FusionTech

SolidX Escrow

Audit Report

www.dapp.solidx.tech

TABLE OF CONTENTS

Summary

Audit Result

Findings

Overview

This report has been prepared for to discover issues and vulnerabilities in the source code of the project as well as any contract dependencies that were not part of an officially recognized library.A comprehensive examination has been performed, utilizing Static Analysisand Manual Reviewtechniques. The auditingprocess pays specialattention to the following considerations:

Testing the smart contracts against both commonand uncommon attackvectors. Assessing the codebase to ensure compliancewith current best practices and industry standards. Ensuring contract logic meets the specifications and intentions of the client. Cross referencing contract structure and implementation againstsimilar smart contracts produced by industryleaders. Thorough line-by-line manual review of the entirecodebase by industryexperts.

The security assessment resulted in findings that ranged from critical to informational. We recommendaddressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could betterserve the projectfrom the securityperspective:

Enhance general coding practices for better structures of source codes; Add enough unit tests to cover the possible use cases; Provide more comments per each function for readability, especiallycontracts that are verified in public; Provide more transparency on privileged activities once the protocolis live.

Audit Summary

Project Name	SolidX (escrow Dapp) - (https://dapp.solidx.tech/)
Platform	ETHEREUM & BINANCE SMART CHAIN
Language	Solidity
Delivery Date	Feb 27, 2024
Audit Methodology	Static Analysis, Manual Review
Key Components	Escrow

Vulnerability Summary

Vulnerability Summary	Total	ⓘ Pending	ⓘ Declined	ⓘ Acknowledged	ⓘ Partially Resolved	ⓘ Resolved
Critical	0	0	0	0	0	0
Major	0	0	0	0	0	0
Medium	0	0	0	0	0	0
Minor	0	0	0	0	0	0
Informational	0	0	0	0	0	0
Discussion	0	0	0	0	0	0

Audit Scope

External Dependencies

This audit focused on identifying security flaws in code and the design of EscrowDapp Contract. It was conducted on the source code provided by the EscrowDapp team. The following files were made available in the course of the review:

Privileged Functions

The contract contains the following privileged functions that are restricted by role with the modifier. Since the contract is the owner cannot modify the contract configurations and address attributes.

Audit methodology

Dependencies

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Integer Overflow and Underflow

- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

About EscrowDapp Contract

EscrowDapp (https://www.dapp.solidx.tech/) is a blockchain escrow service acts as a neutral third party between buyer and seller to protect both parties from potential fraudulent actions of the other.

Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

- **ServiceEscrow.sol**
- **DealContract.sol**
- **Quote.sol**
- **SolidxToken.sol**
- **Ownable.sol**
- **IUniswapV2Router01.sol**
- **IUniswapV2Router02.sol**
- **IUniswapV3Factory.sol**
- **IUniswapV3pool.sol**
- **ReentrancyGuard.sol**
- **SafeMath.sol**

Introduction

Decentralised escrow would refer to an escrow whose operations aren't controlled by anyone, rather being transparent enough to be visible to everyone (or as intended). Ethereum is such a decentralised network (blockchain) which allows transactions to be carried out between any two transacting parties without the need of a centralised go-through intermediary (or so-called banks). Not only that, unlike Bitcoin blockchain, Ethereum network allows execution of code powered by gas. There comes the concept of smart contracts which are task-specific codes written to guarantee the particular task to be carried out in a decentralised way without the intervention of an intermediary. So, escrows fall into this fitting use-case of decentralised network to perform transaction. The core idea is to have the smart contract to take care of the security deposit and prevent the transacting parties to default. Escrow over Ethereum makes sure the service-demander's money isn't fiddled with, the serviceman gets the service charge he demands, and the service-demander gets the service he demands.

Issues

- Either / both of the parties might not be honest in making the deal - and this might result in locking up of one's money in the escrow until the issue is resolved by involvement of legal authorities.

- The third-party entrusted for the escrow might not be honest - and this might result in cases of contract manipulations, collusions, theft, unwanted troubles. Third-party involvements are always security-holes.

- In case the escrow is a physical one, there might be general problems faced by a non-electronic system. There might also be the case of currency disagreements.

Core Functionality

- There's a escrowOwner who is the one who creates the contract. There's a variable eState tracking the current status of the escrow which will be set to unInitialized after this step.

- The escrowOwner will initialize the contract for the two parties by calling the initEscrow() passing seller address, buyer address, fee percentage and the final block number (which is the final block number denoting the service time). eState is set to initialized at this step. It's ensured that none of the addresses of the buyer or seller is equal to that of the escrowOwner.

- Once the escrow is initialized, the buyer can make any number of deposits to the contract using depositToEscrow(); and events are also emitted for the same. This must be noted that the money deposited is owned by the contract and the escrowOwner has no control over it. eState will now hold a status of **buyerDeposited**.

