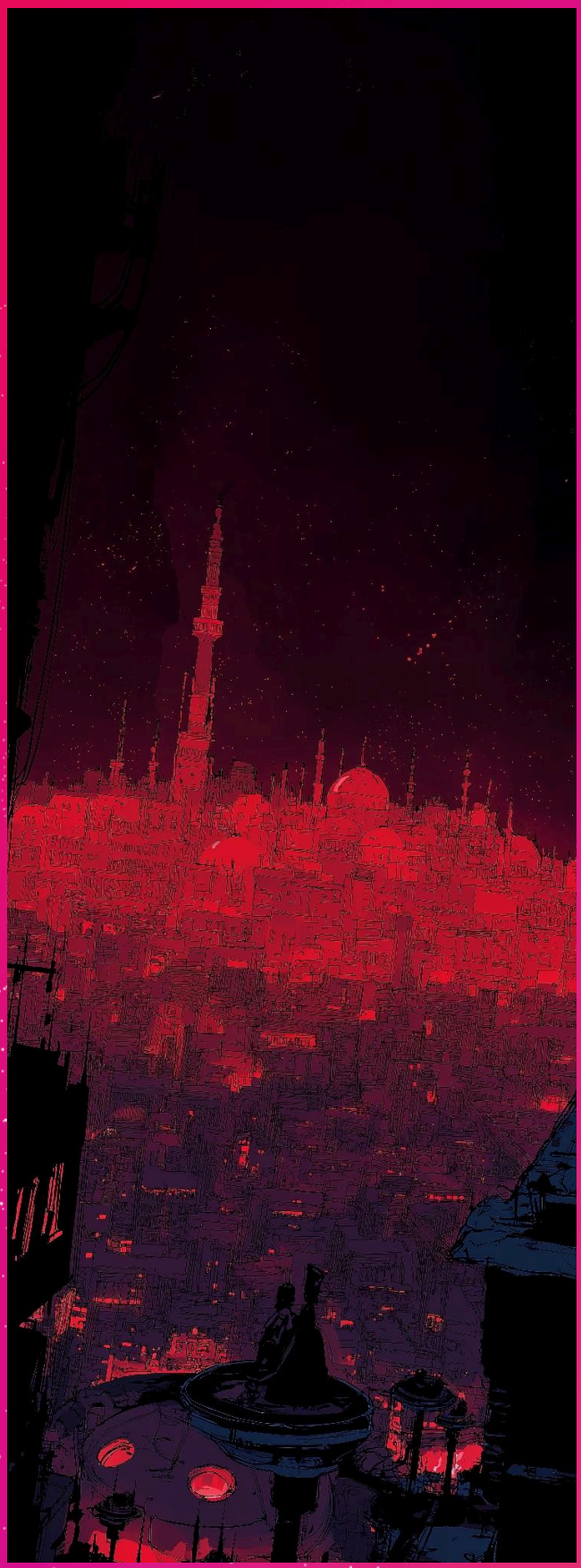
IMMUNEFI AUDIT

Immunefi / Halalfi



DATE November 20, 2025

AUDITOR ExVul

REPORT BY Immunefi

ABOUT IMMUNEFI

Immunefi is the leading onchain security platform, having directly prevented hacks worth more than $25 billion USD. Immunefi security researchers have earned over $120M USD for responsibly disclosing over 4,000 web2 and web3 vulnerabilities, more than the rest of the industry combined.

Through Magnus, Immunefi delivers a comprehensive suite of best-in-class security services through a single command center to more than 300 projects — including Sky (formerly MakerDAO), Optimism, Polygon, GMX, Reserve, Chainlink, TheGraph, Gnosis Chain, Lido, LayerZero, Arbitrum, StarkNet, EigenLayer, AAVE, ZKsync, Morpho, Ethena, USDT0, Stacks, Babylon, Fuel, Sei, Scroll, XION, Wormhole, Firedancer, Jito, Pyth, Eclipse, PancakeSwap and many more.

