SHELLBOXES

PXP Gateway

Smart Contract Security Audit

Prepared by ShellBoxes

May 4th, 2022 - June 15th, 2022

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	PXP Gateway
Version	1.0
Classification	Public

Scope

The PXP Gateway Contract in the PXP Gateway Repository

Files	MD5 Hash
PXPGateWay.sol	3691308F2A4C2F6983F2880D32E29C84
PXPToken.sol	8FCD4C929A5CCE5274D661185D11D43B
signController.go	8FCD4C929A5CCE5274D661185D11D43B
router.go	E68D1EA95D3D55F61C5A704CB7E63551

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction 5

1.1 About PXP 5

1.2 Approach & Methodology 5

1.2.1 Risk Methodology 6

2 Findings Overview 7

2.1 Summary 7

2.2 Key Findings 7

3 Finding Details 9

A signController.go 9

A.1 Missing amount check in signWithdraw [CRITICAL] 9

A.2 Infinite Withdraw Leads To The Drain Of The Contract [CRITICAL] . . 10

A.3 A Malicious User Can Tamper Addresses [HIGH] 12

B routes.go 13

B.1 API Exposed To The Public [CRITICAL] 13

B.2 Missing Middleware For An Inactive User [MEDIUM] 14

C Authorizeservice.go 15

C.1 Wallet Authentication Verifed By The Private Key [HIGH] 15

C.2 Public Key Can Be Tampered [HIGH] 17

C.3 jwtSecret Is Hardcoded In The Authorizeservice [MEDIUM] 18

C.4 HS256 Used As Signing Algorithm [LOW] 19

C.5 Add The Public Address In The JWT Token [INFORMATIONAL] 20

D Authservice.go 21

D.1 getSecretKey Returns Predicted Output [LOW] 21

E PXPGateWay.sol 22

E.1 withdrawToken Can Be Abused [HIGH] 22

E.2 Overriding Completed Transactions [MEDIUM] 24

E.3 Missing Transfer Verification [MEDIUM] 25

E.4 Missing Address Verification [LOW] 27

E.5 Missing Value Verification [LOW] 28

E.6 Floating Pragma [LOW] 29

F PXPToken.sol 30

F.1 Approve Race Condition [LOW] 30
F.2 Owner Can Renounce Ownership [LOW] 31
F.3 Floating Pragma [LOW] 32

4 Best Practices 33
BP.1 Variables should be initialized first 33

5 Static Analysis (Slither) 34

6 Conclusion 55

1 Introduction

PXP engaged ShellBoxes to conduct a security assessment on the PXP Gateway beginning on May 4th, 2022 and ending June 15th, 2022. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About PXP

Pirate X Pirate is a blockchain-based NFT adventure game with a turn-based dice combat system. It is built to be a sustainable platform with long-term updates planned. Pirate X Pirate is a world where you are rewarded with in-game money by adventuring across the high seas. Recruit your crew, form your fleet, then harvest resources or test your skills fighting against other pirates to earn.

Issuer	PXP
Website	https://piratexpirate.io
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact			
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low
	High	Medium	Low

Likelihood

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the PXP Gateway implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general,, these smart contracts are well-designed and constructed,, but their implementation might be improved by addressing the discovered flaws, which include 3 critical-severity, 4 high-severity, 4 medium-severity, 8 low-severity, 1 informational-severity vulnerabilities.

Vulnerabilities	Severity	Status
Missing amount check in signWithdraw	CRITICAL	Fixed
Infinite Withdraw Leads To The Drain Of The Contract	CRITICAL	Fixed
API Exposed To The Public	CRITICAL	Fixed
A Malicious User Can Tamper Addresses	HIGH	Fixed
Wallet Authentication Verifed By The Private Key	HIGH	Fixed
Public Key Can Be Tampered	HIGH	Fixed
withdrawToken Can Be Abused	HIGH	Acknowledged
Missing Middleware For An Inactive User	MEDIUM	Fixed
jwtSecret Is Hardcoded In The Authorizeservice	MEDIUM	Fixed
Overriding Completed Transactions	MEDIUM	Mitigated
Missing Transfer Verification	MEDIUM	Fixed
HS256 Used As Signing Algorithm	LOW	Acknowledged
getSecretKey Returns Predicted Output	LOW	Fixed
Missing Address Verification	LOW	Fixed

Missing Value Verification	LOW	Fixed
Floating Pragma	LOW	Fixed
Approve Race Condition	LOW	Akcnowledged
Owner Can Renounce Ownership	LOW	Acknowledged
Floating Pragma	LOW	Acknowledged
Add The Public Address In The JWT Token	INFORMATIONAL	Acknowledged

3 Finding Details

A signController.go

A.1 Missing amount check in signWithdraw [CRITICAL]

Description:

The signWithdraw API is used to generate signatures for the user to make them able to withdraw tokens. However, there is a missing check on the amount, anyone can generate a signature allowing him to withdraw any amount of tokens.

Code:

Listing 1: signController.go

```
amount, err := strconv.ParseFloat(request.Amount, 64)
if err != nil {
 return c.Status(500).JSON(m.InternalError{Message:
 "cannot parse string to float64"})
}
hash, err := instance.Hash(&bind.CallOpts{
 From: ownerAddress,
}, clientAddress, "Withdraw", tokenAddress, FloatEtherToBigInt(amount),
deadline)
if err != nil {
 log.Println("errorInSignWithdraw: ", err.Error())
 return c.Status(500).JSON(m.InternalError{Message: err.Error()})
}

sig, err := crypto.Sign(hash[:], privateKey)
if err != nil {
 return c.Status(500).JSON(m.InternalError{Message: err.Error()})
}
sig[64] += 27
```

```go

return c.Status(200).JSON(&SignWithdrawResponse{
 Signature: hexutil.Encode(sig[:]),
 Deadline: deadline.Int64(),
 AmountString: FloatEtherToBigInt(amount).String(),
})
```

Risk Level:

Likelihood – 5
Impact - 5

Recommendation:

It is recommended to verify that the user can generate a signature to withdraw only the amount that was already deposited, this can be achieved by first getting a signature from the user and extracting his address, then calling the contract to extract the deposited amount using his address and restricting the user's withdraw to be equal or less than this amount.

Status - Fixed

The PXP team has fixed the issue by adding a verification to the amount that is provided by the user.

A.2 Infinite Withdraw Leads To The Drain Of The Contract [CRITICAL]

Description:

The SignWithdraw function generates the signature that the user will use in the contract to get tokens from the contract. A malicious user can generate an infinite amount of valid signatures and use them multiple times with different amounts to withdraw tokens.

Code:

Listing 2: signController.go

```go
sig, err := crypto.Sign(hash[:], privateKey)
if err != nil {
 return c.Status(500).JSON(m.InternalError{Message: err.Error()})
}
sig[64] += 27

return c.Status(200).JSON(&SignWithdrawResponse{
 Signature: hexutil.Encode(sig[:]),
 Deadline: deadline.Int64(),
 AmountString: FloatEtherToBigInt(amount).String(),
})
```

Exploit Scenario:

1. The malicious user will call the signWithdraw function with the amount 200 and get the associated signature.
2. The malicious user will call a second the signWithdraw but with an amount of 150.
3. The attaquant will submit two requests for withdraw in the contracts with different signatures, the call will succeed since he submitted them with different signatures.

Risk Level:

Likelihood – 4
Impact - 5

Recommendation:

It is recommended to verify from the contract the number of tokens that were already claimed by the user.

Status - Fixed

The PXP team has fixed by adding the verification in the goLang file and also in the contract by adding the following code in the contract.

Listing 3: PXPGateway.sol

```
require(_deadline <= block.timestamp, "Expired!");
```

A.3 A Malicious User Can Tamper Addresses [HIGH]

Description:

In the SignWithdraw you are using the contract address, the token address and the client address, these values are taken from the request. Thus, any user can inject in the body of the request fake values of other addresses and ruin the logic of the contract.

