the actual amount received due to the deduction of fees during token transfers.However, the 0x protocol does not fully support fees-on-transfer tokens. Transactions involving such tokens are likely to revert due to slippage protection.

However, the 0x protocol does not fully support fees-on-transfer tokens. Transactions involving such tokens are likely to revert due to slippage protection (Exceptional ERC20s — 0x Protocol 4.1 documentation).

```
function swapERC20ToStAvail(address allowanceTarget, bytes calldata data)
external {
    (IERC20 tokenIn, IERC20 tokenOut, uint256 inAmount, uint256 minOutAmount,) =
                    abi.decode(data[4:], (IERC20, IERC20, uint256, uint256,
Transformation[]));
    if (address(tokenOut) != address(avail)) revert InvalidOutputToken();
    tokenIn.safeTransferFrom(msg.sender, address(this), inAmount);
    tokenIn.forceApprove(allowanceTarget, inAmount);
    // slither-disable-next-line low-level-calls
    (bool success, bytes memory result) = swapRouter.call(data);
    if (!success) revert SwapFailed(string(result));
    uint256 outAmount = abi.decode(result, (uint256));
    if (outAmount < minOutAmount) revert ExceedsSlippage();
    // slither-disable-next-line unused-return
    avail.approve(address(stAvail), outAmount);
    stAvail.mintTo(msg.sender, outAmount);
}
```

Commentary from the client:

“ - There is no support for fee-on-transfer tokens, this has been updated in the contract NatSpec documentation.”

INCONSISTENCY IN COMMENT AND FUNCTION BEHAVIOR

SEVERITY: Informational

PATH:

AvailWithdrawalHelper.sol::previewFulfill():L69-L71

REMEDIATION:

Update the comment for the previewFulfill() function to reflect the actual behavior.

STATUS: Fixed

DESCRIPTION:

The comment states that the **previewFulfill()** function reverts if **till** is less than or equal to **lastFulfillment**, implying that the function may revert even if **till** equals **lastFulfillment**. However, in the current implementation, the function does not revert when **till** is equal to **lastFulfillment**. Instead, it returns a value of 0, which is inconsistent with the provided comment.

```
 /// @notice Returns fulfilment amount between lastFulfillment and till
/// @dev Reverts if till is less than or equal to lastFulfillment
/// @param till Token ID to iterate till
function previewFulfill(uint256 till) public view returns (uint256 amount) {
                            return       withdrawals[till].accAmount       -
withdrawals[lastFulfillment].accAmount;
}
```

RFRSH-6

THERE ARE NO CHECKS OF THE SELECTOR IN THE DEQROUTER SWAP FUNCTIONS

SEVERITY: Informational

PATH:

src/DeqRouter.sol:L38-L101

REMEDIATION:

Decode the selector and check that it is equal to 0x415565b0 (transformERC20()).

STATUS: Acknowledged, see commentary

DESCRIPTION:

The **swapERC20ToStAvail()**, **swapERC20ToStAvailWithPermit()**, and **swapETHtoStAvail()** functions of DeqRouter don't have any checks of the selector in the **data** argument. It allows users to use any function that has a compatible abi.

For example, a call to some public constant on the 0x side will be successful because the automatically created getters for public variables don't require any arguments, so they'll just skip the part of **data** that contains **tokenIn**, **tokenOut**, **inAmount**, **minOutAmount**, and **transformations** and return some constant value that will be approved to **stAvail**.

It's only an issue if there's AVAIL token on the balance of **DeqRouter**, which is not planned.

