hexens × polygon

FEB.24

SECURITY REVIEW REPORT FOR POLYGON

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 reviewing the code of smart contracts for the WETH Plus and Staking The Bridge projects. The WETH Plus project represents a new version of WETH powered by ERC-7535, ERC-2612 message signing, and EIP-1153 transient storage. The Staking The Bridge project includes L1Escrow, L2Escrow, L2TokenConverter, and L2Token smart contracts, enabling the bridging of tokens between L1 and L2.

Our security assessment involved a comprehensive review of the smart contracts, spanning a total of 1 week. During this audit, we identified two medium-severity vulnerabilities, several low-severity vulnerabilities, and informational issues.

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 of the project have increased after the completion of our audit.

SCOPE

The analyzed resources are located on:

https://github.com/ERC4626-Alliance/WETHPlus/commit/120440fec92b453f34f0356168b7ec9cf6bc00bc

https://github.com/pyk/zkevm-stb/commit/a17f348fc44c9160e48bd78586e56bbd2499cd1e

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

https://github.com/ERC4626-Alliance/WETHPlus/commit/74a7268b533b255a75a612cb7bf090cdd6d9ab5a

https://github.com/pyk/zkevm-stb/commit/30d57cf9298cd468f5637b514fba3098b6560045

AUDITING DETAILS

STARTED
05.02.2024

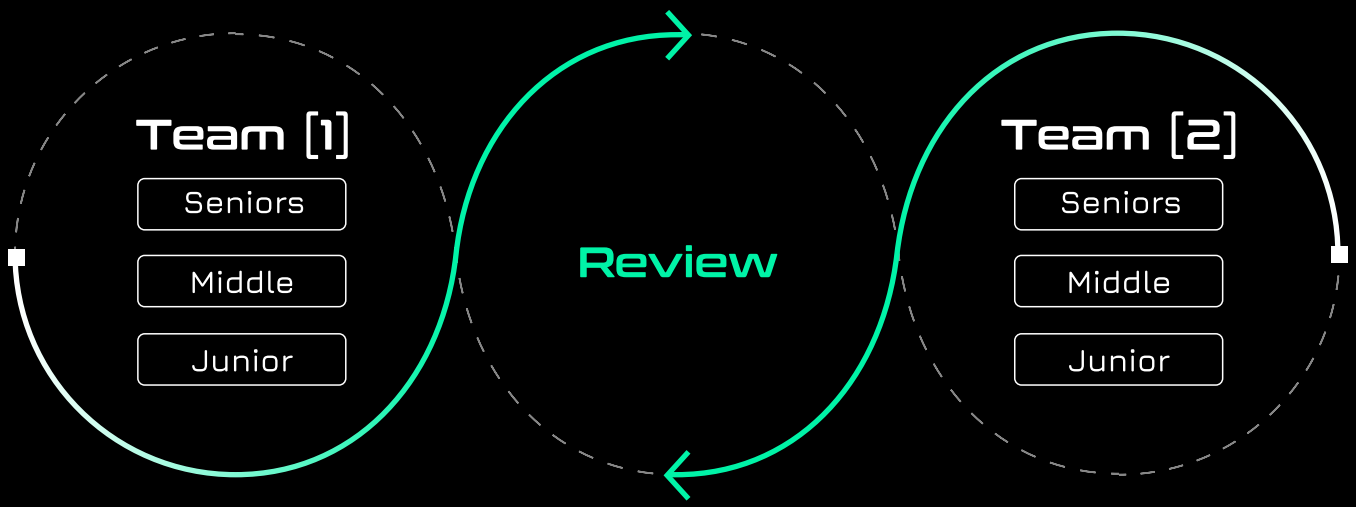
DELIVERED
09.02.2024

Review Led by

MIKHAIL EGOROV
Lead Smart Contract Auditor | 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.