Code:

Listing 4: signController.go

```
contract := common.HexToAddress(request.ContractADDR)
instance, err := PXPGateWayABI.NewPXPGateWayABI(contract, client)
if err != nil {
 return c.Status(500).JSON(m.InternalError{Message: err.Error()})
}

clientAddress := common.HexToAddress(request.ClientADDR)
tokenAddress := common.HexToAddress(request.TokenADDR)
```

Risk Level:

Likelihood – 4
Impact - 4

Recommendation:

The contract and token addresses should be hard-coded and for the user address, it should be extracted from the signature.

Status - Fixed

The PXP team has fixed the issue by hard-coding the contract and token addresses and extracting the user's address from the signature.

B routes.go

B.1 API Exposed To The Public [CRITICAL]

Description:

The two APIs signWithdraw and signDeposit return a signature generated by the server, these APIs are not protected by a middleware or an authorization mechanism, and thus anyone can call them and generate the signature for a particular user and withdraw any number of tokens.

Code:

Listing 5: routes.go

```go
func Setup(app *fiber.App) {
 api := app.Group(path)
 app.Use(logger.New())

 api.Post("/signWithdraw", controller.SignWithdraw)
 api.Post("/signDeposit", controller.SignDeposit)
}
```

Recommendation:

Consider adding an authorization middleware to verify the caller's identity.

Risk Level:

Likelihood – 5
Impact - 5

Status - Fixed

The PXP team has resolved the issue by adding a JWT authorization middleware.

B.2 Missing Middleware For An Inactive User [MEDIUM]

Description:

There is a missing check in the authentication process, the authentication should verify whether the user is Inactive or not. Thus, in this case, an inactive user can interact with these APIs without any restriction.

Code:

Listing 6: routes.go

```go
func Setup(app *fiber.App) {
 api := app.Group(path)
 app.Use(logger.New())

 api.Post("/signWithdraw", controller.SignWithdraw)
 api.Post("/signDeposit", controller.SignDeposit)
}
```

Recommendation:

Consider adding a middleware that verifies the status of the user and makes sure that it is active.

Risk Level:

Likelihood – 3
Impact - 4

Status - Fixed

The PXP team has fixed the issue by adding a verification in the login step that makes sure the user is active before returning the token.

C Authorizeservice.go

C.1 Wallet Authentication Verifed By The Private Key [HIGH]

Description:

The GetAuthWallet function is used to verify the authorization of the user, in the line 68 if the pk is empty the function returns a Bad Credentials message; otherwise it returns the public address of the user. This method will harm the user's privacy, since he will be exposing his private key to the server.

Code:

Listing 7: Authorizeservice.go

```
46 func GetAuthWallet(pk string, c *fiber.Ctx) (string, error) {
47  privateKey, err := crypto.HexToECDSA(pk)
48  if err != nil {
49   c.Status(fiber.StatusBadRequest).JSON(fiber.Map{
50    "error": "error get hexdata",
51    "msg": err.Error(),
52   })
53   return "", nil
54  }
55 
56  publicKey := privateKey.Public()
```

```
  publicKeyECDSA, ok := publicKey.(*ecdsa.PublicKey)
  if !ok {
   if err != nil {
    c.Status(fiber.StatusBadRequest).JSON(fiber.Map{
     "error": "error casting public key to ECDSA",
     "msg": err.Error(),
    })
    return "", nil
   }

  }
  if pk == "" {
   c.Status(fiber.StatusUnauthorized).JSON(fiber.Map{
    "error": "Bad Credentials",
   })
   return "", nil
  }
  fromAddress := crypto.PubkeyToAddress(*publicKeyECDSA)
  return fromAddress.String(), nil
 }
```

Recommendation:

Consider verifying the authorization using only the signature generated by the user's wallet.

Risk Level:

Likelihood – 4
Impact - 4

Status - Fixed

The PXP team has fixed the issue by removing the function.

C.2 Public Key Can Be Tampered [HIGH]

Description:

The query used in the Auth function takes the public key from the request, the issue here is that anyone can tamper this value with another public address other than the intended one.

Code:

Listing 8: Authorizeservice.go

```go
db := db.DBCtx
var user models.User
if err := db.Raw("EXECUTE m_user_login @m_owner = ? , @return_code = ?",
logIn.PublicKey, &returnCode).Scan(&user).Error; err != nil {
 if err != gorm.ErrRecordNotFound {
  log.Println(err.Error())
  return c.Status(500).JSON(fiber.Map{
   "message": "db error -> " + err.Error(),
   "code": 500,
  })
 }
}
```

Recommendation:

Consider extracting the public address from the user's signature.

Risk Level:

Likelihood – 4
Impact - 5

Status - Fixed

The PXP team has fixed the issue by using the user's signature to extract the address.

C.3 jwtSecret Is Hardcoded In The Authorizeservice [MEDIUM]

Description:

The Authorizeservice module contains the jwtSecret used to sign the transactions is hard-coded in the file. Therefore, allowing anyone who had access to the code to generate signed transactions using the secret key.

Code:

Listing 9: Authorizeservice.go

```
const (
 jwtSecret = "ea95b95c1976482f989db81903c01691"
)
```

Recommendation:

Consider removing the jwtSecret from the Authorizeservice.go file and storing it in the .env file, to note also that the .env should be added in the .gitignore.

Risk Level:

Likelihood – 3
Impact - 4

Status - Fixed

The PXP team has fixed the issue by getting the jwtSecret value from an env file.

C.4 HS256 Used As Signing Algorithm [LOW]

Description:

The JWT authentication uses as an algorithm the HS256 which is a symmetric algorithm, that means a single key is used to encrypt and decrypt data. In case of having untrusted entities, this will cause an issue of verifying using only the shared key.

Code:

Listing 10: Authorizeservice.go

```go
func createToken(userUid string, owner string) (MsgToken, error) {
 var msgToken MsgToken
 token := jwt.New(jwt.SigningMethodHS256)
 claims := token.Claims.(jwt.MapClaims)
 claims["sub"] = userUid
 claims["owner"] = owner
 claims["exp"] = time.Now().Add(time.Hour * 24).Unix()
 t, err := token.SignedString([]byte(jwtSecret))
 if err != nil {
  return msgToken, err
 }
 msgToken.AccessToken = t
```

Listing 11: Authorizeservice.go

```go
func AuthorizationRequired() fiber.Handler {
 return jwtware.New(jwtware.Config{
  // Filter: nil,
  SuccessHandler: AuthSuccess,
  ErrorHandler: AuthError,
  SigningKey: []byte(jwtSecret),
  // SigningKeys: nil,
  SigningMethod: "HS256",
  // ContextKey: nil,
  // Claims: nil,
    // TokenLookup: nil,
    // AuthScheme: nil,
  })
}
```

Recommendation:

Consider changing the algorithm to the RS256 which is an asymmetric algorithm.

Risk Level:

Likelihood – 1
Impact - 2

Status - Acknowledged

C.5 Add The Public Address In The JWT Token [INFORMATIONAL]

Description:

The JWT token generated using the createToken function contain the userId and the owner, it is recommended to add the public address of the user to optimize the number of queries.

Code:

Listing 12: Authorizeservice.go

```
func createToken(userUid string, owner string) (MsgToken, error) {
  var msgToken MsgToken
  token := jwt.New(jwt.SigningMethodHS256)
  claims := token.Claims.(jwt.MapClaims)
  claims["sub"] = userUid
  claims["owner"] = owner
  claims["exp"] = time.Now().Add(time.Hour * 24).Unix()
```

```
t, err := token.SignedString([]byte(jwtSecret))
if err != nil {
 return msgToken, err
}
msgToken.AccessToken = t
```

Recommendation:

Consider adding the public address of the user in the JWT token.

