InterFi
NETWORK

SMART CONTRACT AUDIT

interfinetwork

hello@interfi.network

https://interfi.network

PREPARED FOR

SNTL MARKET

INTRODUCTION

Auditing Firm	InterFi Network
Client Firm	SNTL Market
Methodology	Automated Analysis, Manual Code Review
Language	Solidity
Contract	Multiple contracts
Blockchain	
Centralization	Active ownership
Commit	1969bef9a4af91d03a030b435ea141d5a628e565
Website	https://sntl.market/
Telegram	https://t.me/sntlmarkets/
Twitter	https://twitter.com/sntlai/
Report Date	May 11, 2023

ℹ️ Verify the authenticity of this report on our website: https://www.github.com/interfinetwork

EXECUTIVE SUMMARY

InterFi has performed the automated and manual analysis of solidity codes. Solidity codes were reviewed for common contract vulnerabilities and centralized exploits. Here's a quick audit summary:

Status	Critical 🔴	Major 🟠	Medium 🟡	Minor 🟢	Unknown 🟤
Open	0	0	0	5	0
Acknowledged	0	0	0	4	1
Resolved	0	1	1	0	0

ℹ️ Please note that smart contracts deployed on blockchains aren't resistant to exploits, vulnerabilities and/or hacks. Blockchain and cryptography assets utilize new and emerging technologies. These technologies present a high level of ongoing risks. For a detailed understanding of risk severity, source code vulnerability, and audit limitations, kindly review the audit report thoroughly.

ℹ️ Please note that centralization privileges regardless of their inherited risk status - constitute an elevated impact on smart contract safety and security.

TABLE OF CONTENTS

TABLE OF CONTENTS 4

SCOPE OF WORK 5

AUDIT METHODOLOGY 6

RISK CATEGORIES 8

CENTRALIZED PRIVILEGES 9

AUTOMATED ANALYSIS 10

INHERITANCE GRAPH 17

MANUAL REVIEW 18

DISCLAIMERS 35

ABOUT INTERFI NETWORK 38

SCOPE OF WORK

InterFi was consulted by SNTL Market to conduct the smart contract audit of their solidity source codes.

The audit scope of work is strictly limited to mentioned solidity file(s) only:

- GMXListingsDataV2.sol
- GMXTransferEligible.sol
- PayinETHorUSDC.sol
- SNTLMarket2.0.sol

AUDIT METHODOLOGY

Smart contract audits are conducted using a set of standards and procedures. Mutual collaboration is essential to performing an effective smart contract audit. Here's a brief overview of InterFi's auditing process and methodology:

CONNECT

- The onboarding team gathers source codes, and specifications to make sure we understand the size, and scope of the smart contract audit.

AUDIT

- Automated analysis is performed to identify common contract vulnerabilities. We may use the following third-party frameworks and dependencies to perform the automated analysis:
    - Remix IDE Developer Tool
    - Open Zeppelin Code Analyzer
    - SWC Vulnerabilities Registry
    - DEX Dependencies, e.g., Pancakeswap, Uniswap
- Simulations are performed to identify centralized exploits causing contract and/or trade locks.
- A manual line-by-line analysis is performed to identify contract issues and centralized privileges. We may inspect below mentioned common contract vulnerabilities, and centralized exploits:

Centralized Exploits	○ Token Supply Manipulation ○ Access Control and Authorization ○ Assets Manipulation ○ Ownership Control ○ Liquidity Access ○ Stop and Pause Trading ○ Ownable Library Verification

Common Contract Vulnerabilities	○ Integer Overflow ○ Lack of Arbitrary limits ○ Incorrect Inheritance Order ○ Typographical Errors ○ Requirement Violation ○ Gas Optimization ○ Coding Style Violations ○ Re-entrancy ○ Third-Party Dependencies ○ Potential Sandwich Attacks ○ Irrelevant Codes ○ Divide before multiply ○ Conformance to Solidity Naming Guides ○ Compiler Specific Warnings ○ Language Specific Warnings

REPORT

- The auditing team provides a preliminary report specifying all the checks which have been performed and the findings thereof.
- The client's development team reviews the report and makes amendments to solidity codes.
- The auditing team provides the final comprehensive report with open and unresolved issues.

PUBLISH

- The client may use the audit report internally or disclose it publicly.

ℹ️ It is important to note that there is no pass or fail in the audit, it is recommended to view the audit as an unbiased assessment of the safety of solidity codes.