Magnus unifies SecOps across the entire onchain lifecycle, combining Immunefi's market leading products and community of elite security researchers with a curated set of the very best security products and technologies provided by top security firms — including Runtime Verification, Dedaub, Fuzzland, Nexus Mutual, Failsafe, OtterSec and others.

Magnus is powered by Immunefi's proprietary vulnerabilities dataset — the largest and most comprehensive in web3, ensuring that security leaders and teams have the best possible tools for identifying and mitigating life threats before they cause catastrophic harm, all while reducing operational overhead and complexity.

Learn how you can benefit too at immunefi.com.

TERMINOLOGY

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- **Likelihood** represents the likelihood of a finding to be triggered or exploited in practice
- **Impact** specifies the technical and business-related consequences of a finding
- **Severity** is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

LIKELIHOOD	IMPACT		
	HIGH	MEDIUM	LOW
CRITICAL	Critical	Critical	High
HIGH	High	High	Medium
MEDIUM	Medium	Medium	Low
LOW	Low		
NONE	None		

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

EXECUTIVE SUMMARY

Over the course of 2 days in total, HalalFi engaged with Immunefi to review the https://github.com/mazzirak/halalfi-crowdfunding protocol. In this period of time a total of 11 issues were identified.

SUMMARY

Name	HalalFi
Repository	https://github.com/mazzirak/halalfi-crowdfunding
Audit Commit	3ac0bfbcc58c7f272ded67675520395c0da71e53
Type of Project	DeFi, Crowd Funding
Audit Timeline	Nov 5th - Nov 6th
Fix Period	Nov 7th - Nov 13th

ISSUES FOUND

Severity	Count	Fixed	Acknowledged
Critical	3	3	0
High	2	2	0
Medium	2	2	0
Low	2	2	0
INSIGHT	2	2	0

CATEGORY BREAKDOWN

Bug	10
Gas Optimization	0
Informational	1

FINDINGS

IMM-CRIT-01

Admin contract address mismatch breaks authorization #1

Id	IMM-CRIT-01
Severity	Critical
Category	Bug
Status	Fixed: “Added 7-day withdrawal deadline with markAsDefaulted() function allowing admins to refund investors if creator abandons project”

Description

`CrowdfundFactory` sets admin to an EOA, while `CrowdfundProject`'s `onlyAdmin` expects an `ICrowdfundAdmin` contract (`checkAdmin`). Calling `checkAdmin` on an EOA reverts, breaking all admin-gated operations.
Factory sets admin to deployer EOA:

```
TypeScript
constructor() {
    admin = msg.sender;
}
```

Factory passes that EOA into project constructor:

```
TypeScript
CrowdfundProject project = new CrowdfundProject(
    admin,
    init,
```

```
    USDT_TOKEN
);
```

Project-level onlyAdmin expects admin to be an ICrowdfundAdmin contract address:

```TypeScript
modifier onlyAdmin() {
    require(ICrowdfundAdmin(admin).checkAdmin(msg.sender), CrowdfundErrors.NOT_ADMIN);
    _;
}
```

Impact

- Project admin operations revert (DoS).
- Admin whitelist never applies at project level.
- CrowdfundProject.onlyAdmin modifier is useless.

Recommendation

Deploy projects with `CrowdfundAdmin` as the sole admin and add a `withdrawFees()` function to allow admins to forward collected fees to any recipient.

```TypeScript
// contracts/CrowdfundAdmin.sol
+import "@openzeppelin/contracts/token/ERC20/IERC20.sol";


+function withdrawFees(address token, address to, uint256 amount) external onlyAdmin {
+    require(token != address(0), "Invalid token");
+    require(to != address(0), "Invalid recipient");
+    IERC20(token).safeTransfer(to, amount);
+}
```

TypeScript

```
// contracts/CrowdfundFactory.sol
- constructor() {
-     admin = msg.sender;
- }
+ constructor(address adminContract) {
+     require(adminContract != address(0), "adminContract zero");
+     require(adminContract.code.length > 0, "adminContract must be contract");
+     admin = adminContract;
+ }
```

IMM-CRIT-02

Unrestricted project approval/rejection allows unauthorized state transitions #2

Id	IMM-CRIT-02
Severity	Critical
Category	Bug
Status	Fixed: “Added onlyAdmin modifier enforcing admin-only access through ICrowdfundAdmin.checkAdmin()”

Description

`approveProject` and `rejectProject` were externally callable without any access control. enables any address to transition a project from Pending to Waiting or Pending to Rejected, bypassing the required admin review process.

