SALUS SECURITY

JAN 2023

CODE SECURITY ASSESSMENT

UNIPASS

Overview

Project Summary

- Name: Unipass
- Version: v0.0.23
- Platform: EVM-compatible chains
- Language: Solidity
- Repository: https://github.com/UniPassID/Unipass-Wallet-Contract
- Audit Scope: See Appendix - 1

Project Dashboard

Application Summary

Name	Unipass
Version	v3
Type	Solidity
Dates	Jan 19 2023
Logs	Dec 20 2022; Jan 4 2023; Jan 19 2023

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	3
Total Low-Severity issues	5
Total informational issues	8
Total	17

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	syncAccount() may fail to set the new implementation address	High	Business Logic	Resolved
2	getKeysetHash() may return an invalid keysetHash	Medium	Business Logic	Acknowledged
3	Source can be set with no access control	Medium	Access Control	Acknowledged
4	DkimKey and OpenID are susceptible to error-prone admin management during contract upgrade	Medium	Business Logic	Resolved
5	Lack of zero address check in setAdmin()	Low	Data Validation	Acknowledged
6	deleteDKIMKey() triggers DeleteDKIMKey() event with wrong oldKey value	Low	Auditing and Logging	Resolved
7	Incorrect revert reason	Low	Auditing and Logging	Resolved
8	LibOptim.call leaves dirty bits in memory	Low	Low-Level Manipulation	Acknowledged
9	key slot string does not match the contract name	Low	Configuration	Acknowledged
10	error InvalidStatus() can be defined with one less parameter	Informational	Auditing and Logging	Resolved
11	State variable visibility for openIDPublicKey and openIDAudience is not set	Informational	Visibility	Resolved
12	The visibility specifier for _readAdmin() can be changed from public to internal	Informational	Visibility	Resolved
13	Unused import	Informational	Redundancy	Resolved

14	Unused function	Informational	Redundancy	Resolved
15	Unused variable	Informational	Redundancy	Acknowledged
16	Unused error	Informational	Redundancy	Resolved
17	Floating compiler version	Informational	Configuration	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. syncAccount() may fail to set the new implementation address

Severity: High	Category: Business Logic
Target: - contracts/modules/commons/ModuleAccount.sol	

Description

contracts/modules/commons/ModuleAccount.sol:L65-L98

```
function syncAccount(
    uint32 _metaNonce,
    bytes32 _keysetHash,
    uint32 _newTimeLockDuring,
    address _newImplementation,
    bytes calldata _signature
) external override onlySelf {
    _validateMetaNonceForSyncAccount(_metaNonce);
    _requireUnLocked();

    bytes32 digestHash = LibUnipassSig._subDigest(
        keccak256(abi.encodePacked(uint8(SYNC_ACCOUNT), _metaNonce,
_keysetHash, _newTimeLockDuring, _newImplementation)),
        ZERO_CHAINID
    );

    (bool success, IDkimKeys.EmailType emailType, uint32 ownerWeight, ,
) = validateSignature(digestHash, _signature);
    require(
        success && (emailType == IDkimKeys.EmailType.SyncAccount ||
emailType == IDkimKeys.EmailType.None),
        "syncAccount: INVALID_SIG"
    );

    require(ownerWeight >= LibRole.OWNER_THRESHOLD, "syncAccount:
INVALID_WEIGHT");

    if (getImplementation() != _newImplementation) {
        _setImplementation(_newImplementation);
    }
    _updateKeysetHash(_keysetHash);
    if (_getLockDuring() != _newTimeLockDuring) {
        _setLockDuring(_newTimeLockDuring);
    }
    _writeMetaNonce(_metaNonce);

    emit SyncAccount(_metaNonce, _keysetHash, _newTimeLockDuring,
_newImplementation);
}
```

The following lines in **syncAccount()** set the implementation address of the user wallet to a new address.

```
if (getImplementation() != _newImplementation) {
    _setImplementation(_newImplementation);
}
```

But if **ModuleMain** is the current implementation of the user wallet, the next line, **_updateKeysetHash(_keysetHash),** will call the **_updateKeysetHash()** function in **ModuleMain**.
contracts/modules/commons/ModuleAuthFixed.sol:L31-L36

```
function _updateKeysetHash(bytes32 _keysetHash) internal override {
    require(_keysetHash != bytes32(0), "updateKeysetHash
INVALID_KEYSET");
    _writeKeysetHash(_keysetHash);

    _setImplementation(MODULE_MAIN_UPGRADABLE);
}
```

Inside this **_updateKeysetHash()**, the implementation will be overridden to **MODULE_MAIN_UPGRADABLE**, which is the deployed address of **ModuleMainUpgradable** contract.

