WPay - Light Audit + Re-test
Security Assessment Findings Report

Date: October 2, 2024

Version 0.1

Contents

1 Confidentiality statement

This document is the exclusive property of Gateway FM and Sub7 Security. This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires consent of both Gateway FM and Sub7 Security.

2 Disclaimer

A smart contract security audit is considered a snapshot in time. The findings and recommendations reflect the information gathered during the assessment and not any changes or modifications made outside of that period.

Time-limited engagements do not allow for a full evaluation of all security controls. Sub7 Security prioritized the assessment to identify the weakest security controls an attacker would exploit. Sub7 Security recommends conducting similar assessments on an annual basis by internal or third-party assessors to ensure the continued success of the controls

3 About Sub7

Sub7 is a Web3 Security Agency, offering Smart Contract Auditing Services for blockchain-based projects in the DeFi, Web3 and Metaverse space.

Learn more about us at https://sub7.xyz

4 Project Overview

Wpay are experts in payment experiences for large scale, complex businesses. Connected, in-store and online payments, data and insights plus Gifting – all in a single service.

5 Executive Summary

Sub7 Security has been engaged to what is formally referred to as a Security Audit of Solidity Smart Contracts, a combination of automated and manual assessments in search for vulnerabilities, bugs, unintended outputs, among others inside deployed Smart Contracts.

The goal of such a Security Audit is to assess project code (with any associated specification, and documentation) and provide our clients with a report of potential security-related issues that should be addressed to improve security posture, decrease attack surface and mitigate risk.

As well general recommendations around the methodology and usability of the related project are also included during this activity

1 (One) Security Auditors/Consultants were engaged in this activity.

5.1 Scope

https://github.com/gateway-fm/wirex-contracts

5.2 Timeline

from 28/09/2024 to 02/10/2024

5.3 Summary of Findings Identified

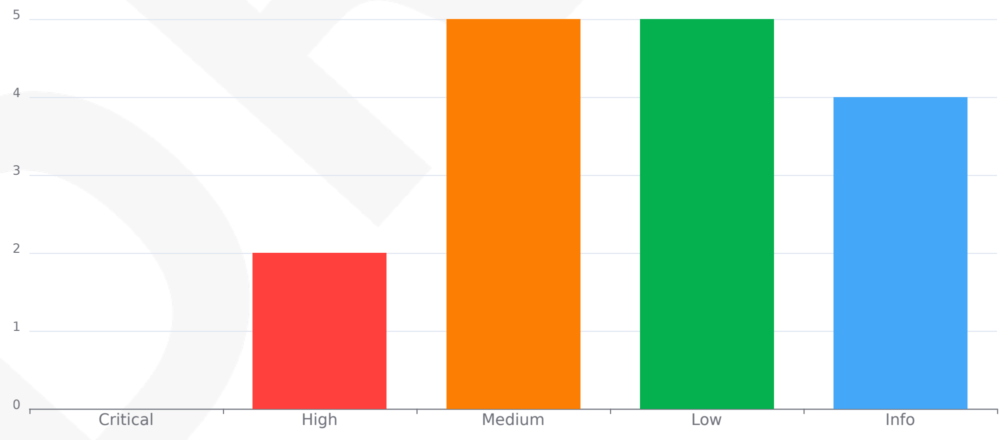

Figure 1: Executive Summary

# 1 High The `issueCard` function is missing a check to validate if the passed owner is the actual owner of the cardWallet. – ***Fixed***

# 2 High prcessTransaction function executes successfully even if the amount is not satisfied. – ***Fixed***

# 3 Medium `ERC20::approve` will revert for some non-standard tokens like USDT. – ***Fixed***

# 4 Medium `addToken` function allows to add same contract but with different names – ***Fixed***

# 5 Medium Solidity version ^0.8.21 might not work on some chain. – ***Acknowledged***

# 6 Medium Sending tokens in a for loop – ***Acknowledged***

# 7 Medium Off-by-one timestamp error – ***Fixed***

# 8 Low Unexpected revert when setting limits in card.sol – ***Open***

# 9 Low Create Account function changes the verification status to unknown instead of Applied – ***Acknowledged***

# 10 Low retrieveCardTransactions can be permanently DOS due to access to unbounded array – ***Fixed***

# 11 Low MIssing zero validations at multiple places – ***Fixed***

# 12 Low `transfer` function can fail silently for card.sol – ***Open***

# 13 Info Getter functions can be renamed better to reprsent what are they meant to do – ***Open***

# 14 Info _checkOwner function is redundant – ***Open***

# 15 Info Missing events for critical operations – ***Open***

16 Info Gas Improvements – ***Open***

5.4 Methodology

SUB7's audit methodology involves a combination of different assessments that are performed to the provided code, including but not limited to the following:

Specification Check

Manual assessment of the assets, where they are held, who are the actors, privileges of actors, who is allowed to access what and when, trust relationships, threat model, potential attack vectors, scenarios, and mitigations. Well-specified code with standards such as NatSpec is expected to save time.