Status - Acknowledged

D Authservice.go

D.1 getSecretKey Returns Predicted Output [LOW]

Description:

The getSecretKey function returns the secret located in the environment variable on the system, if it is empty, it returns the string secret which is predictable and hard-coded.

Code:

Listing 13: Authservice.go

```
func getSecretKey() string {
 secret := os.Getenv("SECRET")
 if secret == "" {
  secret = "secret"
 }
 return secret
}
```

Recommendation:

Remove the empty check, and return an error if secret is empty.

Risk Level:

Likelihood – 1
Impact - 1

Status - Fixed

The PXP team has fixed the issue by removing the function.

E PXPGateWay.sol

E.1 withdrawToken Can Be Abused [HIGH]

Description:

A malicious user can abuse the withdrawToken and withdraw the totality of tokens. A security mechanism was already implemented by verifying that the _amount should be less than _maximumWithdraw and also checking that the _latestWithdrawal was more than 24 hours ago. The issue is that a user can bypass this by sending theirs token to a different wallet and calling a second time the withdrawToken.

Code:

Listing 14: PXPGateWay.sol

```
function withdrawToken(
    address _token,
    uint256 _amount,
    uint256 _deadline,
    bytes memory signature
) external nonReentrant {
    require(
        checkSignature(
            msg.sender,
            "Withdraw",
            _token,
            _amount,
            _deadline,
            signature
        ),
        "!sig"
    );
    require(_token == ACCEPTED_TOKEN, "Token not accepted");
    require(_amount <= _maximumWithdraw, "Over limit");
    require(_amount >= _minimumWithdraw, "Lower Minimum");
    require(
        _latestWithdrawal[msg.sender] == 0 ||
            _latestWithdrawal[msg.sender].add(24 hours) <= block.timestamp,
        "24Hr."
    );
    require(_signatureUsed[signature], "Hacked");

    _latestWithdrawal[msg.sender] = block.timestamp;
    _signatureUsed[signature] = true;

    ERC20(_token).safeTransferFrom(WALLET, msg.sender, _amount);
    emit Withdraw(msg.sender, _token, _amount);
}
```

Risk Level:

Likelihood – 2
Impact - 3

Recommendation:

A change in the architecture will be needed to remediate the risk, but if this feature is not required, the verification on the `_maximumWithdraw` and the `_latestWithdrawal` can be removed.

Status - Acknowledged

E.2 Overriding Completed Transactions [MEDIUM]

Description:

In the depositToken function, we are associating for each transaction the amount deposited in the wallet. The issue here is that the `_transactionId` is inserted by the user. Thus, a malicious user can use an existing transactionId and override the `_transactionIdCompleted` mapping with a lower amount.

Code:

Listing 15: PXPGateWay.sol

```
function depositToken(
    address _token,
    uint256 _amount,
    uint256 _transactionId
) external {
    require(WALLET != address(0), "Gate Closed");
    require(_token == ACCEPTED_TOKEN, "Token not accepted");
    require(_amount >= _minimumDeposit, "Lower Minimum");

    IERC20(_token).transferFrom(msg.sender, WALLET, _amount);
    emit Deposit(msg.sender, _token, _amount);

    _transactionIdCompleted[_transactionId] = _amount;
}
```

Risk Level:

Likelihood – 2
Impact - 3

Recommendation:

Use the transactionId as a private variable in the contract and for each deposit increment the variable.

Status - Mitigated

The PXP team has mitigated the issue by adding a require statement that ensures that the transactionId is not yet completed.

Code:

Listing 16: PXPGateWay.sol

```
    function depositToken(
        address _token,
        uint256 _amount,
        uint256 _transactionId,
        uint256 _deadline,
        bytes memory signature
    ) external {
        require(WALLET != address(0), "Gate Closed");
        require(_token == ACCEPTED_TOKEN, "Token not accepted");
        require(_amount >= _minimumDeposit, "Lower Minimum");
        require(
            _transactionIdCompleted[_transactionId] == 0,
            "Transaction already completed"
        );
```

E.3 Missing Transfer Verification [MEDIUM]

Description:

The ERC20 standard token implementation functions return the transaction status as a Boolean. It is good practice to check for the return status of the function call to ensure that the transaction was successful.

It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller transaction also fails. However, it is mostly missed by developers when they carry out checks in effect, the transaction would always succeed, even if the token transfer did not.

Code:

Listing 17: PXPGateWay.sol

```
IERC20(_token).transferFrom(msg.sender, WALLET, _amount);
emit Deposit(msg.sender, _token, _amount);
```

Listing 18: PXPGateWay.sol

```
IERC20(_token).transferFrom(WALLET, msg.sender, _amount);
emit Withdraw(msg.sender, _token, _amount);
```

Risk Level:

Likelihood – 2
Impact - 4

Recommendation:

It is recommended to use the safeTransfer function from the safeERC20 implementation, or put the transfer call inside an assert or require to verify that the transfer has passed successfully.

Status - Fixed

The PXP team has resolved the issue by using the safeTransfer function from the safeERC20 implementation to ensure that the transfer has passed successfully.

E.4 Missing Address Verification [LOW]

Description:

Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, some of the contract's functionality may become inaccessible. The `_banker`, `_token` and `_signer` arguments should be different from the address(0).

Code:

Listing 19: PXPGateWay.sol

```
function setWalletBanker(address _banker) external onlyRole(ADMIN_ROLE) {
    WALLET = _banker;
}
```

Listing 20: PXPGateWay.sol

```
function setTokenAccept(address _token) external onlyRole(ADMIN_ROLE) {
    ACCEPTED_TOKEN = _token;
}
```

Listing 21: PXPGateWay.sol

```
function setSigner(address _signer) external onlyRole(ADMIN_ROLE) {
    SIGNER = _signer;
}
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

It is recommended to verify that the addresses provided in the arguments are different from the address(0).

Status - Fixed

The PXP team has resolved the issue by adding require statements that verify that the address provided in the arguments are different from the address(0).

E.5 Missing Value Verification [LOW]

Description:

Certain functions lack a safety check in the values, the values of the arguments should be verified to allow only the ones that go with the contract's logic. The _minimum and _maximum variable should be different from 0, and the _maximum should be higher than the _minimum variable.

Code:

Listing 22: PXPGateWay.sol

```
function setMinimumDeposit(uint256 _minimum) external onlyRole(ADMIN_ROLE) {
    _minimumDeposit = _minimum;
}
```

Listing 23: PXPGateWay.sol

```
function setMinimumWithdraw(uint256 _minimum)
    external
    onlyRole(ADMIN_ROLE)
{
    _minimumWithdraw = _minimum;
}
```

Listing 24: PXPGateWay.sol

```
function setMaximumWithdraw(uint256 _maximum)
    external
    onlyRole(ADMIN_ROLE)
{
    _maximumWithdraw = _maximum;}
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

It's recommended to verify the values provided in the arguments. The concerns can be resolved by utilizing a require statement.

Status - Fixed

The PXP team has resolved the issue by adding require statements to verify the values coming from the arguments.

E.6 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.8.4. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version, that may introduce issues in the contract system.

Code:

Listing 25: PXPGateWay.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status - Fixed

The PXP team has resolved the issue by locking the pragma version to 0.8.6.

F PXPToken.sol

F.1 Approve Race Condition [LOW]

Description:

The ERC4626 contract uses the ERC20,the standard ERC20 implementation contains a widely known racing condition in it approve function, wherein a spender can witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender will be able to get both approval amounts of both transactions.

