hexens × WINTERMUTE

SEPT.24

SECURITY REVIEW REPORT FOR WINTERMUTE

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/WintermuteResearch/Outcome-Market

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

AUDITING DETAILS

STARTED
18.09.2024

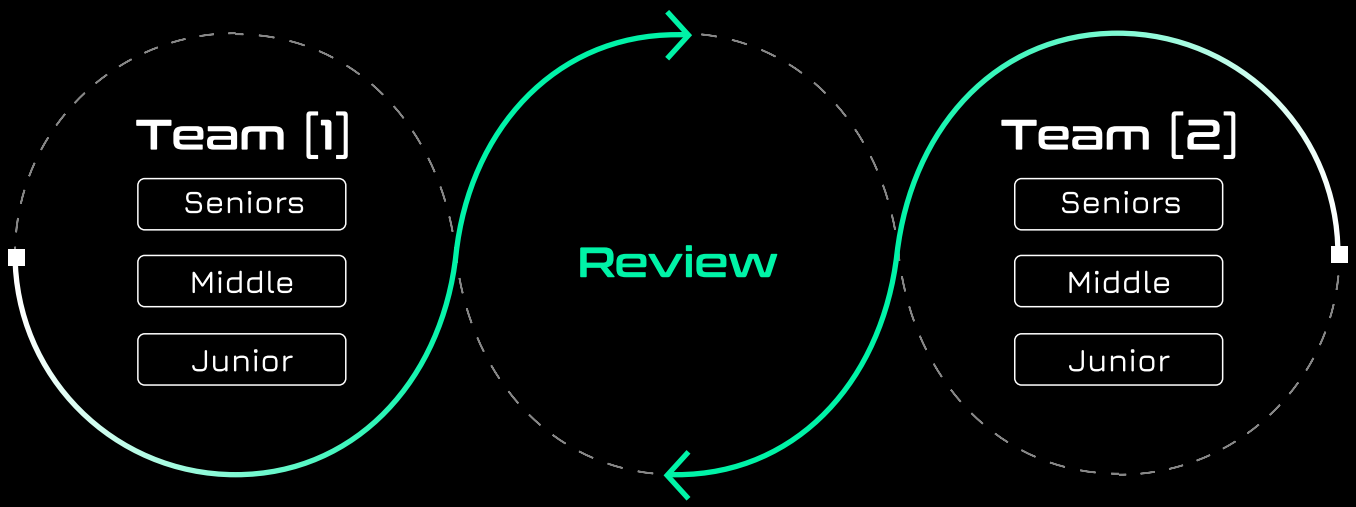
DELIVERED
20.09.2024

Review Led by

KASPER ZWIJSEN
Head of Audits | 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.

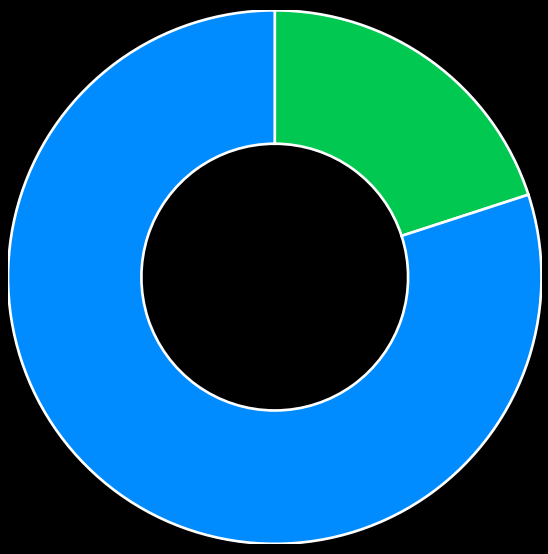
ISSUE SYMBOLIC CODES

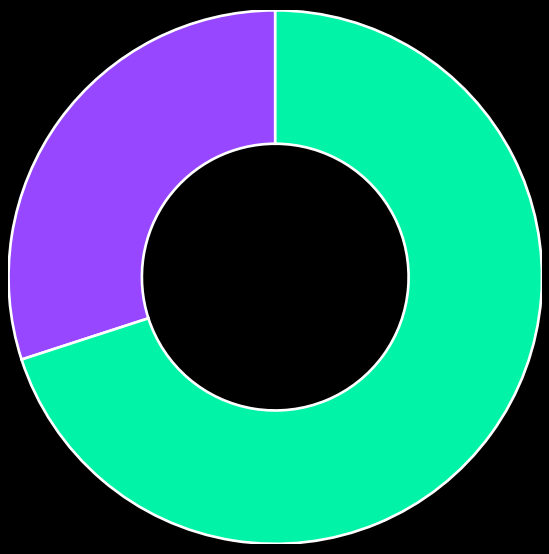
Every issue being identified and validated has its unique symbolic code assigned to the issue at the security research stage. Cause of the vulnerability reporting flow design, some of the rejected issues could be missing.

