hexens × valora

JUNE.24

SECURITY REVIEW REPORT FOR VALORA

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.

SCOPE

The analyzed resources are located on:

https://github.com/valora-inc/swap-contracts

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

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.

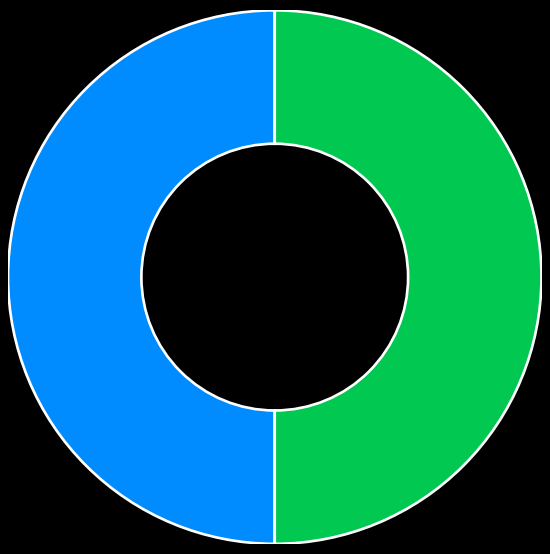
ISSUE SYMBOLIC CODES

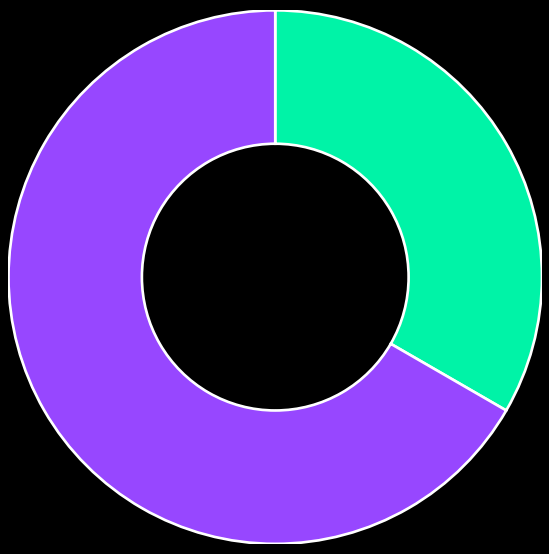
Every issue being identified and validated has its unique symbolic code assigned to the issue at the security research stage. Cause of the vulnerability reporting flow design, some of the rejected issues could be missing.

FINDINGS SUMMARY

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

Total: 6





WEAKNESSES

This section contains the list of discovered weaknesses.

VLR-5

EXCESS ETH FORWARDING TO TARGET CONTRACT

SEVERITY: Low

PATH:

contracts/SwapHandler.sol::swap()#L29-L78

REMEDIATION:

Ensure that only the exact amount required for the transaction (fromAmount) is forwarded to the target contract.

STATUS: Acknowledged

DESCRIPTION:

In the **swap()** function of **SwapHandler.sol**, if **fromToken** is the native token, the contract checks if **msg.value** is greater than or equal to **fromAmount + fee**.

If **msg.value** exceeds **fromAmount + fee**, the excess ETH (**msg.value - (fromAmount + fee)**) is included when calling **target.functionCallWithValue(data, value)**.

As a result excess ETH sent by the user beyond the required amount (**fromAmount + fee**) is forwarded to the **target** contract during the function call.

```
uint256 value = msg.value;
    if (fromToken != Constants.NATIVE_TOKEN) {
      // Take the fee
      _transfer(fromToken, fee, feeWallet);
      // Allow the target to spend fromToken
      fromToken.forceApprove(target, fromAmount);
    } else {
      if (value < fromAmount + fee) {
        revert InsufficientAmount();
      }
      // Take the feeS
      payable(feeWallet).sendValue(fee);
      value -= fee;
    }

    // Do the swap via the target contract
    target.functionCallWithValue(data, value);
```

VLR-6

OWNABLE2STEP NOT USED

SEVERITY: Low

PATH:

contracts/ValoraSwapV2.sol#L12

REMEDIATION:

Use Ownable2Step to prevent accidental loss of ownership.

STATUS: Fixed

DESCRIPTION:

By demanding that the receiver of the owner permissions actively accept via a contract call of its own, **Ownable2Step** and **Ownable2StepUpgradeable** prevent the contract ownership from accidentally being transferred to an address that cannot handle it.

```
contract ValoraSwapV2 is Pausable, Ownable {
```

VLR-9

OWNER CAN RENOUNCE OWNERSHIP WHEN SYSTEM IS PAUSED

SEVERITY: Low

PATH:

contracts/ValoraSwapV2.sol#L18

REMEDIATION:

Remove renounce ownership function from Ownable contract.

STATUS: Acknowledged

DESCRIPTION:

Owner can renounce ownership when system is paused.

```
    constructor(address initialOwner, address payable feeWallet)
Ownable(initialOwner) {
```

VLR-7

NESTED IFS NOT USED

SEVERITY: Informational

PATH:

contracts/SwapHandler.sol#L69-L71

REMEDIATION:

Use nested ifs.

STATUS: Fixed

DESCRIPTION:

Using nested is cheaper than using && multiple check combinations. There are more advantages, such as easier to read code and better coverage reports.

Since the contract relays the swaps to an external DEX, there might be high gas costs during each call, therefore it makes sense to save as much gas as possible.

Here're the gas usage reports before and after the optimization:

```
|    swap    ·     84,252   ·     171,108   ·     126,503   ·     18   ·   –   |
|    swap    ·     84,241   ·     171,099   ·     126,492   ·     18   ·   –   |
```

```
if (toToken != Constants.NATIVE_TOKEN && toNetworkId.equal(fromNetworkId)) {
    _transfer(toToken, toToken.balanceOf(address(this)), recipient);
}
```

VLR-4

MISSING EVENTS

SEVERITY: Informational

PATH:

contracts/ValoraSwapV2.sol::swap()#L34-L65

contracts/SwapHandler.sol::swap()#L29-L78

contracts/SwapHandler.sol::_transfer()#L80-L84

REMEDIATION:

Implement events in the project.

STATUS: Acknowledged

DESCRIPTION:

Events are intended to be added to improve the transparency and trackability of changes. Specifically, functions **ValoraSwapV2.sol::swap()**, **SwapHandler.sol::swap()**, and **SwapHandler.sol::_transfer()** do not emit events upon actions.

VLR-12

ZERO ADDRESS CHECK IS MISSING

SEVERITY: Informational

PATH:

contracts/ValoraSwapV2.sol#L18
contracts/ValoraSwapV2.sol::swap()#L34-L65

REMEDIATION:

Add zero address checks.

STATUS: Acknowledged

DESCRIPTION:

Zero address check is missing in the constructor and the **swap** function. This may lead to the loss of funds, for example, if someone initiates a swap with a `0` address recipient.

```
    constructor(address initialOwner, address payable feeWallet)
Ownable(initialOwner) {
```

```
  function swap(
  [...]
```


hexens × valora