RISK CATEGORIES

Smart contracts are generally designed to hold, approve, and transfer tokens. This makes them very tempting attack targets. A successful external attack may allow the external attacker to directly exploit. A successful centralization-related exploit may allow the privileged role to directly exploit. All risks which are identified in the audit report are categorized here for the reader to review:

Risk Type	Definition
Critical 🔴	These risks could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
Major 🟠	These risks are hard to exploit but very important to fix, they carry an elevated risk of smart contract manipulation, which can lead to high-risk severity.
Medium 🟡	These risks should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution. Low-risk re-entrancy-related vulnerabilities should be fixed to deter exploits.
Minor 🟢	These risks do not pose a considerable risk to the contract or those who interact with it. They are code-style violations and deviations from standard practices. They should be highlighted and fixed nonetheless.
Unknown 🟤	These risks pose uncertain severity to the contract or those who interact with it. They should be fixed immediately to mitigate the risk uncertainty.

All statuses which are identified in the audit report are categorized here for the reader to review:

Status Type	Definition
Open	Risks are open.
Acknowledged	Risks are acknowledged, but not fixed.
Resolved	Risks are acknowledged and fixed.

CENTRALIZED PRIVILEGES

Centralization risk is the most common cause of cryptography asset loss. When a smart contract has a privileged role, the risk related to centralization is elevated.

There are some well-intended reasons have privileged roles, such as:

- Privileged roles can be granted the power to `pause()` the contract in case of an external attack.
- Privileged roles can use functions like, `include()`, and `exclude()` to add or remove wallets from fees, swap checks, and transaction limits. This is useful to run a presale and to list on an exchange.

Authorizing privileged roles to externally-owned-account (EOA) is dangerous. Lately, centralization-related losses are increasing in frequency and magnitude.

- The client can lower centralization-related risks by implementing below mentioned practices:
- Privileged role's private key must be carefully secured to avoid any potential hack.
- Privileged role should be shared by multi-signature (multi-sig) wallets.
- Authorized privilege can be locked in a contract, user voting, or community DAO can be introduced to unlock the privilege.
- Renouncing the contract ownership, and privileged roles.
- Remove functions with elevated centralization risk.

ℹ️ Understand the project's initial asset distribution. Assets in the liquidity pair should be locked. Assets outside the liquidity pair should be locked with a release schedule.

AUTOMATED ANALYSIS

Symbol	Definition
🛑	Function modifies state
💵	Function is payable
🔒	Function is internal
🔐	Function is private
❗	Function is important

GMXListingsDataV2

```
| **GMXListingsData** | Implementation |  |||
| └ | GetGMXListingsData | External ❗ |   |NO❗ |
| └ | GetGMXAccountData | External ❗ |   |NO❗ |
||||||
| **IRewardTracker** | Interface |  |||
| └ | depositBalances | External ❗ |   |NO❗ |
| └ | stakedAmounts | External ❗ | 🛑  |NO❗ |
| └ | updateRewards | External ❗ | 🛑  |NO❗ |
| └ | stake | External ❗ | 🛑  |NO❗ |
| └ | stakeForAccount | External ❗ | 🛑  |NO❗ |
| └ | unstake | External ❗ | 🛑  |NO❗ |
| └ | unstakeForAccount | External ❗ | 🛑  |NO❗ |
| └ | tokensPerInterval | External ❗ |   |NO❗ |
| └ | claim | External ❗ | 🛑  |NO❗ |
| └ | claimForAccount | External ❗ | 🛑  |NO❗ |
| └ | claimable | External ❗ |   |NO❗ |
| └ | averageStakedAmounts | External ❗ |   |NO❗ |
| └ | cumulativeRewards | External ❗️ |   |NO❗️ |
||||||
| **IERC20** | Interface |  |||
| └ | totalSupply | External ❗️ |   |NO❗️ |
| └ | balanceOf | External ❗️ |   |NO❗️ |
| └ | transfer | External ❗️ | 🛑  |NO❗️ |
| └ | allowance | External ❗️ |   |NO❗️ |
| └ | approve | External ❗️ | 🛑  |NO❗️ |
| └ | transferFrom | External ❗️ | 🛑  |NO❗️ |
||||||
| **IVester** | Interface |  |||
| └ | getMaxVestableAmount | External ❗️ |   |NO❗️ |
| └ | getCombinedAverageStakedAmount | External ❗️ |   |NO❗️ |
||||||
| **IGMXVault** | Interface |  |||
| └ | SalePrice | External ❗️ |   |NO❗️ |
| └ | EndAt | External ❗️ |   |NO❗️ |
```