Documentation Review

Manual review of all and any documentation available, allowing our auditors to save time in inferring the architecture of the project, contract interactions, program constraints, asset flow, actors, threat model, and risk mitigation measures

Automated Assessments

The provided code is submitted via a series of carefully selected tools to automatically determine if the code produces the expected outputs, attempt to highlight possible vulnerabilities within non-running code (Static Analysis), and providing invalid, unexpected, and/or random data as inputs to a running code, looking for exceptions such as crashes, failing built-in code assertions, or potential memory leaks.

Examples of such tools are Slither, MythX, 4naly3er, Sstan, Natspec-smells, and custom bots built by partners that are actively competing in Code4rena bot races.

Manual Assessments

Manual review of the code in a line-by-line fashion is the only way today to infer and evaluate business logic and application-level constraints which is where a majority of the serious vulnerabilities are being found. This intensive assessment will check business logics, intended functionality, access control & authorization issues, oracle issues, manipulation attempts and multiple others.

Security Consultants make use of checklists such as SCSVS, Solcurity, and their custom notes to ensure every attack vector possible is covered as part of the assessment

6 Findings and Risk Analysis

6.1 The `issueCard` function is missing a check to validate if the passed owner is the actual owner of the cardWallet.

Severity: High
Status: Fixed

Description

All issued cards of the user are managed by the cardManagement contract. A card contract is deployed by the backend when a card is issued to a user and cards are assigned to users using the `issueCard` function.

However, the function lacks a check to ensure that the `owner` parameter passed to the function is actually the owner of the cardWallet (card contract address).

```
    function issueCard(
        address cardWallet,
        address owner,
        CardType cardType,
        Currency currency,
        uint256 expiration,
        bytes calldata externalID
    ) external onlyOracle {
        // checks
        require(
            cardType != CardType.UNKNOWN,
            ErrorLib.badRequestErr("CardsManagement", "cardType", "cannot be unknown")
        );
        require(
            currency != Currency.UNKNOWN,
            ErrorLib.badRequestErr("CardsManagement", "currency", "cannot be unknown")
        );
        require(
            cardWallet != address(0),
            ErrorLib.badRequestErr("CardsManagement", "cardWallet", "cannot be zero
                address")
        );
        require(
            _cards[cardWallet].cardWallet == address(0),
            ErrorLib.alreadyExistErr("CardsManagement", "cardWallet", "card wallet address
                 already used")
        );
        require(
            _getCardWalletByExternalID(externalID) == address(0),
            ErrorLib.alreadyExistErr("CardsManagement", "externalID", "card external id
                already used")
        );

        IAccounts.Account memory account = IAccounts(_accounts()).getAccount(owner);
        require(
            account.status == IAccounts.AccountStatus.ACTIVE,
            ErrorLib.badStatusErr("CardsManagement", "account.status", "ACTIVE")
        );
        require(
            account.verificationStatus == IAccounts.AccountVerificationStatus.APPROVED,
            ErrorLib.badStatusErr("CardsManagement", "account.verificationStatus", "
                APPROVED")
        );

        // Assignments stuff
```

If an incorrect address is passed, the card will be issued to a user who doesn't actually have authority over that card, making the `card useless to make transactions and there is no option to delete a card`.

Location

implementation/cards/CardsManagement.sol#L71-L108

Recommendation

Enforce a check to validate if the passed owner is the actual owner of cardWallet.

```
    function issueCard(
        address cardWallet,
        address owner,
        CardType cardType,
        Currency currency,
        uint256 expiration,
        bytes calldata externalID
    ) external onlyOracle {
        ...
+        require(
+            IAccountAbstraction(cardWallet).owner() == owner,
+            ErrorLib.unauthorisedErr("CardsManagement", "Owner Mismatch")
+        );
```

Comments

https://github.com/gateway-fm/wirex-contracts/pull/89

6.2 prcessTransaction function executes successfully even if the amount is not satisfied.

Severity: High
Status: Fixed

Description

`processTransaction` is called via oracles that further call the `executeTransaction` function.

