Strategy B…

Try HackMD(https://hackmd.io?utm_source=view-page&utm_medium=logo-nav)

OFFBEAT SECURITY

Strategy Builder Review

May 6, 2024

Prepared for Contango Protocol

Conducted by Richie Humphrey (devtooligan)

About the Strategy Builder Review

Contango builds perps by automating a looping strategy, also known as recursive borrowing and lending. This is achieved using spot and money markets.

This review focused on the `StrategyBuilder` contract enabling a new feature which allows users to create investment strategies with Contango protocol.

About Offbeat Security

Offbeat Security Labs, LLC is a boutique blockchain security company specializing in complex and novel DeFi projects. Our mission is to elevate the blockchain security landscape through innovative, collaborative, and unconventional solutions.

Summary & Scope

The strategies (https://github.com/contango-xyz/core-v2-private/tree/557f61af27ccd69b0a71d74c2a7bf3b7de6df30b/src/strategies) folder of the `core-v2-private` repo was reviewed at commit 557f61af27ccd69b0a71d74c2a7bf3b7de6df30b

The following **3 contracts** were in scope:

- src/strategies/StrategyBuilder.sol
- src/strategies/StrategyBlocks.sol
- src/strategies/PositionPermit.sol

Contango has also requested that we document any assumptions implicit in the code which we encounter during this review.

Note: During the course of the review, the client reported that the `continueActionProcessing` was missing the `validFlashloan` modifier. As such, we did not list it below in the findings.

Summary of Findings

Identifier	Title	Severity	Fixed
H-01	An attacker can seize control of any position	High	Pull 376 (https://github.com/contango-xyz/core-v2-private/pull/376/files)
H-02	The Strategy Builder can be rendered inoperable by a donation attack	High	Pull 376 (https://github.com/contango-xyz/core-v2-private/pull/376/files)
L-01	Native tokens can be left in the contract and anyone can take them	Low	

Additional Recommendations

Consider adding stateful invariant tests. These type of tests are particularly effective in scenarios where different combinations of calls may lead to problems. The testing suite could generate strategies of random step sequences to check if any invariants can be broken. Given the planned further development of the Strategy Builder, implementing these tests would provide additional assurance of the system's correctness and help catch potential issues early in the development process.

Code Assumptions

The client has provided a list of all assumption applicable to the codebase and requested we update the list as we conduct the review.

Assumptions provided by the client:

1. Contract can only act with positionIds it owns, it doesn't have any special permissions on the Contango core contracts.

2. Positions can only be owned by the contract by either the trader sending them or by the contract pulling them via permit2, the contract can't pull positions directly.

3. Funds can only be taken by permit and permit2, the contract can't pull funds directly, so even if there're approvals in place, nobody should be able to steal someone else's money (same applies to positions).

4. Permits can only be executed if the user initiated the trasaction.

5. Risk of funds/positions being stuck in the contract are tackled by the permissioned "retrieve" methods.

6. If funds are left in the vault under the contract's name, then they can be taken by anybody.

Assumption added by us:

7. Permit users will not need to submit two permits with exact same owner, token, amount, and deadline.

8. Signatures are formed in accordance with the Ethereum Yellow Paper using the lower range of the ECDSA elliptic curve.

9. Only wrapped native token contracts (i.e. WETH) will need to send native tokens directly to the contract (with no calldata).

10. More than one flashloan will not need to be processed at a time.

Detailed Findings

High Findings

[H-01] An attacker can seize control of positions by calling `onERC721Received`

Anyone can call `onERC721Received` and provide forged calldata that can be used to control position in the system. This function is intended to be used as the callback for ERC-721 safe transfers.

```
    function onERC721Received(address operator, address from, uint256 tokenId, b
        // When's not a position creation
        if (operator != address(contango)) {
            emit BeginStrategy(fromUint(tokenId), from);
            _onPositionReceived(operator, from, tokenId, data);
            _ensureNorPositionLeftBehind();
            emit EndStrategy(fromUint(tokenId), from);
        }

        return this.onERC721Received.selector;
    }
```

The `_onPositionReceived` function is called, passing on the arguments that `onERC721Received` was called with.

```
    function _onPositionReceived(address, address from, uint256, bytes calldata
        StepCall[] memory steps = abi.decode(data, (StepCall[]));
        StackLib.Stack memory stack;
        (, stack) = _actionProcessor(steps, 0, stack, from);
        _returnPositions(stack.long, stack.short, from);
    }
```

The `_onPositionReceived` function calls `_actionProcessor` using the user provided `from` value as the from. It does not use `msg.sender` as when calling `process` . This allows an attacker to make calls on a position that they do not own.

Exploit scenario

1. Alice calls `process` and includes a step `pullFundsWithPermit` that includes a valid permit.
2. Eve front-runs this transaction, forging a malicious `steps` argument which uses Alice's permit and then swaps Alice's tokens into a low volume meme coin.
3. Alice's transaction fails as her permit is now used. The tokens she permitted have been swapped for highly illiquid tokens.

Recommendation

Add access control to the `onERC721Received` function to only allow it to be called by the position NFT contract.

[H-02] The Strategy Builder functionality can be rendered inoperable

If a `positionNft` is donated to the contract using `transferFrom` , then the StrategyBuilder will no longer be able to receive `positionNft` 's to initiate strategies. When receiving a `positionNft` the contract relies on the `onERC721Received` function which is called by default in the ERC-721 `safeTransfer` method.

```
    function onERC721Received(address operator, address from, uint256 tokenId, b
        // When's not a position creation
        if (operator != address(contango)) {
            emit BeginStrategy(fromUint(tokenId), from);
            _onPositionReceived(operator, from, tokenId, data);
            _ensureNorPositionLeftBehind();
            emit EndStrategy(fromUint(tokenId), from);
        }
        return this.onERC721Received.selector;
    }

    // ======================= Internal functions =======================

    function _ensureNorPositionLeftBehind() internal view {
        if (positionNFT.balanceOf(address(this)) > 0) revert PositionLeftBehind(
    }
```

At the end of that function there is a check `_ensureNorPositionLeftBehind()` which will revert the transaction if there are any `positionNft` 's held by the contract. Unlike `safeTransferFrom` , the `transferFrom` function in the Open Zeppelin ERC-721 implentation does not call `onERC721Received` on the recipient address. Therefore it is possible for anyone to send their `positionNft` to this contract and disable the functionality.

Recommendation

Consider removing the `_ensureNorPositionLeftBehind` function.

Low Findings

[L-01] Native tokens can be left in the contract and anyone can take them

Native tokens can be left in the contract and anyone can retrieve them. This breaks assumption e. above which states:

> Risk of funds/positions being stuck in the contract are tackled by the permissioned "retrieve" methods.

Native tokens can be sent to the payable function 'process' and if there is no step included which uses these tokens then they will remain in the contract. There is a retrieve function but it only handles ERC-20 tokens:

```
    function retrieve(IERC20 token, address to) external onlyRole(DEFAULT_ADMIN_
        token.transferBalance(to);
    }
```

Any other user could steal any native tokens held by the contract by building a strategy that calls `process` and includes two steps such as `_wrapNativeToken` which wraps tokens based on the `msg.value`. If it is added twice, then it will wrap two times the amount of `msg.value`.

Recommendation

Consider updating the `retrieve` functionality or adding a new permissioned function which can retrieve native tokens from the contract.