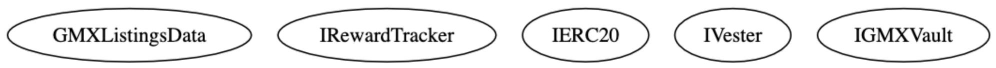
GMXTransferEligible

```
| **AccountEligible** | Implementation |  |||
| └ | TransferEligible | External ❗️ |   |NO❗️ |
||||||
| **IERC20** | Interface |  |||
| └ | totalSupply | External ❗️ |   |NO❗️ |
| └ | balanceOf | External ❗️ |   |NO❗️ |
| └ | transfer | External ❗️ | 🛑  |NO❗️ |
| └ | allowance | External ❗️ |   |NO❗️ |
| └ | approve | External ❗️ | 🛑  |NO❗️ |
| └ | transferFrom | External ❗️ | 🛑  |NO❗️ |
||||||
| **IRewardTracker** | Interface |  |||
| └ | depositBalances | External ❗️ |   |NO❗️ |
| └ | stakedAmounts | External ❗️ |   |NO❗️ |
| └ | updateRewards | External ❗️ | 🛑  |NO❗️ |
| └ | stake | External ❗️ | 🛑  |NO❗️ |
| └ | stakeForAccount | External ❗️ | 🛑  |NO❗️ |
| └ | unstake | External ❗️ | 🛑  |NO❗️ |
| └ | unstakeForAccount | External ❗️ | 🛑  |NO❗️ |
| └ | tokensPerInterval | External ❗️ |   |NO❗️ |
| └ | claim | External ❗️ | 🛑  |NO❗️ |
| └ | claimForAccount | External ❗️ | 🛑  |NO❗️ |
| └ | claimable | External ❗️ |   |NO❗️ |
| └ | averageStakedAmounts | External ❗️ |   |NO❗️ |
| └ | cumulativeRewards | External ❗️ |   |NO❗️ |
||||||
| **IVester** | Interface |  |||
| └ | claimForAccount | External ❗️ | 🛑  |NO❗️ |
| └ | transferredAverageStakedAmounts | External ❗️ |   |NO❗️ |
| └ | transferredCumulativeRewards | External ❗️ |   |NO❗️ |
| └ | cumulativeRewardDeductions | External ❗️ |   |NO❗️ |
| └ | bonusRewards | External ❗️ |   |NO❗️ |
| └ | transferStakeValues | External ❗️ | 🛑  |NO❗️ |
| └ | setTransferredAverageStakedAmounts | External ❗️ | 🛑  |NO❗️ |
| └ | setTransferredCumulativeRewards | External ❗️ | 🛑  |NO❗️ |
| └ | setCumulativeRewardDeductions | External ❗️ | 🛑  |NO❗️ |
| └ | setBonusRewards | External ❗️ | 🛑  |NO❗️ |
| └ | getMaxVestableAmount | External ❗️ |   |NO❗️ |
| └ | getCombinedAverageStakedAmount | External ❗ |   |NO❗ |

PayinETHorUSDC
| **PayinETHorUSDC** | Implementation | ReentrancyGuard |||
| └ | <Receive Ether> | External ❗ |  💵 |NO❗ |
| └ | <Fallback> | External ❗ |  💵 |NO❗ |
| └ | <Constructor> | Public ❗ | 🛑  |NO❗ |
| └ | ETHGMX | External ❗ |  💵 | nonReentrant OnlyEscrows |
| └ | USDCGMX | External ❗ |  💵 | nonReentrant OnlyEscrows |
| └ | WithdrawETH | External ❗ |  💵 | OnlyOwner nonReentrant |
| └ | WithdrawToken | External ❗ | 🛑  | OnlyOwner nonReentrant |
||||||
| **IFactoryContract** | Interface |  |||
| └ | EscrowsToOwners | External ❗ |   |NO❗ |
```