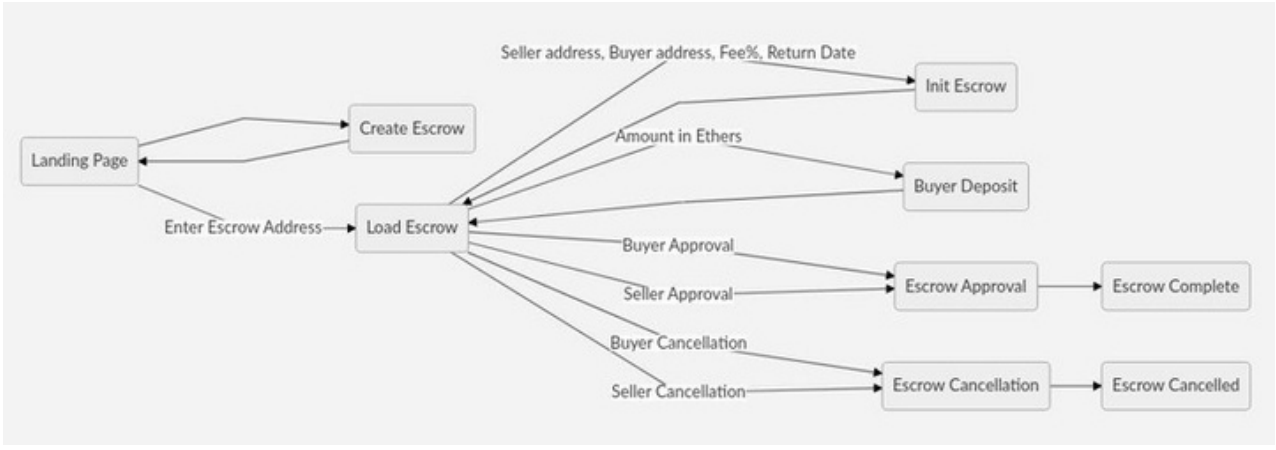
- As transactions take place after this and the latest block number increases (i.e. service time passes).

- Once the service is complete by the seller (before the latest block number exceeds the given block number limit), the seller approves the escrow (i.e. marks the service by him to be complete).

- The escrow is now in steady state and is ready to conduct another escrow.

- Otherwise, the escrow can be ended only by escrowOwner so that the contract is destructed.

- The buyer next reviews the service by the seller and
  - In case the service quality is acceptable,
    - Approves the escrow.
    - Since both the seller and buyer have approved the escrow, eState now changes to serviceApproved.
    - Next, the smart contract automatically initiates payment of fee charges to the escrowOwner - value of which is decided by the pre-decided fee percentage.
    - Next, the smart contract automatically initiates the payment of the remaining balance amount to the seller address. eState is now changed to escrowComplete.
  - In case the service quality is not acceptable,
    - The buyer does not approve the service.
    - After further negotiation, the seller can re-service or decide to cancel the escrow.
    - If the buyer too cancels the escrow, the entire amount of money deposited into the escrow will be refunded back the buyer with the escrowOwner keeping the pre-decided amount of sum as escrow fee charge. eState is now changed to escowCancelled.



- Landing page provides with the options of creating a new Escrow contract or loading an Escrow contract from the given set of already created contracts. Escrow contract creation is managed through a Factory contract which is created as a genesis by the application owner.
- The user of the dapp may:

i. Create a new contract using the button. The latest created contract will appear at the end of the list group. The user who creates this contract becomes the owner of this contract.

ii. Load an already created contract. A contract can be loaded by any user, but the owner remains the owner.

- Once a contract is loaded, the user may:

iii. Initiate the contract. But this can only be done by the owner of the contract. To initialize the contract, the addresses of the seller, buyer, the fee percent charged by the owner and the return date must be specified. Once initiated, the buyer and seller of the escrow are confirmed and cannot be changed.

iv. Deposit the negotiated amount into the escrow. This can only be done by the buyer. The buyer can deposit any number of times into the contract. All the deposit installments will be shown in the deposits section.

v. Approve the contract. Once a contract is initialized, the contract is ready to be approved by the buyer and the seller. The approval of the contract signifies the consent of the buyer as well as the seller to agree with the terms and conditions of the escrow.

a. The completion of the contract requires the approval of both the seller and the buyer. Once both of the parties approve, the contract will pay out. The pay out process of the contract involves the charging of the specified percentage of key being transferred to the contract owner, and the remaining part of the deposit to the deserving seller.

b. The contract won't pay out unless both of the parties approve.

vi. Cancel the contract. The buyer / seller or both might not want to go forward with the escrow at any point of time, and they might then want to cancel the contract. Cancelling the contract requires the action of both the buyer and the seller. Once the contract is cancelled, the entire deposit will be transferred back to the buyer.

- An escrow is said to be complete if either the contract has been approved by both and the service by the seller is received by the buyer or if the escrow has been cancelled by both. In this state, the contract is unusable and cannot be interacted with furthur.

What is Escrow and what is this application?

You can think of the escrow as a middleman. Let's say I want to buy something, and you want to sell something but we both don't trust each other. You don't trust me that I will pay the exact amount of money, and I don't trust you that you will send me the right object I've bought. We decide on a third person/party to be the middleman to prove if the transaction happened as it should be. We both trust that third party, but what if that trusted middleman is dishonest? What if the game is rigged?
Smart contracts come to the rescue right at this point. That middleman might be dishonest but if both parties agree on a smart contract that all the rules of this transaction are set, then there will be no worries about an escrow being dishonest.

This is the smart contract we get from that repository. Yep, that's all. Just a few lines of code, and a basic escrow contract. There are addresses: arbiter, beneficiary and depositor. In the constructor, you can see depositor = msg.value. What happens here? The depositor is the buyer and the beneficiary is the seller. The depositor deploys a contract with some money. It pays the price of the product to the contract (not to the seller) when deploying. There is an arbiter and basically, it acts like a referee. If the seller sends the product to the buyer arbiter approves this transaction. Only the arbiter can call the approve() function. After the arbiter's (referee) approval, the balance of the contract is sent to the beneficiary (seller) and the Approved event is emitted.
Let's make it short. The depositor pays to the contract. Arbiter checks and approves. The seller gets the contract balance after the approval.