FINDINGS SUMMARY

Severity	Number of Findings
Critical	0
High	0
Medium	0
Low	2
Informational	8

Total: 10





WEAKNESSES

This section contains the list of discovered weaknesses.

WMUT-2

OUTCOME MARKET RESOLVE CAN BE FRONTRUN TO ARBITRAGE LEFTOVER VALUE

SEVERITY: Low

REMEDIATION:

See description.

STATUS: Acknowledged, see commentary

DESCRIPTION:

The **onlyBeforeResolution** in the Outcome Market only checks the local **winningOutcomeToken** variable, which only gets set after the **resolve()** function has been called.

However, at some point, the ElectionOracle will already have been resolved and set to the winning answer, while the OutcomeMarket has not yet been resolved.

Since at that point the final answer is known, **resolve()** can be called permissionlessly, an attacker can:

- Wait for the oracle to receive the final answer, after which the result would be a 100% for one candidate.
- Mint a large amount and swap to the winning candidate on the open market (depends on the leftover value).
- Call **resolve** on the OutcomeMarket and consequently call **redeem** to get guaranteed profits in collateral tokens.

```
modifier onlyBeforeResolution() {
    if (address(winningOutcomeToken) != address(0)) {
        revert OutcomeMarket__MarketResolved();
    }
    _;
}
```

Use the Oracle's **isElectionFinalized** in the resolution modifiers and change the requirements for the **resolve** function:

```
modifier onlyBeforeResolution() {
                if    (address(winningOutcomeToken)    !=    address(0)    ||
oracle.isElectionFinalized()) {
        revert OutcomeMarket__MarketResolved();
    }
    _;
}

function resolve() external {
    if (address(winningOutcomeToken) != address(0)) {
        revert OutcomeMarket__MarketResolved();
    }
    if (!oracle.isElectionFinalized()) {
        revert OutcomeMarket__MarketNotResolved();
    }
    [..]
}
```

Commentary from the client:

" - Acknowledged and was the part of initial assumptions. Due to the structure of this market, the outcome will be known some time even before it is finalized in the oracle. We assume that all on-chain liquidity will be removed/drained by MEV bots."

WMUT-11

VARIABLES CAN BE MARKED AS IMMUTABLE

SEVERITY: Low

PATH:

src/ElectionOracle.sol::minEndOfElectionTimestamp#L17

REMEDIATION:

See description.

STATUS: Fixed

DESCRIPTION:

minEndOfElectionTimestamp variable is only set in the constructor and it can therefore be marked as **immutable**.

If a variable is **immutable** it will become a constant and part of the byte code upon creation of the contract. Any subsequent read will cost almost no gas, compared to an **sload** each time, saving a lot of gas.

```
    uint256 public minEndOfElectionTimestamp;
```

WMUT-1

USE CUSTOM ERRORS

SEVERITY: Informational

REMEDIATION:

See description.

STATUS: Acknowledged, see commentary

DESCRIPTION:

Custom Errors, available from Solidity compiler version 0.8.4, provide benefits such as smaller contract size, improved gas efficiency, and better protocol interoperability. Replace require statements with Custom Errors for a more streamlined and user-friendly experience. Furthermore, custom errors are much clearer as they allow for parameter values, making debugging much easier.

For example:

```
require(block.timestamp >= minEndOfElectionTimestamp, "Cannot finalize before the end of the election period.");
```

Replace require statements with custom errors.

For example:

```
require(X == Y, "reason");
```

becomes

```
error XnotY(uint, uint);

if (X != Y)
  revert XnotY(X, Y);
```

Commentary from the client:

“ - Acknowledged. Chaos Labs prefer to use more human-readable errors.”

USE INTERNAL FUNCTIONS TO MANAGE ROLES

SEVERITY: Informational

REMEDIATION:

We recommend replacing the public function calls with internal ones (_grantRole, _revokeRole) to avoid duplication.

STATUS: Fixed

DESCRIPTION:

The roles managing functions in **ElectionOracle** use public calls to the **AccessControl** library (**grantRole**, **revokeRole**), which is unnecessary in this case as the **onlyOwner** modifier will verify that the caller has access to the function.

```
function transferOwnership(address newOwner) external onlyOwner {
    require(newOwner != address(0), "New owner address cannot be the zero
address");

    grantRole(DEFAULT_ADMIN_ROLE, newOwner);
    revokeRole(DEFAULT_ADMIN_ROLE, owner);

    emit OwnershipTransferred(owner, newOwner);
    owner = newOwner;
}

function grantOracleRole(address account) external onlyOwner {
    grantRole(ORACLE_ROLE, account);
}

function revokeOracleRole(address account) external onlyOwner {
    revokeRole(ORACLE_ROLE, account);
}
```

WMUT-4

CONSTANT VARIABLES SHOULD BE MARKED AS PRIVATE

SEVERITY: Informational

PATH:

src/OutcomeMarket.sol
src/OutcomeERC20.sol

REMEDIATION:

The mentioned variables should be marked as private instead of public.

