METATRUST

Security Assessment for

core-contracts

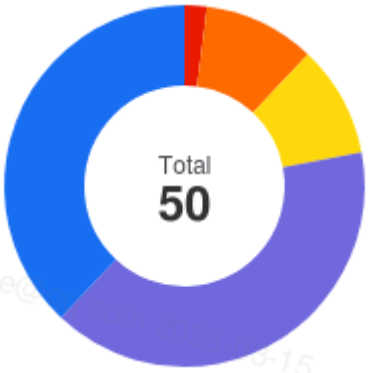
March 15, 2023

Executive Summary

Overview	
Project Name	core-contracts
Codebase Path	git://github.com/syncswap/core-contracts
Scan Engine	Security Analyzer
Scan Time	2023/03/15 23:50:16
Source Code	syncswap/core-contracts commit:f0d0b28

Total	
Critical Issues	1
High risk Issues	5
Medium risk Issues	5
Low risk Issues	20
Informational Issues	19

Critical Issues	The issue can cause large economic losses, large-scale data disorder, loss of control of authority management, failure of key functions, or indirectly affect the correct operation of other smart contracts interacting with it.
High Risk Issues	The issue puts a large number of users' sensitive information at risk or is reasonably likely to lead to catastrophic impacts on clients' reputations or serious financial implications for clients and users.
Medium Risk Issues	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 moderate financial impact.
Low Risk Issues	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 Issue	The issue does not pose an immediate risk but is relevant to security best practices or Defence in Depth.



Critical Issues	2%	1
High risk Issues	10%	5
Medium risk Issues	10%	5
Low risk Issues	40%	20
Informational Issues	38%	19

Summary of Findings

MetaScan security assessment was performed on **March 15, 2023 23:50:16** on project **core-contracts** with the repository **syncswap/core-contracts** on branch **default branch**. The assessment was carried out by scanning the project's codebase using the scan engine **Security Analyzer**. There are in total **50** vulnerabilities / security risks discovered during the scanning session, among which **1** critical vulnerabilities, **5** high risk vulnerabilities, **5** medium risk vulnerabilities, **20** low risk vulnerabilities, **19** informational issues.

ID	Description	Severity
MSA-001	Reentrancy Vulnerability without ETH Transfer in contracts/pool/stable/SyncSwapStablePool.sol	Critical
MSA-002	Call to Arbitrary Addresses with Unchecked Calldata in contracts/SyncSwapRouter.sol	High risk
MSA-003	Call to Arbitrary Addresses with Unchecked Calldata in contracts/master/SyncSwapPoolMaster.sol	High risk
MSA-004	Call to Arbitrary Addresses with Unchecked Calldata in contracts/vault/VaultFlashLoans.sol	High risk
MSA-005	Payable Functions using `delegatecall` inside a Loop in contracts/abstract/Multicall.sol	High risk
MSA-006	Lack of Proper Signature Verification in contracts/libraries/SignatureChecker.sol	High risk
MSA-007	Missing Protection against Signature Replay Attacks in contracts/libraries/ECDSA.sol	Medium risk
MSA-008	Non-standard Encode Method `abi.encodePacked()` in contracts/pool/stable/SyncSwapStablePool.sol	Medium risk
MSA-009	Non-standard Encode Method `abi.encodePacked()` in contracts/pool/classic/SyncSwapClassicPool.sol	Medium risk
MSA-010	Medium Possibility of Price manipulation in contracts/pool/classic/SyncSwapClassicPool.sol	Medium risk
MSA-011	Medium Possibility of Price manipulation in contracts/pool/stable/SyncSwapStablePool.sol	Medium risk
MSA-012	DoS With Failed Call	Low risk
MSA-013	DoS With Failed Call	Low risk
MSA-014	DoS With Failed Call	Low risk
MSA-015	DoS With Failed Call in contracts/libraries/Pausable.sol and other 1 file	Low risk
MSA-016	DoS With Failed Call	Low risk
MSA-017	DoS With Failed Call in contracts/abstract/Multicall.sol	Low risk
MSA-018	Unused Return Value in contracts/pool/classic/SyncSwapClassicPoolFactory.sol	Low risk
MSA-019	Unused Return Value in contracts/SyncSwapRouter.sol	Low risk