In summary, when the user wants to change the implementation of the user wallet from **ModuleMain** to a new address other than **ModuleMain** and **ModuleMainUpgradable** using **syncAccount**(), the implementation will unfortunately be set to **ModuleMainUpgradable**.

Proof of Concept

1. Deploy two ModuleMainUpgradable to address A and address B.
2. Whitelist both address A and address B as whitelisted implementation.
3. Deploy ModuleMain with address A as the _moduleMainUpgradable parameter for the constructor.
4. Create a user wallet.
5. Build a transaction to call syncAccount() to update the implementation to address B.
6. Assert that the implementation of the user wallet is set to address A instead of address B.

Recommendation

Consider putting the **_updateKeysetHash()** line above the **_setImplementation()** logic in **syncAccount()**.

```
_updateKeysetHash(_keysetHash);
if (getImplementation() != _newImplementation) {
    _setImplementation(_newImplementation);
}
```

Status

This issue has been resolved by the team in commit 9748416.

2. getKeysetHash() may return an invalid keysetHash

Severity: Medium	Category: Business Logic

Target:

- contracts/modules/commons/ModuleAuth.sol
- contracts/modules/commons/ModuleAuthFixed.sol
- contracts/modules/commons/ModuleAuthUpgradable.sol

Description

contracts/modules/commons/ModuleAuth.sol:L37-L39

```
function getKeysetHash() public view returns (bytes32 keysetHash) {
    keysetHash = ModuleStorage.readBytes32(KEYSET_HASH_KEY);
}
```

getKeysetHash() returns the value in the **KEYSET_HASH_KEY** slot. However, the **KEYSET_HASH_KEY** slot value is not set during **ModuleMain**’s deployment. As a result, the return value of **getKeysetHash**() defaults to zero bytes32.

contracts/modules/commons/ModuleAuthFixed.sol:L38-L43

```
function isValidKeysetHash(bytes32 _keysetHash) public view override
returns (bool) {
    return
        address(uint160(uint256(keccak256(abi.encodePacked(hex"ff",
FACTORY, _keysetHash, INIT_CODE_HASH))))) ==
        address(this);
}
```

Furthermore, for the user wallet with an initial non-zero keysetHash, **ModuleMain.isValidKeysetHash(getKeysetHash())** returns false.

contracts/modules/commons/ModuleAuthUpgradable.sol:L22-L25

```
function isValidKeysetHash(bytes32 _keysetHash) public view virtual
override returns (bool) {
    return _keysetHash != bytes32(0) && getKeysetHash() == _keysetHash;
}
```

Only when the implementation has been upgraded to **ModuleMainUpgradable** will **getKeysetHash()** return the correct keysetHash, i.e. **ModuleMainUpgradable.isValidKeysetHash(getKeysetHash())** returns true.

Recommendation

We recommend changing the visibility specifier of **getKeysetHash()** from **public** to **internal** since it is more of an internal function than an external one that returns the actual keysetHash.

Status

This issue has been acknowledged by the team. The team has clarified that **getKeysetHash**() is only intended for the SDK to use, and condition checks for the return value of **getKeysetHash**() are done in the SDK.

3. Source can be set with no access control

Severity: Medium	Category: Access Control

Target:
- contracts/modules/commons/ModuleSource.sol

Description

contracts/modules/commons/ModuleSource.sol:L18-L23

```
function setSource(bytes32 _source) external {
    require(_source != bytes32(0), "_setSource: ZERO_SOURCE");
    require(getSource() == bytes32(0), "_setSource: EXISTED_SOURCE");
    _writeSource(_source);
    emit SetSource(_source);
}
```

setSource() only checks against the zero source input and the existing source, meaning anyone can set the source when it hasn't been set yet.

Recommendation

Consider adding proper access control to **setSource()**.

Status

This issue has been acknowledged by the team. The team has clarified that the source is only used to tag information about the user's registration and has no real effect on the user's wallet; thus, no access control is needed for **setSource**().

