Market.xyz - SDK

Penetration Testing

Prepared by: **Halborn**

Date of Engagement: **May 17th, 2022 - May 31st, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	05/20/2022	Elena Calvo
0.2	Document Edits	05/26/2022	Elena Calvo
0.3	Draft Review	05/27/2022	Gabi Urrutia
1.0	Remediation Plan	07/11/2022	Elena Calvo
1.1	Remediation Plan Review	07/11/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com
Elena Calvo	Halborn	Elena.Maranon@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Market.xyz engaged Halborn to conduct a penetration testing on their SDK beginning on May 17th, 2022 and ending on May 31st, 2022 . The security assessment was scoped to the SDK provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned a full-time security engineer to audit the security of the SDK which will be used to communicate with the smart contracts. The security engineer is a blockchain and security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that SDK functions operate as intended
- Identify potential security issues with the SDK

In summary, Halborn identified some security risks that were addressed and accepted by the Market.xyz team. The main ones are the following:

- Outdated and vulnerable package dependencies.
- Lack of input validation and error handling.
- Bad method implementations: wrong inputs, missing parameters, incorrect naming, etc.

1.3 TEST APPROACH & METHODOLOGY

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

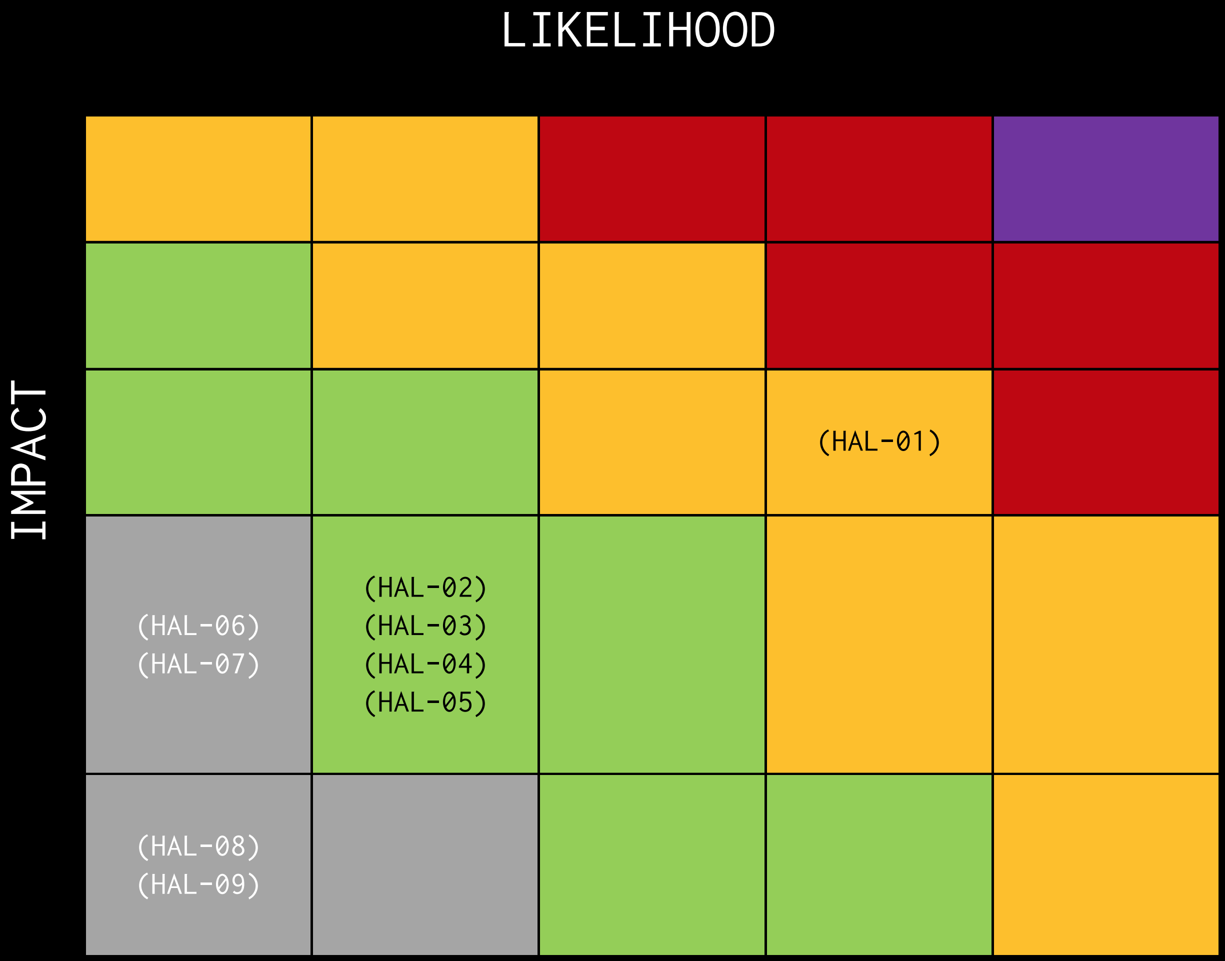
1.4 SCOPE

1. SDK Market.xyz
    (a) Repository: market.xyz-sdk
    (b) Commit ID: 46efbd8e9607710e7211220eecde6fde883a0d95

Out-of-scope: External libraries and financial related attacks

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	4	4



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) MULTIPLE OUTDATED PACKAGE DEPENDENCIES	Medium	SOLVED - 07/04/2022
(HAL-02) LACK OF INPUT VALIDATION	Low	RISK ACCEPTED
(HAL-03) LACK OF ERROR HANDLING	Low	RISK ACCEPTED
(HAL-04) METHOD OVERLOAD AND ERRORS IGNORED	Low	RISK ACCEPTED
(HAL-05) INCORRECT TYPE OF TRANSFER INPUT	Low	RISK ACCEPTED
(HAL-06) MISSING METHODS	Informational	ACKNOWLEDGED
