HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: PreSend
Date: January 30th, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for PreSend
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	Transaction verifier; Affiliate program
Platform	EVM
Network	Ethereum, Polygon
Language	Solidity
Methodology	Link
Website	-
Changelog	17.11.2022 - Initial Review 21.12.2022 - Second Review 30.01.2023 - Third Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by PreSend (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository:
https://github.com/Presend-DeFi/presend-protocol-payment-affiliate
Commit:
0743ffc
Documentation:
Technical description in the README.md
Integration and Unit Tests: Yes
Contracts:
File: ./contracts/PreSendAffiliate.sol
SHA3: f70726703fb04a9f5c15f8a8f22cc74cb318bcb0cfa50df15174812d8a9a2690

File: ./contracts/PreSendPayments.sol
SHA3: 39d1df8c7544a6128519c58c7039ba493a0c72570107cab68e5b1b5a7a5767d5

File: ./contracts/transparent_proxy/ProxyAdmin.sol
SHA3: e29292065af23dea48f09efabb42155a63810583c6afb2a2515d7537e4e8d248

File: ./contracts/transparent_proxy/TransparentUpgradeableProxy.sol
SHA3: 13e717cd1e4c20d24739fde5ce0ee14102cfdec2f7558ad989bf133f2443ab86

File: ./interfaces/AggregatorV3Interface.sol
SHA3: 4f25993d7f224bcb01be58e9cc13ddf3cd9b6ce7ec3ea717175b0c4ca5198a40

Second review scope

Repository:
https://github.com/Presend-DeFi/presend-protocol-payment-affiliate
Commit:
5ffd536b441a5f12044f7b57f0c32e3ef9d69ef2
Documentation:
Technical description in the README.md
Integration and Unit Tests: Yes
Contracts:
File: ./contracts/PreSendAffiliate.sol
SHA3: a044445b99175018e242e5c7b03fda637b1db032531db25e90db837f91e31ee5

File: ./contracts/PreSendPayments.sol
SHA3: 4edb5bdc41b41641a321928197b4a8885f4e269988f2724f681d1041e29e3718

Third review scope

Repository:
https://github.com/Presend-DeFi/presend-protocol-payment-affiliate
Commit:
d2c814f6dd1e574b1d0452c243eca4669be6dd94
Documentation:
Technical description in the README.md

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/PreSendAffiliate.sol

SHA3: 8596d90b541017c42d4aff34739d2287ad953ffc590017aa7fdb08138480d5e2

File: ./contracts/PreSendPayments.sol

SHA3: 314bee2eeeccf4ed5d1543c598918c237241c1077bb7f829f18422e55491f79f

File: ./interfaces/IPreSendAffiliate.sol

SHA3: 2b4397c8452622891908d01105d891bbf35c493a67f63b9474252bd967b7ab5f

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

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

- Technical descriptions are provided.
- Functional requirements are provided.

Code quality

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

- The development environment is configured.

Test coverage

Code coverage of the project is **98.65%**.

- Some error handlers were not tested.

Security score

As a result of the audit, the code contains **0** critical, **0** high, **0** medium, **0** low severity issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.95**.

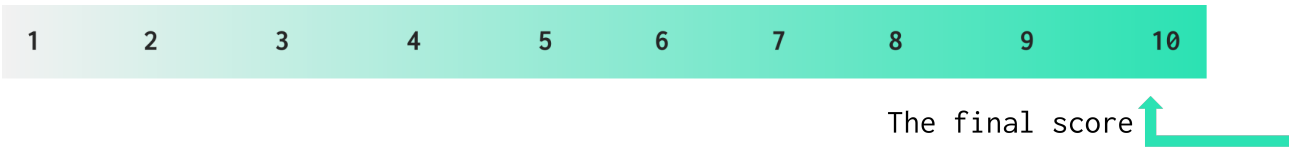

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
17 November 2022	4	8	1	3
21 December 2022	0	2	1	1
06 January 2023	0	0	0	0

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Checked Items

We have audited the Customers' smart contracts for commonly known and more specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Passed
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Not Relevant

Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Not Relevant
Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passe
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

PreSend contracts stand for storing affiliates and fees collecting. They are represented by these contracts:

- *PreSendPayment* – Payment contract gives it permission to increase and decrease the amount of funds affiliates can withdraw. Inherited from OpenZeppelin's Initializable, AccessControl, Chainlink's AutomationCompatible contracts.
  It has the following attributes:
  - affiliate smart contract reference
  - address of the treasury where affiliate payments will go.
  - aggregator to get the price of the native coin in USD.
  - the Default Admin user.
- *PreSendAffilate* - Role to manage affiliates. This role can add/remove affiliates, update the affiliate tier that determines the percentage of fees they get, and add new tiers for affiliates. This role can also update the reference to the payment contract. Inherited from OpenZeppelin's Initializable, AccessControl contracts.
  It has the following attributes:
  - affiliate features and properties
  - the address of the PreSend payments smart contract.
  - the Default Admin user.

