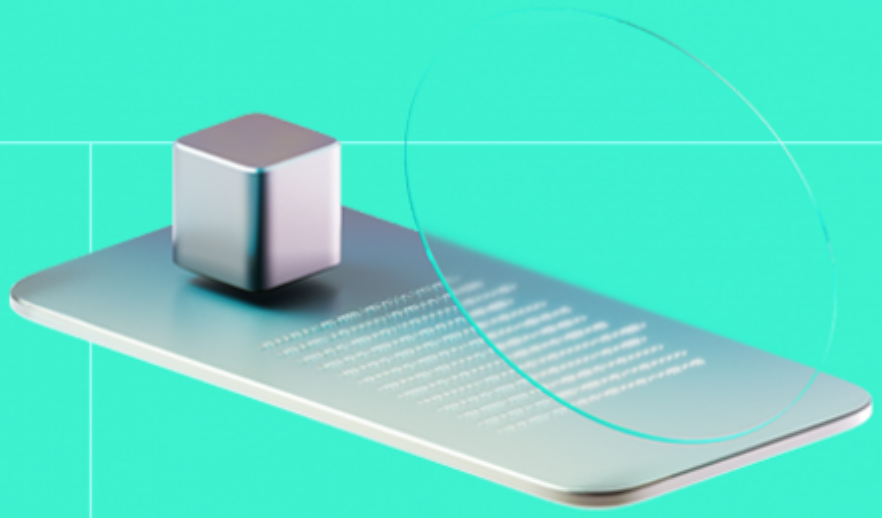
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: OffBlocks

Date: 19/04/2024

We express our gratitude to the OffBlocks team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

OffBlocks is a hybrid platform for digital assets spending, helps wallet and DeFi companies offer trustless crypto-to-fiat experiences as well as payments and card services.

Platform: EVM

Language: Solidity

Tags: Escrow, Smart Wallet, Factory

Timeline: 22/03/2024 - 19/04/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/OffBlocks/offblocks-contracts
Commit	fef94ee4963c49d53c7d217cd6e3e7d5860ec9c3

Audit Summary

10/10	10/10	80.24%	10/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 9.3/10

The system users should acknowledge all the risks summed up in the risks section of the report

4	4	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	1
High	0
Medium	3
Low	0

Vulnerability	Status
F-2024-1695 - The rescueTokens functionn allows owner to drain all tokens from escrow	Fixed
F-2024-1722 - The batchUpdateDepositStatuses function does not updates statuses	Fixed
F-2024-1736 - Lack of deposit status update in reject and settle functions	Fixed
F-2024-1743 - Unchecked transfers of ERC20 tokens	Fixed

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for OffBlocks
Audited By	Niccolò Pozzolini, Kornel Światłowski
Approved By	Przemyslaw Swiatowiec
Website	https://www.offblocks.xyz/
Changelog	27/03/2024 - Preliminary Report
	19/04/2024 - Final Report

Table of Contents

System Overview

OffBlocksEscrow - contract handles token deposits and withdrawals coming from smart wallets, specifically designed for the OffBlocks platform. It supports deposit operations, deposit status updates, withdrawals for rejected deposits, and settlement of approved deposits.

OffBlocksSmartWalletFactory - factory contract for OffBlocks smart wallets. Contract owner can also register and unregister third-party smart wallets that will be allowed to interact with the OffBlocksEscrow contract.

OffBlocksSmartWallet - smart wallet contract deployed within OffBlocksSmartWalletFactory.

PendingWithdrawal - helper contract linked deployed during the creation of each OffBlocksSmartWallet. It manages the withdrawal of tokens from OffBlocksSmartWallet after a declared delay.

Privileged roles

Owner of OffBlocksEscrow can:

- Adds and removes a token to the list of supported tokens.
- Updates the address of the OffBlocks smart wallet factory contract.
- Reduces the amount of the given deposit.
- Batch updates the statuses of the deposits.
- Withdraws the rejected deposit from the escrow to the depositor.
- Transfers the approved deposit from the escrow to the settlement wallet.
- Pause and unpause contract.
- Rescues ERC20 tokens sent to the contract by mistake

Owner of OffBlocksSmartWallet can:

- Initiates a token withdrawal.
- Claims the token withdrawal.

Owner of OffBlocksSmartWalletFactory can:

- Creates a new OffBlocks smart wallet.
- Changes the withdrawal delay.
- Registers and unregisters multiple third-party smart wallets.
- Rescues ERC20 tokens sent to the contract by mistake.