TypeScript

```
   function approveProject() external override {
        ProjectStorage storage ps = projectStorage;
        require(ps.status == CrowdfundStructs.ProjectStatus.Pending,
CrowdfundErrors.INVALID_STATUS);
        ps.status = CrowdfundStructs.ProjectStatus.Waiting;
        emit ProjectApproved(msg.sender);
    }

    function rejectProject(string memory reason) external override {
        ProjectStorage storage ps = projectStorage;
        require(ps.status == CrowdfundStructs.ProjectStatus.Pending,
CrowdfundErrors.INVALID_STATUS);
        ps.status = CrowdfundStructs.ProjectStatus.Rejected;
        ps.rejectionReason = reason;
        emit ProjectRejected(msg.sender, reason);
    }
```

Impact

Any address can approve or reject projects arbitrarily. Admin review is bypassed.

Recommendation

Restrict `approveProject` and `rejectProject` to admin only. Enforce admin membership via the `ICrowdfundAdmin` contract.

```
TypeScript
   function approveProject() external override {
+     require(msg.sender == admin);
        ProjectStorage storage ps = projectStorage;
        require(ps.status == CrowdfundStructs.ProjectStatus.Pending,
CrowdfundErrors.INVALID_STATUS);
        ps.status = CrowdfundStructs.ProjectStatus.Waiting;
        emit ProjectApproved(msg.sender);
    }

    function rejectProject(string memory reason) external override {
+     require(msg.sender == admin);
        ProjectStorage storage ps = projectStorage;
        require(ps.status == CrowdfundStructs.ProjectStatus.Pending,
CrowdfundErrors.INVALID_STATUS);
        ps.status = CrowdfundStructs.ProjectStatus.Rejected;
        ps.rejectionReason = reason;
        emit ProjectRejected(msg.sender, reason);
    }
```

IMM-CRIT-03

USDT incompatible return value breaks all transfers #3

Id	IMM-CRIT-03
Severity	Critical
Category	Bug
Status	Fixed: “Replaced require(IERC20.transfer()) with OpenZeppelin’s SafeERC20.safeTransfer() throughout all contracts”

Description

Contract guards `IERC20.transfer` and `IERC20.transferFrom` calls with `require`, assuming they return `bool`. However, the intended payment token USDT is non-standard and returns no value. When the contract attempts to decode a non-existent return value as `bool`, the call reverts, makes every token movement fail.

TypeScript

```
function invest(uint256 amount) external payable override nonReentrant {
    // ...

    // Transfer USDT from investor to this contract
    require(
        IERC20(paymentToken).transferFrom(msg.sender, address(this), amount),
        "USDT transfer failed"
    );
}

function claimRefund() external nonReentrant override {
    // ...

    require(IERC20(paymentToken).transfer(msg.sender, amount), "USDT transfer failed");
}
```

Impact

- Complete DoS for core flows (invest, refund, fee transfer, creator withdrawal, returns withdrawal).
- Funds become effectively unusable under the intended token configuration, blocking the protocol's operation.

Recommendation

Use `SafeERC20.safeTransfer()` from OpenZeppelin's SafeERC20 library.