Privileged roles

- Default Admin: Admin who can set and change an Affiliate, Treasury, aggregator, Fee divisor, Payment, Fee roles.
- Affiliate_Admin: In PreSendAffiliate.sol can add or update tier and affiliate.
- Payment_Admin: In PreSendPayments.sol this role can increase or decrease currency allowance.
- Fee_Admin: Fee Admin who manually sends PreSend transaction fees from the contract to the treasury address.
- Payment_Contract: In PreSendAffiliate.sol it represents PreSendPayments address.
- User: The person who uses a Payment contract.

Findings

■■■■ Critical

1. Upgradeability Errors

The *@openzeppelin/contracts* should not be used in the OpenZeppelin Upgrades project.

The difference between a *constructor* and a regular function is that Solidity takes care of automatically invoking the constructors of all ancestors of a contract. When writing an *initializer* needs to take special care to manually call the *initializers* of all parent contracts. Note that the initializer modifier can only be called once, even when using inheritance, so parent contracts should use the *onlyInitializing* modifier.

https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable

This violates upgradeability best practices and can lead to issues during an upgrade.

Paths: ./contracts/PreSendPayments.sol

./contracts/PreSendAffiliate.sol

Recommendation: use *@openzeppelin/contracts-upgradeable*, which is an official fork of OpenZeppelin Contracts that has been modified to use initializers instead of constructors.

Status: Fixed (Revised commit: 5ffd536)

2. Insufficient Funds

The *extractFees* function is meant to be used for fee withdrawals but can accept an arbitrary amount that will be transferred to an admin account.

Funds that are not a part of fees can be withdrawn. Users will not be able to claim their affiliate fees by using the *affiliateClaim* function.

Path: ./contracts/PreSendAffiliate.sol : extractFees()

Recommendation: implement proper fees withdrawal.

Status: Fixed (Revised commit: 4fa76a5)

■■■ High

1. Requirements Violation

According to documents and comments, the *affiliateAmount* should be 5% or 10% of the fees. However, this function does not check for these percentages.

The code should not violate the requirements provided by the Customer. This can lead to misallocation of fees.

Path: ./contracts/PreSendPayments.sol : _payPreSendFee()

Recommendation: implement the validation.

Status: Fixed (Revised commit: dc4cf6d)

2. **Requirements Violation**

According to the docs, adding, removing, and updating affiliates associated with a tier can only perform the *AFFILIATE_ADMIN* role, but these functionality are not in this implementation.

A new feature has been added that can only add an affiliate by 5%. It can be called by any person and add any person to the affiliate.

The code should not violate the requirements provided by the Customer. This may lead to incorrect distribution of fees to the affiliates.

Path: ./contracts/PreSendAffiliate.sol : addAffiliate()

Recommendation: change the documentation to match these functionality, or revert to the previous implementation.

Status: Fixed (Revised commit: 5ccc8a1)

3. **Access Control Violation**

This function can be called by any person and add any person to the affiliate.

Anyone cannot call functions to add an affiliate. This may lead to incorrect distribution of fees to the affiliates.

Path: ./contracts/PreSendAffiliate.sol : addAffiliate()

Recommendation: limit access to this function call.

Status: Fixed (Revised commit: 5ccc8a1)

4. **Undocumented Behavior**

This function uses the *affiliatePercentage* parameter. The comment explains that the percentage can be anywhere between 0 and 100 percent, including 0 and 100.

Documents did not describe this feature in *_payPreSendFee()* function.

This may lead to incorrect distribution of fees to the affiliates.

Path: ./contracts/PreSendPayments.sol : _payPreSendFee()

Recommendation: document the functionality according to the current version of the implementation.

Status: Fixed (Revised commit: 5ccc8a1)

Medium

1. **Best Practice Violation**

   Contracts use 2 access options and inherit from both.

   AccessControl makes it possible to add roles, Ownable makes the deployer the owner of the contract.