Challenges to Implement

In the base code, the deposit amount and value are not in Eth but in Wei. Because of that, it was hard to read the values and confirm a transaction. The first challenge is changing the values to Eth which is not a hard thing to do. The previous code was:

```
const value = ethers.BigNumber.from(document.getElementById('wei').value);
```

and it was getting the value typed and changing it to BigNumber using ethers library. The only thing I needed to do was using parseEther and that's all:

```
const value = ethers.utils.parseEther(document.getElementById('eth').value);
```

Audit Result

Overview

The EscrowDapp Contract was written in Solidity language, with the required version to be 0.8.15, and includes 3 contracts: EscrowFactory.sol, Escrow.sol and SafeMath.sol. The EscrowDapp team uses this product to serve as a third party which allows the buyer escrowing tokens or native currency and transfer to the seller that amount after the condition is met (seller delivered and buyer confirmed).

EscrowFactory contract

EscrowFactory contract is initialized with a list of trusted handlers & trusted token addresses.

Trusted handlers can modify the fee percent, trusted handlers, trusted token addresses, and withdraw the amount of the contract.

Due to the appearance of receive() and fallback() functions, the contract is able to receive ether from anyone.

Escrow contract

Escrow contract only accepts buyer, seller, and trusted address for interacting. The buyer deposits tokens for this contract.

The buyer can approve the contract release token after the seller has delivered. But on the other hand, if the buyer rejects and the seller rejects, the status of the escrow is changed to dispute.

When the escrow status is ongoing, revised either the buyer or seller may cancel.

Trusted handlers may add more trusted addresses and call the fund() function while the status is not completed or canceled.

Due to the appearance of receive() and fallback() functions, the contract is able to receive ether from anyone.

Note: The extra ETH that was deposited after the contract was initialized cannot be withdrawn by anyone.

Findings

During the audit process, the audit team found some vulnerability issues in the given files of EscrowDapp Contract.

```
/// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract ServiceEscrow is Ownable, ReentrancyGuard {
    using SafeERC20 for IERC20;
    using SafeMath for uint256;
    using Quote for address;

    struct Milestone {
        string name;
        string description;
        uint256 amount;
        uint256 deadline;
        uint256 status; // 0:Ongoing, 1:Approved
    }

    constructor (address usdc_, IERC20 feeToken_) {
        _feeToken = feeToken_;
        _usdc = usdc_;
        _feePercentage = 50; // (50 / 10000 = 0.5%)
        _minimumFeeAmountInUSD = 5 * 10 ** 6;
        _maximumFeeAmountInUSD = 250 * 10 ** 6;
    }

    event CreateDeal(address creator_, bytes32 dealId_);
    event JoinDeal(address partner_, bytes32 dealId_);
    event ApproveMilestone(address buyer_, bytes32 dealId_, uint256 milestoneIndex_);
    event ApproveDeal(address seller_, bytes32 dealId_);
    event CancelDeal(address seller_, bytes32 dealId_);
    event AchieveDeal(address seller_, bytes32 dealId_);

    modifier _requireDealParticipant(bytes32 dealId_) {
        require(
            _msgSender() == _services[dealId_].creator ||
                _msgSender() == _services[dealId_].partner,
            "Solidx: not a service party"
        );
    }
```

ServiceEscrow.sol - Trusted addresses have more privilege than necessary at fund() function INFORMATIVE

Apart from status completed and canceled, the fund() function allows trusted addresses to call. So, trusted addresses can intervene in between buyer and seller transactions in unnecessary cases.

In the disputing case, the fund() function simply allows trusted addresses to be called. .

```
function fund(address payable toFund) external trusted {
require(toFund == escrowDetail.buyer || toFund == escrowDetail.seller,
'___INVALID_BUYER_SELLER___');
    require(escrowDetail.status == EscrowStatus.Dispute, '___DO_NOT_ALLOW___');
sendAndStatusUpdate(toFund, EscrowStatus.Complete);
}
```