(HAL-07) MISSING INPUT PARAMETER	Informational	ACKNOWLEDGED
(HAL-08) INCORRECT NAMING	Informational	ACKNOWLEDGED
(HAL-09) COMMENTS OF PENDING TASKS	Informational	ACKNOWLEDGED

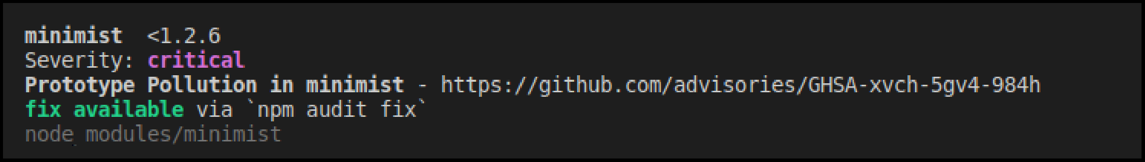
FINDINGS & TECH DETAILS

3.1 (HAL-01) MULTIPLE OUTDATED PACKAGE DEPENDENCIES - MEDIUM

Description:

The automatic analysis of the project dependencies shows that there are some vulnerable dependencies that should be updated to the patched version.

One of these vulnerabilities has been rated as critical and the rest as high.



```
minimist  <1.2.6
Severity: critical
Prototype Pollution in minimist - https://github.com/advisories/GHSA-xvch-5gv4-984h
fix available via `npm audit fix`
node_modules/minimist
```

Figure 1: npm audit evidence

The following table includes the links to the explanation of the vulnerabilities and its remediation.

Result:

Package	**Vulnerability**	**Patched version**
async	Prototype Pollution	=2.6.4 >=3.2.2
cross-fetch	Incorrect Authorization	=2.2.6 >=3.1.5
minimist	Prototype Pollution	>=1.2.6
node-fetch	Exposure of Sensitive Information to an Unauthorized Actor	=2.6.7 >=3.1.1

Risk Level:

Likelihood - 4

Impact - 3

Recommendation:

It is recommended to update all the dependencies of the project to patch security issues. Executing the command npm audit fix would update these dependencies automatically. In addition, it is also worth considering continuously monitor the versions of SDK components and their dependencies using tools like **retire.js** and **Snyk**. Please remember to always obtain components only from official sources over secure links. Prefer signed packages to reduce the chance of including a modified, malicious component.

The async package update should be treated carefully due to its dependency tree. Some packages depend on directly from this one and their update may affect the code performance. The packages affected are: merkle-patricia-tree, ethereumjs-block,ethereumjs-vm, web3-provider-engine and @truffle/hdwallet-provider. It is recommended to study how it could affect the update of this packages to the code before making the upgrade.