Code:

Listing 26: PXPToken.sol

```
contract PXPToken is ERC20, Pausable, Ownable {
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

Use increaseAllowance and decreaseAllowance functions to modify the approval amount instead of using the approve function to modify it.

Status - Akcnowledged

The PXP team has acknowledged the risk.

F.2 Owner Can Renounce Ownership [LOW]

Description:

Typically, the contract's owner is the account that deploys the contract. As a result, the owner can perform certain privileged activities. The renounceOwnership function is used in smart contracts to renounce ownership. However, if the contract's ownership has never been transferred before renouncing it, it will never have an Owner, which may result in a denial of service.

Code:

Listing 27: PXPToken.sol

```
contract PXPToken is ERC20, Pausable, Ownable {
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method will require two or more users to sign the transaction. Alternatively, the renounced ownership functionality can be disabled by overriding it.

Status - Acknowledged

The PXP team has acknowledged the risk.

F.3 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.8.0. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version, that may introduce issues in the contract system.

Code:

Listing 28: PXPToken.sol

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status - Acknowledged

The PXP team has acknowledged the risk.

4 Best Practices

BP.1 Variables should be initialized first

Description:

The `_maximumWithdraw`, `_minimumDeposit`, `_minimumWithdraw`, `ACCEPTED_TOKEN` and WALLET variables should be initialized in the initialize function. Otherwise, if someone calls the depositToken or the withdrawToken function, unexpected behaviors will be generated.

Code:

Listing 29: PXPGateWay.sol (Line 26)

```
address private ACCEPTED_TOKEN;
address private WALLET;

uint256 private _maximumWithdraw;
uint256 private _minimumDeposit;
```

Listing 30: PXPGateWay.sol (Line 37)

```
uint256 private _minimumWithdraw;
```

5 Static Analysis (Slither)

Description:

ShellBoxes expanded the coverage of the specific contract areas using automated testing methodologies. Slither, a Solidity static analysis framework, was one of the tools used. Slither was run on all-scoped contracts in both text and binary formats. This tool can be used to test mathematical relationships between Solidity instances statically and variables that allow for the detection of errors or inconsistent usage of the contracts' APIs throughout the entire codebase.