Owner of PendingWithdrawal can:

- Initiates a token withdrawal.
- Claims the token withdrawal.
- Cancels withdrawal of a token.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are complete.
- Technical description is provided.

Code quality

The total Code Quality score is **10** out of **10**.

Test coverage

Code coverage of the project is **80.24%** (branch coverage).

- Negative cases coverage is missed.

Security score

Upon auditing, the code was found to contain **1** critical, **0** high, **3** medium, and **0** low severity issues. All issues have been fixed, leading to a security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **9.3**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

No additional risks were identified.

Findings

Vulnerability Details

F-2024-1722 - The batchUpdateDepositStatuses function does not updates statuses - Critical

Description:

The `batchUpdateDepositStatuses()` function is designed to update the statuses of provided deposits. The contract owner can apply `Rejected` or `Approved` statuses to given deposits. Updating these statuses is necessary for executing `settle()` and `reject()` functions, which are responsible for releasing tokens to users. However, the function fails to update the statuses of provided deposit IDs. Instead, it creates a copy of a `userDeposit` with the `memory` keyword, performs checks, applies the new status to the `userDeposit` memory variable, and does not update the related storage variable holding deposit information.

This lead to situation where fundamental functions of `OffBlocksEscrow` contract does not work properly.

```
function batchUpdateDepositStatuses(uint256[] calldata _depositIds,
uint256 _status
) external onlyOwner {
if (_depositIds.length == 0) revert ArrayLengthOfZero();
if (
_status != uint256(DepositStatus.Approved) &&
_status != uint256(DepositStatus.Rejected)
) revert InvalidStatus(_status);

for (uint256 i = 0; i < _depositIds.length; i++) {
//@audit copy values from storage to memory type variable
Deposit memory userDeposit = getDepositById(_depositIds[i]);

if (userDeposit.status != DepositStatus.Pending)
revert DepositNotPending();

//@audit memory type variable is updated with new status
userDeposit.status = DepositStatus(_status);

//@audit storage variable is not updated, status is not applied
emit DepositStatusUpdated(
_depositIds[i],
DepositStatus(_status),
block.timestamp
);
}
}
```

Assets:

- File: contracts/core/OffBlocksEscrow.sol [https://github.com/OffBlocks/offblocks-contracts]

Status: Fixed

Classification

Severity: Critical

Impact:
Likelihood [1-5]: 5
Impact [1-5]: 5
Exploitability [0-2]: 0
Complexity [0-2]: 1
Final Score: 4.8 (Critical)
Hacken Calculator Version: 0.6

Recommendations

Remediation: It is recommended to revise the `batchUpdateDepositStatuses()` function to ensure that the statuses of provided deposit IDs are properly updated in the storage variables holding deposit information (`depositById` mapping).

Remediation (commit: 4932655): The `Approved` and `Rejected` statuses are correctly applied in the `batchUpdateDepositStatuses()` function.

Evidences

PoC

Reproduce:

```
import { loadFixture } from "@nomicfoundation/hardhat-toolbox/networ
k-helpers";
import { expect } from "chai";
import { ethers, upgrades } from "hardhat";

describe("poc", function () {
const delay = 15;

async function deployFactoryWithWalletsFixture() {
const [owner, acc1, rec1] = await ethers.getSigners();
const OffBlocksSmartWallet = await ethers.getContractFactory("OffBlo
cksSmartWallet");

const MockToken = await ethers.getContractFactory("MockToken");
const erc20Mock = await MockToken.deploy(
"MockToken1",
"MockToken1",
18,
0,
);

const SmartWalletFactory = await ethers.getContractFactory(
"OffBlocksSmartWalletFactory",
);
const smartWalletFactory = await upgrades.deployProxy(
SmartWalletFactory,
[owner.address, delay],
{ initializer: "initialize" },
);

const Escrow = await ethers.getContractFactory("OffBlocksEscrow");
const escrow = await upgrades.deployProxy(
Escrow,
[smartWalletFactory.target, [erc20Mock.target], owner.address],
{ initializer: "initialize" },
);
```

```
const tx = await smartWalletFactory.connect(owner).deploySmartWallet
(
acc1.address,
escrow.target,
smartWalletFactory.target,
0
);
const receipt = await tx.wait();
const sw1Address = await receipt.logs[2].args[1];
const sw1 = OffBlocksSmartWallet.attach(sw1Address);

await erc20Mock.mint(acc1.address, ethers.parseEther("100"));
return { erc20Mock, smartWalletFactory, escrow, owner, acc1, sw1, re
c1 };
}

describe("batchUpdateDepositStatuses()", function () {
it("Status is not updated", async function () {
const { erc20Mock, escrow, owner, acc1, sw1, rec1 } = await loadFixt
ure(deployFactoryWithWalletsFixture);

const nonce = 0;
const depositAmount = ethers.parseEther("10");

await erc20Mock.connect(acc1).transfer(sw1.target, depositAmount);

const message =
```