4. DkimKey and OpenID are susceptible to error-prone admin management during contract upgrade

Severity: Medium	Category: Business Logic

Target:
- contracts/DkimKeys.sol
- contracts/OpenID.sol
- contracts/modules/commons/ModuleAdminAuth.sol

Description

Both **DkimKey** and **OpenID** are upgradable contracts, and they both inherit **ModuleAdminAuth** for admin management.

contracts/modules/commons/ModuleAdminAuth.sol:L17-L39

```
function _readAdmin() public view returns (address admin) {
    admin = address(bytes20(ModuleStorage.readBytes32(ADMIN_KEY)));
}

function getAdmin() public view returns (address admin) {
    admin = _readAdmin();
    if (admin == address(0)) admin = INIT_ADMIN;
}

constructor(address _admin) {
    require(_admin != address(0), "ModuleAdminAuth#constructor:
INVALID_ADMIN");
    INIT_ADMIN = _admin;
}

modifier onlyAdmin() {
    require(msg.sender == getAdmin(), "NOT_AUTHORIZED");
    _;
}

function setAdmin(address _newAdmin) external onlyAdmin {
    emit SetAdmin(getAdmin(), _newAdmin);
    _writeAdmin(_newAdmin);
}
```

However, the return value of **getAdmin**(), which is used for authorization in the **onlyAdmin**() modifier, depends on whether the **ADMIN_KEY** slot is set to a non-zero value.

- If it is not set, **getAdmin**() returns the **INIT_ADMIN**, which is an immutable variable that initialized to the **_admin** parameter during deployment
- Otherwise, it returns the value in the **ADMIN_KEY** slot

As a result, when upgrading **DkimKeys** or **OpenID**, the admin management could easily get it wrong.

Take **DkimKeys** for example. Assuming the current admin is address A, and we deploy a new implementation of **DkimKeys** with another address B as the **INIT_ADMIN**, then we upgrade the **DkimKeys** proxy to this new implementation. Guess which address will be the admin after the upgrade.

- If the **ADMIN_KEY** slot is set before the upgrade, the admin will remain to be A.
- Otherwise, address B will be the new admin.

Proof of Concept

test/TestDkimVerify.spec.ts:L157-L163

```
it("Upgrade Should Success", async () => {
  expect(await dkimKeys.getAdmin()).equals(signer.address);
  const ret = await (await
dkimKeys.upgradeTo(newDkimkeys.address)).wait();
  expect(ret.status).to.equals(1);
  expect(await dkimKeys.getAdmin()).to.equals(signer1.address);
  expect(await dkimKeys.getAdmin()).to.not.equals(signer.address);
});
```

For this test case, if we add a line, **await dkimKeys.setAdmin(await dkimKeys.getAdmin()),** before the upgrade

```
it("Upgrade Should Success", async () => {
  expect(await dkimKeys.getAdmin()).equals(signer.address);
  await dkimKeys.setAdmin(await dkimKeys.getAdmin());
  const ret = await (await
dkimKeys.upgradeTo(newDkimkeys.address)).wait();
  expect(ret.status).to.equals(1);
  expect(await dkimKeys.getAdmin()).to.equals(signer1.address);
  expect(await dkimKeys.getAdmin()).to.not.equals(signer.address);
});
```

The test case will fail because **setAdmin**() sets the **ADMIN_KEY** slot, and the admin after the upgrade will still be **signer.address** (the original admin) rather than **signer1.address** (the admin of the new implementation)**.**

Recommendation

We recommend initializing the value of the **ADMIN_KEY** slot during deployment of the **DkimKeys** and the **OpenID** so that the admin will not change when upgrading the implementation contract.

Status

This issue has been resolved by the team. The team has updated the deploy script in commit 02b563d0e to avoid the issue.

5. Lack of zero address check in setAdmin()

Severity: Low	Category: Data Validation
Target: - contracts/modules/commons/ModuleAdminAuth.sol	

Description

contracts/modules/commons/ModuleAdminAuth.sol:L36-39

```
function setAdmin(address _newAdmin) external onlyAdmin {
    emit SetAdmin(getAdmin(), _newAdmin);
    _writeAdmin(_newAdmin);
}
```

setAdmin() lacks an input check against the zero address.

When **setAdmin(address(0))** is called, the **ADMIN_KEY** slot will be set to zero. As a result, **getAdmin**() will return **INIT_ADMIN**. In addition, an incorrect **SetAdmin(oldAdmin, 0)** event will be triggered.

Recommendation

Consider adding an input check against the zero address.

```
function setAdmin(address _newAdmin) external onlyAdmin {
    require(_newAdmin != address(0), "ZERO_ADDRESS");
    emit SetAdmin(getAdmin(), _newAdmin);
    _writeAdmin(_newAdmin);
}
```

Status

This issue has been acknowledged by the team. The team has clarified that the zero address check is performed off-chain.