Results:

```
ERC1967UpgradeUpgradeable._functionDelegateCall(address,bytes) (../openzepp
elin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeabl
e.sol#198-204) uses delegatecall to a input-controlled function id
       - (success,returndata) = target.delegatecall(data) (../openzeppelin
-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.so
l#202)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#co
ntrolled-delegatecall

AccessControlUpgradeable.__gap (../openzeppelin-contracts-upgradeable/contr
acts/access/AccessControlUpgradeable.sol#247) shadows:
       - ERC165Upgradeable.__gap (../openzeppelin-contracts-upgradeable/co
ntracts/utils/introspection/ERC165Upgradeable.sol#41)
       - ContextUpgradeable.__gap (../openzeppelin-contracts-upgradeable/c
ontracts/utils/ContextUpgradeable.sol#36)
UUPSUpgradeable.__gap (../openzeppelin-contracts-upgradeable/contracts/prox
y/utils/UUPSUpgradeable.sol#107) shadows:
       - ERC1967UpgradeUpgradeable.__gap (../openzeppelin-contracts-upgrad
eable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#211)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#st
ate-variable-shadowing
```

```
PXPGateWay.depositToken(address,uint256,uint256) (PXPGateWay.sol#108-121) i
gnores return value by IERC20(_token).transferFrom(msg.sender,WALLET,_amoun
t) (PXPGateWay.sol#117)
PXPGateWay.withdrawToken(address,uint256,uint256,bytes) (PXPGateWay.sol#123
-154) ignores return value by IERC20(_token).transferFrom(WALLET,msg.sender
,_amount) (PXPGateWay.sol#152)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#un
checked-transfer

PXPGateWay (PXPGateWay.sol#15-182) is an upgradeable contract that does not
 protect its initiliaze functions: PXPGateWay.initialize() (PXPGateWay.sol#
47-54). Anyone can delete the contract with: UUPSUpgradeable.upgradeTo(addr
ess) (../openzeppelin-contracts-upgradeable/contracts/proxy/utils/UUPSUpgra
deable.sol#72-75)UUPSUpgradeable.upgradeToAndCall(address,bytes) (../openze
ppelin-contracts-upgradeable/contracts/proxy/utils/UUPSUpgradeable.sol#85-8
8)Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#
unprotected-upgradeable-contract

ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool).slot (.
./openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967Upgrade
Upgradeable.sol#98) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#un
initialized-local-variables

ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool) (../ope
nzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgra
deable.sol#87-105) ignores return value by IERC1822ProxiableUpgradeable(new
Implementation).proxiableUUID() (../openzeppelin-contracts-upgradeable/cont
racts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#98-102)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#un
used-return

PXPGateWay.setWalletBanker(address)._banker (PXPGateWay.sol#62) lacks a zer
```

```
o-check on :
              - WALLET = _banker (PXPGateWay.sol#63)
PXPGateWay.setTokenAccept(address)._token (PXPGateWay.sol#66) lacks a zero-
check on :
              - ACCEPTED_TOKEN = _token (PXPGateWay.sol#67)
PXPGateWay.setSigner(address)._signer (PXPGateWay.sol#74) lacks a zero-chec
k on :
              - SIGNER = _signer (PXPGateWay.sol#75)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#mi
ssing-zero-address-validation

Variable 'ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,boo
l).slot (../openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1
967UpgradeUpgradeable.sol#98)' in ERC1967UpgradeUpgradeable._upgradeToAndCa
llUUPS(address,bytes,bool) (../openzeppelin-contracts-upgradeable/contracts
/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#87-105) potentially used befor
e declaration: require(bool,string)(slot == _IMPLEMENTATION_SLOT,ERC1967Upg
rade: unsupported proxiableUUID) (../openzeppelin-contracts-upgradeable/con
tracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#99)
Variable 'ECDSA.tryRecover(bytes32,bytes).r (../openzeppelin-contracts/cont
racts/utils/cryptography/ECDSA.sol#59)' in ECDSA.tryRecover(bytes32,bytes)
(../openzeppelin-contracts/contracts/utils/cryptography/ECDSA.sol#54-83) po
tentially used before declaration: r = mload(uint256)(signature + 0x20) (..
/openzeppelin-contracts/contracts/utils/cryptography/ECDSA.sol#76)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pr
e-declaration-usage-of-local-variables

Reentrancy in PXPGateWay.depositToken(address,uint256,uint256) (PXPGateWay.
sol#108-121):
       External calls:
       - IERC20(_token).transferFrom(msg.sender,WALLET,_amount) (PXPGateWa
y.sol#117)
       State variables written after the call(s):
       - _transactionIdCompleted[_transactionId] = _amount (PXPGateWay.sol
#120)
Reentrancy in PXPGateWay.withdrawToken(address,uint256,uint256,bytes) (PXPG
ateWay.sol#123-154):
	External calls:
	- require(bool,string)(checkSignature(msg.sender,Withdraw,_token,_a
mount,_deadline,signature),!sig) (PXPGateWay.sol#129-139)
		- SignatureChecker.isValidSignatureNow(SIGNER,h,signature)
(PXPGateWay.sol#165)
		- (success,result) = signer.staticcall(abi.encodeWithSelect
or(IERC1271.isValidSignature.selector,hash,signature)) (../openzeppelin-con
tracts/contracts/utils/cryptography/SignatureChecker.sol#30-32)
	State variables written after the call(s):
	- _latestWithdrawal[msg.sender] = block.timestamp (PXPGateWay.sol#1
50)
	- _signatureUsed[signature] = true (PXPGateWay.sol#151)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#re
entrancy-vulnerabilities-2

Reentrancy in PXPGateWay.depositToken(address,uint256,uint256) (PXPGateWay.
sol#108-121):
	External calls:
	- IERC20(_token).transferFrom(msg.sender,WALLET,_amount) (PXPGateWa
y.sol#117)
	Event emitted after the call(s):
	- Deposit(msg.sender,_token,_amount) (PXPGateWay.sol#118)
Reentrancy in PXPGateWay.withdrawToken(address,uint256,uint256,bytes) (PXPG
ateWay.sol#123-154):
	External calls:
	- require(bool,string)(checkSignature(msg.sender,Withdraw,_token,_a
mount,_deadline,signature),!sig) (PXPGateWay.sol#129-139)
		- SignatureChecker.isValidSignatureNow(SIGNER,h,signature)
(PXPGateWay.sol#165)
		- (success,result) = signer.staticcall(abi.encodeWithSelect
or(IERC1271.isValidSignature.selector,hash,signature)) (../openzeppelin-con
```

```
tracts/contracts/utils/cryptography/SignatureChecker.sol#30-32)
	- IERC20(_token).transferFrom(WALLET,msg.sender,_amount) (PXPGateWa
y.sol#152)
	Event emitted after the call(s):
	- Withdraw(msg.sender,_token,_amount) (PXPGateWay.sol#153)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#re
entrancy-vulnerabilities-3

PXPGateWay.withdrawToken(address,uint256,uint256,bytes) (PXPGateWay.sol#123
-154) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(_latestWithdrawal[msg.sender] == 0 || _lates
tWithdrawal[msg.sender].add(86400) <= block.timestamp,24Hr.) (PXPGateWay.so
l#143-147)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#bl
ock-timestamp

AddressUpgradeable.verifyCallResult(bool,bytes,string) (../openzeppelin-con
tracts-upgradeable/contracts/utils/AddressUpgradeable.sol#174-194) uses ass
embly
	- INLINE ASM (../openzeppelin-contracts-upgradeable/contracts/utils
/AddressUpgradeable.sol#186-189)
StorageSlotUpgradeable.getAddressSlot(bytes32) (../openzeppelin-contracts-u
pgradeable/contracts/utils/StorageSlotUpgradeable.sol#52-56) uses assembly
	- INLINE ASM (../openzeppelin-contracts-upgradeable/contracts/utils
/StorageSlotUpgradeable.sol#53-55)
StorageSlotUpgradeable.getBooleanSlot(bytes32) (../openzeppelin-contracts-u
pgradeable/contracts/utils/StorageSlotUpgradeable.sol#61-65) uses assembly
	- INLINE ASM (../openzeppelin-contracts-upgradeable/contracts/utils
/StorageSlotUpgradeable.sol#62-64)
StorageSlotUpgradeable.getBytes32Slot(bytes32) (../openzeppelin-contracts-u
pgradeable/contracts/utils/StorageSlotUpgradeable.sol#70-74) uses assembly
	- INLINE ASM (../openzeppelin-contracts-upgradeable/contracts/utils
/StorageSlotUpgradeable.sol#71-73)
```

```
StorageSlotUpgradeable.getUint256Slot(bytes32) (../openzeppelin-contracts-u
pgradeable/contracts/utils/StorageSlotUpgradeable.sol#79-83) uses assembly
       - INLINE ASM (../openzeppelin-contracts-upgradeable/contracts/utils
/StorageSlotUpgradeable.sol#80-82)
Address.isContract(address) (../openzeppelin-contracts/contracts/utils/Addr
ess.sol#26-36) uses assembly
       - INLINE ASM (../openzeppelin-contracts/contracts/utils/Address.sol
#32-34)
Address.verifyCallResult(bool,bytes,string) (../openzeppelin-contracts/cont
racts/utils/Address.sol#195-215) uses assembly
       - INLINE ASM (../openzeppelin-contracts/contracts/utils/Address.sol
#207-210)
ECDSA.tryRecover(bytes32,bytes) (../openzeppelin-contracts/contracts/utils/
cryptography/ECDSA.sol#54-83) uses assembly
       - INLINE ASM (../openzeppelin-contracts/contracts/utils/cryptograph
y/ECDSA.sol#64-68)
       - INLINE ASM (../openzeppelin-contracts/contracts/utils/cryptograph
y/ECDSA.sol#75-78)
ECDSA.tryRecover(bytes32,bytes32,bytes32) (../openzeppelin-contracts/contra
cts/utils/cryptography/ECDSA.sol#112-124) uses assembly
       - INLINE ASM (../openzeppelin-contracts/contracts/utils/cryptograph
y/ECDSA.sol#119-122)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#as
sembly-usage

Different versions of Solidity is used:
       - Version used: ['^0.8.0', '^0.8.1', '^0.8.2', '^0.8.4']
       - ^0.8.4 (PXPGateWay.sol#2)
       - ^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/access/Ac
cessControlUpgradeable.sol#4)
       - ^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/access/IA
ccessControlUpgradeable.sol#4)
       - ^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/interface
s/draft-IERC1822Upgradeable.sol#4)
```

```
        - ^0.8.2 (../openzeppelin-contracts-upgradeable/contracts/proxy/ERC
1967/ERC1967UpgradeUpgradeable.sol#4)
        - ^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/proxy/bea
con/IBeaconUpgradeable.sol#4)
        - ^0.8.2 (../openzeppelin-contracts-upgradeable/contracts/proxy/uti
ls/Initializable.sol#4)
        - ^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/proxy/uti
ls/UUPSUpgradeable.sol#4)
        - ^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/security/
ReentrancyGuardUpgradeable.sol#4)
        - ^0.8.1 (../openzeppelin-contracts-upgradeable/contracts/utils/Add
ressUpgradeable.sol#4)
        - ^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/utils/Con
textUpgradeable.sol#4)
        - ^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/utils/Sto
rageSlotUpgradeable.sol#4)
        - ^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/utils/Str
ingsUpgradeable.sol#4)
        - ^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/utils/int
rospection/ERC165Upgradeable.sol#4)
        - ^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/utils/int
rospection/IERC165Upgradeable.sol#4)
        - ^0.8.0 (../openzeppelin-contracts/contracts/interfaces/IERC1271.s
ol#3)
        - ^0.8.0 (../openzeppelin-contracts/contracts/token/ERC20/IERC20.so
l#3)
        - ^0.8.0 (../openzeppelin-contracts/contracts/utils/Address.sol#3)
        - ^0.8.0 (../openzeppelin-contracts/contracts/utils/cryptography/EC
DSA.sol#3)
        - ^0.8.0 (../openzeppelin-contracts/contracts/utils/cryptography/Si
gnatureChecker.sol#3)
        - ^0.8.0 (../openzeppelin-contracts/contracts/utils/math/SafeMath.s
ol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#di
```

```
fferent-pragma-directives-are-used

AccessControlUpgradeable.__AccessControl_init_unchained() (../openzeppelin-
contracts-upgradeable/contracts/access/AccessControlUpgradeable.sol#54-55)
is never used and should be removed
AccessControlUpgradeable._setRoleAdmin(bytes32,bytes32) (../openzeppelin-co
ntracts-upgradeable/contracts/access/AccessControlUpgradeable.sol#212-216)
is never used and should be removed
AccessControlUpgradeable._setupRole(bytes32,address) (../openzeppelin-contr
acts-upgradeable/contracts/access/AccessControlUpgradeable.sol#203-205) is
never used and should be removed
Address.functionCall(address,bytes) (../openzeppelin-contracts/contracts/ut
ils/Address.sol#79-81) is never used and should be removed
Address.functionCall(address,bytes,string) (../openzeppelin-contracts/contr
acts/utils/Address.sol#89-95) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (../openzeppelin-contr
acts/contracts/utils/Address.sol#108-114) is never used and should be remov
ed
Address.functionCallWithValue(address,bytes,uint256,string) (../openzeppeli
n-contracts/contracts/utils/Address.sol#122-133) is never used and should b
e removed
Address.functionDelegateCall(address,bytes) (../openzeppelin-contracts/cont
racts/utils/Address.sol#168-170) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (../openzeppelin-contrac
ts/contracts/utils/Address.sol#178-187) is never used and should be removed
Address.functionStaticCall(address,bytes) (../openzeppelin-contracts/contra
cts/utils/Address.sol#141-143) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (../openzeppelin-contracts
/contracts/utils/Address.sol#151-160) is never used and should be removed
Address.isContract(address) (../openzeppelin-contracts/contracts/utils/Addr
ess.sol#26-36) is never used and should be removed
Address.sendValue(address,uint256) (../openzeppelin-contracts/contracts/uti
ls/Address.sol#54-59) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (../openzeppelin-contracts/cont
racts/utils/Address.sol#195-215) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes) (../openzeppelin-contracts-u
pgradeable/contracts/utils/AddressUpgradeable.sol#85-87) is never used and
should be removed
AddressUpgradeable.functionCall(address,bytes,string) (../openzeppelin-cont
racts-upgradeable/contracts/utils/AddressUpgradeable.sol#95-101) is never u
sed and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (../openzep
pelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#114-120)
 is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (../
openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#1
28-139) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes) (../openzeppelin-contr
acts-upgradeable/contracts/utils/AddressUpgradeable.sol#147-149) is never u
sed and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (../openzeppeli
n-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#157-166) is
never used and should be removed
AddressUpgradeable.sendValue(address,uint256) (../openzeppelin-contracts-up
gradeable/contracts/utils/AddressUpgradeable.sol#60-65) is never used and s
hould be removed
ContextUpgradeable.__Context_init() (../openzeppelin-contracts-upgradeable/
contracts/utils/ContextUpgradeable.sol#18-19) is never used and should be r
emoved
ContextUpgradeable.__Context_init_unchained() (../openzeppelin-contracts-up
gradeable/contracts/utils/ContextUpgradeable.sol#21-22) is never used and s
hould be removed
ContextUpgradeable._msgData() (../openzeppelin-contracts-upgradeable/contra
cts/utils/ContextUpgradeable.sol#27-29) is never used and should be removed
ECDSA._throwError(ECDSA.RecoverError) (../openzeppelin-contracts/contracts/
utils/cryptography/ECDSA.sol#20-32) is never used and should be removed
ECDSA.recover(bytes32,bytes) (../openzeppelin-contracts/contracts/utils/cry
ptography/ECDSA.sol#99-103) is never used and should be removed
```

```
ECDSA.recover(bytes32,bytes32,bytes32) (../openzeppelin-contracts/contracts
/utils/cryptography/ECDSA.sol#131-139) is never used and should be removed
ECDSA.recover(bytes32,uint8,bytes32,bytes32) (../openzeppelin-contracts/con
tracts/utils/cryptography/ECDSA.sol#182-191) is never used and should be re
moved
ECDSA.toTypedDataHash(bytes32,bytes32) (../openzeppelin-contracts/contracts
/utils/cryptography/ECDSA.sol#216-218) is never used and should be removed
ERC165Upgradeable.__ERC165_init() (../openzeppelin-contracts-upgradeable/co
ntracts/utils/introspection/ERC165Upgradeable.sol#24-25) is never used and
should be removed
ERC165Upgradeable.__ERC165_init_unchained() (../openzeppelin-contracts-upgr
adeable/contracts/utils/introspection/ERC165Upgradeable.sol#27-28) is never
 used and should be removed
ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init() (../openzeppelin-contract
s-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#21-22)
is never used and should be removed
ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init_unchained() (../openzeppeli
n-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.s
ol#24-25) is never used and should be removed
ERC1967UpgradeUpgradeable._changeAdmin(address) (../openzeppelin-contracts-
upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#139-142)
is never used and should be removed
ERC1967UpgradeUpgradeable._getAdmin() (../openzeppelin-contracts-upgradeabl
e/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#122-124) is never u
sed and should be removed
ERC1967UpgradeUpgradeable._getBeacon() (../openzeppelin-contracts-upgradeab
le/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#158-160) is never
used and should be removed
ERC1967UpgradeUpgradeable._setAdmin(address) (../openzeppelin-contracts-upg
radeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#129-132) is
never used and should be removed
ERC1967UpgradeUpgradeable._setBeacon(address) (../openzeppelin-contracts-up
gradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#165-172) is
 never used and should be removed
```

```
ERC1967UpgradeUpgradeable._upgradeBeaconToAndCall(address,bytes,bool) (../o
penzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpg
radeable.sol#180-190) is never used and should be removed
Initializable._disableInitializers() (../openzeppelin-contracts-upgradeable
/contracts/proxy/utils/Initializable.sol#129-131) is never used and should
be removed
PXPGateWay._authorizeUpgrade(address) (PXPGateWay.sol#56-60) is never used
and should be removed
ReentrancyGuardUpgradeable.__ReentrancyGuard_init() (../openzeppelin-contra
cts-upgradeable/contracts/security/ReentrancyGuardUpgradeable.sol#40-42) is
 never used and should be removed
ReentrancyGuardUpgradeable.__ReentrancyGuard_init_unchained() (../openzeppe
lin-contracts-upgradeable/contracts/security/ReentrancyGuardUpgradeable.sol
#44-46) is never used and should be removed
SafeMath.div(uint256,uint256) (../openzeppelin-contracts/contracts/utils/ma
th/SafeMath.sol#134-136) is never used and should be removed
SafeMath.div(uint256,uint256,string) (../openzeppelin-contracts/contracts/u
tils/math/SafeMath.sol#190-199) is never used and should be removed