`executeTransaction` function calls the `_checkCardWalletFunds` function that returns the amount of each token to be transferred and if the tokens in the card can satisfy the required amount based on the exchange rate.

then if there is some unsatisfied amount it emits the event with the transaction failure reason `FUNDS`.And tries to transfer the tokens as follows:

```
        // If not enough funds to satisfy the amount, fail the transaction
        if (unsatisfied > 0) {
            return (false, TransactionFailReason.FUNDS);
        }

        // Transfer funds
        for (uint256 i = 0; i < tokens.length; i++) {
            // If no tokens to transfer, continue
            if (toTransfer[i] == 0) continue;

            IERC20 token = IERC20(tokens[i].tokenAddress);
            // @audit-issue silent failure
            token.transferFrom(cardWallet, _treasureStorage, toTransfer[i]);
        }
```

The issue here is, that some tokens can fail silently and return false on failure. If the transfer fails due to some reason, it will go unnoticed and the contracts will assume the transfer happened successfully and will emit the event NONE representing that the transaction is successful while the funds are still in the card wallet.

Location

implementation/processing/Processing.sol

Recommendation

Use the OZ safe transfer library.

Comments

6.3 `ERC20::approve` will revert for some non-standard tokens like USDT.

Severity: Medium
Status: Fixed

Description

The code uses the `approve` method to set allowance for ERC20 tokens in the `setLimits` function. This will cause a revert if the token registry contains an ERC20 token that is non-standard and has a different function signature for `approve()`.

```
    function setLimits(uint256[] calldata amounts) external onlyOwner {
        ITokensRegistry.Token[] memory tokens = ITokensRegistry(_erc20Registry()).
            retrieveAllTokens();
        require(
            tokens.length == amounts.length,
            ErrorLib.badRequestErr("Card", "limits length", "not equal to tokens length")
        );

        for (uint256 i = 0; i < tokens.length; i++) {
            IERC20(tokens[i].tokenAddress).approve(_processing(), amounts[i]);
        }
    }
```

Tokens like USDT will cause a revert for this function because the USDT contract on Ethereum doesn't implement the IERC20 interface correctly.

Impact: Medium, because the function `setLimits` won't work, if the token registry contains tokens like USDT.

Likelihood: Medium, because USDT is a common token. Also confirmed from the team, the answer is `technically it can be any token` but not no.

Location

implementation/cards/Card.sol#L60-L67

Recommendation

Use OpenZeppelin's `SafeERC20.forceApprove` method instead to support all the ERC20 tokens.

Comments

https://github.com/gateway-fm/wirex-contracts/pull/88

6.4 addToken function allows to add same contract but with different names

Severity: Medium
Status: Fixed

Description

when `addToken` function is called admin can pass in symbol, tokenaddress and decimals of token, than it checks if that symbol already exist or not as follow

```
        (Token memory token, ) = _getTokenBySymbol(symbol);
        require(
            token.tokenAddress == address(0),
            ErrorLib.alreadyExistErr("TokensRegistry", "tokenAddress", "token already
                supported")
        );
    function _getTokenBySymbol(string calldata symbol) internal view returns (Token memory
         token, uint256 index) {
        bytes32 symbolHash = keccak256(bytes(symbol));
        for (uint256 i = 0; i < _tokens.length; i++) {
            if (keccak256(bytes(_tokens[i].symbol)) == symbolHash) {
                token = _tokens[i];
                index = i;
                break;
            }
        }

        return (token, index);
    }
```

Now consider the following scenario:

1. admin adds a token with symbole WETH and address 0x123 and decimals 18.
2. admin adds another token with symbol WETH2. and same address as before 0x123 and decimals 18.

Now there are two tokens in _tokens array with different symbol but the same address which should not happen.

Same thing can happen in the `ContractRegistery.sol` in the following function

```
    function setContract(string calldata name, address proxy) external onlyOwner {
        require(proxy != address(0), "ContractRegistry: proxy address cannot be zero
            address");

        // @audit - here we should add a check that the address doesn't already exists but
             with a different name

        if (_contracts[name] == address(0)) {
            _names.push(name);
        }

        // proxy address always remains the same, the implementation can change
        _contracts[name] = proxy;
    }
```

Location

implementation/tokensRegistry/TokensRegistry.sol#L33-L42

contracts/implementation/ContractRegistry.sol#L104-L111

Recommendation

When adding the token and contract if the token/contract with the same address already exists, if so revert.