See more

Results:

```
poc
batchUpdateDepositStatuses()
> Status before: 0n
Deposit before: Result(9) [
1n,
'0x3FF3F9F6b31b1691F72b8639860aD73e957C13AE',
'0x5FbDB2315678afecb367f032d93F642f64180aa3',
10000000000000000000n,
Result(1) [ 10000000000000000000n ],
Result(1) [ '0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC' ],
0n,
1711445067n,
1711445067n
]
> Status after : 0n
Deposit after: Result(9) [
1n,
'0x3FF3F9F6b31b1691F72b8639860aD73e957C13AE',
'0x5FbDB2315678afecb367f032d93F642f64180aa3',
10000000000000000000n,
Result(1) [ 10000000000000000000n ],
Result(1) [ '0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC' ],
0n,
1711445067n,
1711445067n
]
```

F-2024-1695 - The rescueTokens functionn allows owner to drain all tokens from escrow - Medium

Description:

The `OffBlocksEscrow` contract functions as an escrow and is designed to accept, hold, and release tokens with dedicated Smart Wallets. According to the NatSpec of `rescueTokens()` function: `Rescues ERC20 tokens sent to the contract by mistake`. However, the contract does not keep track of the sum of deposited tokens by users and allows the contract owner to drain all tokens deposited by users.

```
/**
* @notice Rescues ERC20 tokens sent to the contract by mistake
* @param _token address - The address of the token to be rescued
* @dev Only owner can call this function
* @dev Emits a {RescueTokens} event
*/
function rescueTokens(address _token) external onlyOwner {
if (_token == address(0)) revert ZeroAddress();
IERC20 token = IERC20(_token);

uint256 balance = token.balanceOf(address(this));
if (balance == 0) revert ZeroTokens();

//@audit whole contract balance is transfered to owner address
bool success = token.transfer(owner(), balance);
if (!success) revert ERC20TransferFailed();

emit RescueTokens(owner(), _token, balance);
}
```

Assets:

- File: contracts/core/OffBlocksEscrow.sol [https://github.com/OffBlocks/offblocks-contracts]

Status: Fixed

Classification

Severity: Medium

Impact:

Likelihood [1-5]: 2
Impact [1-5]: 5
Exploitability [0-2]: 1
Complexity [0-2]: 0
Final Score: 2.7 (Medium)
Hacken Calculator Version: 0.6

Recommendations

Remediation:

It is recommended to track the sum of deposited tokens by users and update the `rescueTokens()` function so that only tokens sent by mistake

can be withdrawn via this function.

Remediation (commit: 4f97593): The `currentEscrowBalanceDepositedByUsers` mapping has been added to track the sum of deposited tokens by users. The `rescueTokens()` will withdraw only the redundant tokens deposited by mistake.