6. deleteDKIMKey() triggers DeleteDKIMKey() event with wrong oldKey value

Severity: Low	Category: Auditing and Logging

Target:
- contracts/DkimKeys.sol

Description

contracts/DkimKeys.sol:L106-109

```
function deleteDKIMKey(bytes calldata _emailServer) external onlyAdmin {
    delete dkimKeys[_emailServer];
    emit DeleteDKIMKey(_emailServer, dkimKeys[_emailServer]);
}
```

When emitting **DeleteDKIMKey(bytes emailServer, bytes oldKey)** event in **deleteDKIMKey**(), **dkimKeys[_emialSever]** has already been deleted (set to zero bytes), so zero bytes instead of the old key is emitted with **DeleteDKIMKey** event.

Recommendation

Consider adding a local variable to store the oldKey

```
function deleteDKIMKey(bytes calldata _emailServer) external onlyAdmin {
    bytes memory oldKey = dkimKeys[_emailServer];
    delete dkimKeys[_emailServer];
    emit DeleteDKIMKey(_emailServer, oldKey);
}
```

Status

This issue has been resolved by the team in commit d0bfb7ac4.

7. Incorrect revert reason

Severity: Low	Category: Auditing and Logging

Target:
- contracts/modules/commons/ModuleCall.sol
- contracts/modules/commons/ModuleAuth.sol

Description

contracts/modules/commons/ModuleCall.sol:L46-L68

```
function execute(
    Transaction[] calldata _txs,
    uint256 _nonce,
    bytes calldata _signature
) external payable {
    _validateNonce(_nonce);

    bytes32 txhash =
LibUnipassSig._subDigest(keccak256(abi.encode(_nonce, _txs)),
block.chainid);

    (
        bool succ,
        IDkimKeys.EmailType emailType,
        uint32 ownerWeight,
        uint32 assetsOpWeight,
        uint32 guardianWeight
    ) = validateSignature(txhash, _signature);
    require(
        succ && (emailType == IDkimKeys.EmailType.None || emailType ==
IDkimKeys.EmailType.CallOtherContract),
        "execute: INVALID_SIG_WEIGHT"
    );

    _execute(txhash, _txs, ownerWeight, assetsOpWeight, guardianWeight);
}
```

In **execute**(), the require statement checks the **succ** and **emailType** returned by **validateSignature**(). However, the revert reason, "**execute: INVALID_SIG_WEIGHT**," indicates that signature weight has been checked, which is not the case.

contracts/modules/commons/ModuleAuth.sol:L21-L24

```
constructor(IDkimKeys _dkimKeys, IOpenID _openID) {
    require(address(_dkimKeys) != address(0), "INVALID_DKIMKEYS");
    dkimKeys = _dkimKeys;
    require(address(_openID) != address(0), "INVALID_DKIMKEYS");
    openID = _openID;
}
```

In **ModuleAuth**’s constructor, the second require statement checks the **_openID** against the zero address, but the revert reason, “**INVALID_DKIMKEYS**”, is about dkimKeys.

Recommendation

Consider using revert reasons that match the checking logic.

Status

This issue has been resolved by the team in commit d322e4c.

8. LibOptim.call leaves dirty bits in memory

Severity: Low	Category: Low-Level Manipulation
Target: - contracts/utils/LibOptim.sol	

Description

contracts/utils/LibOptim.sol:L16-L28

```
function call(
    address _to,
    uint256 _val,
    uint256 _gas,
    bytes calldata _data
) internal returns (bool r) {
    assembly {
        let tmp := mload(0x40)
        calldatacopy(tmp, _data.offset, _data.length)

        r := call(_gas, _to, _val, tmp, _data.length, 0, 0)
    }
}
```

This internal function copies the calldata into memory and does not reset the free memory pointer to an empty space before exit. As a result, the next memory allocated will have dirty bits in it.

Recommendation

Consider resetting the free memory pointer to an empty place.

```
function call(
    address _to,
    uint256 _val,
    uint256 _gas,
    bytes calldata _data
) internal returns (bool r) {
    assembly {
        let tmp := mload(0x40)
        calldatacopy(tmp, _data.offset, _data.length)

        r := call(_gas, _to, _val, tmp, _data.length, 0, 0)
        mstore(0x40, add(tmp, _data.length))
    }
}
```

If it is intentional to save gas by overwriting used memory, consider documenting and commenting on the potential risk of this memory usage.

Status

This issue has been acknowledged by the team.

9. key slot string does not match the contract name

Severity: Low	Category: Configuration
Target: - contracts/modules/commons/ModuleAccount.sol - contracts/modules/commons/ModuleHooks.sol	

Description

In the Unipass project, the slot value for the key is computed by hashing a certain string. The string follows the pattern “unipass-wallet:<contract-name>:<key-name>”.

