hexens × polygon

Aug.23

SECURITY REVIEW REPORT FOR POLYGON TECHNOLOGY

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.

AUDIT LED BY

VAHE KARAPETYAN

Co-founder / CTO | Hexens

Audit Starting Date
07.08.2023

Audit Completion Date
15.08.2023

METHODOLOGY

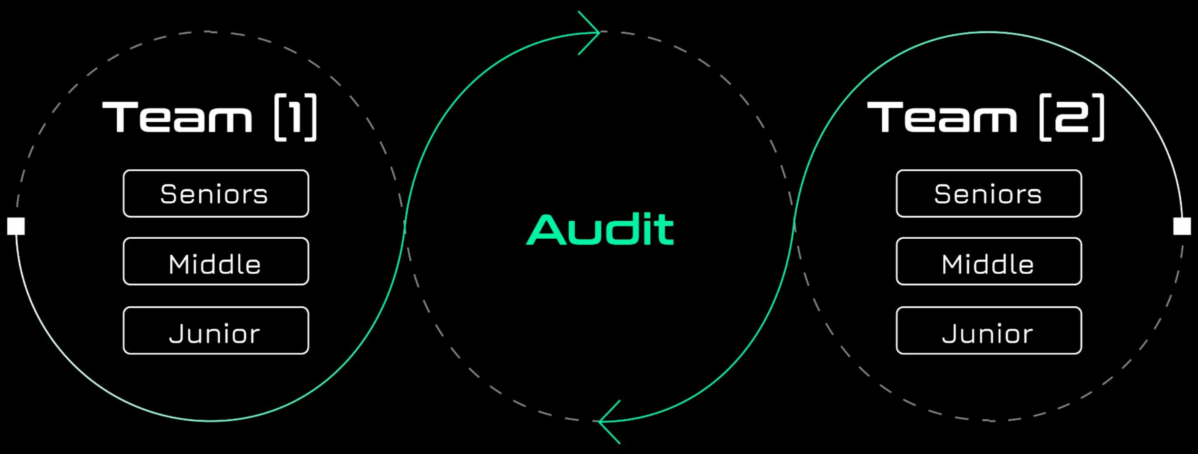
COMMON AUDIT PROCESS

Companies often assign just one engineer to one security assessment with no specified level. Despite the possible impeccable skills of the assigned engineer, it carries risks of the human factor that can affect the product's lifecycle.

Auditor*
Audit

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

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 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.

It's important to consider all types of vulnerabilities, including informational ones, when assessing the security of the project. A comprehensive security audit should consider all types of vulnerabilities to ensure the highest level of security and reliability.

EXECUTIVE SUMMARY

OVERVIEW

This audit covered AAVE's and ZKEVM-DAI's bridge integrations with Polygon zkEVM.

Our security assessment was a full review of these smart contracts.

During our audit no high impact vulnerabilities were found, nevertheless we have identified several low severity 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 completion of our audit.

SCOPE

The analyzed resources are located on:
https://github.com/aave/governance-crosschain-bridges/pull/78
https://github.com/bgd-labs/aave-helpers/pull/123
https://github.com/pyk/zkevm-dai/commit/5406d716537bf5b57ca94
61214232d91c6c02703

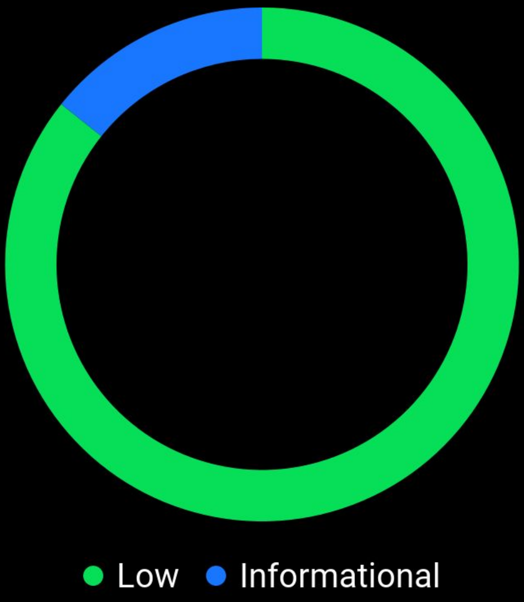
The issues described in this report were fixed in the following commit:
https://github.com/pyk/zkevm-dai/commit/ec405b5bb50fc6d6c4fb
98052a5b6a94cfe6ec52