Remediation Plan:

SOLVED: The Market.xyz team solved the issue by updating the outdated package dependencies.

3.2 (HAL-02) LACK OF INPUT VALIDATION - LOW

Description:

During the SDK analysis it has been noticed that most of the method's implementation consists in a direct call to the contract functions without any kind of input validation, error handling or data operation.

The lack of input validation on the SDK leaves all kind of validations to the contract itself, leading to an unnecessary computation waste in the communication to the blockchain. A simple validation of input parameters before the contract call and using a proper error handling methodology will reduce the computational waste and highly improve the quality and utility of the SDK, even some security problems could be avoided, depending on the contracts code.

The next section contains some examples of this issue.

Code Location:

The function convertIRMtoCurve from **market-sdk-main/src/lib/JumpRateModel.ts** implements multiple mathematical operations using values which are extracted from the input, and they are not validated. Due to the use of Big Number library, the overflow errors are discarded, but the conversion of the input values to BN could raise an error due to the mentioned lack of validation.

Listing 1: market-sdk-main/src/lib/JumpRateModels.ts (Lines 107-112,127-131)

```
async convertIRMtoCurve(cToken: CToken) {
  const [
    reserveFactorMantissa,
    baseRatePerBlock,
    kink,
      multiplierPerBlock,
      jumpMultiplierPerBlock
    ] = await Promise.all([
      cToken.reserveFactorMantissa(),
      this.baseRatePerBlock(),
      this.kink(),
      this.multiplierPerBlock(),
      this.jumpMultiplierPerBlock()
    ]);

    const borrowerRates: { x: number; y: number }[] = [];
    const supplierRates: { x: number; y: number }[] = [];

    for (let i = 0; i <= 100; i++) {
      const supplyLevel =
        (Math.pow(
          (Number(
            this._getSupplyRate(
              Web3.utils.toBN((i * 1e16).toString()),
              Web3.utils.toBN(reserveFactorMantissa),
              Web3.utils.toBN(kink),
              Web3.utils.toBN(multiplierPerBlock),
              Web3.utils.toBN(baseRatePerBlock),
              Web3.utils.toBN(jumpMultiplierPerBlock)
            ).toString(),
          ) /
            1e18) *
            (this.sdk.options!.blocksPerMin * 60 * 24) +
            1,
          365,
        ) -
          1) *
        100;

      const borrowLevel =
        (Math.pow(
          (Number(
            this._getBorrowRate(
              Web3.utils.toBN((i * 1e16).toString()),
              Web3.utils.toBN(kink),
              Web3.utils.toBN(multiplierPerBlock),
              Web3.utils.toBN(baseRatePerBlock),
              Web3.utils.toBN(jumpMultiplierPerBlock)
            ).toString(),
          ) /
            1e18) *
            (this.sdk.options!.blocksPerMin * 60 * 24) +
            1,
          365,
        ) -
          1) *
        100;

      supplierRates.push({ x: i, y: supplyLevel });
      borrowerRates.push({ x: i, y: borrowLevel });
    }
    return { borrowerRates, supplierRates };
  }
}
```

The following methods are two examples extracted from Comptroller class (**market-sdk-main/src/lib/Comptroller.ts**). The first method makes a partial input validation, checking whether the input is an instance of CToken class or not. In case of negative result, it assigns the input parameter “cToken” directly to the contract call, without verifying if it is a real address or an empty string.

The second method assigns directly the input parameter to the contract call without any kind of checking if the value could be converted to number or BN.

Listing 2: market-sdk-main/src/lib/Comptroller.ts (Lines 47,54)

```
_setBorrowPaused(
    cToken: CToken | string,
    state: boolean,
    tx?: NonPayableTx
  ): PromiEvent<TransactionReceipt> {
    cToken = cToken instanceof CToken ? cToken.address : cToken;
    return this.contract.methods._setBorrowPaused(cToken, state).send(tx);
  }

  _setCloseFactor(
    newCloseFactorMantissa: number | string | BN,
    tx?: NonPayableTx
  ): PromiEvent<TransactionReceipt> {
    return this.contract.methods._setCloseFactor(
↳ newCloseFactorMantissa).send(tx);
  }

```

This example seems to obtain the input parameter from standard input (because of the name arg0). That input is directly passed to the contract, and any type of validation depends on how the client uses the SDK.