ID	Description	Severity
MSA-020	Default Function Visibility in contracts/pool/BasePoolFactory.sol	Low risk
MSA-021	Default Function Visibility in contracts/libraries/Ownable.sol	Low risk
MSA-022	Missing Input Validation in contracts/libraries/ERC20Permit2.sol	Low risk
MSA-023	Missing Input Validation in contracts/vault/SyncSwapVault.sol	Low risk
MSA-024	Missing Input Validation in contracts/master/SyncSwapPoolMaster.sol	Low risk
MSA-025	Missing Input Validation in contracts/master/SyncSwapFeeRecipient.sol	Low risk
MSA-026	Shadowing using Local Variables in contracts/vault/SyncSwapVault.sol and other 1 file	Low risk
MSA-027	Difficult-to-Understand Magic Number in contracts/libraries/ERC20Permit2.sol and other 1 file	Low risk
MSA-028	Difficult-to-Understand Magic Number in contracts/master/SyncSwapFeeManager.sol	Low risk
MSA-029	Difficult-to-Understand Magic Number in contracts/SyncSwapRouter.sol	Low risk
MSA-030	Difficult-to-Understand Magic Number in contracts/libraries/Math.sol	Low risk
MSA-031	Reentrancy Vulnerability in contracts/vault/SyncSwapVault.sol	Low risk
MSA-032	Uninitialized Local Variables in contracts/pool/classic/SyncSwapClassicPool.sol	Informational
MSA-033	Uninitialized Local Variables in contracts/vault/VaultFlashLoans.sol	Informational
MSA-034	Uninitialized Local Variables in contracts/libraries/StableMath.sol	Informational
MSA-035	Uninitialized Local Variables in contracts/pool/stable/SyncSwapStablePool.sol	Informational
MSA-036	Uninitialized Local Variables in contracts/SyncSwapRouter.sol	Informational
MSA-037	Uninitialized Local Variables in contracts/master/SyncSwapFeeRecipient.sol	Informational
MSA-038	Tips for Gas Optimisation in contracts/SyncSwapRouter.sol	Informational
MSA-039	Tips for Gas Optimisation in contracts/pool/stable/SyncSwapStablePool.sol	Informational
MSA-040	Tips for Gas Optimisation in contracts/master/SyncSwapFeeManager.sol	Informational
MSA-041	Tips for Gas Optimisation in contracts/pool/classic/SyncSwapClassicPool.sol	Informational
MSA-042	Tips for Gas Optimisation in contracts/master/ForwarderRegistry.sol	Informational
MSA-043	Tips for Gas Optimisation in contracts/master/SyncSwapFeeRecipient.sol	Informational
MSA-044	Tips for Gas Optimisation in contracts/vault/SyncSwapVault.sol	Informational
MSA-045	Error-prone Assembly Usage in contracts/libraries/SignatureChecker.sol	Informational
MSA-046	Unused Internal Functions in contracts/libraries/ReentrancyGuard.sol	Informational
MSA-047	Missing Mutability Specifier in contracts/pool/BasePoolFactory.sol	Informational

ID	Description	Severity
MSA-048	Inappropriate Solidity Naming Conventions in contracts/pool/classic/SyncSwapClassicPool.sol	Informational
MSA-049	Variables with Similar Names in contracts/pool/stable/SyncSwapStablePool.sol and other 1 file	Informational
MSA-050	DoS with Block Gas Limit in contracts/SyncSwapRouter.sol	Informational

Findings

Critical (1)

1. Reentrancy Vulnerability without ETH Transfer in contracts/pool/stable/SyncSwapStablePool.sol

Critical Security Analyzer

A state variable is changed after a contract calls another contract function. The target contract can callback and reenter before the state variable is updated. This may lead to an unexpected result.

For example: `solidity function withdrawBalance(){ // send userBalance[msg.sender] Ether to msg.sender // if mgs.sender is a contract, it will call its fallback function if( ! (msg.sender.call.value(userBalance[msg.sender])() ) ){ throw; } userBalance[msg.sender] = 0; }` `msg.sender` can reenter the withdrawBalance function and withdraw all ether in the contract.

File(s) Affected

contracts/pool/classic/SyncSwapClassicPool.sol #165-220 #225-304 #308-372
contracts/pool/stable/SyncSwapStablePool.sol #180-235 #240-319 #323-387

Examples

```
203             params.sender = _getVerifiedSender(_sender);
204             params.callbackData = _callbackData;
205
206             ICallback(_callback).syncSwapBaseBurnCallback(params);
207         }
208
209         // Updates reserves and last invariant with up-to-date balances (after transfers).
```

Recommendation

Apply the `check-effects-interactions pattern`.