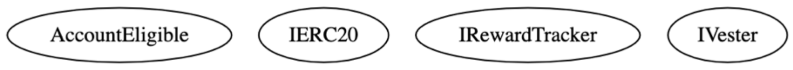
```
SNTLMarket2.0
| **DeployEscrow** | Implementation | ReentrancyGuard |||
| └ | <Receive Ether> | External ❗ |  💵 |NO❗ |
| └ | <Fallback> | External ❗ |  💵 |NO❗ |
| └ | <Constructor> | Public ❗ | 🛑  |NO❗ |
| └ | SetMaxOffers | Public ❗ | 🛑  | nonReentrant OnlyOwner |
| └ | SetMinGMXGLP | Public ❗ | 🛑  | nonReentrant OnlyOwner |
| └ | SetFeeAmount | Public ❗ | 🛑  | nonReentrant OnlyOwner |
| └ | EmitListed | External ❗ | 🛑  | nonReentrant OnlyEscrows |
| └ | EmitPurchased | External ❗ | 🛑  | nonReentrant OnlyEscrows |
| └ | EmitPriceChange | External ❗ | 🛑  | nonReentrant OnlyEscrows |
| └ | EmitOfferMade | External ❗ | 🛑  | nonReentrant OnlyEscrows |
| └ | List | External ❗ | 🛑  | nonReentrant |
| └ | DeployBuyerEscrow | External ❗ | 🛑  | OnlyEscrows nonReentrant |
| └ | GetListings | External ❗ |   |NO❗ |
| └ | GetNumberOfListings | External ❗ |   |NO❗ |
| └ | ResetCloseEscrow | External ❗ | 🛑  | OnlyEscrows nonReentrant |
| └ | DeleteListing | External ❗ | 🛑  | OnlyEscrows nonReentrant |
| └ | SetListingsToOwners | External ❗ | 🛑  | OnlyEscrows nonReentrant |
| └ | PushListing | External ❗ | 🛑  | OnlyEscrows nonReentrant |
| └ | SetFeeAddress | External ❗ | 🛑  | OnlyOwner nonReentrant |
| └ | SetAllowPurchases | External ❗ | 🛑  | OnlyOwner nonReentrant |
| └ | ComputeFutureEscrowAddress | Public ❗ |   |NO❗ |
| └ | WithdrawETH | External ❗ |  💵 | OnlyOwner nonReentrant |
| └ | WithdrawToken | External ❗ | 🛑  | OnlyOwner nonReentrant |
| └ | _IndexOfEscrowOwnerArray | Private 🔐 |   | |
| └ | _IndexOfListingArray | Private 🔐 |   | |
||||||
| **GMXEscrow** | Implementation | ReentrancyGuard |||
| └ | <Constructor> | Public ❗ | 🛑  |NO❗ |
| └ | <Receive Ether> | External ❗ |  💵 |NO❗ |
| └ | <Fallback> | External ❗ |  💵 |NO❗ |
| └ | MakeOffer | External ❗ | 🛑  | nonReentrant ClosedEscrow |
| └ | AcceptOffer | External ❗ | 🛑  | nonReentrant ClosedEscrow OnlyEscrowOwner |
| └ | CompoundAndClaim | External ❗ |  💵 | nonReentrant ClosedEscrow OnlyEscrowOwner |
| └ | TransferIn | Public ❗ | 🛑  | nonReentrant ClosedEscrow |
| └ | TransferOut | External ❗ | 🛑  | nonReentrant ClosedEscrow OnlyEscrowOwner |
| └ | TransferOutEscrowOwner | External ❗ | 🛑  | nonReentrant ClosedEscrow |
| └ | SetForSale | External ❗ | 🛑  | nonReentrant ClosedEscrow |
| └ | ChangePrice | External ❗ | 🛑  | nonReentrant ClosedEscrow OnlyEscrowOwner |
| └ | EndEarly | External ❗ | 🛑  | nonReentrant ClosedEscrow OnlyEscrowOwner |
| └ | MakePurchase | External ❗ |  💵 | nonReentrant ClosedEscrow |
| └ | CloseEscrow | External ❗ | 🛑  | nonReentrant ClosedEscrow OnlyEscrowOwner |
| └ | CancelAllOffers | External ❗ | 🛑  | nonReentrant ClosedEscrow OnlyEscrowOwner |
| └ | CancelOffer | External ❗ | 🛑  | nonReentrant ClosedEscrow |
| └ | WithdrawETH | External ❗ |  💵 | nonReentrant OnlyEscrowOwner |
| └ | WithdrawToken | External ❗ | 🛑  | nonReentrant OnlyEscrowOwner |
| └ | SetIsPurchased | External ❗ | 🛑  | nonReentrant ClosedEscrow |
| └ | GetOffers | External ❗ |   |NO❗ |
| └ | _IndexOfOfferArray | Private 🔐 |   | |
| └ | _ClearOffers | Private 🔐 | 🛑  | |
| └ | _ClearOffer | Private 🔐 | 🛑  | |
| └ | GetNumberOfOffers | External ❗ |   |NO❗ |
||||||
| **IERC20** | Interface |  |||
| └ | totalSupply | External ❗ |   |NO❗ |
| └ | balanceOf | External ❗ |   |NO❗ |
| └ | transfer | External ❗ | 🛑  |NO❗ |
| └ | allowance | External ❗ |   |NO❗ |
| └ | approve | External ❗ | 🛑  |NO❗ |
| └ | transferFrom | External ❗ | 🛑  |NO❗ |
||||||
| **IGMXRewardRouter** | Interface |  |||
| └ | stakeGmx | External ❗ | 🛑  |NO❗ |
| └ | stakeEsGmx | External ❗ | 🛑  |NO❗ |
| └ | unstakeGmx | External ❗ | 🛑  |NO❗ |
| └ | unstakeEsGmx | External ❗ | 🛑  |NO❗ |
| └ | claim | External ❗ | 🛑  |NO❗ |
| └ | claimEsGmx | External ❗ | 🛑  |NO❗ |
| └ | claimFees | External ❗ | 🛑  |NO❗ |
| └ | compound | External ❗ | 🛑  |NO❗ |

| └ | handleRewards | External ❗ | 🛑  |NO❗ |

| └ | signalTransfer | External ❗ | 🛑  |NO❗ |

| └ | acceptTransfer | External ❗ | 🛑  |NO❗ |

||||||

| **IWETH** | Interface | IERC20 |||

| └ | deposit | External ❗ |  💵 |NO❗ |

| └ | withdraw | External ❗ | 🛑  |NO❗ |

||||||

| **IGMXEscrow** | Interface |  |||

| └ | TransferOutEscrowOwner | External ❗ | 🛑  |NO❗ |

| └ | TransferIn | External ❗ | 🛑  |NO❗ |

| └ | SetIsPurchased | External ❗ | 🛑  |NO❗ |

||||||

| **IGMXEligible** | Interface |  |||

| └ | TransferEligible | External ❗ |   |NO❗ |

||||||

| **IPayinETHorUSDC** | Interface |  |||

| └ | ETHGMX | External ❗ |  💵 |NO❗ |

| └ | USDCGMX | External ❗ | 🛑  |NO❗ |
```