APPENDIX

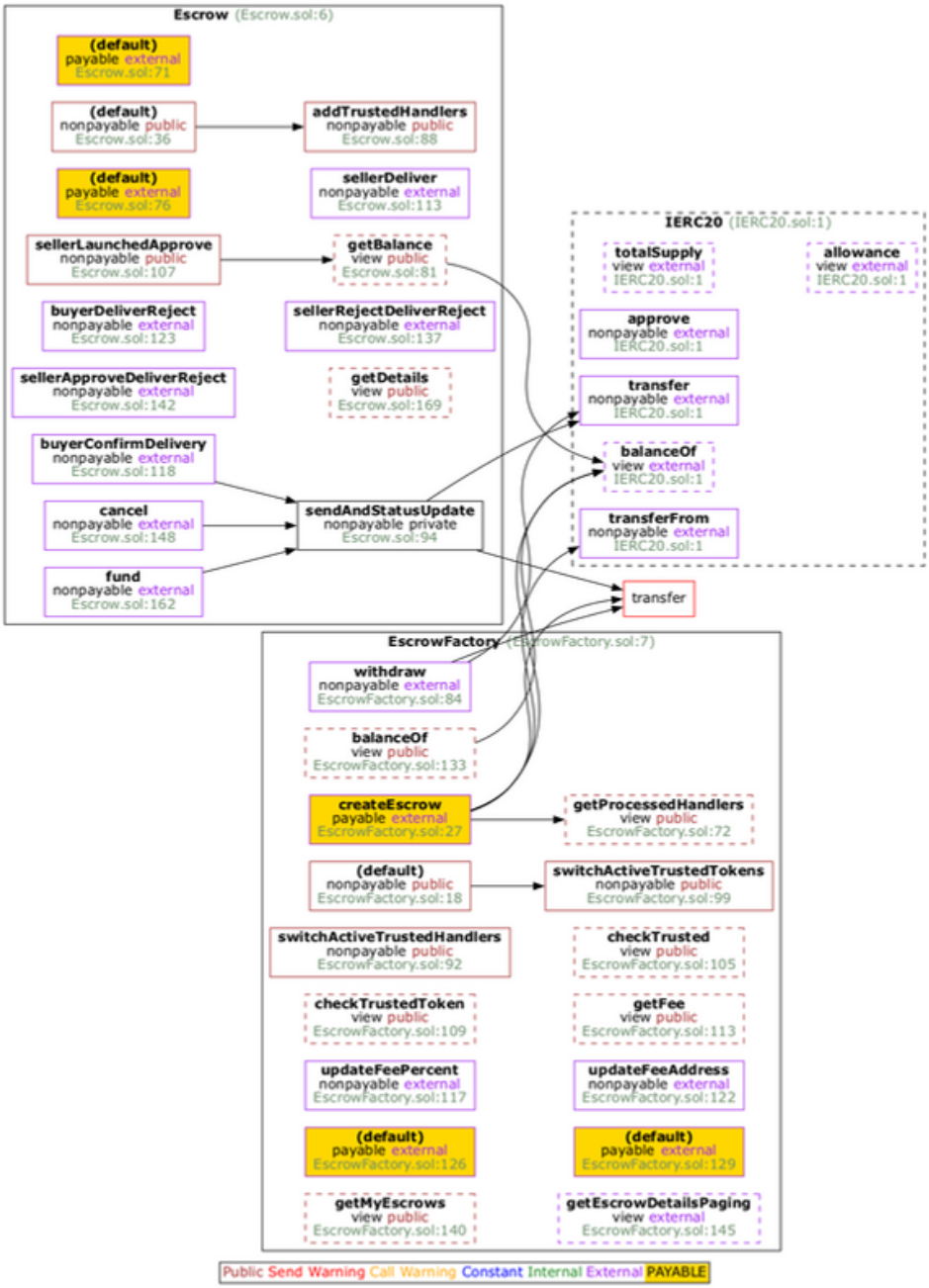


Image 1. Escrow Factory Smart contract call graph

Escrow.sol - Unnecessary states in contract INFORMATIVE

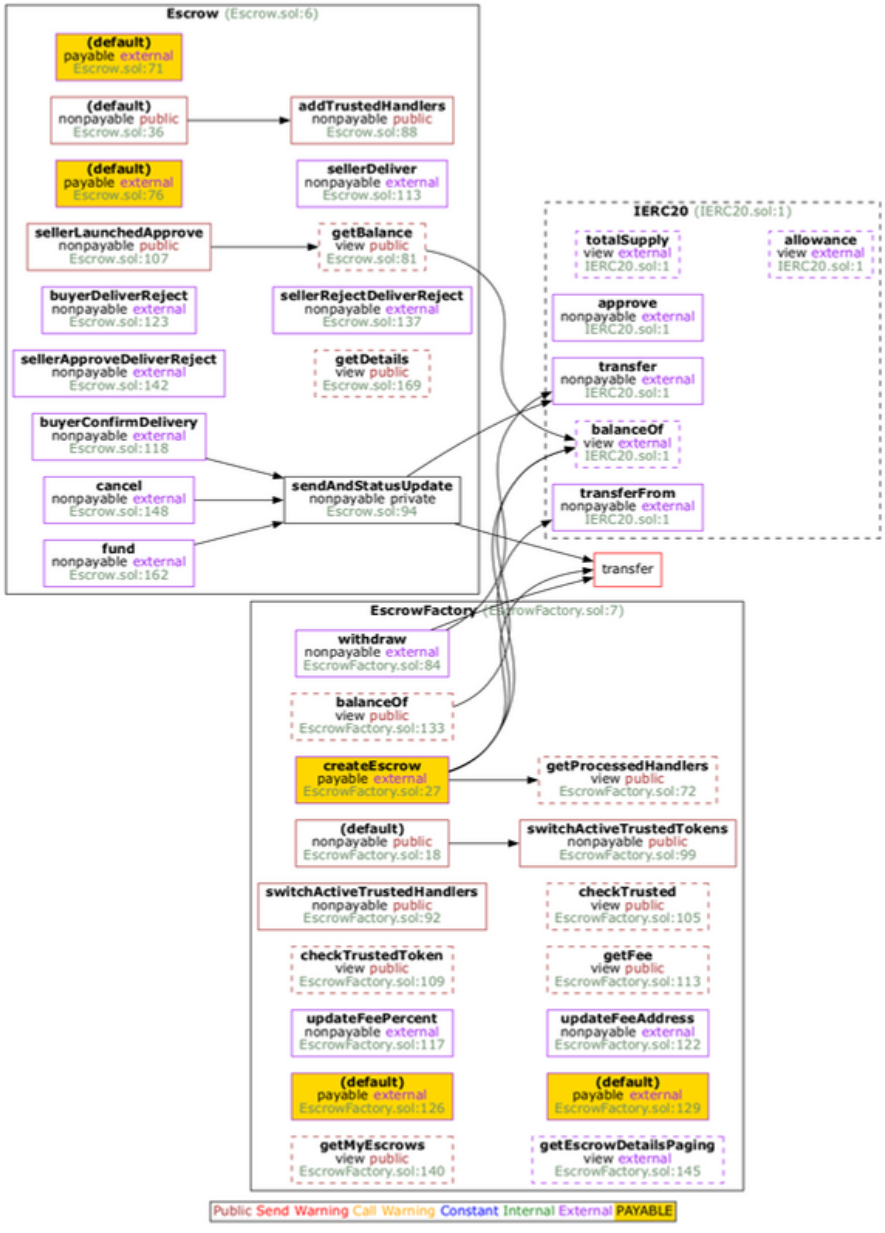
In escrow.sol, the mutable duration, feePercent and deliverRejectDuration states are superfluous. All of them waste gas from the contract.