High risk (5)

1. Call to Arbitrary Addresses with Unchecked Calldata in contracts/SyncSwapRouter.sol

High risk Security Analyzer

Call to arbitrary addresses with malicious calldata may incur unexpected behavior.

File(s) Affected

contracts/SyncSwapRouter.sol #386-388

Examples

```
384
385     /// @notice Wrapper function to allow pool deployment to be batched.
386     function createPool(address _factory, bytes calldata data) external payable returns (address) {
387         return IPoolFactory(_factory).createPool(data);
388     }
389
390     function stake(address stakingPool, address token, uint amount, address onBehalf) external {
```

Recommendation

Check the target address and calldata when use call

2. Call to Arbitrary Addresses with Unchecked Calldata in contracts/master/SyncSwapPoolMaster.sol

High risk Security Analyzer

Call to arbitrary addresses with malicious calldata may incur unexpected behavior.

File(s) Affected

contracts/master/SyncSwapPoolMaster.sol #108-112

Examples

```
108     function createPool(address factory, bytes calldata data) external override returns (address pool)
109         // The factory have to call `registerPool` to register the pool.
110         // The pool whitelist is checked in `registerPool`.
111         pool = IPoolFactory(factory).createPool(data);
112     }
113
114     /// @dev Register a pool to the mapping by its config. Can only be called by factories.
```

Recommendation

Check the target address and calldata when use call

3. Call to Arbitrary Addresses with Unchecked Calldata in contracts/vault/VaultFlashLoans.sol

High risk Security Analyzer

Call to arbitrary addresses with malicious calldata may incur unexpected behavior.

File(s) Affected

contracts/vault/VaultFlashLoans.sol #86-149

Examples

```
120             }
121         }
122
123         recipient.receiveFlashLoan(tokens, amounts, feeAmounts, userData);
124
125         uint preLoanBalance;
126         uint postLoanBalance;
```

Recommendation

Check the target address and calldata when use call

4. Payable Functions using `delegatecall` inside a Loop in contracts/abstract/Multicall.sol

High risk Security Analyzer

Detect `delegatecall` inside a loop in a payable function. When using `delegatecall` in a payable function, the msg.value will be passed to the target contract. The amount or balance will be accredited multiple times if we use this inside a loop.

For example:

```
contract DelegatecallInLoop{

    mapping (address => uint256) balances;

    function bad(address[] memory receivers) public payable {
        for (uint256 i = 0; i < receivers.length; i++) {
            address(this).delegatecall(abi.encodeWithSignature("addBalance(address)", receivers[i]));
        }
    }

    function addBalance(address a) public payable {
        balances[a] += msg.value;
    }

}
```

When calling bad the same `msg.value` amount will be accredited multiple times.

File(s) Affected

contracts/abstract/Multicall.sol #9-31

Examples

```
10          results = new bytes[](data.length);
11
12          for (uint i; i < data.length;) {
13              (bool success, bytes memory result) = address(this).delegatecall(data[i]);
14
15              if (!success) {
16                  // Next 5 lines from https://ethereum.stackexchange.com/a/83577
```

Recommendation

Carefully check that the function called by `delegatecall` is not payable/doesn't use `msg.value`

5. Lack of Proper Signature Verification in contracts/libraries/SignatureChecker.sol

High risk Security Analyzer

It is a common pattern for smart contract systems to allow users to sign message off-chain instead of directly requesting users to do an on-chain transaction because of the flexibility and increased transferability that this provides. Smart contract systems that process signed messages must implement their own logic to recover the authenticity of the sign messages. Some signature verification implementations attempt to solve this problem by assuming the validity of a signed message based on other methods that do not have this limitation. An example of such a method is to rely on msg.sender and assume that if a signed message originated from the sender address, it has also been created by the sender address. This can lead to vulnerabilities, especially in scenarios where proxies can be used to relay transactions.

File(s) Affected

contracts/libraries/SignatureChecker.sol #25-108

Examples

```
25      function isValidSignatureNow(address signer, bytes32 hash, bytes memory signature)
26          internal
27          view
    ...
106             }
107         }
108     }
```

Recommendation

It is not recommended to use alternate verification schemes that do not require proper signature verification through `ecrecover()`.