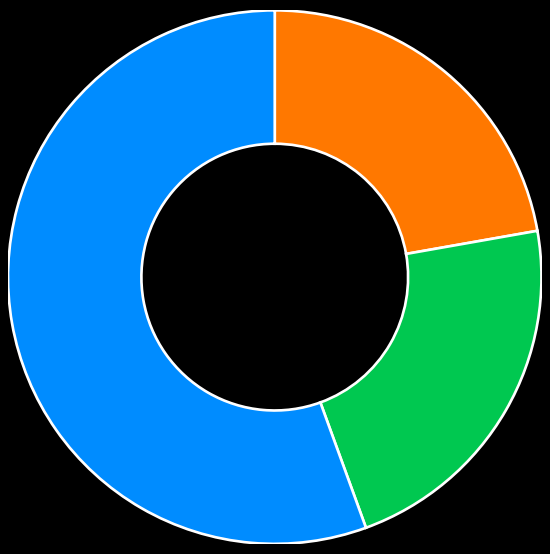
ISSUE SYMBOLIC CODES

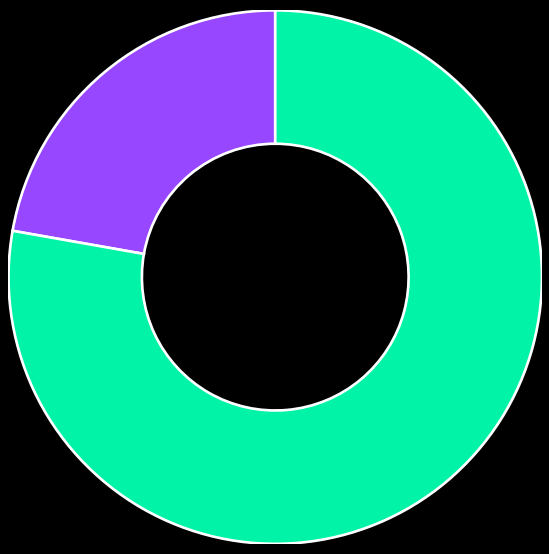
Every issue being identified and validated has its unique symbolic code assigned to the issue at the security research stage. Cause of the vulnerability reporting flow design, some of the rejected issues could be missing.

FINDINGS SUMMARY

Severity	Number of Findings
Critical	0
High	0
Medium	2
Low	2
Informational	5

Total: 9





WEAKNESSES

This section contains the list of discovered weaknesses.

PLFD-2

ERC7575WITHDRAWVAULT INTERFACEID DIFFERS FROM SPECIFICATION

SEVERITY: Medium

PATH:

WETHPlus.sol:supportsInterface():L75-80

REMEDIATION:

Refrain from using hardcoded values for interface ids. Instead, calculate them as follows.

type(IERC7575Withdraw).interfaceId

STATUS: Fixed

DESCRIPTION:

The **WETHPlus** contract is designed to support **ERC-165**, which allows a smart contract to indicate which interfaces it implements. According to the specification, the value of interfaceId should be calculated as the XOR of all function selectors in the interface. However, in the case of **ERC7575WithdrawVault** interface, the calculation of interfaceId is incorrect.

The current value of **0xe1550342** is incorrect, and the correct value should be **0x70dec094**.

ERC-7575: Partial and Extended ERC-4626 Vaults by Joeysantoro · Pull Request #157 · ethereum/ERCs

As a result, the call to **supportsInterface()** with the correct value of **interfaceId** for the **ERC7575WithdrawVault** interface will return **false**.

```
function supportsInterface(bytes4 interfaceId) external pure returns (bool) {
        return interfaceId == 0x01ffc9a7 // //@note ok
                || interfaceId == 0x50a526d6 // ERC7575MinimalVault //@note ok
                || interfaceId == 0xc1f329ef // ERC7575DepositVault //@note ok
                   || interfaceId == 0xe1550342; // ERC7575WithdrawVault //@audit
type(IERC7575Withdraw).interfaceId == 0x70dec094
    }
```

PLFD-7

DEPOSITALL CALL CAN BE BLOCKED

SEVERITY: Medium

PATH:

WETHUpgradeRouter.sol:L32

REMEDIATION:

To mitigate this issue, implement the following solution. Before calling permit(), verify if the msg.sender has type(uint256).max approval for Permit2. If this condition is satisfied, skip the call to pemit() altogether.

STATUS: Fixed

DESCRIPTION:

The **depositAll()** function currently utilizes an inner call to the **permit()** function of the **WETHPLus** contract, which grants unlimited approval to the **Permit2** contract. However, this flow exposes **depositAll()** to a griefing attack, where an attacker can forcibly block the victim's transaction.