   Paths: ./contracts/PreSendPayments.sol

   ./contracts/PreSendAffiliate.sol

   Recommendation: *DEFAULT_ADMIN* can be used as an owner.

   Status: Fixed (Revised commit: 5ffd536)

2. **Best Practice Violation**

   The Checks-Effects-Interactions pattern is violated. During the function, some state variables are updated after the external calls.

   Path: ./contracts/PreSendPayments.sol : _payPreSendFee()

   Recommendation: implement the function according to the Checks-Effects-Interactions pattern.

   Status: Fixed (Revised commit: 5ffd536)

3. **Missing Event for Critical Value Updation**

   Critical state changes should emit events for tracking things off-chain.

   The functions do not emit events on change of important values.

   Paths: ./contracts/PreSendAffiliate.sol : addOrRemoveAffiliate()

   ./contracts/PreSendAffiliate.sol : addOrUpdateTier()

   ./contracts/PreSendPayments.sol : setTreasuryAddress()

   ./contracts/PreSendPayments.sol : setPreSendAffiliate()

   ./contracts/PreSendPayments.sol : decreaseCurrencyAllowance()

   ./contracts/PreSendPayments.sol : increaseCurrencyAllowance()

   ./contracts/PreSendPayments.sol : extractFees()

   ./contracts/PreSendPayments.sol : setFeeDivisor()

   Recommendation: emit events on critical state changes.

   Status: Fixed (Revised commit: 5ffd536)

4. **Unscalable Functionality**

Well-known contracts from projects like OpenZeppelin, ChainLink should be imported directly from the source as the projects are in development and may update the contracts in the future.

https://github.com/smartcontractkit/chainlink/blob/develop/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol

Path: ./interfaces/AggregatorV3Interface.sol

Recommendation: import the contracts directly from the source, avoid modifying them.

Status: Fixed (Revised commit: 5ffd536)

5. **Inefficient Gas Model**

For external interaction with the contract, you can use interfaces, which makes it possible to abstract from private methods and variables.

Using contracts instead of interfaces adds a lot of unnecessary bytes to this contract that are taken from private methods and variables, which increases the cost of deployment.

In *PreSendPayments.sol* contract imported directly *PreSendAffiliate.sol* instead of the interface of this contract.

Path: ./contracts/PreSendPayments.sol

Recommendation: import the contracts interface from the source instead of import contracts directly.

Status: Fixed (Revised commit: dc4cf6d)

6. **Contradiction**

The *addOrRemoveAffiliate()* also allows updating the existing records. Its name does not state this feature. So it should be renamed accordingly.

Path: ./contracts/PreSendAffiliate.sol : addOrRemoveAffiliate

Recommendation: rename the aforementioned function accordingly.

Status: Fixed (Revised commit: 5ffd536)

7. **Requirements Violation**

The role granting and revoking flows have separate functions controlled by the owner. It is a long shot, but it is possible for an owner to revoke a role and not assign a new address for it. This will prevent some admin features from being unusable for the time being.

Paths: ./contracts/PreSendAffiliate.sol : removePaymentAdmin(), addPaymentAdmin(), removeAffiliateAdmin(), addAffliateAdmin()

./contracts/PreSendPayments.sol: removePaymentAdmin(), addPaymentAdmin(), removeFeeAdmin(), addFeeAdmin()

Recommendation: merge role changing flows into a single function for each role.

Status: Fixed (Revised commit: 5ffd536)

8. Missing Validation

According to the documentation, the tier parameter should be checked before updating. However, in the *addOrRemoveAffiliate()*, *addOrUpdateTier()* functions, the validation is missed. In the future, this can cause unexpected scenarios(for example, admin can change percentage to 1000)

Paths: ./contracts/PreSendAffiliate.sol : addOrRemoveAffiliate()
./contracts/PreSendAffiliate.sol : addOrUpdateTier()

Recommendation: implement the validation for tear parameters.

Status: Fixed (Revised commit: 5ffd536)

9. Contradiction

The comment on the variable *aggregatorCoinPriceMult* does not match this value.

This can lead to incorrect calculations.

Path: ./contracts/PreSendPayments.sol

Recommendation: edit this comment according to the implementation.

Status: Fixed (Revised commit: 5ccc8a1)

10. Contradiction

The documentation has not been updated to the latest version and is missing information about all changed functions and variables, as well as their interactions.