Medium risk (5)

1. Missing Protection against Signature Replay Attacks in contracts/libraries/ECDSA.sol

Medium risk | Security Analyzer

Sometimes it is necessary to perform signature verification in smart contracts to achieve better usability or to save gas costs. A secure implementation must protect against Signature Replay Attacks by, for example, keeping track of all processed message hashes and only allowing new message hashes. A malicious user could attack a contract without such control and get a message hash sent by another user processed multiple times.

File(s) Affected

contracts/libraries/ECDSA.sol #14-71

Examples

```
31       * - with https://web3js.readthedocs.io/en/v1.3.4/web3-eth-accounts.html#sign[Web3.js]
32       * - with https://docs.ethers.io/v5/api/signer/#Signer-signMessage[ethers]
33       */
34      function recover(bytes32 hash, bytes memory signature) internal pure returns (address) {
35          // Check the signature length
36          if (signature.length != 65) {
37              return address(0);
38          }
39
40          // Divide the signature in r, s and v variables
41          bytes32 r;
42          bytes32 s;
43          uint8 v;
44
45          // ecrecover takes the signature parameters, and the only way to get them
46          // currently is to use assembly.
47          /// @solidity memory-safe-assembly
48          // solhint-disable-next-line no-inline-assembly
49          assembly {
50              r := mload(add(signature, 0x20))
51              s := mload(add(signature, 0x40))
52              v := byte(0, mload(add(signature, 0x60)))
53          }
54
55          // EIP-2 still allows signature malleability for ecrecover(). Remove this possibility and make
56          // unique. Appendix F in the Ethereum Yellow paper (https://ethereum.github.io/yellowpaper/pape
57          // the valid range for s in (301): 0 < s < secp256k1n ÷ 2 + 1, and for v in (302): v ∈ {27, 28,
58          // signatures from current libraries generate a unique signature with an s-value in the lower h
59          //
60          // If your library generates malleable signatures, such as s-values in the upper range, calcula
61          // with 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141 - s1 and flip v fro
62          // vice versa. If your library also generates signatures with 0/1 for v instead 27/28, add 27 t
63          // these malleable signatures as well.
64          if (uint256(s) > 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0) {
65              return address(0);
66          }
67
68          return ecrecover(hash, v, r, s);
69      }
70  }
```

Recommendation

Store every message hash that the smart contract has processed. When new messages are received, check against the already existing ones and only proceed with business logic if it's a new message hash.\n Include the address of the contract that processes the message. This ensures that the message can only be used in a single contract.\n- Under no circumstances generate the message hash, including the signature. The `ecrecover` function is susceptible to signature malleability.

2. Non-standard Encode Method `abi.encodePacked()` in contracts/pool/stable/SyncSwapStablePool.sol

Medium risk Security Analyzer

The non-standard encode method `abi.encodePacked()` does not meet the ABI requirement. It encodes the parameters without padding for parameters less than 32 bytes and length information of dynamic arrays. As a result, we should not use it to encode parameters for any call preparation.\nAdditionally, it may be vulnerable to increasing the possibility of hash collision.

File(s) Affected

contracts/pool/stable/SyncSwapStablePool.sol #74-77

Examples

```
74              _initialize(
75                  string(abi.encodePacked("SyncSwap ", _symbol0, "/", _symbol1, " Stable LP")),
76                  string(abi.encodePacked(_symbol0, "/", _symbol1, " sSLP"))
77              );
```

Recommendation

Avoid using the non-standard ABI encode method.

3. Non-standard Encode Method `abi.encodePacked()` in contracts/pool/classic/SyncSwapClassicPool.sol

Medium risk Security Analyzer

The non-standard encode method `abi.encodePacked()` does not meet the ABI requirement. It encodes the parameters without padding for parameters less than 32 bytes and length information of dynamic arrays. As a result, we should not use it to encode parameters for any call preparation.\nAdditionally, it may be vulnerable to increasing the possibility of hash collision.

File(s) Affected

contracts/pool/classic/SyncSwapClassicPool.sol #59-62

Examples

```
59              _initialize(
60                  string(abi.encodePacked("SyncSwap ", _symbol0, "/", _symbol1, " Classic LP")),
61                  string(abi.encodePacked(_symbol0, "/", _symbol1, " cSLP"))
62              );
```

Recommendation

Avoid using the non-standard ABI encode method.

4. Medium Possibility of Price manipulation in contracts/pool/classic/SyncSwapClassicPool.sol

Medium risk Security Analyzer

Please check all child function call based on this expression or expression itself, the potential price manipulation risk may in it. for example, in some functions, certain variables used in `transfer` or `mint` or `return` procedures depend on another dangerous variable that derives its data from `balanceof`, `getReserve`, `totalSupply()` or `address(someAddress).balance` and is vulnerable to manipulation by flash loan.

