Uniswap V4 Periphery Security Review

Pashov Audit Group

Conducted by: defsec, dirk_y, Shaka

October 11th - October 16th

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **timeless-fi/bunni-v2** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Uniswap V4 Periphery

Uniswap v4 retains the capital efficiency improvements of Uniswap v3, while introducing flexibility through hooks and optimizing gas usage throughout the entire process. SVG contract in periphery contracts defines a library called NFTSVG, which provides functions for generating SVG images used in Uniswap NFTs, combining customizable graphical elements like curves, colors, and positions based on various parameters such as token IDs, price ranges, and token symbols.

5. Risk Classification

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash - 7faae4718eecda1b33dc3abd894431ed2d16c929

fixes review commit hash - 1a21920085fc712ca745361bf397e8a7be25dc1c

Scope

The following smart contracts were in scope of the audit:

- `PositionDescriptor`
- `PositionManager`
- `ERC721Permit_v4`
- `SafeCurrencyMetadata`
- `AddressStringUtils`
- `HexStrings`
- `Descriptor`
- `SVG`
- `SafeCurrencyMetadata`

7. Executive Summary

Over the course of the security review, defsec, dirk_y, Shaka engaged with Uniswap to review Uniswap V4 Periphery. In this period of time a total of **7** issues were uncovered.

Protocol Summary

Protocol Name	Uniswap V4 Periphery
Repository	https://github.com/Uniswap/v4-periphery
Date	October 11th - October 16th
Protocol Type	DEX

Findings Count

Severity	**Amount**
Medium	1
Low	6
Total Findings	**7**

Summary of Findings

ID	Title	Severity	Status
[M-01]	Hook address is not correctly represented in the SVG	Medium	Resolved
[L-01]	Hook address could be unset	Low	Resolved
[L-02]	Breaking change to permit hashes	Low	Acknowledged
[L-03]	Using zero address for native token can be confusing for end users	Low	Resolved
[L-04]	Special characters not escaped can produce invalid JSON	Low	Resolved
[L-05]	Long symbol strings can cause tokenURI to revert or artifacts in the SVG image	Low	Resolved
[L-06]	currencyDecimals does not check if the value returned is uint8	Low	Resolved

8. Findings

8.1. Medium Findings

[M-01] Hook address is not correctly represented in the SVG

Severity

Impact: Low

Likelihood: High

Description

The `generateSVGPositionDataAndLocationCurve` of the `SVG` library generates the SVG for the position data. This data includes the address of the hook contract, which is not represented in full, but only the first and last characters with an ellipsis in between.

To process the hook address, first, the `toHexString` function is called to transform the address into a string, so the `hookStr` variable is 42 characters long and the range of its bytes indexes is from 0 to 41.

Then the sliced address is generated by concatenating the first 5 characters, the ellipsis, and the last 3 characters of the `hookStr` variable. However, the values passed to the `substring` function are incorrect for the last characters, as `endIndex` should be 42 instead of 40.

```
function generateSVGPositionDataAndLocationCurve(
        string memory tokenId,
        address hook,
        int24 tickLower,
        int24 tickUpper
    ) private pure returns (string memory svg) {
@>      string memory hookStr = (uint256(uint160(hook))).toHexString(20);
        string memory tickLowerStr = tickToString(tickLower);
        string memory tickUpperStr = tickToString(tickUpper);
        uint256 str1length = bytes(tokenId).length + 4;
@>      string memory hookSlice = string(abi.encodePacked(substring
  (hookStr, 0, 5), "...", substring(hookStr, 37, 40)));
```

This means that if the hook address is `0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48`, the SVG will show the following text:

```
Hook: 0xa0b...6eb
```

This will result in users not being able to identify the hook contract.

Recommendations

```
-     string memory hookSlice = string(abi.encodePacked(substring
- (hookStr, 0, 5), "...", substring(hookStr, 37, 40)));
+     string memory hookSlice = string(abi.encodePacked(substring
+ (hookStr, 0, 5), "...", substring(hookStr, 39, 42)));
```

8.2. Low Findings

[L-01] Hook address could be unset

In `Descriptor.sol` and `SVG.sol` it appears that there is the expectation that every position will have an associated hook contract, however as stated in this page, hooks are optional for liquidity pools, so positions may not have an associated hook.