Comments

https://github.com/gateway-fm/wirex-contracts/pull/91#pullrequestreview-2093427854

6.5 Solidity version ˆ0.8.21 might not work on some chain.

Severity: Medium
Status: Acknowledged

Description

some chains like zksync don't support the version 0.8.20 and above due to incompatibility with the push0 opcode, which is also the case with the wpay contracts.

Location

implementation/cards/Card.sol#L2

Recommendation

Downgrade the version below 0.8.0 and also lock the pragma like 0.8.19 instead of ˆ0.8.19

Comments

Project uses OpenZeppelin proxy contracts with 0.8.20 solidity version

6.6 Sending tokens in a for loop

Severity: Medium
Status: Acknowledged

Description

Performing token transfers in a loop in a Solidity contract is generally not recommended due to various reasons. One of these reasons is the "Fail-Silently" issue.

In a Solidity loop, if one transfer operation fails, it causes the entire transaction to fail. This issue can be particularly troublesome when you're dealing with multiple transfers in one transaction. For instance, if you're looping through an array of recipients to distribute dividends or rewards, a single failed transfer will prevent all the subsequent recipients from receiving their transfers. This could be due to the recipient contract throwing an exception or due to other issues like a gas limit being exceeded.

Moreover, such a design could also inadvertently lead to a situation where a malicious contract intentionally causes a failure when receiving Ether to prevent other participants from getting their rightful transfers. This could open up avenues for griefing attacks in the system.

Location

implementation/processing/Processing.sol#L212-L217

Recommendation

To mitigate this problem, it's typically recommended to follow the "withdraw pattern" in your contracts instead of pushing payments. In this model, each recipient would be responsible for triggering their own payment. This separates each transfer operation, so a failure in one doesn't impact the others. Additionally, it greatly reduces the chance of malicious interference as the control over fund withdrawal lies with the intended recipient and not with an external loop operation.

Comments

Unfortunatly it doesn't seem to be a direct solution to this, but indirectly a solution already exist. If at some point some token is paused, admin can simply remove that token from the list and the system will keep working fine.

6.7 Off-by-one timestamp error

Severity: Medium
Status: Fixed

Description

In Solidity, using `>=` or `<=` to compare against `block.timestamp` (alias `now`) may introduce off-by-one errors due to the fact that `block.timestamp` is only updated once per block and its value remains constant throughout the block's execution. If an operation happens at the exact second when `block.timestamp` changes, it could result in unexpected behavior.

Location

implementation/processing/Processing.sol#L64-L94

Recommendation

To avoid this, it's safer to use strict inequality operators (`>` or `<`). For instance, if a condition should only be met after a certain time, use `block.timestamp > time` rather than `block.timestamp >= time`. This way, potential off-by-one errors due to the exact timing of block mining are mitigated, leading to safer, more predictable contract behavior.

Comments

https://github.com/gateway-fm/wirex-contracts/pull/90/files

6.8 Unexpected revert when setting limits in card.sol

Severity: Low
Status: Open

Description

When setting the limit the following approval is granted for each token:

```
        for (uint256 i = 0; i < tokens.length; i++) {
            IERC20(tokens[i].tokenAddress).approve(_processing(), amounts[i]);
        }
```

The issue here is some tokens like USDT will revert here because it requires the approval to be set to 0 first to prevent the approval race condition vulnerability. But in this case, reporting is low because in such a scenario user can simply call the other approve function exposed, but the user should be made aware of that scenario.

Location

implementation/cards/Card.sol#L59-L69

Recommendation

In this specific scenario granting the 0 approval is not ideal because a token like BNB will revert to doing that and will DOS the functionality for such a token, so the best solution seems to be just to make the user aware of the problem and guide them to use approve function in that scenario.

Comments

6.9 Create Account function changes the verification status to unknown instead of Applied

Severity: Low
Status: Acknowledged

Description

AccountStatus and VerficationStatus have the following properties:

```
    enum AccountStatus {
        UNKNOWN,
        PENDING,
        ACTIVE,
        BLOCKED,
        DELETED
    }
       enum AccountVerificationStatus {
        UNKNOWN,
        APPLIED,
        IN_REVIEW,
        APPROVED,
        CANCELLED,
        REJECTED
    }
```

A user can request to create an account via the `createAccount` function which performs the following action

```
    function createAccount() external {
        require(
            _isAccountExists(msg.sender) == false,
            ErrorLib.alreadyExistErr("Accounts", "account", "already exists")
        );

        // @audit-issue should change the verification status to applied instead of
            unknown
        _accounts[msg.sender] = Account(AccountStatus.PENDING, AccountVerificationStatus.
            UNKNOWN);

        emit AccountCreated(msg.sender);
    }
```

Now here we can see that the account status is changed to pending but the verification status is still set to it's default 0 of unknown.