INHERITANCE GRAPH

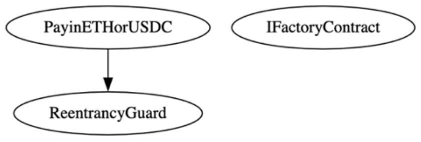
GMXListingsDataV2



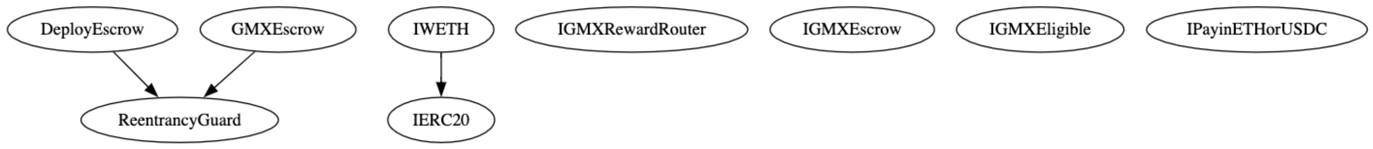
GMXTransferEligible



PayinETHorUSDC



SNTLMarket2.0



MANUAL REVIEW

Identifier	Definition	Severity
CEN-01	Centralized privileges	Major 🟠
CEN-07	Authorizations and access controls	

`OnlyOwner` modifier checks that `msg.sender` is the contract owner, and `OnlyEscrows` modifier checks that `msg.sender` is an escrow account. While this is an ideal way, it's important to ensure that `msg.sender` cannot be easily spoofed or manipulated.

`onlyOwner` centralized privileges are listed below:

```
WithdrawETH()
WithdrawToken()
SetMaxOffers()
SetMinGMXGLP()
SetFeeAmount()
SetFeeAddress()
SetAllowPurchases()
WithdrawETH()
WithdrawToken()
```

`OnlyEscrows` access control is provided to below mentioned functions:

```
ETHGMX()
USDCGMX()
EmitListed()
EmitPurchased()
EmitPriceChange()
EmitOfferMade()
DeployBuyerEscrow()
ResetCloseEscrow()
DeleteListing()
SetListingsToOwners()
```

`OnlyEscrowOwner` access control is provided to below mentioned functions:

```
WithdrawETH()
WithdrawToken()
```

`ClosedEscrow and OnlyEscrowOwner` access controls are provided to below mentioned functions:

```
AcceptOffer()
CompoundAndClaim()
TransferOut()
ChangePrice()
EndEarly()
CloseEscrow()
CancelAllOffers()
```

`ClosedEscrow` access control is provided to below mentioned functions:

```
MakeOffer()
TransferIn()
TransferOutEscrowOwner()
SetForSale()
MakePurchase()
CancelOffer()
SetIsPurchased()
```

RECOMMENDATION

Contract creator, contract owner, administrator and all privileged roles' private keys should be secured carefully. Please refer to PAGE-09 CENTRALIZED PRIVILEGES for a detailed understanding.

PARTIAL RESOLUTION*

SNTL Market team uses multi-sig wallet for ownership controls, where 2 out of 3 signatures are required to access `OnlyOwner` centralized privileges.

It is recommended to ensure that Escrow account owners are trusted addresses and privileged `msg.sender` cannot be easily manipulated.