Evidences

PoC

Reproduce:

```
import { loadFixture } from "@nomicfoundation/hardhat-toolbox/networ
k-helpers";
import { expect } from "chai";
import { ethers, upgrades } from "hardhat";
import { smartWalletAbi } from "../utils/abis";

describe("poc", function () {

const delay = 15;

async function deployFactoryWithWalletsFixture() {
const [owner, proxyAdmin, acc1, acc2, acc3] = await ethers.getSigner
s();
const OffBlocksSmartWallet = await ethers.getContractFactory("OffBlo
cksSmartWallet");

const MockToken = await ethers.getContractFactory("MockToken");
const erc20Mock = await MockToken.deploy(
"MockToken1",
"MockToken1",
18,
0,
);

const SmartWalletFactory = await ethers.getContractFactory(
"OffBlocksSmartWalletFactory",
);
const smartWalletFactory = await upgrades.deployProxy(
SmartWalletFactory,
[owner.address, delay],
{ initializer: "initialize" },
);

const Escrow = await ethers.getContractFactory("OffBlocksEscrow");
const escrow = await upgrades.deployProxy(
Escrow,
[smartWalletFactory.target, [erc20Mock.target], owner.address],
{ initializer: "initialize" },
);

const tx = await smartWalletFactory.connect(owner).deploySmartWallet
(
acc1.address,
escrow.target,
smartWalletFactory.target,
0
);
const receipt = await tx.wait();
const sw1Address = await receipt.logs[2].args[1];
const sw1 = OffBlocksSmartWallet.attach(sw1Address);

const tx2 = await smartWalletFactory.connect(owner).deploySmartWalle
t(
acc2.address,
escrow.target,
smartWalletFactory.target,
0
);
const receipt2 = await tx2.wait();
```

```
const sw2Address = await receipt2.logs[2].args[1];
const sw2 = OffBlocksSmartWallet.attach(sw2Address);

const tx3 = await smartWalletFactory.connect(owner).deploySmartWalle
t(
acc3.address,
escrow.target,
smartWalletFactory.target,
0
```

See more

Results:

```
poc
Owner can steal funds
ERC20 balance of escrow contract before: 10000000000000000000n
ERC20 balance of owner address before: 0n
ERC20 balance of escrow contract after: 0n
ERC20 balance of owner address after: 10000000000000000000n
✔ PASS (9085ms)
```

F-2024-1736 - Lack of deposit status update in reject and settle functions - Medium

Description:

In the `OffBlocksEscrow.sol` file, the `reject`, `settle`, and `batchSettle` functions have a flaw that could lead to the escrow being drained if the escrow owner wallet is compromised.

The mentioned three functions check the current deposit status but do not update it after the function is called. This means that if the owner's account is compromised, an attacker could repeatedly call these functions, causing funds to flow out of the escrow to an attacker-controlled wallet with each invocation.

```
function reject(uint256 _depositId) external nonReentrant onlyOwner
{
Deposit memory userDeposit = getDepositById(_depositId);

if (userDeposit.status != DepositStatus.Rejected)
revert DepositNotRejected();

bool success = IERC20(userDeposit.token).transfer(
userDeposit.depositor,
userDeposit.amount
);
if (!success) revert ERC20TransferFailed();

emit Withdrawal(...);
}


function settle(uint256 _depositId) external nonReentrant onlyOwner
{
Deposit memory userDeposit = getDepositById(_depositId);

if (userDeposit.status != DepositStatus.Approved)
revert DepositNotApproved();

for (uint256 j = 0; j < userDeposit.splitAddresses.length; j++) {
bool success = IERC20(userDeposit.token).transfer(
userDeposit.splitAddresses[j],
userDeposit.splitAmounts[j]
);
if (!success) revert ERC20TransferFailed();
}

emit Settlement(...);
}


function batchSettle(uint256[] calldata _depositIds) external onlyOw
ner {
if (_depositIds.length == 0) revert ArrayLengthOfZero();

for (uint256 i = 0; i < _depositIds.length; i++) {
Deposit memory userDeposit = getDepositById(_depositIds[i]);

if (userDeposit.status != DepositStatus.Approved)
revert DepositNotApproved();

for (uint256 j = 0; j < userDeposit.splitAddresses.length; j++) {
bool success = IERC20(userDeposit.token).transfer(
userDeposit.splitAddresses[j],
userDeposit.splitAmounts[j]
);
if (!success) revert ERC20TransferFailed();
}

emit Settlement(...);
```

```
}
}
```

Assets:

- File: contracts/core/OffBlocksEscrow.sol [https://github.com/OffBlocks/offblocks-contracts]

Status: Fixed

Classification

Severity: Medium

Impact:

Likelihood [1-5]: 2
Impact [1-5]: 5
Exploitability [0-2]: 0
Complexity [0-2]: 0
Final Score: 3.5 (Medium)
Hacken Calculator Version: 0.6

Recommendations

Remediation:

To fix the issue, it is recommended to implement two new deposit statuses (e.g. `RejectedExecuted` and `AppovedExecuted`), and update the deposit status after the `reject` and `settle` functions are invoked. This would ensure that subsequent invocations would fail, preventing the potential draining of the escrow.

