HMX Security Assessment

Jun 20, 2023

Prepared for

HMX

Prepared by:

0xfoobar

Table of Contents

Table of Contents 2
Summary 4
Overview 4
Project Scope 4
Summary of Findings 4
Disclaimer 5
Key Findings and Recommendations 6
1. Malicious strategy could steal all tokens 6
Description 6
Impact 6
Recommendation 6
Fix 6
2. Compromised whitelistedExecutor address could drain all tokens 7
Description 7
Impact 7
Recommendation 7
Fix 7
3. TraderLoyaltyCredit should inherit IERC20 interface 8
Description 8
Impact 8
Recommendation 8
Fix 8
4. Use of transfer to send ETH can cause reverts 9
Description 9
Impact 9
Recommendation 9
Response 9
5. Use latest compiler version and associated features 10
Description 10
Impact 10
Recommendation 10
Response 10
6. Use named mappings 11
Description 11
Impact 11
Recommendation 11
Response
HMX implemented the suggested change. 11

7. Remove unused files and rename imported files 12
Description 12
Impact 12
Recommendation 12
Response 12
8. Use custom errors instead of require strings 13
Description 13
Impact 13
Recommendation 13
Response 13
9. Correct typos and method names 14
Description 14
Impact 14
Recommendation 14
Response 14
10. Redundant function modifiers 15
Description 15
Impact 15
Recommendation 15
Response 16

Summary

Overview

HMX is a decentralized perpetual exchange with a cross-margin and multi-asset collateral support on Arbitrum.

Project Scope

We reviewed the core smart contracts and test suite contained within commit hash e20d9c23d5f6dfde1c6b6eb771a5f60ce567c735 at https://github.com/perp88/v2-evm, and commit hash 4ccf695b73148276388b3252b9e17117c03796e7 at https://github.com/perp88/hmx-tokenomics.

Summary of Findings

Severity	Findings	Resolved
High	**1**	**All Resolved**
Medium	**1**	**All Resolved**
Low	**2**	**1 Resolved, 1 Acknowledged**
Informational	**6**	**5 Resolved, 1 Acknowledged**

Disclaimer

This security assessment should not be used as investment advice.

We do not provide any guarantees on eliminating all possible security issues. foostudio recommends proceeding with several other independent audits and a public bug bounty program to ensure smart contract security.

We did not assess the following areas that were outside the scope of this engagement:

- Website frontend components
- Offchain order management infrastructure
- Multisig or EOA private key custody