For example:

docs have *_payPreSendFee(address user, address currency, uint256 amount, uint256 payment, uint256 currencyPrice, address affiliate)*

but in the current implementation has been added a new parameter *affiliatePercentage* and the current function has *_payPreSendFee(address user, address currency, uint256 amount, uint256 payment, uint256 currencyPrice, address affiliate, uint256 affiliatePercentage)*

This may lead to a misunderstanding by the user of the product and its capabilities.

Path: ./PreSend Payment and Affiliate Smart Contract Requirements.pdf

Recommendation: change the document to match the current version of the implementation.

Status: Fixed (Revised commit: 5ccc8a1)

Low

1. Missing Zero Address Validation

Address parameters are being used without checking against the possibility of 0x0.

Paths: ./contracts/PreSendAffiliate.sol : initialize()

./contracts/PreSendAffiliate.sol : addOrRemoveAffiliate()

./contracts/PreSendAffiliate.sol : increaseAffiliateAmount()

./contracts/PreSendAffiliate.sol : decreaseAffiliateAmount()

./contracts/PreSendAffiliate.sol : updatePaymentAddress()

./contracts/PreSendAffiliate.sol : addAffiliateAdmin()

./contracts/PreSendAffiliate.sol : removeAffiliateAdmin()

./contracts/PreSendAffiliate.sol : addPaymentAdmin()

./contracts/PreSendAffiliate.sol : removePaymentAdmin()

./contracts/PreSendPayments.sol : initialize()

./contracts/PreSendPayments.sol : setPreSendAffiliate()

./contracts/PreSendPayments.sol : setTreasuryAddress()

./contracts/PreSendPayments.sol : decreaseCurrencyAllowance()

./contracts/PreSendPayments.sol : increaseCurrencyAllowance()

./contracts/PreSendPayments.sol : addPaymentAdmin()

./contracts/PreSendPayments.sol : removePaymentAdmin()

./contracts/PreSendPayments.sol : addFeeAdmin()

./contracts/PreSendPayments.sol : removeFeeAdmin()

Recommendation: implement zero address checks.

Status: Fixed (Revised commit: 5ffd536)

2. Redundant Import

The use of unnecessary imports will increase the Gas consumption of the code. Thus they should be removed from the code.

Paths: ./contracts/PreSendAffiliate.sol : IERC20

./contracts/PreSendPayments.sol : IERC20

./contracts/PreSendPayments.sol : OwnableUpgradeable

Recommendation: remove redundant import or create new behavior for this contract.

Status: Fixed (Revised commit: 5ffd536)

3. **Redundant Address**

   The use of unnecessary addresses will increase the Gas consumption of the code. Thus they should be removed from the code.

   Path: ./contracts/PreSendPayments.sol : preSendAffiliateAddress

   Recommendation: to replace the address for this contract, you can use the interface, or the contract itself, which has already been added.

   Status: Fixed (Revised commit: 5ffd536)

4. **Use of Hard-Coded Values**

   Using hardcoded values in the computations and comparisons is not the best practice.

   Path: ./contracts/PreSendPayments.sol : _payPreSendFee()

   Recommendation: convert these variables into constants.

   Status: Fixed (Revised commit: 5ffd536)

5. **Redundant Variable Update**

   The initializer function updates *affiliateRegistrationTime*, *aggregatorCoinPriceMult*, *aggregatorCoinPriceSub*, *netRevenue*, *grossRevenue* variables with the same values as defined above.

   Path: ./contracts/PreSendPayments.sol : initialize()

   Recommendation: remove redundant variables update.

   Status: Fixed (Revised commit: 5ccc8a1)

6. **State Variables Can Be Declared Constant**

   aggregatorCoinPriceMult, *aggregatorCoinPriceSub* variables do not change anywhere and can be declared as constant.

   Path: ./contracts/PreSendPayments.sol

   Recommendation: declare mentioned variables as constant.

   Status: Fixed (Revised commit: 5ccc8a1)

7. **Misleading Error Messages**

   The require handler implementation is *(affiliatePercentage <= 100)* and can be explained as "affiliatePercentage must be less than or equal to 100", but the error message is "Affiliate Percentage must be less than 100".

This makes code harder to understand, test and debug.

Path: ./contracts/PreSendPayments.sol : _payPreSendFee()

Recommendation: refactor messages in require conditions to fit code behavior.

Status: Fixed (Revised commit: 5ccc8a1)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date 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 to 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, Consultant cannot guarantee the explicit security of the audited smart contracts.