hexens × PancakeSwap

AUG.24

SECURITY REVIEW REPORT FOR PANCAKESWAP

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/pancakeswap/pancake-v4-periphery/tree/4d97f3a33a1aa5eabd7d1c4f843b528bf342cc5e

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

https://github.com/pancakeswap/pancake-v4-periphery/commit/5a904ecdd6b445fc88d600bcf659209f534ad589

https://github.com/pancakeswap/pancake-v4-periphery/commit/7ca22462252ba93f86a4a1bd702c5548ea3cc330

https://github.com/pancakeswap/pancake-v4-periphery/commit/50e30cb7eb536d6f586d917fd061f9861c6d83c2

AUDITING DETAILS

STARTED
19.08.2024

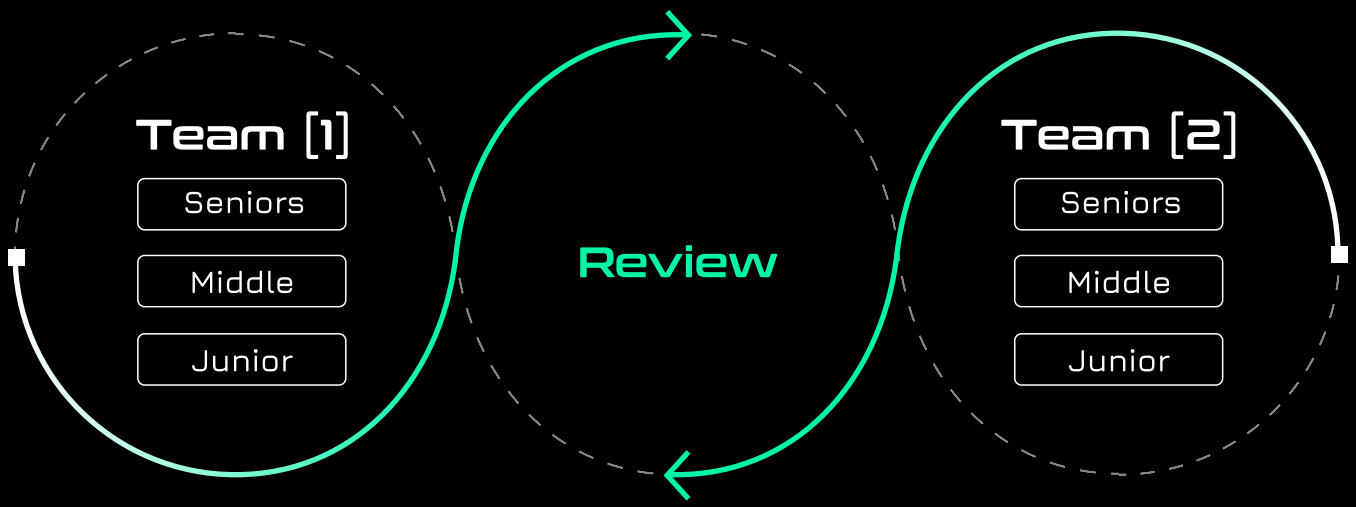
DELIVERED
04.09.2024

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.

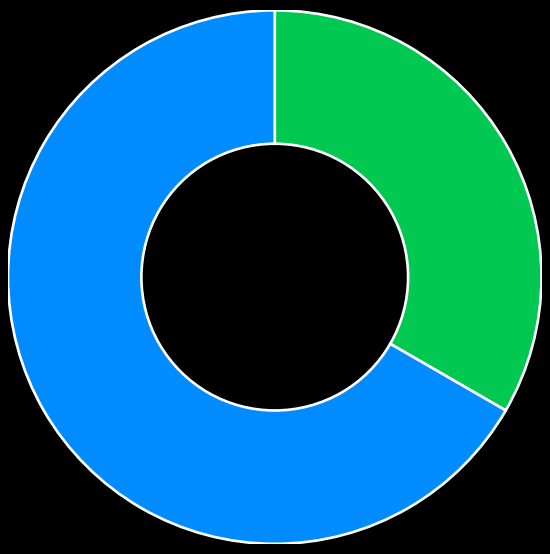
ISSUE SYMBOLIC CODES

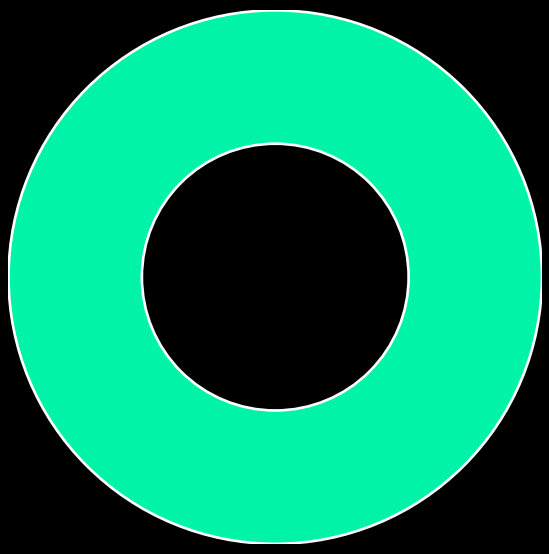
Every issue being identified and validated has its unique symbolic code assigned to the issue at the security research stage. Cause of the vulnerability reporting flow design, some of the rejected issues could be missing.