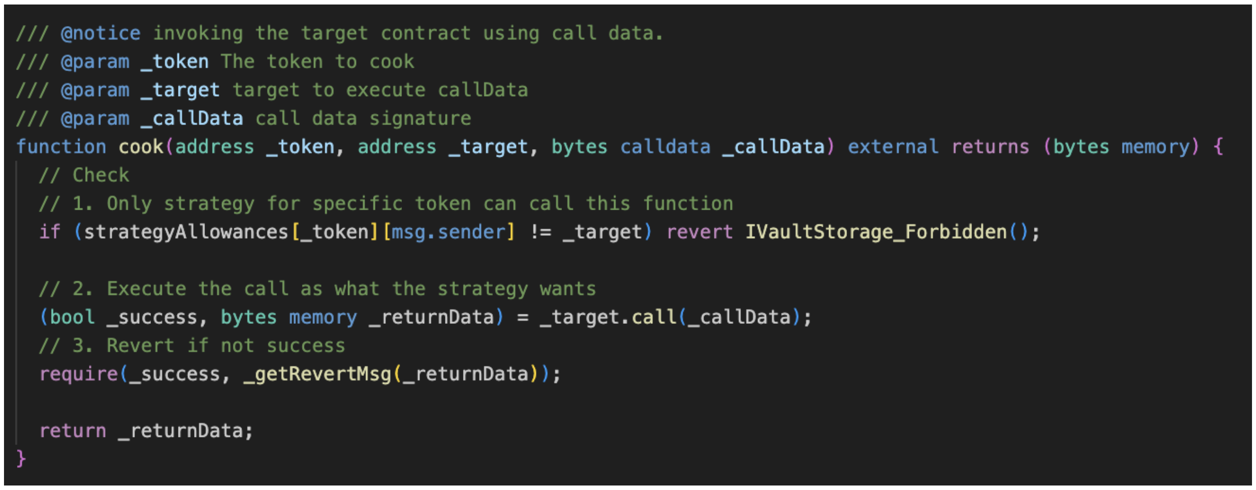
Key Findings and Recommendations

1. Malicious strategy could steal all tokens

Severity: **High**
Files: VaultStorage.sol

Description

VaultStorage is the contract that runs token accounting, and also holds the tokens. It has an access-gated method called cook(), where whitelisted strategies can pass in arbitrary calldata that the VaultStorage contract will then invoke on whitelisted target contracts. While all existing strategies are currently developed by the team, it's possible that in the future third-party strategies will exist. And if a team there is able to sneak in malicious code via an upgradeable proxy, a delegatecall, or even obfuscation of immutable logic, they would be able to drain the entire vault pool of all its tokens through a series of IERC20().transfer() calls.



```
/// @notice invoking the target contract using call data.
/// @param _token The token to cook
/// @param _target target to execute callData
/// @param _callData call data signature
function cook(address _token, address _target, bytes calldata _callData) external returns (bytes memory) {
  // Check
  // 1. Only strategy for specific token can call this function
  if (strategyAllowances[_token][msg.sender] != _target) revert IVaultStorage_Forbidden();

  // 2. Execute the call as what the strategy wants
  (bool _success, bytes memory _returnData) = _target.call(_callData);
  // 3. Revert if not success
  require(_success, _getRevertMsg(_returnData));

  return _returnData;
}
```

Impact

All tokens would be drained from the vault.

Recommendation

Define a standard interface for all strategies to use, instead of trusting arbitrary calldata. If this is not possible because of the need to maintain future flexibility, all future strategies must be carefully audited and use only immutable logic to prevent abuse.

Fix

HMX created a whitelist of function selectors that can be called, to prevent abuse.

2. Compromised whitelistedExecutor address could drain all tokens

Severity: **Medium**
Files: VaultStorage.sol

Description

VaultStorage whitelistedExecutor can rug all tokens. Other ownership, like proxy admins, can be gated behind a timelocked multisig, so less of an issue. But the whitelistedExecutor here will have to be a hot wallet pushing transactions frequently, so compromise here could be fatal. The increaseTraderBalance() function here would enable simple draining of more tokens than a user is entitled to.

Impact

All tokens could be drained from the vault if the keeper private keys are compromised.

Recommendation

Only assign other smart contracts (whose proxy logic changes are gated behind a timelocked multisig) to the VaultStorage whitelistedExecutor role. Those intermediate smart contracts can be called by a keeper EOA but they'll do important token transfer checks before increasing user balances.

Fix

HMX added a validation check requiring all whitelistedExecutors to be smart contracts not EOAs.

3. TraderLoyaltyCredit should inherit IERC20 interface

Severity: **Low**
Files: TraderLoyaltyCredit.sol

Description

This token is intended to be an ERC20, and implements the requisite methods, but should be explicitly marked as such by inheriting the proper interface. This gives compile-time checks and behavior guarantees for users.

Impact

Integrators may not use TLC as an ERC20.

Recommendation

Inherit IERC20 in the contract declaration.

Fix

HMX inherited the interface.

4. Use of transfer to send ETH can cause reverts

Severity: **Low**
Files: CrossMarginHandler.sol

Description

The protocol employs Solidity’s .transfer method to send ETH. However, .transfer is limited to a hardcoded gas amount of 2300, which can break if contracts have fallback logic. This is also potentially incompatible with future changes to Ethereum’s gas costs.