SUMMARY

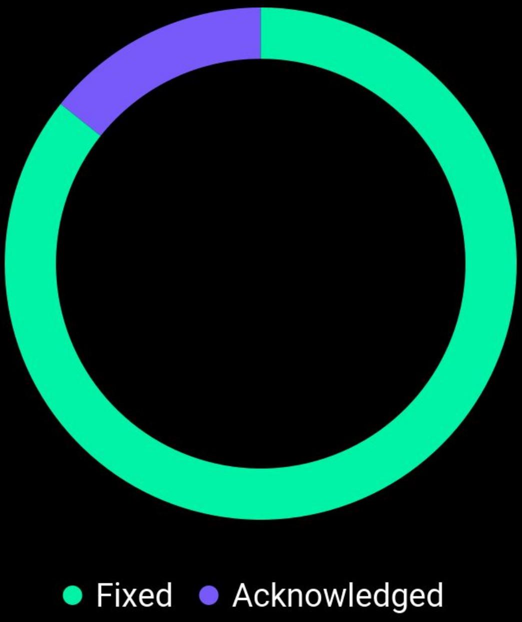
SEVERITY	NUMBER OF FINDINGS
CRITICAL	0
HIGH	0
MEDIUM	0
LOW	6
INFORMATIONAL	1

TOTAL: 7

SEVERITY



STATUS



WEAKNESSES

This section contains the list of discovered weaknesses.

POLAU-10. LACK OF AUTOMATIC REBALANCING IN L1ESCROW

SEVERITY: Low

PATH: L1Escrow.sol

REMEDIATION: recommend either incorporating rebalancing logic within the onMessageReceived function, or withdrawing (partially/fully) the non-rebalanced DAI

STATUS: acknowledged

DESCRIPTION:

The **L1Escrow** contract lacks automatic rebalancing logic in scenarios where the available balances of **dai** and **sdai** are insufficient to cover withdrawals triggered by the **onMessageReceived** function. This can result in asset management inefficiency.

When the **onMessageReceived** function is triggered and requires a withdrawal of funds, the contract should automatically perform rebalancing actions to ensure that the required funds are available for withdrawal, otherwise the non-rebalanced DAI can be used for the withdrawal (as the token depositing is done via try/catch construct there is possibility for DAI token to be present on the contract).

```
function onMessageReceived(
  address originAddress,
  uint32 originNetwork,
  bytes memory metadata
) external payable virtual {
    if (msg.sender != address(zkEvmBridge)) revert MessageInvalid();
  if (originAddress != destAddress) revert MessageInvalid();
  if (originNetwork != destId) revert MessageInvalid();

  (address recipient, uint256 amount) =
    abi.decode(metadata, (address, uint256));

  uint256 sdaiBalance = IERC20(address(sdai)).balanceOf(address(this));
  uint256 savingsBalance = sdai.previewRedeem(sdaiBalance);
  if (amount > savingsBalance) {
    sdai.withdraw(savingsBalance, recipient, address(this));
    dai.transfer(recipient, amount - savingsBalance);
  } else {
    sdai.withdraw(amount, recipient, address(this));
  }

  totalBridgedDAI -= amount;
  emit DAIClaimed(recipient, amount, totalBridgedDAI);
}
```

POLAU-6. MISMANAGEMENT OF THE ADMIN_ROLE ROLE

SEVERITY: Low

PATH: L1Escrow.sol

REMEDIATION: use AccessControlDefaultAdminRulesUpgradeable instead of AccessControlUpgradeable

STATUS: fixed

DESCRIPTION:

The role **ADMIN_ROLE** is crucial for the **L1Escrow.sol** contract, in the event it accidentally renounced, revoked, or assigned to a wrong address, it becomes impossible to upgrade the contract.

Precaution measures should be taken to avoid such situations, including: enforcing 2-step process to transfer the role; not allowing to revoke/renounce the role, or allowing to revoke/renounce the role but with a delay and possibility to cancel the revocation/renouncement before the delay ends.

```
contract L1Escrow is Initializable, UUPSUpgradeable, AccessControlUpgradeable {
    ...
    function _authorizeUpgrade(address v) internal override onlyRole(ADMIN_ROLE) {}
    ...
}
```

POLAU-7. INCONSISTENT USAGE OF SAFEERC20

SEVERITY: Low

PATH: L1Escrow.sol

REMEDIATION: use safeTransfer() instead of transfer()

STATUS: fixed

DESCRIPTION:

SafeERC20 library is used inconsistently. In some cases the **safeApprove()** and **safeTransferFrom()** functions are used with DAI token (L146, L153, L155 of **L1Escrow.sol**). In other cases plain **transfer()** function is used instead (L181, L248 of **L1Escrow.sol**) and its return value isn't checked.

```
contract L1Escrow is Initializable, UUPSUpgradeable, AccessControlUpgradeable {
 using SafeERC20 for IERC20;

 ...

 function bridgeToken(
   address recipient,
   uint256 amount,
   bool forceUpdateGlobalExitRoot
 ) external virtual {
   ...
>>  dai.safeTransferFrom(msg.sender, address(this), amount);
   ...
>>  dai.safeApprove(address(sdai), amount);
   try sdai.deposit(amount, address(this)) returns (uint256) {} catch {}
>>  dai.safeApprove(address(sdai), 0);
   ...
 }

 function onMessageReceived(
   address originAddress,
   uint32 originNetwork,
   bytes memory metadata
 ) external payable virtual {
   ...
   if (amount > savingsBalance) {
     sdai.withdraw(savingsBalance, recipient, address(this));
>>    dai.transfer(recipient, amount - savingsBalance);
   } else {
     sdai.withdraw(amount, recipient, address(this));
   }

   totalBridgedDAI -= amount;
   emit DAIClaimed(recipient, amount, totalBridgedDAI);
 }
}
```