IMM-HIGH-01

Hardcoded chain specific token address breaks multi chain deployment and causes DoS on Ethereum #4

Id	IMM-HIGH-01
Severity	High
Category	Bug
Status	Fixed: “Made paymentToken immutable and configurable per deployment via constructor”

Description

`CrowdfundFactory` hardcodes a payment token address intended to be USDT, but the value is chain specific and not valid on ethereum mainnet. The comment and user facing errors indicate “ETH/BNB not accepted, use USDT only” across networks, so the contract will be deployed on BSC mainnet and ethereum mainnet.

```TypeScript
    function invest(uint256 amount) external payable override nonReentrant {
        ProjectStorage storage ps = projectStorage;

        // Reject any ETH/BNB sent
        require(msg.value == 0, "ETH/BNB not accepted, use USDT only");
        require(amount > 0, "Amount must be greater than 0");
```

`0x55d398326f99059fF775485246999027B3197955` is BUSD token address on BSC.
https://bscscan.com/address/0x55d398326f99059fF775485246999027B3197955
`0x55d398326f99059fF775485246999027B3197955` is not an erc20 token on Ethereum mainnet and address has been blocked by a custodial stablecoin provider (such as USDT and USDC).
https://etherscan.io/address/0x55d398326f99059fF775485246999027B3197955

Impact

- On Ethereum mainnet, the hardcoded address is not USDT, so ERC20 calls (e.g., `transferFrom`, `transfer`) revert, making investments, refunds, fee transfers, returns, and withdrawals impossible.
- Any deployment to other networks will also fail unless the address happens to match a valid ERC20 there.

Recommendation

Make the payment token address configurable per deployment and store it as `immutable`.

```
TypeScript
contract CrowdfundFactory {
    address public admin;
    address[] public allProjects;

-   address public constant USDT_TOKEN = 0x55d398326f99059fF775485246999027B3197955;
+   address public immutable paymentToken;

-   constructor() {
-       admin = msg.sender;
-   }
+   constructor(address _paymentToken, address _admin) {
+       require(_paymentToken != address(0), "Payment token cannot be zero address");
+       require(_admin != address(0), "Admin cannot be zero address");
+       admin = _admin;
+       paymentToken = _paymentToken;
+   }

    function createProject(
        // ...
    ) external returns (address) {
        // ...
        CrowdfundProject project = new CrowdfundProject(
            admin,
            init,
-           USDT_TOKEN
+           paymentToken
        );
```

IMM-HIGH-02

Immutable project admin prevents registry migration #11

Id	IMM-HIGH-02
Severity	High
Category	Bug
Status	Fixed: “Made Admin Mutable in CrowdfundProject

```TypeScript
// Before
address public immutable admin;

// After
address public admin;
2.Added Admin Update Function
function updateAdmin(address newAdmin) external
{
    require(
        msg.sender == admin ||
        msg.sender == factory ||

ICrowdfundAdmin(admin).checkAdmin(msg.sender),
        CrowdfundErrors.NOT_ADMIN
    );
    require(newAdmin != address(0),
CrowdfundErrors.INVALID_ADDRESS);
    require(newAdmin.code.length > 0,
CrowdfundErrors.ADMIN_MUST_BE_CONTRACT);
    require(newAdmin != admin,
CrowdfundErrors.ALREADY_EXISTS);

    address oldAdmin = admin;
    admin = newAdmin;

    emit AdminUpdated(oldAdmin, newAdmin,
msg.sender);
```

```
}
3.Added Batch Migration Functions to Factory
// Update factory admin and optionally migrate
all projects
function updateAdmin(address newAdminContract,
bool migrateExisting) external

// Batch update multiple projects
function batchUpdateProjectAdmin(address[]
calldata projects, address newAdminContract)
external

// Update single project
function updateProjectAdmin(address project,
address newAdminContract) external

// Get projects still using old admin
function getProjectsWithOldAdmin(address
oldAdmin) external view returns (address[]
memory)

4. Added Events for Audit Trail
event AdminUpdated(address indexed oldAdmin,
address indexed newAdmin);
event ProjectAdminUpdated(address indexed
project, address indexed newAdmin);
event AdminUpdated(address indexed oldAdmin,
address indexed newAdmin, address indexed
updater);
```

Description:
all three functions are properly protected
CrowdfundProject.updateAdmin() Multi-Level Authorization
function updateAdmin(address newAdmin) external