We identified two places that do not follow this convention.
The first one is **META_NONCE_KEY** in **ModuleAccount**.
contracts/modules/commons/ModuleAccount.sol:L18-L19

```
//                      META_NONCE_KEY =
keccak256("unipass-wallet:module-auth:meta-nonce")
bytes32 private constant META_NONCE_KEY =
bytes32(0x0ca6870aa26ec991ce7fe5a2fe6d18a240f46fa28d3c662b0a534d670d38ad
09);
```

The contract name is ModuleAccount, but the slot string is about module-auth.

Another one is **HOOKS_KEY** in **ModuleHooks**.
contracts/modules/commons/ModuleHooks.sol:L21-L22

```
//                      HOOKS_KEY =
keccak256("org.arcadeum.module.hooks.hooks");
bytes32 private constant HOOKS_KEY =
bytes32(0xbe27a319efc8734e89e26ba4bc95f5c788584163b959f03fa04e2d7ab4b9a1
20);
```

Recommendation

Fix the slot string and recompute the hash for the key.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

10. error InvalidStatus() can be defined with one less parameter

Severity: Informational	Category: Auditing and Logging

Target:
- contracts/modules/commons/ModuleWhiteList.sol

Description

ModuleWhiteList contract defined a custom error

```
error InvalidStatus(bool _status, bool _changeStatus);
```

It's used in **updateHookWhiteList()**
contracts/modules/commons/ModuleWhiteList.sol:27-35

```
function updateHookWhiteList(address _addr, bool _isWhite) external
onlyAdmin {
    bool isWhite = hooks[_addr];
    if (isWhite != _isWhite) {
        hooks[_addr] = _isWhite;
        emit UpdateHookWhiteList(_addr, _isWhite);
    } else {
        revert InvalidStatus(isWhite, _isWhite);
    }
}
```

and **updateImplementationWhiteList**()
contracts/modules/commons/ModuleWhiteList.sol:L45-54

```
function updateImplementationWhiteList(address _addr, bool _isWhite)
external onlyAdmin {
    bool isWhite = implementations[_addr];
    if (isWhite != _isWhite) {
        implementations[_addr] = _isWhite;
        emit UpdateImplementationWhiteList(_addr, _isWhite);
    } else {
        revert InvalidStatus(isWhite, _isWhite);
    }
}
```

In both cases, error **InvalidStatus**() will have the same bool value for **_status** and **_changeStatus**. So one of the parameters can be removed. **InvalidStatus**() could be defined as

```
error InvalidStatus(bool _status);
```

Status

This issue has been resolved by the team in commit 735641e.

11. State variable visibility for openIDPublicKey and openIDAudience is not set

Severity: Informational	Category: Visibility
Target: - contracts/OpenID.sol	

Description

Visibility is not set for the **openIDPublicKey** and **openIDAudience** state variable **contracts/OpenID.sol:L41,L46**

```
mapping(bytes32 => bytes) openIDPublicKey;

mapping(bytes32 => bool) openIDAudience;
```

It is best practice to set the visibility of state variables explicitly. The default visibility is **internal**. Other possible visibility settings are **public** and **private**.

Status

This issue has been resolved by the team in commit bc66753.

12. The visibility specifier for _readAdmin() can be changed from public to internal

Severity: Informational	Category: Visibility
Target: - contracts/modules/commons/ModuleAdminAuth.sol	

Description

In ModuleAdminAuth, there is already a public get function for admin contracts/modules/commons/ModuleAdminAuth.sol:L21-L24,L17-L19

```
function getAdmin() public view returns (address admin) {
    admin = _readAdmin();
    if (admin == address(0)) admin = INIT_ADMIN;
}

function _readAdmin() public view returns (address admin) {
    admin = address(bytes20(ModuleStorage.readBytes32(ADMIN_KEY)));
}
```

Therefore, the visibility specifier for _readAdmin() can be changed from **public** to **internal**.

Status

This issue has been resolved by the team in commit 264302e.