Impact

Sending ETH to contracts with fallback logic will revert.

Recommendation

Use the .call() method as described in https://solidity-by-example.org/sending-ether/.

Response

“We decided to keep the code as is. We use the specified gas limit to ensure that very little malicious code could be executed. In case, there is a failure, we also do not want denial of service. So, we will return fund in the form of wrapped native token to the users instead.”

5. Use latest compiler version and associated features

Severity: **Informational**
Files: foundry.toml, *.sol

Description

All files currently pin the solidity pragma to 0.8.18. Best practice is to use a floating pragma (prefixed with ^) targeting the latest 0.8.20 version. This will have three benefits: cheaper deployment costs, gas optimization and new language features. Since this protocol targets Arbitrum rather than Ethereum, it'll be important to explicitly set the compiler's target EVM version as "paris" rather than "shanghai" to avoid PUSH0 opcode incompatibility.

Impact

Increased deployment size and gas costs.

Recommendation

Use a floating ^0.8.20 pragma everywhere, and target the paris EVM version with `evm_version = "paris"` in foundry.toml

Response

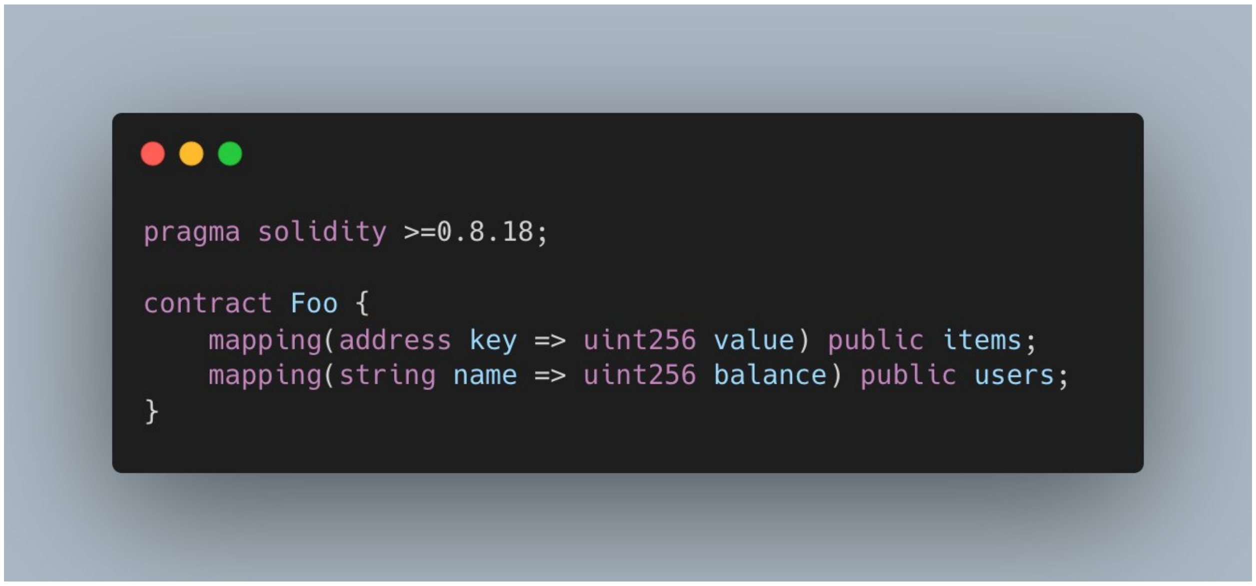
"We would like to stick with this version of compiler for consistency with our deployed contracts and tests."

6. Use named mappings

Severity: **Informational**
Files: *.sol

Description

Solidity 0.8.18 and later support a language feature called named mappings, where keys and values can be named to give devs better documentation of what's expected in each.



```
pragma solidity >=0.8.18;

contract Foo {
    mapping(address key => uint256 value) public items;
    mapping(string name => uint256 balance) public users;
}
```

Impact

Extraneous comments that could be inlined.