Admin contract (msg.sender == admin) - Allows cross-contract calls
Factory contract (msg.sender == factory) - Enables batch migrations
Registered admins (checkAdmin(msg.sender)) - Individual

	admin addresses
	
	the msg.sender == admin approach ONLY
	modifier onlyAdmin() {
	require(
	msg.sender == admin ||
	ICrowdfundAdmin(admin).checkAdmin(msg.sender),
	CrowdfundErrors.NOT_ADMIN
	);
	_;
	}
	The msg.sender == admin check in the modifier handles cross-contract authorization
	Self-registration (isAdmin[address(this)] = true) is unnecessary and creates state bloat”

Description

CrowdfundProject stores admin as immutable and has no updater; Factory updateAdmin only affects future deployments, leaving existing projects pinned to the old registry.

Impact

Old projects cannot move to the new admin registry; governance splits and can stall if the old admin contract is deprecated.

Recommendation

Make project admin mutable and add an onlyAdmin updater. Also change declaration to address public admin;.

```
TypeScript
// contracts/CrowdfundProject.sol
+ function updateAdmin(address newAdmin) external onlyAdmin {
+     require(newAdmin != address(0), CrowdfundErrors.INVALID_ADDRESS);
+     require(newAdmin.code.length > 0, CrowdfundErrors.ADMIN_MUST_BE_CONTRACT);
+     admin = newAdmin;
+ }
```

IMM-MED-01

Skipping withdrawFunds before returnFunds permanently locks raised funds #5

Id	IMM-MED-01
Severity	Medium
Category	Bug
Status	Fixed: “Added prerequisite check: require(hasCreatorWithdrawnFunds)”

Description

`returnFunds()` can be called directly without first calling `withdrawFunds()`, causing the original raised funds to be permanently locked. When `returnFunds()` executes, it sets status to `Completed`, which prevents `withdrawFunds()` from ever executing (it requires `Active` status), leaving the original raised funds and platform fees permanently frozen in the contract.

```
TypeScript
   function withdrawFunds() external nonReentrant onlyCreator {
        require(projectStorage.status == CrowdfundStructs.ProjectStatus.Active,
CrowdfundErrors.PROJECT_NOT_ACTIVE);
        require(block.timestamp >= projectInfo.endDate, CrowdfundErrors.INVALID_DATE);
        require(!hasCreatorWithdrawnFunds, "Funds already withdrawn");

        uint256 balance = IERC20(paymentToken).balanceOf(address(this));
        require(balance > 0, CrowdfundErrors.NOT_ENOUGH_BALANCE);


        uint256 feeAmount = (projectStorage.totalRaised * PLATFORM_FEE_PERCENT) / 100;
        uint256 creatorAmount = balance - feeAmount;

        hasCreatorWithdrawnFunds = true;

        // Transfer fee to admin
        require(IERC20(paymentToken).transfer(admin, feeAmount),
CrowdfundErrors.TRANSFER_FAILED);
```

```
        // Transfer remaining to creator
        require(IERC20(paymentToken).transfer(creator, creatorAmount),
CrowdfundErrors.TRANSFER_FAILED);

        emit FundsWithdrawn(creator, creatorAmount);
        emit PlatformFeeTransferred(admin, feeAmount);
    }

    function returnFunds(uint256 amount) external payable nonReentrant onlyCreator onlyActive
{
        require(msg.value == 0, "ETH/BNB not accepted, use USDT only");
        require(!returnInfo.isReturned, CrowdfundErrors.ALREADY_RETURNED);

        uint256 totalRaised = projectStorage.totalRaised;
        require(amount >= totalRaised, "Must repay at least the raised amount");

        require(
            IERC20(paymentToken).transferFrom(msg.sender, address(this), amount),
            "USDT transfer failed"
        );

        uint256 profit = amount - totalRaised;
        uint256 profitPercentage = (profit * 100) / totalRaised;

        returnInfo = CrowdfundStructs.ProjectReturn({
            returnedAmount: amount,
            profitPercentage: profitPercentage,
            returnDate: block.timestamp,
            isReturned: true
        });

        projectStorage.status = CrowdfundStructs.ProjectStatus.Completed;

        emit FundsReturned(returnInfo.returnedAmount, profitPercentage);
    }
```

Impact

`returnFunds()` transitions to `Completed` without ensuring the original raised funds have been withdrawn first, and there is no recovery path for locked funds after status becomes `Completed`.