13. Unused import

Severity: Informational	Category: Redundancy

Target:
- contracts/modules/commons/ModuleTransaction.sol
- contracts/DkimKeys.sol
- contracts/DkimZK.sol
- contracts/OpenID.sol
- contracts/modules/commons/ModuleAuth.sol
- contracts/modules/commons/ModuleAuthBase.sol
- contracts/modules/commons/ModuleCall.sol
- contracts/modules/commons/ModuleHookEIP4337Wallet.sol
- contracts/modules/commons/ModuleHooks.sol
- contracts/modules/commons/ModuleSource.sol
- contracts/modules/commons/ModuleTimeLock.sol
- contracts/modules/commons/ModuleWhiteList.sol
- contracts/modules/utils/GasEstimator.sol
- contracts/modules/utils/LibDkimAuth.sol
- contracts/modules/utils/LibEmailHash.sol
- contracts/modules/utils/LibOpenIDAuth.sol
- contracts/modules/utils/LibUnipassSig.sol
- contracts/utils/LibBase64.sol
- contracts/utils/LibBytes.sol

Description

contracts/modules/commons/ModuleTransaction.sol:L4

```
import "@openzeppelin/contracts/interfaces/IERC20.sol";
```

IERC20.sol is imported in **ModuleTransaction** but never used, therefore, it can be removed.

```
import "hardhat/console.sol";
```

console.sol is imported in the following contracts
- contracts/DkimKeys.sol:L16
- contracts/DkimZK.sol:L8
- contracts/OpenID.sol:L13
- contracts/modules/commons/ModuleAuth.sol:L13
- contracts/modules/commons/ModuleAuthBase.sol:L6
- contracts/modules/commons/ModuleCall.sol:L20
- contracts/modules/commons/ModuleHookEIP4337Wallet.sol:L21
- contracts/modules/commons/ModuleHooks.sol:L17
- contracts/modules/commons/ModuleSource.sol:L6
- contracts/modules/commons/ModuleTimeLock.sol:L7
- contracts/modules/commons/ModuleWhiteList.sol:L6

- contracts/modules/utils/GasEstimator.sol:L4
- contracts/modules/utils/LibDkimAuth.sol:L6
- contracts/modules/utils/LibEmailHash.sol:L7
- contracts/modules/utils/LibOpenIDAuth.sol:L6
- contracts/modules/utils/LibUnipassSig.sol:L9
- contracts/utils/LibBase64.sol:L5
- contracts/utils/LibBytes.sol:L6

console.sol is used only for debugging and should be removed before deployment.

contracts/modules/commons/ModuleCall.sol:L18

```
import "../../interfaces/IEIP4337Wallet.sol";
```

The interface **IEIP4337Wallet** is imported in **ModuleCall** but not used; therefore, it can be removed.

contracts/modules/commons/ModuleRole.sol:L6

```
import "../../interfaces/IModuleCall.sol";
```

The interface **IModuleCall** is imported and not used in **ModuleRole**; therefore, it can be removed.

Status

This issue has been resolved by the team in commit 0cebc9a.

14. Unused function

Severity: Informational	Category: Redundancy
Target: - contracts/modules/commons/ModuleAuth.sol - contracts/modules/commons/ModuleTimeLock.sol	

Description

contracts/modules/commons/ModuleAuth.sol:L164-L177

```
function _parseRoleWeight(uint256 _index, bytes calldata _signature)
    private
    pure
    returns (
        uint32 ownerWeight,
        uint32 assetsOpWeight,
        uint32 guardianWeight,
        uint256 index
    )
{
    (ownerWeight, index) = _signature.cReadUint32(_index);
    (assetsOpWeight, index) = _signature.cReadUint32(index);
    (guardianWeight, index) = _signature.cReadUint32(index);
}
```

_parseRoleWeight() is an unused private function; therefore, it can be removed.

contracts/modules/commons/ModuleTimeLock.sol:L60-L62,L68-L70

```
function _unlockKeysetHash() internal {
    isLocked = false;
}

function _setUnLock() internal {
    isLocked = false;
}
```

In **ModuleTimeLock**, there exist two internal functions (**_unlockKeysetHash**() and **_setUnLock**()) performing the same task. What's more, **_setUnLock**() is never used throughout the Unipass contracts. Therefore, **_setUnLock**() can be removed.

Status

This issue has been resolved by the team in commit a98e2cf.

15. Unused variable

Severity: Informational	Category: Redundancy
Target: - contracts/modules/commons/ModuleHookEIP4337Wallet.sol	

Description

contracts/modules/commons/ModuleHookEIP4337Wallet.sol:L159-167

```
function _payPrefund(uint256 missingWalletFunds) internal virtual {
    if (missingWalletFunds != 0) {
        //pay required prefund. make sure NOT to use the "gas" opcode, which is banned during validateUserOp
        // (and used by default by the "call")
        (bool success, ) = payable(msg.sender).call{value: missingWalletFunds, gas: type(uint256).max}("");
        (success);
        //ignore failure (its EntryPoint's job to verify, not wallet.)
    }
}
```

In **_payPrefund**(), the line, `(success)`;, performs nothing; therefore, it can be removed. Also, the local variable **success** is not actually used and can be removed.