The attack proceeds as follows: the attacker front-runs the victim's transactions, extracts the signature from the mempool, and places a transaction that directly calls **WETHPLus.permit()** with the victim's signature. Consequently, the victim's transaction reverts since the signature has already been used for **permit()** in the attacker's transaction.

```
function depositAll(
        address to,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) public payable returns (uint256 shares) {

          // If the user passes in a permit of permit2, then attempt to
execute it
        if (deadline >= block.timestamp) {
             IWETHPlus(share).permit(msg.sender, PERMIT2, type(uint256).max,
deadline, v, r, s);
        }

        // Withdraw all WETH9 balance of caller
        uint256 weth9Amount = IWETH9(asset).balanceOf(msg.sender);
        return deposit(weth9Amount, to);
    }
```

PLFD-1

APPROVE RACE CONDITION IN WETHPLUS

SEVERITY: Low

PATH:

WETHPlus.sol

REMEDIATION:

Consider adding the increaseAllowance() and decreaseAllowance() functions to WETHPlus.

STATUS: Acknowledged

DESCRIPTION:

The WETHPlus contract inherits from the **ERC20.sol** contract. However, this implementation lacks protection against the well-known "Multiple Withdrawal Attack" on the **Approve/TransferFrom** methods of the **ERC20** standard. This issue allows a malicious actor to front-run an approve transaction of the victim in an attempt to gain additional tokens.

```
function approve(address spender, uint256 amount) public virtual returns
(bool) {//@audit vulnarable for race condition or front running
        allowance[msg.sender][spender] = amount;

        emit Approval(msg.sender, spender, amount);

        return true;
    }
```

```
function transferFrom(
        address from,
        address to,
        uint256 amount
    ) public virtual returns (bool) {
           uint256 allowed = allowance[from][msg.sender]; // Saves gas for
limited approvals.

           if (allowed != type(uint256).max) allowance[from][msg.sender] =
allowed - amount;

        balanceOf[from] -= amount;

        // Cannot overflow because the sum of all user
        // balances can't exceed the max uint256 value.
        unchecked {
            balanceOf[to] += amount;
        }

        emit Transfer(from, to, amount);

        return true;
    }
```

PLFD-10

INCORRECT MAXISSUANCE CHECK MAY BLOCK FUNDS UNTIL THE RISK MANAGER TAKES ACTION

SEVERITY: Low

PATH:

L2TokenConverter.sol:L161

REMEDIATION:

Remove the if (_amount > maxIssuance) revert MaxIssuance(); condition from the withdraw() function at line 161.

STATUS: Fixed

DESCRIPTION:

Incorrect maxIssuance check may block funds until the risk manager takes action.

Consider the following scenario:

1. The risk manager sets maxIssuance for **[token][l2token]** to 1000.
2. The user deposits 1000 tokens by calling the **deposit()** function.
3. maxIssuance decreases to 0.
4. The user tries to withdraw tokens, but as there is a check **if (amount > maxIssuacne)** revert and **maxIssuance** is 0, the **withdraw()** function call will revert until the risk manager sets a new **maxIssuance** value.

```
    /// @dev User can deposit ERC-20 in exchange for L2Token
       function deposit(IERC20Metadata _token, IL2Token _l2Token, address
_recipient, uint256 _amount) external virtual {
        L2TokenConverterStorage storage $ = _getL2TokenConverterStorage();
        uint256 maxIssuance = $.issuances[_token][_l2Token];
        if (_amount > maxIssuance) revert MaxIssuance();

        // Reduce max issuance
        $.issuances[_token][_l2Token] -= _amount;

        _token.safeTransferFrom(msg.sender, address(this), _amount);
        _l2Token.converterMint(_recipient, _amount);

        emit Deposit(_token, _l2Token, msg.sender, _recipient, _amount);
    }

    /// @dev User can withdraw ERC-20 by burning L2Token
      function withdraw(IERC20Metadata _token, IL2Token _l2Token, address
_recipient, uint256 _amount) external virtual {
        L2TokenConverterStorage storage $ = _getL2TokenConverterStorage();
        uint256 maxIssuance = $.issuances[_token][_l2Token];
        if (_amount > maxIssuance) revert MaxIssuance();

        // Freed up some issuance quota
        $.issuances[_token][_l2Token] += _amount;

        _l2Token.converterBurn(msg.sender, _amount);
        _token.safeTransfer(_recipient, _amount);

        emit Withdraw(_token, _l2Token, msg.sender, _recipient, _amount);
    }
```