Recommendation

Experiment with named mappings where the storage layout otherwise requires comment explanations.

Response

HMX implemented the suggested change.

7. Remove unused files and rename imported files

Severity: **Informational**
Files: .gitkeep, interfaces/gmx/*

Description

The .gitkeep files are unnecessary now that the project is fully fleshed out, and any files from GMX used as inspiration in interfaces/gmx should be renamed to reflect the focus on HMX.

Impact

Bloated project directory layout.

Recommendation

Delete or rename these files and folders.

Response

HMX implemented the suggested change.

8. Use custom errors instead of require strings

Severity: **Informational**
Files: BaseAccount.sol, HmxAccount.sol, CrossMarginHandler.sol, LimitTradeHandler.sol, TraderLoyaltyCredit.sol

Description

Custom errors are slimmer from a deployment codesize perspective, and also use less gas on reverts. They're also better for grouping similar errors into canonical categories without the repetition of similar-but-distinct revert strings.

Impact

Lower deployment and usage gas costs.

Recommendation

Update require statements to revert with custom errors.

Response

HMX implemented the suggested change.

9. Correct typos and method names

Severity: **Informational**
Files: BaseAccount.sol, HmxAccount.sol, CrossMarginHandler.sol, LimitTradeHandler.sol, TraderLoyaltyCredit.sol

Description

VaultStorage line 33: dept -> debt
LHMX line 8, 34: transferror -> transferrer
LHMXVester line 53, Vester line 62: santy -> sanity
CrossMarginHandler line 44, 510: chuck -> check

Also recommend updating the VaultStorage method name pullToken() to be syncBalance(), reflecting that it doesn't attempt to transfer any tokens from the user but merely checks its current ones and updates its own tracking variables accordingly.

Impact

Less professional code.

Recommendation

Update with fixes.

Response

HMX implemented the suggested change.

10. Redundant function modifiers

Severity: **Informational**
Files: Compounder.sol, HLP.sol, BotHandler.sol, CrossMarginHandler.sol, LimitTradeHandler.sol, LiquidityHandler.sol, MarketTradeHandler.sol, TradeHelper.sol, VaultStorage.sol, PerpStorage.sol

Description

Compounder.removeToken() can be marked external not public, since it's never called internally. The nonReentrant modifier is only necessary when there's an untrusted external function call (such as an ERC20 token transfer or oracle call), which is not the case in most setter methods or internal accounting. The following methods can remove it:

- HLP.setMinter()
- HLP.mint()
- HLP.burn()
- BotHandler.updateLiquidityEnabled()
- BotHandler.updateDynamicEnabled()
- BotHandler.setTradeService()
- BotHandler.setPositionManagers()
- BotHandler.setLiquidationService()
- CrossMarginHandler.setCrossMarginService()
- CrossMarginHandler.setMinExecutionFee()
- CrossMarginHandler.setMaxExecutionChuck()
- CrossMarginHandler.setOrderExecutor()
- LimitTradeHandler.setMinExecutionFee()
- LimitTradeHandler.setIsAllowAllExecutor()
- LimitTradeHandler.setOrderExecutor()
- LiquidityHandler.setMinExecutionFee()
- LiquidityHandler.setMaxExecutionChunk()
- LiquidityHandler.setOrderExecutor()
- MarketTradeHandler.setTradeService()
- TradeHelper.updateFeeStates()
- TradeHelper.setConfigStorage()
- PerpStorage.setServiceExecutors()
- PerpStorage.setServiceExecutorBatch()
- PerpStorage.savePosition()
- VaultStorage.setServiceExecutors()
- VaultStorage.setServiceExecutorBatch()

Impact

Increased deployment code size and gas costs for end users.

Recommendation

Remove redundant modifiers.

Response

HMX updated Compounder.removeToken().