Listing 3: market-sdk-main/src/lib/Comptroller.ts (Line 413)

```
  markets(
    arg0: string,
    tx?: NonPayableTx
  ): Promise<{
    isListed: boolean;
    collateralFactorMantissa: string;
    0: boolean;
    1: string;
  }> {
    return this.contract.methods.markets(arg0).call(tx);
  }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to implement input validation inside methods and functions of the SDK to avoid non-necessary communications with the blockchain and even preventing security issues (this affirmation depends on how the contracts are implemented). Currently, all kinds of input validation, as well as error handling, depend on how the user makes use of the SDK, being responsible for adding the extra code needed for all the checks, as it can be shown on the examples folder.

Remediation Plan:

RISK ACCEPTED: The Market.xyz team accepted the risk of this finding.

3.3 (HAL-03) LACK OF ERROR HANDLING - LOW

Description:

The complete analysis of the SDK has revealed that there is not any kind of error handling along the code. As mentioned on the previous section, most of the methods consist on direct calls to the contract without any process of data inside, so the error handling is left to the contract itself. However, there are some functions that implement mathematical operations or data assignments whose potential errors are not handled.

The next section contains some examples of this issue.

Code Location:

The normalizePoolAsset function from **market-sdk-main/src/lib/Pools.ts** file is a normalization function with 20 input parameters which are directly assigned to variables. None of them are validated and there is no error handling neither. Working with an array of such dimensions could easily rise a **NullPointer** error in case of having any input missed.

Listing 4: market-sdk-main/src/lib/Pools.ts

```
export function normalizePoolAsset(raw: {
  0: string;
  1: string;
  2: string;
  3: string;
  4: string;
  5: string;
  6: string;
  7: string;
  8: string;
  9: string;
  10: string;
  11: string;
  12: string;
  13: boolean;
  14: string;
  15: string;
  16: string;
  17: string;
  18: string;
  19: string;
  20: string;
}, sdk: MarketSDK): PoolAsset {
  return {
    cToken: new CToken(sdk, raw[0]),
    underlyingToken: raw[1],
    underlyingName: raw[2],
    underlyingSymbol: raw[3],
    underlyingDecimals: new BN(raw[4]),
    underlyingBalance: new BN(raw[5]),
    supplyRatePerBlock: new BN(raw[6]),
    borrowRatePerBlock: new BN(raw[7]),
    totalSupply: new BN(raw[8]),
    totalBorrow: new BN(raw[9]),
    supplyBalance: new BN(raw[10]),
    borrowBalance: new BN(raw[11]),
    liquidity: new BN(raw[12]),
    membership: raw[13],
    exchangeRate: new BN(raw[14]),
    underlyingPrice: new BN(raw[15]),
    oracle: raw[16],
    collateralFactor: new BN(raw[17]),
    reserveFactor: new BN(raw[18]),
    adminFee: new BN(raw[19]),
    fuseFee: new BN(raw[20]),
  }
}
```

The method getPoolsByAccount from file **market-sdk-main/src/lib/PoolDirectory.ts** implements some data processing. The input parameter is used to make the contract call, and the result obtained is processed over two 'for' loops and pushed into a couple of arrays before being returned. Due to the lack of input validation, in case of a wrong account parameter, the result of the call would be an error. This potential error is not handled.

Listing 5: market-sdk-main/src/lib/PoolDirectory.ts

```
async getPoolsByAccount(
  account: string,
  tx?: NonPayableTx
): Promise<{
  indexes: BN[];
  pools: Pool[];
}> {
  const { 0: indexesRaw, 1: poolsRaw } = await this.contract.
↳ methods.getPoolsByAccount(account).call(tx);

  const indexes: BN[] = [];
  const pools: Pool[] = [];

  for(const pool of poolsRaw){
    pools.push(normalizePool(pool, this.sdk));
  }
  for(const index of indexesRaw){
    indexes.push(new BN(index));
  }
  return { indexes, pools };
}
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

As mentioned on input validation issue, it is recommended to implement error handling inside methods and functions of the SDK to facilitate to the user working with the SDK. In case of error, if error handling is applied, it would be easier to debug the code and find the root of the problem. Currently, error handling, as well as input validation, depends on how the user makes use of the SDK, being responsible for adding the extra code needed for all the checks as it can be shown on the examples folder.