Unnecessary view function INFORMATIVE

Contracts can directly access to public states, so we do not need to attend public functions to view them.

Unnecessary payable modifier INFORMATIVE

The functions do not receive any ETH, but they continue to function in the absence of the "payable" modifier.



ServiceEscrow.sol - Buyer abuse cancel() function perform theft of seller's amount after delivered INFORMATIVE

After the seller has delivered the products or services, the buyer waits for the deadline to end and cancels the payment. Furthermore, the cancel() function does not check for revised request when buyer rejected delivered status.

Reproduce

After the seller has delivered the products or services, the buyer waits for the deadline to end and cancels the payment. Furthermore, the cancel() function does not check for revised request when buyer rejected delivered status.

Step 1: The buyer waits for the deadline to end when the seller delivered.

Step 2: The buyer calls cancel() function.

Result: The buyer receives the products or services, and the seller does not receive any amount

Do not allow cancel when escrow status is delivered. Furthermore, the contract must compare the revised request deadline to the timestamp

```
function cancel() external {
require(uint8(escrowDetail.status) < 4 && escrowDetail.status !=
EscrowStatus.Delivered, '___NOT_ELIGIBLE___');
   require(msg.sender == escrowDetail.buyer || msg.sender == escrowDetail.seller,
'___INVALID_BUYER_SELLER___');

 if (
msg.sender == escrowDetail.buyer &&
       (escrowDetail.status == EscrowStatus.Ongoing || escrowDetail.status ==
EscrowStatus.RequestRevised)
) {
       require(block.timestamp >= escrowDetail.deadline && block.timestamp >=
escrowDetail.requestRevisedDeadline, '___NOT_EXPIRED___');
}

 sendAndStatusUpdate(escrowDetail.buyer, EscrowStatus.Cancelled); }
```

ServiceEscrow.sol - The fallback() function causes the permanent freezing of the contract's amount INFORMATIVE

Description

The attacker can control the amount by sending arbitrary ETHs to the contract. This led to a permanent freeze on the contract's amount. Both the seller and the buyer are unable to withdraw any funds.

Reproduce

Step 1: The attacker sends 0 (or any) ETH to the contract.

Result: Because the amount is zero (or any), neither the buyer nor the seller can withdraw correct funds from the contract.

```
fallback() external payable {
    escrowDetail.amount += msg.value;
}

receive() external payable {
    escrowDetail.amount += msg.value; //Code Consistency
}
```

ServiceEscrow.sol - Unnecessary usage of SafeMath library INFORMATIVE

All SafeMath usages in the contract are for overflow checking, only usage of SafeMath now is to have a custom revert message which isn't the case in the auditing contracts.

ServiceEscrow.sol - Do not specify a time limit for the duration in constructor INFORMATIVE

The buyer has control over the duration time, so he can pass a small number such as 1. This causes too fast a performance of the deal between buyer and seller.

The duration time buyer passed should be at least one day, according to _deliverRejectDuration variable at _buyerDeliverReject() function.

Unnecessary view function INFORMATIVE

In escrow.sol, the mutable duration, feePercent and deliverRejectDuration states are superfluous. All of them waste gas from the contract