PLFD-3

UNUSED INIT FUNCTION

SEVERITY: Informational

PATH:

zkevm-stb/src/PolygonBridgeBaseUpgradeable.sol:L52-61

REMEDIATION:

See description

STATUS: Acknowledged

DESCRIPTION:

In the **zkevm-stb/src/PolygonBridgeBaseUpgradeable.sol** there exists two init functions which are meant to be used in the escrow contracts or to be called from the contract **PolygonERC20BridgeBaseUpgradeable** which inherits the base contract. The two init functions are:

- **__PolygonBridgeBase_init**
- **__PolygonBridgeBase_init_unchained**

The **__PolygonBridgeBase_init** is never called in any of the contracts making it obsolete.

```
        function __PolygonBridgeBase_init(address _polygonZkEVMBridge, address
_counterpartContract, uint32 _counterpartNetwork) internal onlyInitializing {
                          __PolygonBridgeBase_init_unchained(_polygonZkEVMBridge,
_counterpartContract, _counterpartNetwork);
    }

       function __PolygonBridgeBase_init_unchained(address _polygonZkEVMBridge,
address     _counterpartContract,     uint32     _counterpartNetwork)     internal
onlyInitializing {
        PolygonBridgeBaseStorage storage $ = _getPolygonBridgeBaseStorage();
        $.polygonZkEVMBridge = IPolygonZkEVMBridge(_polygonZkEVMBridge);
        $.counterpartContract = _counterpartContract;
        $.counterpartNetwork = _counterpartNetwork;
    }
```

Either remove unused __PolygonBridgeBase_init() function or use it inside __PolygonERC20BridgeBase_init() as follows.

```
  function __PolygonERC20BridgeBase_init(address _polygonZkEVMBridge, address
  _counterpartContract, uint32 _counterpartNetwork) internal onlyInitializing
  {
      __PolygonERC20BridgeBase_init(_polygonZkEVMBridge, _counterpartContract,
  _counterpartNetwork);
  }
```

PLFD-4

UNUSED EVENTS

SEVERITY: Informational

PATH:

WETHUpgradeRouter.sol:L44

REMEDIATION:

Either emit the Deposit event or remove the declaration if it's not required.

STATUS: Fixed

DESCRIPTION:

The **WETHUpgradeRouter.sol** contract declares a **Deposit** event , but it doesn't actually emit the event.

```
event Deposit(address indexed caller, address indexed owner, uint256 assets,
uint256 shares);
```

PLFD-6

UNUSED VARIABLES

SEVERITY: Informational

PATH:

WETHUpgradeRouter.sol:L50

REMEDIATION:

Delete unused variables to reduce the deployment costs.

STATUS: Fixed

DESCRIPTION:

The **WETHUpgradeRouter** contract declares **totalAssets** variable, but it is never used.

```
uint256 public constant totalAssets = 0;
```

PLFD-8

REDUNDANT USAGE OF SAFEERC20

SEVERITY: Informational

PATH:

L2TokenConverter.sol;L23, L2Escrow.sol:L22

REMEDIATION:

Delete the redundant line of code associated with the unused SafeERC20 library.

STATUS: Fixed

DESCRIPTION:

The **L2TokenConverter** and **L2Escrow** contracts both declare the usage of the SafeERC20 library for the **IL2Token** interface. However, none of the functions from the **SafeERC20** library are actually used, making the declaration obsolete.

```
using SafeERC20 for IL2Token;
```

PLFD-9

MISSING ADDRESS ZERO CHECK

SEVERITY: Informational

PATH:

PolygonERC20BridgeBaseUpgradeable.sol:bridgeToken():L44-54

REMEDIATION:

Consider adding an address zero check.

STATUS: Fixed

DESCRIPTION:

The bridgeToken function in the PolygonERC20BridgeBaseUpgradeable contract lacks zero address validation for the destinationAddress parameter. For the L2Token, PolygonZkEVMBridge.claimMessage() call will revert since minting to a 0 address is not allowed.

```
function bridgeToken(address destinationAddress, uint256 amount, bool
forceUpdateGlobalExitRoot) external {
        _receiveTokens(amount);

        // Encode message data
        bytes memory messageData = abi.encode(destinationAddress, amount);

        // Send message data through the bridge
        _bridgeMessage(messageData, forceUpdateGlobalExitRoot);

        emit BridgeTokens(destinationAddress, amount);
    }
```


hexens
×
polygon