Identifier	Definition
LOG-03	Re-entrancy

In **`SNTLMarket2.0`**, below mentioned functions are attributed with `nonReentrant` modifier:

```
SetMaxOffers
SetMinGMXGLP
SetFeeAmount
EmitListed
EmitPurchased
EmitPriceChange
EmitOfferMade
List
DeployBuyerEscrow
ResetCloseEscrow
DeleteListing
SetListingsToOwners
PushListing
SetFeeAddress
SetAllowPurchases
WithdrawETH
WithdrawToken
MakeOffer
AcceptOffer
CompoundAndClaim
TransferIn
TransferOut
TransferOutEscrowOwner
SetForSale
ChangePrice
EndEarly
MakePurchase
CloseEscrow
CancelAllOffers
CancelOffer
WithdrawETH
WithdrawToken
SetIsPurchased
```

In **`PayingETHforUSDC`**, below mentioned functions are attributed with `nonReentrant` modifier:

```
ETHGMX
USDCGMX
WithdrawETH
WithdrawToken
```

The contract uses the Re-entrancy Guard from the OpenZeppelin library to prevent re-entrancy attacks.

Identifier	Definition	Severity
LOG-04	Checks Effects Interactions	Medium 🟡

Some of these functions modify the state of the contract, and also interact with other contracts. They should be validated for Checks Effects Interactions:

`withdrawETH()` - Sending ETH to the owner is done before the `require(sent)` statement, which should ideally be placed before sending ETH to the owner.

In `AcceptOffer()`, the code transfers the balance of the contract to the `EscrowOwner`. If the contract receives Ether from an unexpected source, it will be transferred to the `EscrowOwner`. Consider implementing a more secure way to handle Ether payments.

RECOMMENDATION

Follow checks effects interactions pattern while handling contract flow.

RESOLUTION

SNTL Market team has updated `AcceptOffer()` function with recommended change. Regarding `withdrawETH()`, development team has followed this to send ETH, https://solidity-by-example.org/sending-ether/

Identifier	Definition	Severity
COD-01	Use of `.call()`	Minor 🟢

Smart contracts **`SNTLMarket2.0`** and **`PayingETHforUSDC`** use `.call()` in some instances to send ether.

RECOMMENDATION

Avoid using `.call()` whenever possible when executing another contract function as it bypasses type checking, function existence check, etc.

ACKNOWLEDGEMENT

SNTL Market team has acknowledged that `.call()` is more useful than `transfer` in this case, hence, `call()` is used to send ether.

Identifier	Definition	Severity
COD-02	Timestamp manipulation via `block.timestamp`	Minor 🟢

Be aware that the timestamp of the block can be manipulated by a miner. When the contract uses the timestamp to seed a random number, the miner can actually post a timestamp within 15 seconds of the block being validated, effectively allowing the miner to precompute an option more favorable to their chances.

RECOMMENDATION

To maintain block integrity, follow 15 seconds rule, and scale time dependent events accordingly.

Identifier	Definition	Severity
COD-05	Misleading nomenclature	Minor 🟢

`TransferEligible` function is a view function, which does not modify the state of the contract. However, the function name can be misleading, as it implies that it transfers something.

RECOMMENDATION

Use appropriate function names for better readability.

Identifier	Definition	Severity
SNT-01	Hardcoded addresses of contracts, tokens, routers	Minor ●
COD-06	Unknown externally owned addresses	