SafeMath.mod(uint256,uint256) (../openzeppelin-contracts/contracts/utils/ma
th/SafeMath.sol#150-152) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (../openzeppelin-contracts/contracts/u
tils/math/SafeMath.sol#216-225) is never used and should be removed
SafeMath.mul(uint256,uint256) (../openzeppelin-contracts/contracts/utils/ma
th/SafeMath.sol#120-122) is never used and should be removed
SafeMath.sub(uint256,uint256) (../openzeppelin-contracts/contracts/utils/ma
th/SafeMath.sol#106-108) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (../openzeppelin-contracts/contracts/u
tils/math/SafeMath.sol#167-176) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (../openzeppelin-contracts/contracts/utils
/math/SafeMath.sol#21-27) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (../openzeppelin-contracts/contracts/utils
/math/SafeMath.sol#63-68) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (../openzeppelin-contracts/contracts/utils
/math/SafeMath.sol#75-80) is never used and should be removed
```

```
SafeMath.tryMul(uint256,uint256) (../openzeppelin-contracts/contracts/utils
/math/SafeMath.sol#46-56) is never used and should be removed
SafeMath.trySub(uint256,uint256) (../openzeppelin-contracts/contracts/utils
/math/SafeMath.sol#34-39) is never used and should be removed
StorageSlotUpgradeable.getBytes32Slot(bytes32) (../openzeppelin-contracts-u
pgradeable/contracts/utils/StorageSlotUpgradeable.sol#70-74) is never used
and should be removed
StorageSlotUpgradeable.getUint256Slot(bytes32) (../openzeppelin-contracts-u
pgradeable/contracts/utils/StorageSlotUpgradeable.sol#79-83) is never used
and should be removed
StringsUpgradeable.toHexString(uint256) (../openzeppelin-contracts-upgradea
ble/contracts/utils/StringsUpgradeable.sol#40-51) is never used and should
be removed
StringsUpgradeable.toString(uint256) (../openzeppelin-contracts-upgradeable
/contracts/utils/StringsUpgradeable.sol#15-35) is never used and should be
removed
UUPSUpgradeable.__UUPSUpgradeable_init_unchained() (../openzeppelin-contrac
ts-upgradeable/contracts/proxy/utils/UUPSUpgradeable.sol#26-27) is never us
ed and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#de
ad-code

Pragma version^0.8.4 (PXPGateWay.sol#2) necessitates a version too recent t
o be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/acces
s/AccessControlUpgradeable.sol#4) necessitates a version too recent to be t
rusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/acces
s/IAccessControlUpgradeable.sol#4) necessitates a version too recent to be
trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/inter
faces/draft-IERC1822Upgradeable.sol#4) necessitates a version too recent to
 be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.2 (../openzeppelin-contracts-upgradeable/contracts/proxy
/ERC1967/ERC1967UpgradeUpgradeable.sol#4) necessitates a version too recent
 to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/proxy
/beacon/IBeaconUpgradeable.sol#4) necessitates a version too recent to be t
rusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.2 (../openzeppelin-contracts-upgradeable/contracts/proxy
/utils/Initializable.sol#4) necessitates a version too recent to be trusted
. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/proxy
/utils/UUPSUpgradeable.sol#4) necessitates a version too recent to be trust
ed. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/secur
ity/ReentrancyGuardUpgradeable.sol#4) necessitates a version too recent to
be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.1 (../openzeppelin-contracts-upgradeable/contracts/utils
/AddressUpgradeable.sol#4) necessitates a version too recent to be trusted.
 Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/utils
/ContextUpgradeable.sol#4) necessitates a version too recent to be trusted.
 Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/utils
/StorageSlotUpgradeable.sol#4) necessitates a version too recent to be trus
ted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/utils
/StringsUpgradeable.sol#4) necessitates a version too recent to be trusted.
 Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/utils
/introspection/ERC165Upgradeable.sol#4) necessitates a version too recent t
o be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts-upgradeable/contracts/utils
/introspection/IERC165Upgradeable.sol#4) necessitates a version too recent
to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts/contracts/interfaces/IERC12
71.sol#3) necessitates a version too recent to be trusted. Consider deployi
```

```
ng with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts/contracts/token/ERC20/IERC2
0.sol#3) necessitates a version too recent to be trusted. Consider deployin
g with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts/contracts/utils/Address.sol
#3) necessitates a version too recent to be trusted. Consider deploying wit
h 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts/contracts/utils/cryptograph
y/ECDSA.sol#3) necessitates a version too recent to be trusted. Consider de
ploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts/contracts/utils/cryptograph
y/SignatureChecker.sol#3) necessitates a version too recent to be trusted.
Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts/contracts/utils/math/SafeMa
th.sol#3) necessitates a version too recent to be trusted. Consider deployi
ng with 0.6.12/0.7.6
solc-0.8.6 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#in
correct-versions-of-solidity

Low level call in ERC1967UpgradeUpgradeable._functionDelegateCall(address,b
ytes) (../openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC196
7UpgradeUpgradeable.sol#198-204):
       - (success,returndata) = target.delegatecall(data) (../openzeppelin
-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.so
l#202)
Low level call in AddressUpgradeable.sendValue(address,uint256) (../openzep
pelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#60-65):
       - (success) = recipient.call{value: amount}() (../openzeppelin-cont
racts-upgradeable/contracts/utils/AddressUpgradeable.sol#63)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,ui
nt256,string) (../openzeppelin-contracts-upgradeable/contracts/utils/Addres
sUpgradeable.sol#128-139):
       - (success,returndata) = target.call{value: value}(data) (../openze
ppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#137)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,strin
g) (../openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeabl
e.sol#157-166):
	- (success,returndata) = target.staticcall(data) (../openzeppelin-c
ontracts-upgradeable/contracts/utils/AddressUpgradeable.sol#164)
Low level call in Address.sendValue(address,uint256) (../openzeppelin-contr
acts/contracts/utils/Address.sol#54-59):
	- (success) = recipient.call{value: amount}() (../openzeppelin-cont
racts/contracts/utils/Address.sol#57)
Low level call in Address.functionCallWithValue(address,bytes,uint256,strin
g) (../openzeppelin-contracts/contracts/utils/Address.sol#122-133):
	- (success,returndata) = target.call{value: value}(data) (../openze
ppelin-contracts/contracts/utils/Address.sol#131)
Low level call in Address.functionStaticCall(address,bytes,string) (../open
zeppelin-contracts/contracts/utils/Address.sol#151-160):
	- (success,returndata) = target.staticcall(data) (../openzeppelin-c
ontracts/contracts/utils/Address.sol#158)
Low level call in Address.functionDelegateCall(address,bytes,string) (../op
enzeppelin-contracts/contracts/utils/Address.sol#178-187):
	- (success,returndata) = target.delegatecall(data) (../openzeppelin
-contracts/contracts/utils/Address.sol#185)
Low level call in SignatureChecker.isValidSignatureNow(address,bytes32,byte
s) (../openzeppelin-contracts/contracts/utils/cryptography/SignatureChecker
.sol#20-34):
	- (success,result) = signer.staticcall(abi.encodeWithSelector(IERC1
271.isValidSignature.selector,hash,signature)) (../openzeppelin-contracts/c
ontracts/utils/cryptography/SignatureChecker.sol#30-32)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#lo
w-level-calls

Parameter PXPGateWay.setWalletBanker(address)._banker (PXPGateWay.sol#62) i
s not in mixedCase
Parameter PXPGateWay.setTokenAccept(address)._token (PXPGateWay.sol#66) is
```

```
not in mixedCase
Parameter PXPGateWay.setMinimumDeposit(uint256)._minimum (PXPGateWay.sol#70
) is not in mixedCase
Parameter PXPGateWay.setSigner(address)._signer (PXPGateWay.sol#74) is not
in mixedCase
Parameter PXPGateWay.setMinimumWithdraw(uint256)._minimum (PXPGateWay.sol#8
2) is not in mixedCase
Parameter PXPGateWay.setMaximumWithdraw(uint256)._maximum (PXPGateWay.sol#8
9) is not in mixedCase
Parameter PXPGateWay.getTransactionIdAmount(uint256)._transactionId (PXPGat
eWay.sol#100) is not in mixedCase
Parameter PXPGateWay.depositToken(address,uint256,uint256)._token (PXPGateW
ay.sol#109) is not in mixedCase
Parameter PXPGateWay.depositToken(address,uint256,uint256)._amount (PXPGate
Way.sol#110) is not in mixedCase
Parameter PXPGateWay.depositToken(address,uint256,uint256)._transactionId (
PXPGateWay.sol#111) is not in mixedCase
Parameter PXPGateWay.withdrawToken(address,uint256,uint256,bytes)._token (P
XPGateWay.sol#124) is not in mixedCase