FINDINGS SUMMARY

Severity	Number of Findings
Critical	0
High	0
Medium	0
Low	1
Informational	2

Total: 3





WEAKNESSES

This section contains the list of discovered weaknesses.

CAKE2-1

FRONT RUNNING THE PERMIT FUNCTION CAN TRIGGER A DOS ATTACK

SEVERITY: Low

PATH:

src/base/SelfPermitERC721.sol#L15-L21

src/base/Permit2Forwarder.sol#L26-L31

src/base/Permit2Forwarder.sol#L17-L22

REMEDIATION:

Consider using the trustlessPermit() function, or ensure the contract does not revert when the permit() call fails.

STATUS: Fixed

DESCRIPTION:

The **Permit2Forwarder** and **SelfPermitERC721** contracts integrate the EIP-2612 Permit function, It transfers the burden of holding native (gas) tokens away from users, by allowing them to sign an approval off-chain and send it to a trusted service, which could use the funds as if the user called approve(). Because transactions are in the mempool and accessible to anyone, an attacker can extract the **_signature** parameters from the current call and front-run it with a direct **permit()** transaction. In this case, the result is harmful, as the user loses the functionality that follows the **permit()**.

For example, all functions that use the **permit** functionality for gasless transactions can be blocked, preventing users from using this feature within the project.

```
        function selfPermitERC721(address token, uint256 tokenId, uint256
deadline, uint8 v, bytes32 r, bytes32 s)
        public
        payable
        override
    {
         IERC721Permit(token).permit(address(this), tokenId, deadline, v, r,
s);
    }
```

```
    /// @notice allows forwarding batch permits to permit2
     /// @dev this function is payable to allow multicall with NATIVE based
actions
        function permitBatch(address owner, IAllowanceTransfer.PermitBatch
calldata _permitBatch, bytes calldata signature)
        external
        payable
    {
        permit2.permit(owner, _permitBatch, signature);
    }
```

```
    /// @notice allows forwarding a single permit to permit2
     /// @dev this function is payable to allow multicall with NATIVE based
actions
     function permit(address owner, IAllowanceTransfer.PermitSingle calldata
permitSingle, bytes calldata signature)
        external
        payable
    {
        permit2.permit(owner, permitSingle, signature);
    }
```

UNUSED IMPORTS

SEVERITY: Informational

PATH:

src/V4Router.sol::SafeCastTemp#L16

src/V4Router.sol::BalanceDelta#L7

src/V4Router.sol::PoolKey#L8

src/V4Router.sol::ActionConstants#17

src/pool-bin/BinMigrator.sol::SafeTransferLib#L5

src/pool-cl/CLMigrator.sol::SafeTransferLib#L5

src/pool-cl/CLMigrator.sol::Currency#L6

src/pool-cl/CLMigrator.sol::PoolKey#L8

REMEDIATION:

Remove unused imports.

STATUS: Fixed

DESCRIPTION:

The **V4Router.sol** contract imports **./libraries/SafeCast.sol**, but does not use it anywhere, moreover, the **SafeCastTemp** is already imported in the **BinRouterBase.sol** and **CLRouterBase.sol** contracts, from which the **V4Router.sol** inherits from.

There are also a couple of not-used imports, which are mentioned in the path.

```
import {SafeCastTemp} from "./libraries/SafeCast.sol";
```

```
import {BalanceDelta} from "pancake-v4-core/src/types/BalanceDelta.sol";
import {PoolKey} from "pancake-v4-core/src/types/PoolKey.sol";
```

```
import {ActionConstants} from "./libraries/ActionConstants.sol";
```

```
import {SafeTransferLib} from "solmate/src/utils/SafeTransferLib.sol";
```

```
import {SafeTransferLib} from "solmate/src/utils/SafeTransferLib.sol";
import {Currency} from "pancake-v4-core/src/types/Currency.sol";
```

```
import {PoolKey} from "pancake-v4-core/src/types/PoolKey.sol";
```

REDUNDANT ASSIGNMENT LIBRARY TO TYPE

SEVERITY: Informational

PATH:

src/types/Currency.sol#L10

REMEDIATION:

Remove redundant assignments of CurrencyLibrary wherever they occur.

STATUS: Fixed

DESCRIPTION:

The **CurrencyLibrary** library is assigned to the **Currency** type in multiple contracts, but in the **Currency.sol** contract, this library is already assigned globally to the **Currency** type.

```
using CurrencyLibrary for Currency global;
```


hexens
×
PancakeSwap