Addresses of the contracts and tokens are hardcoded. If any of these addresses change, the contract will need to be redeployed.

```
SNTLMarket2.0
    address constant private GMXEligible = 0x0Da48adDA1F4374F110EfD49A02d6665c6B69805;
    address constant private EsGMX = 0xf42Ae1D54fd613C9bb14810b0588FaAa09a426cA;
    address constant private WETH = 0x82aF49447D8a07e3bd95BD0d56f35241523fBab1;
    address constant private GMX = 0xfc5A1A6EB076a2C7aD06eD22C90d7E710E35ad0a;
    address constant private USDC = 0xFF970A61A04b1cA14834A43f5dE4533eBDDB5CC8;
    address constant private GMXRewardRouter = 0xA906F338CB21815cBc4Bc87ace9e68c87eF8d8F1;
    address constant private stakedGmxTracker = 0x908C4D94D34924765f1eDc22A1DD098397c59dD4;
    address constant private bonusGmxTracker = 0x4d268a7d4C16ceB5a606c173Bd974984343fea13;
    address constant private feeGmxTracker = 0xd2D1162512F927a7e282Ef43a362659E4F2a728F;
    address constant private gmxVester = 0x199070DDfd1CFb69173aa2F7e20906F26B363004;
    address constant private stakedGlpTracker = 0x1aDDD80E6039594eE970E5872D247bf0414C8903;
    address constant private feeGlpTracker = 0x4e971a87900b931fF39d1Aad67697F49835400b6;
    address constant private glpVester = 0xA75287d2f8b217273E7FCD7E86eF07D33972042E;
    address constant private GMXRewardContract = 0xA906F338CB21815cBc4Bc87ace9e68c87eF8d8F1;
    Owner = 0xeA4D1a08300247F6298FdAF2F68977Af7bf93d01;
    FeeAddress = 0xc7a5e5E3e2aba9aAa5a4bbe33aAc7ee2b2AA7bE4;

GMXTransferEligible
    address constant private EsGMXAddress = 0xf42Ae1D54fd613C9bb14810b0588FaAa09a426cA;
    address constant private WETHAddress = 0x82aF49447D8a07e3bd95BD0d56f35241523fBab1;
    address constant private GMXAddress = 0xfc5A1A6EB076a2C7aD06eD22C90d7E710E35ad0a;
    address constant private GMXRewardRouterAddress =
0xA906F338CB21815cBc4Bc87ace9e68c87eF8d8F1;
    address constant private stakedGmxTracker = 0x908C4D94D34924765f1eDc22A1DD098397c59dD4;
    address constant private bonusGmxTracker = 0x4d268a7d4C16ceB5a606c173Bd974984343fea13;
    address constant private feeGmxTracker = 0xd2D1162512F927a7e282Ef43a362659E4F2a728F;
    address constant private gmxVester = 0x199070DDfd1CFb69173aa2F7e20906F26B363004;
    address constant private stakedGlpTracker = 0x1aDDD80E6039594eE970E5872D247bf0414C8903;
    address constant private feeGlpTracker = 0x4e971a87900b931fF39d1Aad67697F49835400b6;
    address constant private glpVester = 0xA75287d2f8b217273E7FCD7E86eF07D33972042E;
```

```
PayingETHforUSDC

    address constant private WETH = 0x82aF49447D8a07e3bd95BD0d56f35241523fBab1;
    address constant private GMX = 0xfc5A1A6EB076a2C7aD06eD22C90d7E710E35ad0a;
    address constant private USDC = 0xFF970A61A04b1cA14834A43f5dE4533eBDDB5CC8;
    ISwapRouter constant router = ISwapRouter(0xE592427A0AEce92De3Edee1F18E0157C05861564);
    IPeripheryPayments constant refundrouter =
IPeripheryPayments(0xE592427A0AEce92De3Edee1F18E0157C05861564);
        Owner = 0xeA4D1a08300247F6298FdAF2F68977Af7bf93d01;
```

SNT-01 RECOMMENDATION

Use contract registry or configuration file to make the contract more flexible. Hardcoded addresses may be changed by an attacker with access to the contract bytecode.

COD-06 RECOMMENDATION

Private keys of externally owned accounts must be secured carefully. Use only trusted token and contract addresses in the code.

Identifier	Definition	Severity
COD-07	Lack of Natspec comments	Minor 🟢

Smart contracts are missing Natspec comments.

RECOMMENDATION

Follow coding conventions for writing solidity code. Provide appropriate Natspec comments for functions.

Identifier	Definition	Severity
COD-08	Very large number	Minor ●

`uint256` `MaxApproveValue` is set to the maximum value and is a very large number.

```
    uint256 constant private MaxApproveValue =
115792089237316195423570985008687907853269984665640564039457584007913129639935;
```

By setting the `MaxApproveValue`, contract ensures that the spender can spend an unlimited number of tokens.

COMMENT

Consider the potential security implications of granting unlimited approval to a contract.

ACKNOWLEDGEMENT

SNTL Market team has acknowledged this finding, and iterated that the `MaxApproveValue` is used by "trusted" GMX contracts and application. Hence, risk severity is amended from medium to minor.

Identifier	Definition	Severity
COD-10	Third Party Dependencies	Unknown ●

Smart contract is interacting with third party protocols, APIs, market makers, front-end applications, contracts, addresses, and interfaces. The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised, and exploited. Moreover, upgrades in third parties can create severe impacts, e.g., increased transactional fees, deprecation of previous routers, etc.

RECOMMENDATION

Inspect third party dependencies regularly, and mitigate severe impacts whenever necessary.

Identifier	Definition	Severity
COD-12	Lack of event-driven architecture	Minor 🟢

Smart contract uses function calls to update state, which can make it difficult to track and analyze changes to the contract over time.

RECOMMENDATION

Use events to track state changes. Events improve transparency and provide a more granular view of contract activity.