Status

This issue has been acknowledged by the team. The team has clarified that the line was intentionally left there to align with the comment below.

16. Unused error

Severity: Informational	Category: Redundancy

Target:
- contracts/modules/commons/ModuleCall.sol
- contracts/modules/commons/ModuleAccount.sol

Description

contracts/modules/commons/ModuleCall.sol:L29-L31

```
error UnknownCallDataSelector(bytes4 _selector);
error SelectorDoesNotExist(bytes4 _selector);
error ImmutableSelectorSigWeight(bytes4 _selector);
```

These errors are defined but not used; therefore, they can be removed.

contracts/modules/commons/ModuleAccount.sol:L37

```
error InvalidActionType(uint256 _actionType);
```

InvalidActionType() is defined but never used; therefore, it can be removed.

Status

This issue has been resolved by the team in commit 2ab5bc9.

17. Floating compiler version

Severity: Informational	Category: Configuration
Target: - all	

Description

```
pragma solidity ^0.8.0;
```

The Unipass contracts use a floating compiler version ^0.8.0.

However, it is always recommended that contracts be deployed with the same compiler version and flags that they have been tested with the most.

We recommend locking the compiler version to 0.8.15, which is the version specified in the hardhat.config.ts file.

```
pragma solidity 0.8.15;
```

Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Status

This issue has been acknowledged by the team in commit a35b348.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
contracts/DkimKeys.sol	fff7c89ce373c525facc46c621df96d669e50ee7
contracts/DkimZK.sol	06fd79bc07323b30f63d6fe6868d356dd5fc4052
contracts/interfaces/IDkimKeys.sol	095a847ddf19800eb2448cd241bc43c1ecce52e2
contracts/interfaces/IDkimZK.sol	4906a30e26e1a021ccc8622f2bf30ca0706a4724
contracts/interfaces/IEIP4337Wallet.sol	6d728c7b5f118c0c0ef6d1963d210f5f1b6cabe4
contracts/interfaces/IERC223Receiver.sol	bd5ba69ab421698e96f83abc39118b1d223d78a0
contracts/interfaces/IModuleAccount.sol	08c3915fbaee6d88a03d77170f6cba8cafb78ef0
contracts/interfaces/IModuleAuth.sol	f68ca9463d6acb9e547db7b73921f7e9a039530c
contracts/interfaces/IModuleCall.sol	adaaec372b79f43744804bdc0950f062e3910b9f
contracts/interfaces/IModuleHooks.sol	948af5b992beb1a7e6eaeee5a5619f319676b9d3
contracts/interfaces/IModuleWhiteList.sol	2af786aea6b5039ffd5aa86674b7ca2b75687f54
contracts/interfaces/IOpenID.sol	ef259b77b2056928cfb3c80dad3c093b22e0227f
contracts/interfaces/IPaymaster.sol	e140a2ae0232ace05e6913fd0492d9e33200a111
contracts/modules/commons/Implementation.sol	21161c73e985befb70827655608a8391dc39df82
contracts/modules/commons/ModuleAccount.sol	aba4c35c076b509f31ccb696ee47e0695138a0ec

contracts/modules/commons/ModuleAdminAuth.sol	2f23f687802999cd1382acb25626de2cff37fe98
contracts/modules/commons/ModuleAuthBase.sol	196b79a0a617c77e68d4fbef68116aa0f1f4a5e8
contracts/modules/commons/ModuleAuthFixed.sol	440b38d6759031646f656187df2aea150f0e57ef
contracts/modules/commons/ModuleAuth.sol	3b09bfe3e14ae52046441c0b1ad638e7b74e3973
contracts/modules/commons/ModuleAuthUpgradable.sol	0386d0e2ee6d49b6fd1836eea51bf8cadbb0bbc5
contracts/modules/commons/ModuleCall.sol	f2f1dd5d0fd59703e508ce2f7b23d4a59ded3691
contracts/modules/commons/ModuleERC165.sol	887791f16f95c6a244464cc3ea4f2cf6e26bf975
contracts/modules/commons/ModuleHookEIP4337Wallet.sol	e6b48ed4191d537ed561f96a46c1e35719657ebe
contracts/modules/commons/ModuleHooks.sol	3d61c04b8c768438efe16c44f785897628c19558
contracts/modules/commons/ModuleIgnoreAccount.sol	5cd61af2c2c3ad7a0ad3ff7af8664176ab3183cf
contracts/modules/commons/ModuleIgnoreAuthUpgradable.sol	96911a39f05eeb73045c7bcad90fdad4949b452e
contracts/modules/commons/ModuleRole.sol	899e9d685a832006d1e02eab78a1661753a13dae
contracts/modules/commons/ModuleSelfAuth.sol	63e114dfeb0aa678d5a5749606ad538389bbac5d
contracts/modules/commons/ModuleSource.sol	29ed74b6b25dc5a734c2dfa650434f0559f8f893
contracts/modules/commons/ModuleStorage.sol	99d38ab1ca62f8c1d1a2489b04c4a5b0e97253c9
contracts/modules/commons/ModuleTimeLock.sol	c157df93397dfdc603be79fdec8b57c5b2fc2380
contracts/modules/commons/ModuleTransaction.sol	18b2fe58cc1de9f34e1f2c957bd004b680dae92b
contracts/modules/commons/ModuleWhiteList.sol	864a09b105b7a34424d6d14f2544f7d5733b6251
contracts/modules/ModuleGuest.sol	c93f3e2284b0817bb71c449903b01092ea3d68a7