Parameter PXPGateWay.withdrawToken(address,uint256,uint256,bytes)._amount (
PXPGateWay.sol#125) is not in mixedCase
Parameter PXPGateWay.withdrawToken(address,uint256,uint256,bytes)._deadline
 (PXPGateWay.sol#126) is not in mixedCase
Variable PXPGateWay.ACCEPTED_TOKEN (PXPGateWay.sol#26) is not in mixedCase
Variable PXPGateWay.WALLET (PXPGateWay.sol#27) is not in mixedCase
Variable PXPGateWay.SIGNER (PXPGateWay.sol#32) is not in mixedCase
Function AccessControlUpgradeable.__AccessControl_init() (../openzeppelin-c
ontracts-upgradeable/contracts/access/AccessControlUpgradeable.sol#51-52) i
s not in mixedCase
Function AccessControlUpgradeable.__AccessControl_init_unchained() (../open
zeppelin-contracts-upgradeable/contracts/access/AccessControlUpgradeable.so
l#54-55) is not in mixedCase
Variable AccessControlUpgradeable.__gap (../openzeppelin-contracts-upgradea
ble/contracts/access/AccessControlUpgradeable.sol#247) is not in mixedCase
```

```
Function ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init() (../openzeppelin
-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.so
l#21-22) is not in mixedCase
Function ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init_unchained() (../op
enzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgr
adeable.sol#24-25) is not in mixedCase
Variable ERC1967UpgradeUpgradeable.__gap (../openzeppelin-contracts-upgrade
able/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#211) is not in m
ixedCase
Function UUPSUpgradeable.__UUPSUpgradeable_init() (../openzeppelin-contract
s-upgradeable/contracts/proxy/utils/UUPSUpgradeable.sol#23-24) is not in mi
xedCase
Function UUPSUpgradeable.__UUPSUpgradeable_init_unchained() (../openzeppeli
n-contracts-upgradeable/contracts/proxy/utils/UUPSUpgradeable.sol#26-27) is
 not in mixedCase
Variable UUPSUpgradeable.__gap (../openzeppelin-contracts-upgradeable/contr
acts/proxy/utils/UUPSUpgradeable.sol#107) is not in mixedCase
Variable UUPSUpgradeable.__self (../openzeppelin-contracts-upgradeable/cont
racts/proxy/utils/UUPSUpgradeable.sol#29) is not in mixedCase
Function ReentrancyGuardUpgradeable.__ReentrancyGuard_init() (../openzeppel
in-contracts-upgradeable/contracts/security/ReentrancyGuardUpgradeable.sol#
40-42) is not in mixedCase
Function ReentrancyGuardUpgradeable.__ReentrancyGuard_init_unchained() (../
openzeppelin-contracts-upgradeable/contracts/security/ReentrancyGuardUpgrad
eable.sol#44-46) is not in mixedCase
Variable ReentrancyGuardUpgradeable.__gap (../openzeppelin-contracts-upgrad
eable/contracts/security/ReentrancyGuardUpgradeable.sol#74) is not in mixed
Case
Function ContextUpgradeable.__Context_init() (../openzeppelin-contracts-upg
radeable/contracts/utils/ContextUpgradeable.sol#18-19) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (../openzeppelin-con
tracts-upgradeable/contracts/utils/ContextUpgradeable.sol#21-22) is not in
mixedCase
Variable ContextUpgradeable.__gap (../openzeppelin-contracts-upgradeable/co
ntracts/utils/ContextUpgradeable.sol#36) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init() (../openzeppelin-contracts-upgra
deable/contracts/utils/introspection/ERC165Upgradeable.sol#24-25) is not in
 mixedCase
Function ERC165Upgradeable.__ERC165_init_unchained() (../openzeppelin-contr
acts-upgradeable/contracts/utils/introspection/ERC165Upgradeable.sol#27-28)
 is not in mixedCase
Variable ERC165Upgradeable.__gap (../openzeppelin-contracts-upgradeable/con
tracts/utils/introspection/ERC165Upgradeable.sol#41) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#co
nformance-to-solidity-naming-conventions

PXPGateWay (PXPGateWay.sol#15-182) does not implement functions:
	- UUPSUpgradeable._authorizeUpgrade(address) (../openzeppelin-contr
acts-upgradeable/contracts/proxy/utils/UUPSUpgradeable.sol#100)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#un
implemented-functions

UUPSUpgradeable.__gap (../openzeppelin-contracts-upgradeable/contracts/prox
y/utils/UUPSUpgradeable.sol#107) is never used in PXPGateWay (PXPGateWay.so
l#15-182)
PXPGateWay._withdrawAllowance (PXPGateWay.sol#34) is never used in PXPGateW
ay (PXPGateWay.sol#15-182)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#un
used-state-variable

initialize() should be declared external:
	- PXPGateWay.initialize() (PXPGateWay.sol#47-54)
grantRole(bytes32,address) should be declared external:
	- AccessControlUpgradeable.grantRole(bytes32,address) (../openzeppe
lin-contracts-upgradeable/contracts/access/AccessControlUpgradeable.sol#148
-150)
revokeRole(bytes32,address) should be declared external:
	- AccessControlUpgradeable.revokeRole(bytes32,address) (../openzepp
elin-contracts-upgradeable/contracts/access/AccessControlUpgradeable.sol#16
1-163)
renounceRole(bytes32,address) should be declared external:
       - AccessControlUpgradeable.renounceRole(bytes32,address) (../openze
ppelin-contracts-upgradeable/contracts/access/AccessControlUpgradeable.sol#
179-183)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pu
blic-function-that-could-be-declared-external

Context._msgData() (../openzeppelin-contracts/contracts/utils/Context.sol#2
0-22) is never used and should be removed
ERC20._burn(address,uint256) (../openzeppelin-contracts/contracts/token/ERC
20/ERC20.sol#274-289) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#de
ad-code

Pragma version^0.8.0 (PXPToken.sol#2) necessitates a version too recent to
be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts/contracts/access/Ownable.so
l#3) necessitates a version too recent to be trusted. Consider deploying wi
th 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts/contracts/security/Pausable
.sol#3) necessitates a version too recent to be trusted. Consider deploying
 with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts/contracts/token/ERC20/ERC20
.sol#3) necessitates a version too recent to be trusted. Consider deploying
 with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts/contracts/token/ERC20/IERC2
0.sol#3) necessitates a version too recent to be trusted. Consider deployin
g with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts/contracts/token/ERC20/exten
sions/IERC20Metadata.sol#3) necessitates a version too recent to be trusted
. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../openzeppelin-contracts/contracts/utils/Context.sol
#3) necessitates a version too recent to be trusted. Consider deploying wit
h 0.6.12/0.7.6
solc-0.8.6 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#in
correct-versions-of-solidity

PXPToken.constructor() (PXPToken.sol#10-12) uses literals with too many dig
its:
       - _mint(msg.sender,100000000 * 10 ** decimals()) (PXPToken.sol#11)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#to
o-many-digits

pause() should be declared external:
       - PXPToken.pause() (PXPToken.sol#14-16)
unpause() should be declared external:
       - PXPToken.unpause() (PXPToken.sol#18-20)
renounceOwnership() should be declared external:
       - Ownable.renounceOwnership() (../openzeppelin-contracts/contracts/
access/Ownable.sol#53-55)
transferOwnership(address) should be declared external:
       - Ownable.transferOwnership(address) (../openzeppelin-contracts/con
tracts/access/Ownable.sol#61-64)
name() should be declared external:
       - ERC20.name() (../openzeppelin-contracts/contracts/token/ERC20/ERC
20.sol#61-63)
symbol() should be declared external:
       - ERC20.symbol() (../openzeppelin-contracts/contracts/token/ERC20/E
RC20.sol#69-71)
totalSupply() should be declared external:
       - ERC20.totalSupply() (../openzeppelin-contracts/contracts/token/ER
C20/ERC20.sol#93-95)
balanceOf(address) should be declared external:
       - ERC20.balanceOf(address) (../openzeppelin-contracts/contracts/tok
en/ERC20/ERC20.sol#100-102)
```

```
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (../openzeppelin-contracts/contra
cts/token/ERC20/ERC20.sol#112-115)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (../openzeppelin-contracts/contr
acts/token/ERC20/ERC20.sol#120-122)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (../openzeppelin-contracts/contrac
ts/token/ERC20/ERC20.sol#131-134)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (../openzeppelin-cont
racts/contracts/token/ERC20/ERC20.sol#149-163)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (../openzeppelin-contrac
ts/contracts/token/ERC20/ERC20.sol#177-180)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (../openzeppelin-contrac
ts/contracts/token/ERC20/ERC20.sol#196-204)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pu
blic-function-that-could-be-declared-external
. analyzed (28 contracts with 75 detectors), 186 result(s) found
```

Conclusion:

Most of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

6 Conclusion

In this audit, we examined the design and implementation of PXP Gateway contract and discovered several issues of varying severity. PXP team addressed 13 issues raised in the initial report and implemented the necessary fixes, while classifying the rest as a risk with low-probability of occurrence. Shellboxes' auditors advised PXP Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

SHELLBOXES

For a Contract Audit, contact us at contact@shellboxes.com