Remediation (commit: aa88adf): The `ApprovedExecuted` and `RejectedExecuted` statuses have been added, and they are applied in the `reject()`, `settle()`, and `batchSettle()` functions.

F-2024-1743 - Unchecked transfers of ERC20 tokens - Medium

Description:

The analysis identified that there are omitted verifications for the return values of ERC20 transfer functions. This oversight can lead to vulnerabilities since certain tokens might deviate from the ERC20 standards, either by returning `false` upon a transfer failure or by not issuing any return value whatsoever.

The most popular way to handle such tokens is by utilizing the `SafeERC20` library,

The following functions are affected:

- OffBlocksEscrow: deposit(), topUpDeposit(), reduceDeposit(), reject(), settle(), batchSettle(), rescueTokens()
- PendingWithdrawal: cancelTokenWithdrawal(), claimTokenWithdrawal()
- OffBlocksSmartWallet: initiateTokenWithdrawal()
- OffBlocksSmartWalletFactory: rescueTokens()

What is more, the function `OffBlocksSmartWallet.approveTokensToEscrow` invokes `token.approve` which should be replaced by its safe version `SafeERC20.safeIncreaseAllowance` to make it compatible with non-standard tokens like USDT (USDT requires the allowance to be set to 0 before being changed)

Assets:

- File: contracts/core/OffBlocksEscrow.sol [https://github.com/OffBlocks/offblocks-contracts]
- File: contracts/core/OffBlocksSmartWallet.sol [https://github.com/OffBlocks/offblocks-contracts]
- File: contracts/core/OffBlocksSmartWalletFactory.sol [https://github.com/OffBlocks/offblocks-contracts]
- File: contracts/core/PendingWithdrawal.sol [https://github.com/OffBlocks/offblocks-contracts]

Status: Fixed

Classification

Severity: Medium

Impact:

Likelihood [1-5]: 3
Impact [1-5]: 3
Exploitability [0-2]: 0
Complexity [0-2]: 0

Final Score: 3.0 (Medium)
Hacken Calculator Version: 0.6

Recommendations

Remediation: Consider using `SafeERC20` library in the aforementioned functions or implement a custom check for the result of the ERC20 transfer.

Remediation (commit: bf41e08): The `SafeERC20` library has been implemented and used when tokens are transferred or approved.

Observation Details

F-2024-1680 - Floating Pragma - Info

Description:

The project uses floating pragmas `^0.8.21`

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Assets:

- File: contracts/core/OffBlocksEscrow.sol [https://github.com/OffBlocks/offblocks-contracts]
- File: contracts/core/OffBlocksSmartWallet.sol [https://github.com/OffBlocks/offblocks-contracts]
- File: contracts/core/OffBlocksSmartWalletFactory.sol [https://github.com/OffBlocks/offblocks-contracts]
- File: contracts/core/PendingWithdrawal.sol [https://github.com/OffBlocks/offblocks-contracts]

Status: Fixed

Recommendations

Remediation:

Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs for the compiler version that is chosen.

Remediation (commit: 9060389): Floating pragma has been locked in `OffBlocksEscrow`, `OffBlocksSmartWallet`, `OffBlocksSmartWalletFactory`, and `PendingWithdrawal` contracts.

F-2024-1735 - Repeated smart wallet check could be a modifier - Info

Description:

In the `OffBlocksEscrow.sol` file, the following code snippet is repeated in both the `deposit` and `topUpDeposit` functions:

```
bool isSmartWallet = IOffBlocksSmartWalletFactory(smartWalletFactory
)
.getIsSmartWallet(msg.sender);
if (!isSmartWallet) revert MsgSenderIsNotASmartWallet();
```

This snippet checks if the `msg.sender` is a smart wallet.

Assets:

- File: contracts/core/OffBlocksEscrow.sol [https://github.com/OffBlocks/offblocks-contracts]

Status: Fixed

Recommendations

Remediation:

To improve code readability and maintainability, it would be beneficial to refactor this check into a modifier or a utility function. This would avoid code repetition and make the code cleaner.

Remediation (commit: 35af86b): Repeated smart wallet checks have been transformed and used as a modifier.

F-2024-1737 - Unoptimized gas consumption in _checkForInvalidAddress function - Info

Description:

In the `OffBlocksEscrow.sol` file, the `_checkForInvalidAddress` function checks for invalid addresses in an array. However, when an invalid address is found (i.e., the zero address), the function continues to iterate over the remaining addresses in the array, even though the result is already determined.