Version	Date	Status/Change	Created by
1.0	Feb 27, 2024	Public Repory	FusionTech

Table 4. Report versions history

APPENDIX

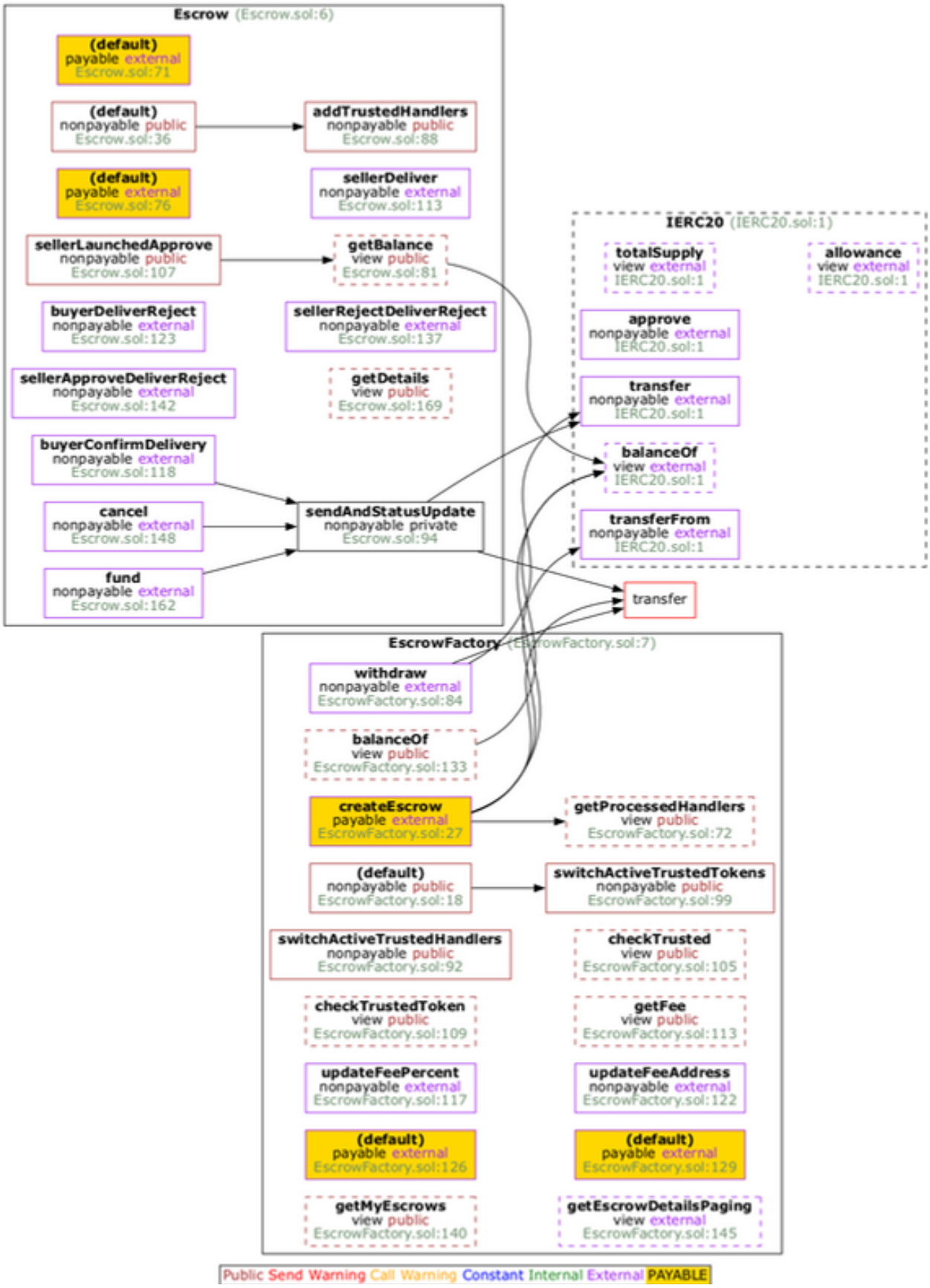


Image 1. Escrow Factory Smart contract call graph

In summary, this escrow system offers a secure, flexible, and customizable escrow solution that caters to a diverse range of transaction types and user requirements. It introduces user-defined release conditions and advanced transaction tracking, ensuring a more versatile escrow experience for all parties involved.

Escrow.sol Contract Cannot Be Paused

Smart contracts can contain vulnerabilities, bugs, or under specific circumstances such as technical issues, it's essential to be able to temporarily stop the execution of the contract to prevent any further exploitation. The pause function allows developers to react quickly to such situations and protect the involved users and assets. This promotes better risk management and enhances user experience. We strongly recommend implementing a pause function.

Uninitialized state

Proper initialization of state variables in a Solidity smart contract is essential to prevent undesired behaviors, ensure security, facilitate interoperability, and improve code readability. This contributes to the contract's robustness, security, and maintainability. For these reasons, it is crucial to correctly initialize the variables Escrow.pause #L25 and Factory.pause #L9 .

Variable Name Error

If we follow the logic, it appears that in the Escrow.sol contract, in the function, the transfer should be made to the address of the the admin's address #L507.

The buyer and seller can request a settlement offer by calling the `settlementRequest` method.

The buyer and seller can initiate an arbitration process by calling the `addDispute` or `addDisputeByToken` methods.

After the inspection period, the seller can claim the funds for the transaction by calling the claim method.

A final claim can be made after one year by the admin of the Factory by calling the superClaim method.

Medium Severity Issues

Arrays Length

The changeConfig function allows the admin to update token statuses and the fees_paid_byvalue. However, there is no validation to ensure that the _tokens and _tokenStatus arrays have the same length. If they have different lengths, this can lead to unexpected behavior. You should add a check to ensure that the lengths are equal.

Low Severity Issues

Shadowing Local Variable

In the changeConfig function, the variable _status#L202 has the same name as a variable present in the ReentrancyGuard.sol contract. In Solidity, it is possible to use the same variable name in different scopes; however, this can potentially lead to undesirable effects. To improve readability and minimize errors, we recommend renaming the variable to _tokensStatus.

Reentrancy

In the addOrderByToken function, even though a reentrancy attack has no interest in this function, it is still preferable to follow the **checks-effects-interactions** pattern to maintain consistency in the code and avoid any potential attack vectors.

Missing Zero Check Address

By checking addresses are not equal to address 0, we avoid transfer errors to an invalid address and prevent the loss of funds. Furthermore, an invalid address such as address 0 can be exploited by attackers to manipulate the smart contract's functionality. It's important to verify that addresses are not equal to address 0 in order to ensure data validity, security, and enhance the contract's robustness.

Escrow.constructor :

- admin = _admin #L92
- affiliation_address = _affiliation_address #L101
- fee_collector = _fee_collector #L94
- arbitrator_address = _arbitrator_address #L93

Escrow.changeAffiliate function :

- affiliation_address = _address #L191

Escrow.changeFeeCollector function :

- fee_collector = _address #L195

Escrow.changeSuperAdmin function :

- super_admin = _address #L183

Escrow.changeArbitrator function :

- arbitrator_address = _address #L187

Factory.constructor :

- arbitrator_address = _arbitrator_address #L21
- fee_collector = _fee_collector #L22

Factory.changeArbitrator

- arbitrator_address = _address #L63

Factory.changeAdmin

- admin = _address #L40

Factory.changeFeeCollector

- fee_collector = _address #L43

After discussing with the development team, adding verification in correlation with the other modifications would have resulted in excessive contract size. In the interest of optimization, this issue is therefore not resolved but also not a security issue here.