Identifier	Definition	Severity
VOL-01	Irrelevant code	Minor 🟢

In **`SNTLMarket2.0`** redundancy found in below mentioned interfaces:

```
IGMXRewardRouter
IWETH
```

In **`GMXTransferEligible`**, redundancy found in below mentioned contract:

```
AccountEligible
```

RECOMMENDATION

Remove redundant and dead code.

Identifier	Definition	Severity
VOL-02	Code improvements	Minor ●

Use `SafeERC20` library:

Replace `TransferHelper` with `SafeERC20` and update the related function calls accordingly.

Gas Optimization:

`_IndexOfEscrowOwnerArray` and `_IndexOfListingArray` functions use a loop to find the index of a given address. This can lead to high gas consumption when the arrays grow in size. A better approach would be to maintain a separate mapping that maps the addresses to their indices in the arrays, which would allow for constant-time lookups.

Array Management:

When you remove an element from an array, like in the `ResetCloseEscrow` and `DeleteListing` functions, you replace the removed element with the last element of the array and then pop the last element. This may cause an issue if the order of elements in the array is important. However, if the order doesn't matter, this is an acceptable way to remove elements from an array.

RECOMMENDATION

Add recommended improvements to the code.

Identifier	Definition	Severity
COM-01	Floating compiler status	

Compilers are set to `^0.8.19`

RECOMMENDATION

Pragma should be fixed to the version that you're indenting to deploy your contracts with.

DISCLAIMERS

InterFi Network provides the easy-to-understand audit of solidity source codes (commonly known as smart contracts).

The smart contract for this particular audit was analyzed for common contract vulnerabilities, and centralization exploits. This audit report makes no statements or warranties on the security of the code. This audit report does not provide any warranty or guarantee regarding the absolute bug-free nature of the smart contract analyzed, nor do they provide any indication of the client's business, business model or legal compliance. This audit report does not extend to the compiler layer, any other areas beyond the programming language, or other programming aspects that could present security risks. Cryptographic tokens are emergent technologies, they carry high levels of technical risks and uncertainty. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. This audit report could include false positives, false negatives, and other unpredictable results.

CONFIDENTIALITY

This report is subject to the terms and conditions (including without limitations, description of services, confidentiality, disclaimer and limitation of liability) outlined in the scope of the audit provided to the client. This report should not be transmitted, disclosed, referred to, or relied upon by any individual for any purpose without InterFi Network's prior written consent.

NO FINANCIAL ADVICE

This audit report does not indicate the endorsement of any particular project or team, nor guarantees its security. No third party should rely on the reports in any way, including to make any decisions to buy or sell a product, service or any other asset. The information provided in this report does not constitute investment advice, financial advice, trading advice, or any other sort of advice and you should not treat any of the report's content as such. This audit report should not be used in any way to make decisions around investment or involvement. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort.

FOR AVOIDANCE OF DOUBT, SERVICES, INCLUDING ANY ASSOCIATED AUDIT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

TECHNICAL DISCLAIMER

ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, INTERFI NETWORK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO SERVICES, AUDIT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, INTERFI NETWORK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM THE COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

WITHOUT LIMITING THE FOREGOING, INTERFI NETWORK MAKES NO WARRANTY OF ANY KIND THAT ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET THE CLIENT'S OR ANY OTHER INDIVIDUAL'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE.

TIMELINESS OF CONTENT

The content contained in this audit report is subject to change without any prior notice. InterFi Network does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following the publication.

LINKS TO OTHER WEBSITES

This audit report provides, through hypertext or other computer links, access to websites and social accounts operated by individuals other than InterFi Network. Such hyperlinks are provided for your reference and convenience only and are the exclusive responsibility of such websites' and social accounts' owners. You agree that InterFi Network is not responsible for the content or operation of such websites and social accounts and that InterFi Network shall have no liability to you or any other person or entity for the use of third-party websites and social accounts. You are solely responsible for determining the extent to which you may use any content at any other websites and social accounts to which you link from the report.

ABOUT INTERFI NETWORK

InterFi Network provides intelligent blockchain solutions. We provide solidity development, testing, and auditing services. We have developed 150+ solidity codes, audited 1000+ smart contracts, and analyzed 500,000+ code lines. We have worked on major public blockchains e.g., Ethereum, Binance, Cronos, Doge, Polygon, Avalanche, Metis, Fantom, Bitcoin Cash, Velas, Oasis, etc.

InterFi Network is built by engineers, developers, UI experts, and blockchain enthusiasts. Our team currently consists of 4 core members, and 6+ casual contributors.

Website: https://interfi.network

Email: hello@interfi.network

GitHub: https://github.com/interfinetwork

Telegram (Engineering): https://t.me/interfiaudits

Telegram (Onboarding): https://t.me/interfisupport