Location

implementation/accounts/Accounts.sol#L35-L44

Recommendation

Change verification status to applied on create account request.

Comments

Fix requires big changes in backend services which need to be done before

6.10 retrieveCardTransactions can be permanently DOS due to access to unbounded array

Severity: Low
Status: Fixed

Description

`retrieveCardTransactions` gets the ids of all the transaction id's from _cardTransactions and than traverse all the id's to get the transaction data from the mapping:

```
    function retrieveCardTransactions(address cardWallet) external view returns (
        Transaction[] memory transactions) {
        // Collects an array of transactions using transactions ids and returns
        uint256[] memory transactionIDs = _cardTransactions[cardWallet];

        if (transactionIDs.length == 0) return transactions;

        transactions = new Transaction[](transactionIDs.length);

        for (uint256 i = 0; i < transactionIDs.length; i++) {
            transactions[i] = _transactions[transactionIDs[i]];
        }
    }
```

The issue is the id's array could grow so big that looping it could always run out of block transaction limit in which case calling this function will always revert.

Location

implementation/processing/Processing.sol#L160-L171

Recommendation

Modify this function in a way that it allows to extract transaction id's from certain index to a certain index, in this way user could always extract partial array in case the array becomes too big.

Comments

https://github.com/gateway-fm/wirex-contracts/pull/90/files

6.11 MIssing zero validations at multiple places

Severity: Low
Status: Fixed

Description

In the above provided functions, add the necessary zero address, zero value, and empty string sanity checks so the admin may not mistakenly add those or can't be passed via other functions.

Location

implementation/processing/Processing.sol#L64-L70 implementation/limits/Limits.sol#L28-L32

implementation/tokensRegistry/TokensRegistry.sol#L33-L43

implementation/tokensRegistry/TokensRegistry.sol#L48-L60

implementation/tokensRegistry/TokensRegistry.sol#L65-L73

Recommendation

Add the validation sanity checks.

Comments

https://github.com/gateway-fm/wirex-contracts/pull/90/files

6.12 `transfer` function can fail silently for card.sol

Severity: Low
Status: Open

Description

`card` acts as an account abstraction wallet and allows the owner to transfer any erc20 token but it uses the `transfer` function which could fail silently for some tokens and will not revert on failure.

```
    function transfer(address token, address to, uint256 amount) external onlyOwner {
        // @audit - this could silently fail, use safeTransfer
        IERC20(token).transfer(to, amount);
    }
```

Location

implementation/cards/Card.sol#L82-L84

Recommendation

use the oz safe transfer library that handles such cases.

Comments

6.13 Getter functions can be renamed better to reprsent what are they meant to do

Severity: Info
Status: Open

Description

Some functions are meant to get the storage variables in contractRegisterry but they don't have a get prefix, unlike setterr functions. Add a get prefix to increase the readability of the code. For example in the contract registry following functions can be renamed:

```

function oracleProcessing() external view returns (address) {
        return _oracleProcessing;
    }

    /*
     * See {IContractRegistry - oracleCards}
     */
    function oracleCards() external view returns (address) {
        return _oracleCards;
    }

    /*
     * See {IContractRegistry - oracleKYC}
     */
    function oracleKYC() external view returns (address) {
        return _oracleKYC;
    }
```

Location

contracts/implementation/ContractRegistry.sol#L43-L57

Recommendation

Add the get prefix to functions.

Comments

6.14 _checkOwner function is redundant

Severity: Info
Status: Open

Description

The function `Card._checkOwner` is not used anywhere, rendering it redundant.

Location

implementation/cards/Card.sol#L86-L89

Recommendation

Remove `_checkOwner` function.

Comments

6.15 Missing events for critical operations

Severity: Info
Status: Open

Description

Without events, users and blockchain-monitoring systems will not be able to easily detect suspicious behavior.

Several critical functions are not triggerring events, which will make it difficult to check the behavior of the contracts once they have been deployed.

Location

contracts/implementation/ContractRegistry.sol#L64-L80

Recommendation

Ideally, the following critical operations should trigger events.

Comments

6.16 Gas Improvements

Severity: Info
Status: Open

Description

The following are gas improvements for your Smart Contracts

https://gist.github.com/tonysub7/4d5eb68639a05a236579ce1b14559777

Location

wirex-contracts/blob/a0f8fe77772d9c088c8c86aabf2964fbb8b8ff19

Recommendation

NA

Comments