Here is the relevant code snippet:

```
for (uint i = 0; i < _splitAddresses.length; i++) {
if (_splitAddresses[i] == address(0)) {
valid = false;
}
}
```

Assets:

- File: contracts/core/OffBlocksEscrow.sol [https://github.com/OffBlocks/offblocks-contracts]

Status: Fixed

Recommendations

Remediation:

To optimize gas usage, it is suggested to return `false` immediately when an invalid address is found, skipping the unnecessary iterations. This change would make the function more efficient by reducing the amount of computation required.

Remediation (commit: f6ecf91): The `_checkForInvalidAddress()` function returns `false` immediately when an invalid address is found.

F-2024-1740 - PendingWithdrawal only supports one withdrawal per token at a time - Info

Description:

In the `PendingWithdrawal.sol` file, the `PendingWithdrawal` contract currently only supports one withdrawal per token at a time. If a new withdrawal is initiated while a withdrawal is already pending, the timelock resets for the funds already pending. This is due to the fact that the `timestamps` mapping only stores a single timestamp per token.

This limitation leads to a less optimal user experience, as users may need to initiate multiple withdrawals of the same token concurrently.

```
function initiateTokenWithdrawal(
address _token,
uint256 _delay
) external nonReentrant onlyOwner {
timestamps[_token] = block.timestamp + _delay;

emit InitiateWithdrawal(_token, _delay);
}
```

Assets:

- File: contracts/core/PendingWithdrawal.sol [https://github.com/OffBlocks/offblocks-contracts]

Status: Accepted

Recommendations

Remediation:

To improve the user experience, it is suggested to refactor the `PendingWithdrawal` contract to support multiple parallel withdrawals.

This could be achieved by changing the `timestamps` mapping to map a token address to an array of withdrawal objects, where each object contains the timestamp and the amount of the withdrawal. This change would allow the contract to handle multiple withdrawals per token concurrently, without resetting the timelock for funds already pending.

F-2024-1741 - Redundant getters in OffBlocksSmartWalletFactory.sol - Info

Description: In the `OffBlocksSmartWalletFactory.sol` file, the `getWithdrawalDelay`, `getSmartWalletsOfUser`, and `getIsSmartWallet` functions are redundant. These functions are used to return the values of `withdrawalDelay` and `isSmartWallet` respectively. However, `withdrawalDelay`, `smartWallets`, and `isSmartWallet` are declared as public variables, which means Solidity automatically generates public getter functions for them. Therefore, the `getWithdrawalDelay`, `getSmartWalletsOfUser`, and `getIsSmartWallet` functions are unnecessary and can be removed to simplify the code.

Assets:

- File: contracts/core/OffBlocksSmartWalletFactory.sol [https://github.com/OffBlocks/offblocks-contracts]

Status: Fixed

Recommendations

Remediation: It is suggested to remove the redundant getters.

Remediation (commit: 35a34fa): The redundant getters functions have been removed.

F-2024-1744 - Gas usage of updating Deposit struct can be optimized - Info

Description:

The functions responsible for updating values in the `Deposit` struct consumes a significant amount of gas, indicating room for optimization. All functions in the contract follow the same pattern:

1. Create a copy of the entire `Deposit` struct and assign it to the `userDeposit` memory variable.
2. Perform checks to ensure that all necessary conditions are met.
3. Update the `userDeposit` variable with new values.
4. Assign the entire `userDeposit` variable back to the corresponding storage variable, even though many fields in the `Deposit` struct are reassigned to the same value.

Redundant gas consumption is observed in step 4 and skipping step 1. Gas optimization can be achieved by saving only necessary `Deposit` fields instead of saving the entire struct and reading directly from storage variable. For example, in the `batchUpdateDepositStatuses()` function, gas usage can be reduced from 79055 to 54117.

```
function batchUpdateDepositStatuses(
uint256[] calldata _depositIds,
uint256 _status
) external onlyOwner {
if (_depositIds.length == 0) revert ArrayLengthOfZero();
if (
_status != uint256(DepositStatus.Approved) &&
_status != uint256(DepositStatus.Rejected)
) revert InvalidStatus(_status);
//@audit Deposit struct is not copied to memory
for (uint256 i = 0; i < _depositIds.length; i++) {
if (depositById[_depositIds[i]].status != DepositStatus.Pending)
revert DepositNotPending();
//@audit update only 1 Deposit field
depositById[_depositIds[i]].status = DepositStatus(_status);

emit DepositStatusUpdated(
_depositIds[i],
DepositStatus(_status),
block.timestamp
);
}
}
```