Remediation Plan:

RISK ACCEPTED: The Market.xyz team accepted the risk of this finding.

3.4 (HAL-04) METHOD OVERLOAD AND ERRORS IGNORED - LOW

Description:

The method init, belonging to MarketSDK class, from the file **market-sdk-main/src/lib/MarketSDK.ts** presents the same method name with two different signatures and also two different implementations.

According to Typescript documentation, function/method overload is allowed having different signatures as log as the implementation signature is only defined once.

Reference:https://www.typescriptlang.org/docs/handbook/2/functions.html#function-overloads

In addition, the init() method contains multiple @ts-ignore tags for almost all the option parameters. According to Typescript documentation: this comment only suppresses the error reporting, and we recommend you use this comments very sparingly; therefore, it’s recommended to limit the use of this tag and, in case of use, always reporting which error will be suppressed.

Reference:https://www.typescriptlang.org/docs/handbook/release-notes/typescript-2-6.html#suppress-errors-in-ts-files-using--ts-ignore-comments

Code Location:

First init method:

Listing 6: market-sdk-main/lib/MarketSDK.ts (Lines 23,27,30)

```
async init(){
  if(!this.options){
    const chainId = await this.web3.eth.getChainId() as keyof
↳ typeof Addrs;

    // @ts-ignore
    if (Addrs[chainId].v2) {
      this.options = {
        // @ts-ignore
        poolDirectory: Addrs[chainId].v2.poolDirectory,
        // @ts-ignore
        poolLens: Addrs[chainId].v2.poolLens,
        // @ts-ignore
        marketLens: Addrs[chainId].v2.marketLens,
        blocksPerMin: Addrs[chainId].blocksPerMin
      };
    // @ts-ignore
    } else if(Addrs[chainId].v1) {
      this.options = {
        // @ts-ignore
        poolDirectory: Addrs[chainId].v1.poolDirectory,
        // @ts-ignore
        poolLens: Addrs[chainId].v1.poolLens,
        // @ts-ignore
        marketLens: Addrs[chainId].v1.marketLens,
        blocksPerMin: Addrs[chainId].blocksPerMin
      };
    }}}
```

Second init method with different signature implementation:

Listing 7: market-sdk-main/lib/MarketSDK.ts (Line 58)

```
  static async init(web3: Web3, options?: MarketOptions){
    const sdk = new MarketSDK(web3, options);
    await sdk.init();

    return sdk;
  }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

For the duplicity of init signature, it is recommended to rename one of the methods to avoid possible errors.

The excessive use of @ts-ignore tag would be fixed using a proper error handling methodology like try-catch structure, as mentioned on finding HAL-03.

Remediation Plan:

RISK ACCEPTED: The Market.xyz team accepted the risk of this finding.

3.5 (HAL-05) INCORRECT TYPE OF TRANSFER INPUT - LOW

Description:

The methods liquidateBorrow, mint, repayBorrow and repayBorrowBehalf belonging to class CTokenV2 from **market-sdk-main/src/lib/CToken.ts** are defined on the ABI of the contract as payable functions; however, the interface implemented by send parameter (tx) is NonPayableTX. This interface does not accept the parameter value, which is already considered on the interface PayableTx.

Code Location:

Declaration of liquidateBorrow method:

Listing 8: market-sdk-main/src/lib/CToken.ts (Lines 638,647)

```
  liquidateBorrow(
    borrower: string,
    cTokenCollateral: CTokenV2 | string,
    tx?: NonPayableTx
  ): PromiEvent<TransactionReceipt> {

```

ABI interface for liquidateBorrow function of the contract:

Listing 9: market-sdk-main/abi/CTokenV2.json (Lines 1070-1071,1079-1080)

```
          "name": "liquidateBorrow",
          "outputs": [],
          "payable": true,
          "stateMutability": "payable",
          "type": "function"
```

Declaration of mint method:

Listing 10: market-sdk-main/src/lib/CToken.ts (Line 647)

```
646   mint(
647     tx?: NonPayableTx
648   ): PromiEvent<TransactionReceipt> {
649     return this.contract.methods.mint().send(tx);
650   }
```

ABI interface for mint function of the contract:

Listing 11: market-sdk-main/abi/CTokenV2.json (Lines 1078-1079)