The users can also set **allowanceTarget** to their address in this case to get their transferred tokens back because 0x will not use any allowance.

```
/// @notice Swaps an ERC20 token to staked Avail
/// @param allowanceTarget Address of the allowance target from 0x API
/// @param data Data for the swap from 0x API
function swapERC20ToStAvail(address allowanceTarget, bytes calldata data)
external {
    (IERC20 tokenIn, IERC20 tokenOut, uint256 inAmount, uint256 minOutAmount,) =
                    abi.decode(data[4:], (IERC20, IERC20, uint256, uint256,
Transformation[]));
    if (address(tokenOut) != address(avail)) revert InvalidOutputToken();
    tokenIn.safeTransferFrom(msg.sender, address(this), inAmount);
    tokenIn.forceApprove(allowanceTarget, inAmount);
    // slither-disable-next-line low-level-calls
    (bool success, bytes memory result) = swapRouter.call(data);
    if (!success) revert SwapFailed(string(result));
    uint256 outAmount = abi.decode(result, (uint256));
    if (outAmount < minOutAmount) revert ExceedsSlippage();
    // slither-disable-next-line unused-return
    avail.approve(address(stAvail), outAmount);
    stAvail.mintTo(msg.sender, outAmount);
}

/// @notice Swaps an ERC20 token to staked Avail with permit
/// @param allowanceTarget Address of the allowance target from 0x API
/// @param data Data for the swap from 0x API
function swapERC20ToStAvailWithPermit(
    address allowanceTarget,
    bytes calldata data,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) external {
    (IERC20 tokenIn, IERC20 tokenOut, uint256 inAmount, uint256 minOutAmount,) =
                    abi.decode(data[4:], (IERC20, IERC20, uint256, uint256,
Transformation[]));
    if (address(tokenOut) != address(avail)) revert InvalidOutputToken();
    // if permit fails, assume executed
          try IERC20Permit(address(tokenIn)).permit(msg.sender, address(this),
inAmount, deadline, v, r, s) {} catch {}
    tokenIn.safeTransferFrom(msg.sender, address(this), inAmount);
    tokenIn.forceApprove(allowanceTarget, inAmount);
```

```
    // slither-disable-next-line low-level-calls
    (bool success, bytes memory result) = swapRouter.call(data);
    if (!success) revert SwapFailed(string(result));
    uint256 outAmount = abi.decode(result, (uint256));
    if (outAmount < minOutAmount) revert ExceedsSlippage();
    // slither-disable-next-line unused-return
    avail.approve(address(stAvail), outAmount);
    stAvail.mintTo(msg.sender, outAmount);
}

/// @notice Swaps ETH to staked Avail
/// @param data Data for the swap from 0x API
function swapETHtoStAvail(bytes calldata data) external payable {
    (address tokenIn, IERC20 tokenOut, uint256 inAmount, uint256 minOutAmount,) =
                  abi.decode(data[4:],  (address,  IERC20,  uint256,  uint256,
Transformation[]));
     if (address(tokenIn) != 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE) revert
InvalidInputToken();
    if (address(tokenOut) != address(avail)) revert InvalidOutputToken();
    if (msg.value != inAmount) revert InvalidInputAmount();
    // slither-disable-next-line low-level-calls
      (bool success, bytes memory result) = swapRouter.call{value: msg.value}
(data);
    if (!success) revert SwapFailed(string(result));
    uint256 outAmount = abi.decode(result, (uint256));
    if (outAmount < minOutAmount) revert ExceedsSlippage();
    // slither-disable-next-line unused-return
    avail.approve(address(stAvail), outAmount);
    stAvail.mintTo(msg.sender, outAmount);
}
```

Commentary from the client:

“ - The selector would differ based on the type of swap call and can change in the future due to changes in the 0x proxy contract. Hence, we are leaving this as-is.”

INITIALIZERS ARE NOT DISABLED

SEVERITY: Informational

PATH:

src/AvailDepository.sol#L35-L39
src/StakedAvail.sol#L40-L43
src/AvailWithdrawalHelper.sol#L42-L45

REMEDIATION:

Add _disableInitializers() to the constructors of the contracts.

STATUS: Fixed

DESCRIPTION:

AvailDepository, **StakedAvail**, and **AvailWithdrawalHelper** have initializer functions open on the implementation contracts which can cause unnecessary confusion in case some user calls them to set storage variables or claim the ownership.

```
constructor(IERC20 newAvail, IAvailBridge newBridge) {
    if (address(newAvail) == address(0) || address(newBridge) == address(0))
revert ZeroAddress();
    avail = newAvail;
    bridge = newBridge;
}
```

```
constructor(IERC20 newAvail) {
    if (address(newAvail) == address(0)) revert ZeroAddress();
    avail = newAvail;
}
```

```
  constructor(IERC20 newAvail) {
      if (address(newAvail) == address(0)) revert ZeroAddress();
      avail = newAvail;
  }
```

RFRSH-10

EXTERNAL DEPENDENCY ON THE UPGRADEABLE BRIDGE CONTRACT

SEVERITY: Informational

PATH:

src/AvailDepository.sol:L85-L91

REMEDIATION:

We recommend adding an alternative function that will send the funds directly to the AvailWithdrawalHelper without involving the bridge.

STATUS: Fixed

DESCRIPTION:

The only way to withdraw the deposited AVAIL in **AvailDepository** is through the **deposit()** function. If there is an external issue with the bridge or the L2, the users will not be able to withdraw the funds.

```
function deposit() external whenNotPaused onlyRole(DEPOSITOR_ROLE) {
    uint256 amount = avail.balanceOf(address(this));
     // keep 1 wei so slot stays warm, intentionally leave return unused,
since OZ impl does not return false
    // slither-disable-next-line unused-return
    avail.approve(address(bridge), amount - 1);
    bridge.sendAVAIL(depository, amount - 1);
}
```


hexens
×
deq.fi