Variable Initialization

The initialization of the counter variable before the start of the loop ensures that its value is correct from the beginning, thus avoiding unexpected behaviors or incorrect calculations. As best practice, we recommend to properly initializing the i variable in the Escrow.addToken function #L158.

After discussing with the development team, due to the fact that a uint type variable is initialized by default to the value of 0, this issue has been decided to be marked as solved.

Low Level Calls

When performing a transfer or any other interaction with an external smart contract, it is preferable to use an IERC20 interface to interact with it for several reasons.
Using an IERC20 interface ensures clarity and readability of the code by concisely declaring the specific functions and events of an ERC20 contract. Additionally, it provides enhanced security and reusability by utilizing standardized functions proven by the community. The use of an IERC20 interface is recommended to improves
code quality, security, maintainability, and interoperability of the smart contract.

- Factory.recoverTokens #L68
- Escrow.addOrderByToken #L223

- Escrow.cancelOrder #L285
- Escrow.completeOrder #L313
- Escrow.completeOrder #L316
- Escrow.completeOrder #L319
- Escrow.acceptSettlementRequest #L370
- Escrow.acceptSettlementRequest #L373
- Escrow.acceptSettlementRequest #L376
- Escrow.acceptSettlementRequest #L379
- Escrow.addDisputeByToken #L438
- Escrow.addDisputeByToken #L440
- Escrow.claim #L478
- Escrow.claim #L481
- Escrow.claim #L484
- Escrow.superClaim #L509

Settlement Request Percentage Check

The settlementRequest function allows the buyer or the admin to request a settlement for the order, but it does not have a check to ensure that the _percentage argument is within a valid range.

Notes & Additional Information

Checks-effects-interactions pattern

In these two functions addDisputeByToken and addOrderByToken, the non-reentrancy modifier has been added, however the **checks-effects-interactions** pattern is not followed. Even though the modifier serves as a security measure, it is still preferable to adhere to this pattern to minimize the potential surface for a reentrancy attack.

> After discussing with the development team, the non-reentrancy modifier is deemed sufficient. Therefore, this issue is considered solved.

Naming Convention

Throughout the Escrow.sol smart contract, you use different naming conventions, such as snake_case and camelCase (for example: extensionRequests #L60, order_ids #L57). For better code readability and maintainability, its preferable to use a single and consistent naming convention across all smart contracts.

> After discussing with the development team, this issue is considered resolved.

Immutable States

Immutable variables cannot be modified after their initialization. This ensures their integrity and prevents potential errors from manipulation or accidental modification. Additionally, immutable variables are optimized by the compiler, which can improve performance and reduce gas costs during contract deployment and execution. We recommend declaring the following variables as immutable:

- Escrow.delivery_period #L19
- Escrow.extension_period #L20
- Escrow.request_reply_period #L21
  - **Escrow.admin #L15**
  - **Escrow.factory_contract_address** #L16
  - **Escrow.withdrawal_period #L17**
  - **Escrow.inspection_period #L18**

Error Messages

require statements use short error codes, more descriptive error messages can improve readability and debugging.

After discussing with the development team, with a focus on optimizing contract size, the team has chosen to keep short error codes. A comprehensive documentation will be provided for users. This issue is therefore considered resolved.

Upgradability

The Escrow.sol contract does not have any built-in upgradability features. If you anticipate needing to make changes to the contract's logic after deployment, consider implementing a proxy pattern or using a more modular design that allows individual components to be upgraded.

After discussing with the project team, the topic of contract update was not included in this version. Therefore, this issue is not considered resolved.

Smart Contract Documentation

The Escrow.sol & Factory.sol smart contracts does not provide any documentation. Ensure that the smart contract provide clear and comprehensive documentation to guide users and developers on how to interact with the contract.

Corrected : The development team has chosen to place the documentation outside the smart contracts.

Function To Modifier

You can replace the following functions with modifier to ensure consistency, improved readability, and enhanced maintainability of the code :

- Escrow.validStatus #L128
- Escrow.validOrder #L139
- Escrow.validSettlement #L143

Event Not Emitted

Events provide transparency and traceability by recording important actions on the blockchain. They allow users, third-party applications, and contracts to monitor and react to changes in the contract's state.

We recommend adding the emission of an event for each of the following functions :

- Escrow.changeSuperAdmin #L182
- Escrow.changeArbitrator #L186
- Escrow.changeAffiliate #L190
- Escrow.changeFeeCollector #L194
- Escrow.changeSuperAdminClaimPeriod #L198

Factory.changeAdmin #L39

Factory.changeFeeCollector #L42

Factory.changeArbitrator #L62

> After discussing with the project team, emitting events for these functions was deemed unnecessary. Therefore, this issue is considered resolved.

Duplicated Tokens Data

The token information is stored as a copy during the deployment of an Escrow.sol contract :

- The status can be modified independently of the Factory using the changeConfig function, which is accessible only to the Escrow contract administrator. If desired, in the case where a token is blacklisted by modifying its status in the Factory, this status change will not be automatically reflected in the already deployed Escrow contracts.
- Once the token fees have been initialized at instantiation, it is not possible to update them from the Escrow. Only the Factory can update the fees, and the new configuration will be used only for deploying future Escrow contracts.
- Adding a new token is possible through the changeConfig function, which verifies in the Factory that the token's status is valid. This means that for each token added to the Factory, as long as the administrators of the Escrow contracts have not called the changeConfig function, the token will not be available in their Escrow contract.