```
1076        "name": "mint",
1077        "outputs": [],
1078        "payable": true,
1079        "stateMutability": "payable",
1080        "type": "function"
```

Declaration of repayBorrow method:

Listing 12: market-sdk-main/src/lib/CToken.ts (Line 679)

```
678   repayBorrow(
679     tx?: NonPayableTx
680   ): PromiEvent<TransactionReceipt> {
681     return this.contract.methods.repayBorrow().send(tx);
682   }
```

ABI interface for repayBorrow function of the contract:

Listing 13: market-sdk-main/abi/CTokenV2.json (Lines 1159-1160,1174-1175)

```
1157        "name": "repayBorrow",
1158        "outputs": [],
1159        "payable": true,
1160        "stateMutability": "payable",
1161        "type": "function"
```

Declaration of repayBorrowBehalf method:

Listing 14: market-sdk-main/src/lib/CToken.ts (Line 685)

```
  repayBorrowBehalf(
    borrower: string,
    tx?: NonPayableTx
  ): PromiEvent<TransactionReceipt> {
    return this.contract.methods.repayBorrowBehalf(borrower).send(
  ↳ tx);
  }
```

ABI interface for repayBorrowBehalf function of the contract:

Listing 15: market-sdk-main/abi/CTokenV2.json (Lines 1174-1175)

```
      "name": "repayBorrowBehalf",
      "outputs": [],
      "payable": true,
      "stateMutability": "payable",
      "type": "function"
    },
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to change the tx type from NonPayableTx to PayableTx.

Remediation Plan:

RISK ACCEPTED: The Market.xyz team accepted the risk of this finding.

3.6 (HAL-06) MISSING METHODS - INFORMATIONAL

Description:

The purpose of an SDK is to facilitate the use of the contract interface to any user, translating the functions contained in the ABI files to a group of Typescript classes and methods to make an easier integration within the web environment.

During the analysis of the SDK, almost all the functions contained in the ABI have been converted to typescript methods, but some of them have been missed:

Missing methods:

Contract	**Function from ABI**
ComptrollerV2	supplyCaps
CToken	addReserves
CToken	comptroller
CV2Token	comptroller
JumpRateModel	isInterestRateModel
PoolDirectory	setAdminDeployer
PoolLensv1	directory

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to review the missed functions to verify if the absence is due to a mistake during development, or it was on purpose.

Remediation Plan:

ACKNOWLEDGED: The Market.xyz team acknowledged this finding.

3.7 (HAL-07) MISSING INPUT PARAMETER - INFORMATIONAL

Description:

The method getPoolSummary belonging to the class PoolLensV2 from file **market-sdk-main/src/lib/PoolLens.ts** only have the 'tx' input parameter defined, however, the ABI interface of the contract (PoolLensV2.json) has defined another input parameter called comptroller from contract Comptroller.

Code Location:

Method getPoolSummary from class PoolLensV2:

Listing 16: market-sdk-main/src/lib/PoolLens.ts (Lines 328,335)

```
  async getPoolSummary(
    tx?: NonPayableTx
  ): Promise<{
    totalSupply: BN,
    totalBorrow: BN,
    underlyingTokens: string[],
    underlyingSymbols: string[],
  }> {
    const raw = await this.contract.methods.getPoolSummary(this.
  ↳ address).call(tx);

    return {
      totalSupply: new BN(raw[0]),
      totalBorrow: new BN(raw[1]),
      underlyingTokens: raw[2],
      underlyingSymbols: raw[3],
    };
  }
```

ABI interface for getPoolSummary from PoolLensV2:

Listing 17: market-sdk-main/abi/PoolLensV2.json (Lines 87-89)

```
    {
      "inputs": [
        {
          "internalType": "contract Comptroller",
          "name": "comptroller",
          "type": "address"
        }
      ],
      "name": "getPoolSummary",
      "outputs": [
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to modify the signature definition on the method getPoolSummary from class PoolLensV2 to add the missing comptroller input parameter.

Remediation Plan:

ACKNOWLEDGED: The Market.xyz team acknowledged this finding.

3.8 (HAL-08) INCORRECT NAMING - INFORMATIONAL

Description:

The initialize method, belonging to CTokenV2 class, from file **market-sdk-main/src/lib/CToken.ts**, has not been properly named regarding the ABI interface of the contract.

Code Location:

Nomenclature of initialize function in CTokenV2 class:

Listing 18: market-sdk-main/src/lib/CToken.ts (Lines 586,599,602,613)