Assets:

- File: contracts/core/OffBlocksEscrow.sol [https://github.com/OffBlocks/offblocks-contracts]

Status: Fixed

Recommendations

Remediation: It is recommended to refactor the function responsible for updating values in the `Deposit` struct to optimize gas usage. This can be achieved by saving only the necessary `Deposit` fields instead of saving the entire struct and avoiding to copy whole `Deposit` struct in memory during function execution. Gas optimization can be performed in: `batchUpdateDepositStatuses()`, `reduceDeposit()`, `topUpDeposit()`, `reject()`, `settle()` and `batchSettle()`.

Remediation (commit: 4932655): Gas usage has been optimized.

F-2024-1750 - Missing two-step ownership transfer process - Info

Description:

The `OffBlocksEscrow` and `OffBlocksSmartWalletFactory` contracts currently utilizes the `OwnableUpgradeable` library from OpenZeppelin for managing contract ownership.

The `PendingWithdrawal` and `OffBlocksSmartWallet` contracts currently utilizes the `Ownable` library from OpenZeppelin for managing contract ownership.

However, the `OwnableUpgradeable` and `Ownable` libraries lacks a safety mechanism that prevent the contract ownership from mistakenly being transferred to an address that cannot handle it (e.g. due to a typo in the address), by requiring that the recipient of the owner permissions actively accept via a contract call of its own.

Assets:

- File: contracts/core/OffBlocksEscrow.sol [https://github.com/OffBlocks/offblocks-contracts]
- File: contracts/core/OffBlocksSmartWallet.sol [https://github.com/OffBlocks/offblocks-contracts]
- File: contracts/core/OffBlocksSmartWalletFactory.sol [https://github.com/OffBlocks/offblocks-contracts]
- File: contracts/core/PendingWithdrawal.sol [https://github.com/OffBlocks/offblocks-contracts]

Status: Fixed

Recommendations

Remediation:

Consider using `Ownable2StepUpgradeable` and `Ownable2Step` libraries from OpenZeppelin Contracts to enhance the security of your contract ownership management. These contracts prevent the accidental transfer of ownership to an address that cannot handle it, such as due to a typo, by requiring the recipient of owner permissions to actively accept ownership via a contract call. This two-step ownership transfer process adds an additional layer of security to your contract's ownership management.

Remediation (commit: 1115a7d): `Ownable` and `OwnableUpgradeable` libraries have been replaced with `Ownable2Step` and `Ownable2StepUpgradeable` libraries.

F-2024-1779 - Unnecessary external call in the smartWalletOwner function - Info

Description:

In the `OffBlocksSmartWallet.sol` file, the `smartWalletOwner` function is performing an unnecessary external call to read a storage variable in the same contract.

Here is the relevant code snippet:

```
function smartWalletOwner() public view returns (address) {
return IOffBlocksSmartWallet(address(this)).owner();
}
```

This external call is a waste of gas because a simple storage read would be sufficient.

Assets:

- File: contracts/core/OffBlocksSmartWallet.sol [https://github.com/OffBlocks/offblocks-contracts]

Status: Fixed

Recommendations

Remediation:

It is recommended to refactor the `smartWalletOwner` function to directly return the `owner()` storage variable, eliminating the need for the external call and optimizing gas usage.

Remediation (commit: 011243b): The redundant external call has been removed, and replaced with access to local variable instead.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/OffBlocks/offblocks-contracts
Commit	fef94ee4963c49d53c7d217cd6e3e7d5860ec9c3
Whitepaper	https://docs.offblocks.xyz/overview/whitepaper
Requirements	https://docs.offblocks.xyz/developer-guides/smart-contracts
Technical Requirements	https://docs.offblocks.xyz/developer-guides/smart-contracts

Contracts in Scope

contracts/core/OffBlocksEscrow.sol

contracts/core/OffBlocksSmartWallet.sol

contracts/core/OffBlocksSmartWalletFactory.sol

contracts/core/PendingWithdrawal.sol

contracts/interfaces/IOffBlocksEscrow.sol

contracts/interfaces/IOffBlocksSmartWalletFactory.sol

contracts/interfaces/IOffBlocksSmartWallet.sol