POLAU-5. REDUNDANT APPROVE

SEVERITY: Low

PATH: L1Escrow.sol

REMEDIATION: eliminate the redundant call to reduce the cost of executing the function

STATUS: fixed

DESCRIPTION:

The purpose of the **rebalance()** function is to ensure the equilibrium of the total locked **DAI** within the **L1Escrow** contract.

Upon execution, the **rebalance()** function adds **DAI** tokens to the **SDAI** contract. Following this deposit, the **rebalance()** function redundantly invokes the **sdai.approve()** function with a value of 0. This redundant call is unnecessary, as the **sdai.deposit()** function already employs the **transferFrom()** mechanism, which inherently diminishes the allowance by the amount being transferred.

```
function rebalance() public {
 uint256 balance = IERC20(address(dai)).balanceOf(address(this));
 if (balance > totalProtocolDAI) {
  uint256 targetDepositAmount = balance - totalProtocolDAI;
  if (targetDepositAmount > 0.05 ether) {
   // Leave smol amount of DAI in this smart contract
   uint256 depositAmount = targetDepositAmount - 0.01 ether;
   dai.safeApprove(address(sdai), depositAmount);
   sdai.deposit(depositAmount, address(this));
   dai.safeApprove(address(sdai), 0);
   emit AssetRebalanced();
  }
 } else {
  uint256 sdaiBalance = IERC20(address(sdai)).balanceOf(address(this));
  uint256 savingsBalance = sdai.previewRedeem(sdaiBalance);
  uint256 withdrawAmount = totalProtocolDAI - balance;
  if (withdrawAmount > 0 && savingsBalance > withdrawAmount) {
   sdai.withdraw(withdrawAmount, address(this), address(this));
   emit AssetRebalanced();
  }
 }
}
```

POLAU-9. MISMANAGEMENT OF THE CONTRACT OWNERSHIP

SEVERITY: Low

PATH: L2Dai.sol

REMEDIATION: use Ownable2StepUpgradeable instead of OwnableUpgradeable and override renounceOwnership() with implementation that reverts

STATUS: fixed

DESCRIPTION:

The owner account is crucial for the **L2Dai.sol** contract, in the event it accidentally renounced, or transferred to a wrong address, it becomes impossible to upgrade the contract.

Precaution measures should be taken to avoid such situations, including: enforcing 2-step process to transfer the ownership; not allowing to renounce the ownership, or allowing to renounce it but with a delay and possibility to cancel the renouncement before the delay ends.

```
contract L2Dai is
  Initializable,
  UUPSUpgradeable,
  OwnableUpgradeable,
  ERC20Upgradeable
{
    ...
    function _authorizeUpgrade(address v) internal override onlyOwner {}
    ...
}
```

POLAU-8. ADD DEFAULT CONSTRUCTOR THAT CALLS _DISABLEINITIALIZERS()

SEVERITY: Low

PATH: L1Escrow.sol, L2Dai.sol

REMEDIATION: see description

STATUS: fixed

DESCRIPTION:

Malicious actor can initialize implementation contract of a proxy by calling **initialize()**. If the implementation contract makes a controlled delegatecall or performs self-destruct, it's possible to delete the implementation.

Constructor of the implementation contract should call **_disableInitializers()** function.

```
contract L1Escrow is Initializable, UUPSUpgradeable, AccessControlUpgradeable {...}
```

POLAU-11. SAFEAPPROVE WITH 0 AMOUNT SHOULD BE INSIDE THE CATCH BLOCK

SEVERITY: Informational

PATH: L1Escrow.sol

REMEDIATION: put dai.safeApprove() (L155) with 0 amount inside catch

STATUS: fixed

DESCRIPTION:

Calling **dai.safeApprove()** (L155, **bridgeToken()** function) with **0** amount is necessary only in case **sdai.deposit()** (L154) reverts. In a positive scenario **sdai.deposit()** calls **transferFrom()** internally that adjusts the allowance.

```
function bridgeToken(
    address recipient,
    uint256 amount,
    bool forceUpdateGlobalExitRoot
  ) external virtual {
    ...
    dai.safeApprove(address(sdai), amount);
    try sdai.deposit(amount, address(this)) returns (uint256) {} catch {}
    dai.safeApprove(address(sdai), 0);
    ...
  }
```