contracts/modules/ModuleMainGasEstimator.sol	2076d1ac6a0c86c8afb4f46542e16370e9449b94
contracts/modules/ModuleMain.sol	aa29d1bb23826c85ab8018e0dd63518885b08e17
contracts/modules/ModuleMainUpgradable.sol	e73fd31c3cdeef420fd05fdb1c6937e212bf141f
contracts/modules/utils/FeeEstimator.sol	f3df2be754ccb092427bb7788b0fdb348838a4b7
contracts/modules/utils/GasEstimator.sol	57f1fb769ce50c08372f6158b6180484d1bc4f63
contracts/modules/utils/LibDkimAuth.sol	34f5215a7ad2f7ce244c330f5a365ae24fee8104
contracts/modules/utils/LibEmailHash.sol	08307cbf912e8a2e185da3c14aef708ce75b72bb
contracts/modules/utils/LibOpenIDAuth.sol	054743cf77183dfb20e51cdb67d73ad54904630d
contracts/modules/utils/LibUnipassSig.sol	b7e4bd8fe13932b1a38369da2ed9c225bf62da8f
contracts/OpenID.sol	c8325566fccee172b0375fe0afdad50b9d04f8e3
contracts/tests/CallReceiverMock.sol	ad7fa2189d665a9c97fcc4ab37ade349cf10b40e
contracts/tests/EntryPoint.sol	c083b3705ab253de8b3bce2e5ecaf288dfc43818
contracts/tests/Greeter.sol	977db478c677b135113180759054e9621127fe2f
contracts/tests/ICreate2Deployer.sol	ed0e1c89120c6ed5a4f0f5d49bed3853a7e80b1d
contracts/tests/StakeManager.sol	51d23781e0fabe36f965d98693758b45183343a4
contracts/tests/TestDkimVerify.sol	de091bdcc328c295d5894ad6f1fa8eab1376a5fe
contracts/tests/TestERC1271Wallet.sol	750d9c3b2f7d84e1f5fff963bbb1dc286a697704
contracts/tests/TestModuleCall.sol	93e13994c8de9fa76f45ad1bc1c1049f71d8ddb3
contracts/tests/Tokens/TestERC1155.sol	f0f16a629fb581f61863dc14fce7f2f37e87e4a5

contracts/tests/Tokens/TestERC20.sol	9ee522b7159b0b48ed0984a4e80eff6a60e73070
contracts/tests/Tokens/TestERC721.sol	2ee07270de98fceeba1dffde2963a1420e1a90a8
contracts/UserOperation.sol	6888506f0d6ba7a3fac1502fdce290e1a6fb7834
contracts/utils/LibBase64.sol	6e0887f83085e58984fae9bdae858537352915d4
contracts/utils/LibBytes.sol	7a77ef60a2854aac742ca8fd396cb5653e844a19
contracts/utils/LibModexpPrecompile.sol	559f72c4d13f5487c04755409e8bbb55963b6df6
contracts/utils/LibOptim.sol	d09a1d8cda0d498ee189d58bf9c50d1ae2ce3fd1
contracts/utils/LibRole.sol	b1666c9a3560e3ee330bb3e92f37eeb7f1ea2a7e
contracts/utils/LibRsa.sol	a31382d7bf878be7b734a087d6a5591f7d306172
contracts/utils/LibSignatureValidator.sol	d3578b9a9f75b49acde4cd22d28f8e1b284a6790
contracts/Wallet.sol	0d2d7f8a6149a1ebd205aadb4ff2add8ade80a81