Unused/Missing Statuses

The Escrow.Status.SR (SETTLEMENT_REQUEST) status is not defined in the order statuses when calling the Escrow.settlementRequest function.

The Escrow.Status.NA status is not used #L63.

Configuration Update

The configuration functions in the Escrow contract (only the Factory admin), must be called on each deployed Escrow contract in case of modification : Escrow.changeSuperAdmin, Escrow.changeArbitrator, Escrow.changeAffiliate, Escrow.changeFeeCollector, Escrow.changeSuperAdminClaimPeriod.

After discussing with the project team as well as the development team, the chosen behavior is intentional. Therefore, this issue is considered resolved.

Constructor Arguments

The times #L90 parameter is not clearly defined, and index 5 of this parameter is used for fees_paid_by, which does not correspond to a time period. One solution could be to separate the fees_paid_by parameters from the time periods or rename the times parameter to options to avoid any confusion. This would make the code clearer and easier to understand for developers and users of the escrow contract.

Fallback & Receive Functions

The Factory.sol & Escrow.sol contract don't require to receive standalone Ether payments or perform any action when it receives Ether, you don't need to explicitly include the receive function.

On the other hand, the fallback function i used to handle function calls that didn't match any other defined function in the contract. It is considered good practice to define a fallback function that reverts in case of function calls that didn't match any other defined function in the contract. This ensures that any unexpected or invalid function calls to your contract are handled properly.

Test Execution Failures

The following tests failed during our execution :

- Factory.DeployEscrow #L197 “Should deploy escrow with new address”
- Factory.DeployEscrow #L209 : “Should deploy escrow having two token”

Conclusions

In conclusion, the Factory.sol contract doesn’t contain any major issues, only a few minor improvements could be made.

On the other hand, the Escorw.sol contract seems to have three high-severity issues. These issues have been processed. Besides this, the contract is using an up-to-date and appropriate version of the Solidity compiler which includes many safety features.

Visibility specifiers are well defined in all functions which is a good practice and can reduce the chance of unintentional behavior. The contract uses modifiers correctly to handle access control and checks for conditions before execution of function bodies. This helps in reducing code repetition and improving readability.

Revert statements with error messages are used in the contract which is a good practice for debugging and understanding why a transaction might have failed. But it is necessary to provide more explicit error messages and/or comprehensive documentation to clearly and easily identify the errors returned by the smart contract. Given these insights, measures have been taken to prevent any potential reentrancy attack. The "Checks-Effects-Interactions" pattern has been mostly adhered to. Additionally, to ensure the logic of the contract works as intended, further testing should be conducted, including unit testing, integration testing, and stress testing.

Due to the weight of the Escrow.sol smart contract code, some improvements and/or optimizations regarding the code and gas consumption are currently not possible. The development team has already implemented some optimizations; however, it is important to consider this aspect in the context of future improvement and optimization efforts to ensure the best user experience.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanismto relocate funds.

Logical Issue

Logical Issue findingsdetail a fault in the logic of the linked code, such as an incorrect notion on how block.times tamp works.

Volatile Code

Volatile Code findingsrefer to segmentsof code that behave unexpectedly on certain edge cases that may resultin a vulnerability.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather commenton how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings referto functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setterfunction.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexa-decimal encoded and is the same as the output of the Linux "sha256sum" commandagainst the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimerand limitation of liability) set forth in the ServicesAgreement, or the scope of services, and terms and conditions providedto you ("Customer" or the "Company") in connection with the Agreement. This report providedin connection with the Servicesset forth in the Agreementshall be used by the Company only to the extent permittedunder the terms and conditions set forth in the Agreement.
This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes,nor may copiesbe delivered to any otherperson other than the Company,without FusionTech prior written consentin each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts FusionTech to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, businessmodel or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor shouldbe leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. FusionTech's position is that each companyand individual are responsible for their own due diligenceand continuous security.
FusionTech's goal is to help reduce the attack vectorsand the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by FusionTech is subject to dependencies and under continuing development. You agree that your access and/or use, includingbut not limited to any services, reports,and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokensare emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon,multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTSOR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS

AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUTWARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, FusionTech HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, FusionTech SPECIFICALLY DISCLAIMS ALL IMPLIEDWARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, FusionTech MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE,ACCURATE, COMPLETE, FREE OF HARMFULCODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, FusionTech PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDSOR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHERFusionTech NOR ANY OF FusionTech'SAGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. FusionTech WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTINGFROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES,ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED"AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THETHIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSENOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIESBE DELIVERED TO, ANY OTHERPERSON WITHOUT FusionTech'SPRIOR WRITTEN CONSENTIN EACH INSTANCE.

NOTHIRD PARTY OR ANYONE ACTINGON BEHALF OF ANY THEREOF,SHALL BE A THIRD PARTYOR OTHER BENEFICIARY OF SUCH SERVICES,ASSESSMENT REPORT, AND ANY ACCOMPANYING

FORAVOIDANCE OF DOUBT, THE SERVICES,INCLUDING ANY ASSOCIATEDASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2022 by leading academics in the field of Computer Science, FusionTech is going to be a leading blockchain security company that serves to verify the security and correctness of smart contracts KYC and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of ourclients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

FusionTech

https://fusiontech.live

https://twitter.com/fusiontechh

https://t.me/fusiontechofficial

https://github.com/fusiontechofficial