Recommendation

Require `withdrawFunds()` to be called before `returnFunds()`.

```
TypeScript
   function returnFunds(uint256 amount) external payable nonReentrant onlyCreator onlyActive
{
        require(msg.value == 0, "ETH/BNB not accepted, use USDT only");
        require(!returnInfo.isReturned, CrowdfundErrors.ALREADY_RETURNED);
+       require(hasCreatorWithdrawnFunds, "Must withdraw funds first");
```

IMM-MED-02

Admin registry mismatch blocks approvals via admin contract #10

Id	IMM-MED-02
Severity	Medium
Category	Bug

Status	Fixed:
	
	“CrowdfundProject.sol
	modifier onlyAdmin() {
	require(ICrowdfundAdmin(admin).checkAdmin(msg.sender
	), CrowdfundErrors.NOT_ADMIN);
	require(
	msg.sender == admin ||
	ICrowdfundAdmin(admin).checkAdmin(msg.sender),
	CrowdfundErrors.NOT_ADMIN
	);
	_;
	}
	
	CrowdfundAdmin.sol
	constructor() {
	admin = msg.sender;
	isAdmin[msg.sender] = true;
	isAdmin[address(this)] = true;
	emit AdminAdded(address(this));
	}”

Description

Description

Project-level onlyAdmin trusts the admin registry, but when CrowdfundAdmin calls approveProject/rejectProject the observed msg.sender is the CrowdfundAdmin contract itself, which is never auto-added to the registry. As a result, every cross-contract approval or rejection reverts with NOT_ADMIN.

```
TypeScript
// contracts/CrowdfundProject.sol
```

```
modifier onlyAdmin() {
    require(ICrowdfundAdmin(admin).checkAdmin(msg.sender), CrowdfundErrors.NOT_ADMIN);
    _;
}
// contracts/CrowdfundAdmin.sol
function approveProject(address project, string calldata reason) external override onlyAdmin
{
    ICrowdfundProject(project).approveProject();
    allProjects.push(project);
    emit ProjectApproved(project, msg.sender, reason);
}
```

Impact
Admin governance flows are frozen: no project can move past Pending via the intended admin contract.
Recommend

Recommendation

Register CrowdfundAdmin itself as an admin (constructor) or let onlyAdmin accept calls whose msg.sender equals the admin contract, ensuring cross-contract approvals succeed.

TypeScript

```
// contracts/CrowdfundProject.sol
modifier onlyAdmin() {
-    require(ICrowdfundAdmin(admin).checkAdmin(msg.sender), CrowdfundErrors.NOT_ADMIN);
+    require(
+        msg.sender == admin || ICrowdfundAdmin(admin).checkAdmin(msg.sender),
+        CrowdfundErrors.NOT_ADMIN
+    );
    _;
}
// contracts/CrowdfundAdmin.sol
constructor() {
    admin = msg.sender;
    isAdmin[msg.sender] = true;
+    isAdmin[address(this)] = true;
```

```
+    emit AdminAdded(address(this));
}
```

IMM-LOW-01

Boundary time overlap allows fund lock after withdrawal #6

Id	IMM-LOW-01
Severity	LOW
Category	Bug
Status	Fixed: “Changed withdrawFunds() from = to endDate, creating exclusive time windows”