```
586   "initialize(address,address,uint256,string,string,uint8,uint256,uint256)"(
587     comptroller: ComptrollerV2 | string,
588     interestRateModel: string,
589     initialExchangeRateMantissa: number | string | BN,
590     name: string,
591     symbol: string,
592     decimals: number | string | BN,
593     reserveFactorMantissa: number | string | BN,
594     adminFeeMantissa: number | string | BN,
595     tx?: NonPayableTx
596   ): PromiEvent<TransactionReceipt> {
597     comptroller = comptroller instanceof ComptrollerV2 ? comptroller.address : comptroller;
598
599     return this.contract.methods["initialize(address,address,uint256,string,string,uint8,uint256,uint256)"](comptroller, interestRateModel, initialExchangeRateMantissa, name, symbol, decimals, reserveFactorMantissa, adminFeeMantissa).send(tx);
600   }
601
602   "initialize(address,address,string,string,uint256,uint256)"(
603     comptroller: ComptrollerV2 | string,
604     interestRateModel: string,
605     name: string,
606     symbol: string,
    reserveFactorMantissa: number | string | BN,
    adminFeeMantissa: number | string | BN,
    tx?: NonPayableTx
  ): PromiEvent<TransactionReceipt> {
    comptroller = comptroller instanceof ComptrollerV2 ?
↳ comptroller.address : comptroller;

    return this.contract.methods["initialize(address,address,
↳ string,string,uint256,uint256)"](comptroller, interestRateModel,
↳ name, symbol, reserveFactorMantissa, adminFeeMantissa).send(tx);
  }
```

initialize function in ABI file:

Listing 19: market-sdk-main/src/abi/CToken.json (Line 962)

```
      "name": "initialize",
      "outputs": [],
      "payable": false,
      "stateMutability": "nonpayable",
      "type": "function"
    },
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to change the name of the initialize method of CTokenV2 class in order to correspond with the ABI contract interface.

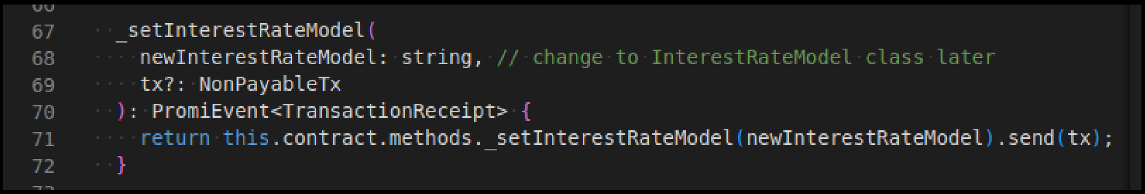
Remediation Plan:

ACKNOWLEDGED: The Market.xyz team acknowledged this finding.

3.9 (HAL-09) COMMENTS OF PENDING TASKS - INFORMATIONAL

Description:

The method _setInterestRateModel belonging to the classes CToken and CTokenV2 from file **market-sdk-main/src/lib/CToken.ts** has the comment change to InterestRateModel class later. This is not a security issue itself, but it denotes a lack of quality for a project that it is supposed to be deployed on production.



```
  _setInterestRateModel(
    newInterestRateModel: string, // change to InterestRateModel class later
    tx?: NonPayableTx
  ): PromiEvent<TransactionReceipt> {
    return this.contract.methods._setInterestRateModel(newInterestRateModel).send(tx);
  }
```

Figure 2: Comment on the code

Code Location:

Listing 20: market-sdk-main/src/lib/CToken.ts (Line 428)

```
427   _setInterestRateModel(
428     newInterestRateModel: string, // change to InterestRateModel
    ↳ class later
429     tx?: NonPayableTx
430   ): PromiEvent<TransactionReceipt> {
431     return this.contract.methods._setInterestRateModel(
    ↳ newInterestRateModel).send(tx);
432   }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to review the comments in the code and apply those pending changes that would affect to the future functionality before get into production environment.

Remediation Plan:

ACKNOWLEDGED: The Market.xyz team acknowledged this finding.

THANK YOU FOR CHOOSING

HALBORN