For example, the SVGs that are generated in `SVG.sol` could be confusing to users who have positions in liquidity pools without a hook contract. In the currently generated SVGs there will be sections that look something like `Hook: 0x0000...0000`. To the uneducated user, it might look like the SVG was generated incorrectly or that the hook is indeed at the displayed address.

It would be better if there was conditional logic for when a position does not have an associated hook. Some possibilities include:

- Generating SVGs (or descriptors) without hook-specific sections if the relevant liquidity pool does not use a hook contract
- Instead of displaying `0x0000...00000` display a string like `No hook`

Of course, it might be easier/better to stick with the 0 address for some components like URIs that need to be confirmed to a specific, expected format. However, for user-visible content, it would be better to be more transparent/explicit.

[L-02] Breaking change to permit hashes

Although v4 has not yet been deployed to mainnet, it has been deployed to testnet, so it is worth calling out that the small change to the name string in `PositionManager.sol` from `V4` to `v4` will break all existing permits that have been signed but not been used yet. This may impact any integrators that are testing with v4.

If this change is understood and communicated to any downstream testnet integrators then this change can persist.

[L-03] Using zero address for native token can be confusing for end users

Internally, Uniswap v4 uses the zero address to represent the native token of the chain. This means that `tokenURI` for NFTs of pools using the native token will contain the following text in their metadata's "description" field:

```
ETH Address: 0x0000000000000000000000000000000000000000
```

In the same way, the SVG image for the NFT will contain the following text on the border of the image:

```
0x0000000000000000000000000000000000000000 • ETH
```

This might be confusing for end users, as they might interpret the zero address as the actual address of the native token.

Consider managing the special case of the currency being the native token and not displaying the address in that case or using a different representation, like the word "Native".

[L-04] Special characters not escaped can produce invalid JSON

The `Descriptor.constructTokenURI` generates a Base64-encoded JSON that will be returned by the `PositionDescriptor.tokenURI` function. The function does not fully sanitize the input symbols, which may contain special characters that will produce an invalid JSON.

The following characters should be escaped in the same way as it is done for the double quote character:

- `\u000c` (form feed)
- `\n` (newline)
- `\r` (carriage return)
- `\t` (tab)

[L-05] Long symbol strings can cause `tokenURI` to revert or artifacts in the SVG image

The `currencySymbol` function in the `SafeCurrencyMetadata` library is used to extract the token symbol from the token contract. If the value returned by the contract is too long it can cause the following issues:

1. If the length of the symbol is greater than 255 characters, the `Descriptor.escapeQuotes` function will revert due to an overflow error, as `symbolBytes.length` will not fit in a `uint8`:

```
for (uint8 i = 0; i < symbolBytes.length; i++) {
        if (symbolBytes[i] == '"') {
            quotesCount++;
        }
    }
```

2. For lengths lower than 255 but still long, the text with the tokens data placed in the border of the SVG image will overlap and the output will be unreadable (see example).

Consider trimming the output of the `currencySymbol` function to a a sensible length to avoid these issues.

[L-06] `currencyDecimals` does not check if the value returned is uint8

In the `SafeCurrencyMetadata` the NatSpec of the `currencyDecimals` function states the following:

```
/// @notice attempts to extract the token decimals, returns 0 if not implemented
// or not a uint8
```

However, the function does not check if the value returned by the token contract is a uint8, so in case the value is greater than 255 the function will revert in the decoding phase:

```
if (data.length == 32) {
        return abi.decode(data, (uint8));
    }
```

While a decimal value greater than 255 might not be expected, as it is not possible to represent a number with more than 77 decimal places in 32 bytes, the function should avoid reverting and return the fallback value of 0 instead.

Consider adding the following changes to the code:

```diff
if (data.length == 32) {
-       return abi.decode(data, (uint8));
+       uint256 decimals = abi.decode(data, (uint256));
+       if (decimals <= type(uint8).max) {
+           return uint8(decimals);
+       }
    }
    return (false, 0);
```