Description

Time boundary checks in `invest()` and `withdrawFunds()` overlap at endDate, enabling a race condition where funds can be locked. If `withdrawFunds()` executes first at `block.timestamp == endDate`, it sets `hasCreatorWithdrawnFunds = true`, but `invest()` can still accept investments at the same timestamp, and those funds become permanently locked as they cannot be withdrawn.

```TypeScript
    function invest(uint256 amount) external payable override nonReentrant {
        ProjectStorage storage ps = projectStorage;

        // Reject any ETH/BNB sent
        require(msg.value == 0, "ETH/BNB not accepted, use USDT only");
        require(amount > 0, "Amount must be greater than 0");
        require(ps.status == CrowdfundStructs.ProjectStatus.Waiting || ps.status ==
CrowdfundStructs.ProjectStatus.Active, "Project not available for investment");
        require(block.timestamp >= projectInfo.startDate, "Investment not started");
        require(block.timestamp <= projectInfo.endDate, "Investment period ended");
```

```TypeScript
   function withdrawFunds() external nonReentrant onlyCreator {
        require(projectStorage.status == CrowdfundStructs.ProjectStatus.Active,
CrowdfundErrors.PROJECT_NOT_ACTIVE);
        require(block.timestamp >= projectInfo.endDate, CrowdfundErrors.INVALID_DATE);
        require(!hasCreatorWithdrawnFunds, "Funds already withdrawn");

        uint256 balance = IERC20(paymentToken).balanceOf(address(this));
        require(balance > 0, CrowdfundErrors.NOT_ENOUGH_BALANCE);


        uint256 feeAmount = (projectStorage.totalRaised * PLATFORM_FEE_PERCENT) / 100;
        uint256 creatorAmount = balance - feeAmount;

        hasCreatorWithdrawnFunds = true;
```

Recommendation

Use strict inequalities to create mutually exclusive time windows.

```TypeScript
   function withdrawFunds() external nonReentrant onlyCreator {
        require(projectStorage.status == CrowdfundStructs.ProjectStatus.Active,
CrowdfundErrors.PROJECT_NOT_ACTIVE);
-       require(block.timestamp >= projectInfo.endDate, CrowdfundErrors.INVALID_DATE);
+       require(block.timestamp > projectInfo.endDate, CrowdfundErrors.INVALID_DATE);
        require(!hasCreatorWithdrawnFunds, "Funds already withdrawn");

        uint256 balance = IERC20(paymentToken).balanceOf(address(this));
        require(balance > 0, CrowdfundErrors.NOT_ENOUGH_BALANCE);


        uint256 feeAmount = (projectStorage.totalRaised * PLATFORM_FEE_PERCENT) / 100;
        uint256 creatorAmount = balance - feeAmount;

        hasCreatorWithdrawnFunds = true;
```

IMM-LOW-02

Insufficient parameter validation allows nonsensical projects #7

Id	IMM-LOW-02
Severity	LOW
Category	Bug
Status	Fixed: “Added comprehensive validation in createProject() : startDate >= block.timestamp raiseAmount > 0 minInvestment > 0”

Description

Missing key checks: `startDate >= block.timestamp`, `raiseAmount > 0`, `minInvestment > 0`.

```
TypeScript
require(endDate > startDate, CrowdfundErrors.INVALID_DATE);
require(returnNotifyDate > endDate, CrowdfundErrors.INVALID_DATE);
require(minInvestment <= maxInvestment, CrowdfundErrors.INVALID_AMOUNT);
require(maxInvestment <= raiseAmount, CrowdfundErrors.INVALID_AMOUNT);
```

Impact

Projects may be created expired or non investable (`raiseAmount == 0`, `minInvestment == 0`).

Adds invalid on-chain states and handling complexity.

Recommendation

Add parameter constraints.

```
TypeScript
// contracts/CrowdfundFactory.sol
```

```
function createProject(
    string memory title,
    string memory description,
    CrowdfundStructs.Document[] memory documents,
    uint256 startDate,
    uint256 endDate,
    uint256 returnNotifyDate,
    uint256 raiseAmount,
    uint256 minInvestment,
    uint256 maxInvestment
) external returns (address) {
+    require(startDate >= block.timestamp, CrowdfundErrors.INVALID_DATE);
+    require(raiseAmount > 0, CrowdfundErrors.INVALID_AMOUNT);
+    require(minInvestment > 0, CrowdfundErrors.INVALID_AMOUNT);
     require(endDate > startDate, CrowdfundErrors.INVALID_DATE);
     require(returnNotifyDate > endDate, CrowdfundErrors.INVALID_DATE);
     require(minInvestment <= maxInvestment, CrowdfundErrors.INVALID_AMOUNT);
     require(maxInvestment <= raiseAmount, CrowdfundErrors.INVALID_AMOUNT);
     // ...
}
```