STATUS: Partially fixed

DESCRIPTION:

In the mentioned contracts, there are **constant** and **immutable** variables that are declared **public**. However, setting **constants** and **immutables** to **private** will save deployment gas. This is because the compiler won't have to create non-payable getter functions for deployment calldata, won't need to store the bytes of the value outside of where it's used, and won't add another entry to the method ID table. If necessary, the values can still be read from the verified contract source code:

```
    address public constant OTHER_WINNER = address(0xdEaD);

    uint256 public constant COLLATERAL_TOKEN_DECIMAL_DIFF = 1e12;
```

Commentary from the client:

“ - Fixed for one case, and acknowledged for other ones to not break tests and make it more difficult to extract data. The deployments costs are not concern, and decreasing runtime costs is not enough to justify worsening user/dev experience.”

WMUT-5

SINGLE-STEP OWNERSHIP CAN INTRODUCE POTENTIAL RISKS

SEVERITY: Informational

PATH:

src/ElectionOracle.sol:L70-78

REMEDIATION:

Implement two step ownership transfer.

STATUS: Fixed

DESCRIPTION:

The **transferOwnership** in the **ElectionOracle.sol** contract is used to transfer the ownership of the contract to the new owner, however the owner is being transferred instantly without the new owner verifying their new ownership.

src/ElectionOracle.sol:L70-78

```
    function transferOwnership(address newOwner) external onlyOwner {
          require(newOwner != address(0), "New owner address cannot be the
zero address");

        grantRole(DEFAULT_ADMIN_ROLE, newOwner);
        revokeRole(DEFAULT_ADMIN_ROLE, owner);

        emit OwnershipTransferred(owner, newOwner);
        owner = newOwner;
    }
```

This can potentially introduce the risk of accidentally transferring the owner role to a non existing address or to another person if enough care is not taken when transferring the owner role.

WMUT-6

REDUNDANT INITIALIZATION

SEVERITY: Informational

PATH:

src/ElectionOracle.sol#L41-42

REMEDIATION:

There is no need to initialize result and isResultFinalized.

STATUS: Fixed

DESCRIPTION:

Declared variables are by default initialized to their default value.

```
result = ElectionResult.NotSet;
isResultFinalized = false;
```

WMUT-8

IMPLEMENT PAUSING MECHANISM IN OUTCOMEMARKET CONTRACT

SEVERITY: Informational

REMEDIATION:

Incorporate a pausing mechanism using the Pausable.sol contract from OpenZeppelin, with the ability to pause the key functions (mint, redeem, resolve).

STATUS: Acknowledged, see commentary

DESCRIPTION:

The **OutcomeMarket** contract manages a prediction market where participants can **mint** and **redeem** outcome tokens based on election results. Given the potential risks of unexpected behavior, vulnerabilities, or external dependency failures (such as oracle malfunctions), it is recommended to implement a pausing mechanism. This mechanism will allow the contract owner to temporarily halt key functions (minting, redeeming, and resolving the market) in the event of emergencies.

Commentary from the client:

" - Acknowledged. The set of OutcomeMarket contracts should be permissionless after the deployment."

WMUT-12

THE ENUM ELECTIONRESULT IS DEFINED TWICE

SEVERITY: Informational

PATH:

IElectionOracle.solL#4-L9

ElectionOracle.so#L7-L12

REMEDIATION:

Consider adjusting ElectionOracle.sol to import the enum ElectionResult from IElectionOracle.sol, and also removing the duplicate enum definition on line 7-12 from ElectionOracle.sol.

STATUS: Fixed

DESCRIPTION:

The enum **ElectionResult** is defined twice in the codebase inside IElectionOracle.sol (line 4-9) and ElectionOracle.sol (line 7-12). This may make it harder to maintain and may introduce future issues in the case where the enum is changed only in one of the two locations.

```
enum ElectionResult {
    NotSet,  // Initial/default value when the election result has not been
set yet
    Trump,   // Election result for candidate Trump
    Harris,  // Election result for candidate Harris
    Other    // Election result for any other candidate
}
```

WMUT-13

WRONG COMMENT IN IOUTCOMEMARKET.SOL SUGGESTS THAT EITHER KAMALA OR HARRIS TOKENS ARE MINTED

SEVERITY: Informational

REMEDIATION:

Consider adjusting the comment for the mint() function:

// IOutcomeMarket.sol (line 6):

/// @notice Takes USDC and mints both TRUMP and HARRIS for each unit

STATUS: Fixed

DESCRIPTION:

There is a wrong comment in IOutcomeMarket.sol line 6 for the **mint()** function, which suggests that either Kamala or Harris tokens are minted for the USDC received.

IOutcomeMarket.sol

```
/// @notice Takes USDC and mints both KAMALA and HARRIS for each unit
/// @param usdcAmount The USDC amount for conversion to conditional tokens
function mint(uint256 usdcAmount) external;
```


hexens
x
WINTERMUTE