IMM-INSIGHT-01

Inconsistent raiseAmount sources enable premature state transitions #8

Id	IMM-INSIGHT-01
Severity	INSIGHT
Category	Bug
Status	Fixed “Added explicit validation: require(init.raiseAmount <= type(uint128).max)”

Description

`raiseAmount` is stored as `uint128` in `ProjectStorage` (downcast from `uint256`), but investment cap checks use `projectInfo.raiseAmount` (uint256). State transitions and post endDate status determination use the truncated `ps.raiseAmount`. When `init.raiseAmount > type(uint128).max`, the value is silently truncated, causing investment cap enforcement and state activation logic to diverge.

```TypeScript
    struct ProjectStorage {
        CrowdfundStructs.ProjectStatus status;
        uint128 totalRaised;
        uint128 raiseAmount;
        string rejectionReason;
    }

    CrowdfundStructs.ProjectInfo private projectInfo;
    ProjectStorage private projectStorage;

    constructor(
        address _admin,
        CrowdfundStructs.ProjectInfo memory init,
        address _paymentToken
    ) {
        // ...
        _initProjectInfo(init);
```

```
        projectStorage = ProjectStorage({
            status: CrowdfundStructs.ProjectStatus.Pending,
            totalRaised: 0,
            raiseAmount: uint128(init.raiseAmount),
            rejectionReason: ""
        });
        hasCreatorWithdrawnFunds = false;
    }

    function _initProjectInfo(CrowdfundStructs.ProjectInfo memory init) internal {
        CrowdfundStructs.ProjectInfo storage pi = projectInfo;
        pi.creator = init.creator;
        pi.title = init.title;
        pi.description = init.description;
        pi.startDate = init.startDate;
        pi.endDate = init.endDate;
        pi.returnNotifyDate = init.returnNotifyDate;
        pi.raiseAmount = init.raiseAmount;
        pi.minInvestment = init.minInvestment;
        pi.maxInvestment = init.maxInvestment;
        for (uint256 i = 0; i < init.documents.length; i++) {
            pi.documents.push(init.documents[i]);
        }
    }
```

Recommendation

add explicit validation.

TypeScript

```
   constructor(
        address _admin,
        CrowdfundStructs.ProjectInfo memory init,
        address _paymentToken
    ) {
        require(init.endDate > init.startDate, CrowdfundErrors.INVALID_DATE);
        require(init.returnNotifyDate > init.endDate, CrowdfundErrors.INVALID_DATE);
        require(init.minInvestment <= init.maxInvestment, CrowdfundErrors.INVALID_AMOUNT);
        require(init.maxInvestment <= init.raiseAmount, CrowdfundErrors.INVALID_AMOUNT);
```

```
        require(_paymentToken != address(0), "Payment token cannot be zero address");

        admin = _admin;
        factory = msg.sender;
        creator = init.creator;
        paymentToken = _paymentToken;

        _initProjectInfo(init);

+       require(init.raiseAmount <= type(uint128).max, "raiseAmount exceeds uint128 max");
        projectStorage = ProjectStorage({
            status: CrowdfundStructs.ProjectStatus.Pending,
            totalRaised: 0,
            raiseAmount: uint128(init.raiseAmount),
            rejectionReason: ""
        });
        hasCreatorWithdrawnFunds = false;
    }
```

IMM-INSIGHT-02

Unused onlyAdmin modifier in factory #9

Id	IMM-INSIGHT-02
Severity	INSIGHT
Category	Informational
Status	Fixed: “Removed”

Description

`onlyAdmin` modifier is defined but never used at `CrowdfundFactory`.

```
TypeScript
   modifier onlyAdmin() {
        require(msg.sender == admin, "Only admin can call this function");
        _;
    }
```

Recommendation

Remove if unused.