File(s) Affected

contracts/pool/classic/SyncSwapClassicPool.sol #102-102 #174-174 #234-234 #318-318 #401-401 #402-402

Examples

```
102         (params.balance0, params.balance1) = _balances();
```

Recommendation

It is recommended to use the chainlink oracle to obtain data, or to avoid relying on easily manipulated variables, or to use the TWAP mechanism.

5. Medium Possibility of Price manipulation in contracts/pool/stable/SyncSwapStablePool.sol

Medium risk | Security Analyzer

Please check all child function call based on this expression or expression itself, the potential price manipulation risk may in it. for example, in some functions, certain variables used in `transfer` or `mint` or `return` procedures depend on another dangerous variable that derives its data from `balanceof`, `getReserve`, `totalSupply()` or `address(someAddress).balance` and is vulnerable to manipulation by flash loan.

File(s) Affected

contracts/pool/stable/SyncSwapStablePool.sol #117-117 #189-189 #249-249 #333-333 #416-416 #417-417

Examples

```
117         (params.balance0, params.balance1) = _balances();
```

Recommendation

It is recommended to use the chainlink oracle to obtain data, or to avoid relying on easily manipulated variables, or to use the TWAP mechanism.

Low risk (20)

1. DoS With Failed Call

Low risk | Security Analyzer

External calls can fail accidentally or deliberately, which can cause a DoS condition in the contract. To minimize the damage caused by such failures, it is better to isolate each external call into its transaction that the recipient can initiate. Namely, the state change of the recipient should be initiated by himself. \nThis is especially relevant for payments, where it is better to let users withdraw funds rather than push funds to them automatically (this also reduces the chance of problems with the gas limit).

File(s) Affected

contracts/master/SyncSwapFeeRecipient.sol #97-97 #100-100 #102-102

Examples

```
97                 TransferHelper.safeTransferETH(to, amount);
```

Recommendation

Avoid combining multiple calls in a single transaction, especially when calls are executed as part of a loop Always assume that external calls can fail Implement the contract logic to handle failed calls

2. DoS With Failed Call

Low risk | Security Analyzer

External calls can fail accidentally or deliberately, which can cause a DoS condition in the contract. To minimize the damage caused by such failures, it is better to isolate each external call into its transaction that the recipient can initiate. Namely, the state change of the recipient should be initiated by himself. \nThis is especially relevant for payments, where it is better to let users withdraw funds rather than push funds to them automatically (this also reduces the chance of problems with the gas limit).

File(s) Affected

contracts/libraries/StableMath.sol #31-31 #33-33 #56-56 #59-64 #66-66

Examples

```
28          for (uint i; i < 256; ) {
29              yPrev = y;
30              //y = (y * y + c) / (y * 2 + b - d);
31              y = Math.div(Math.mul(y, y) + c, Math.mulUnsafeFirst(2, y) + b - d);
32
33              if (Math.within1(y, yPrev)) {
34                  break;
```

Recommendation

Avoid combining multiple calls in a single transaction, especially when calls are executed as part of a loop Always assume that external calls can fail Implement the contract logic to handle failed calls

3. DoS With Failed Call

Low risk Security Analyzer

External calls can fail accidentally or deliberately, which can cause a DoS condition in the contract. To minimize the damage caused by such failures, it is better to isolate each external call into its transaction that the recipient can initiate. Namely, the state change of the recipient should be initiated by himself. \nThis is especially relevant for payments, where it is better to let users withdraw funds rather than push funds to them automatically (this also reduces the chance of problems with the gas limit).

File(s) Affected

contracts/libraries/Pausable.sol #49-49
contracts/vault/VaultFlashLoans.sol #70-70 #112-112 #116-116 #123-123

Examples

```
49          return _paused;
```

Recommendation

Avoid combining multiple calls in a single transaction, especially when calls are executed as part of a loop Always assume that external calls can fail Implement the contract logic to handle failed calls

4. DoS With Failed Call in contracts/libraries/Pausable.sol and other 1 file

Low risk Security Analyzer

External calls can fail accidentally or deliberately, which can cause a DoS condition in the contract. To minimize the damage caused by such failures, it is better to isolate each external call into its transaction that the recipient can initiate. Namely, the state change of the recipient should be initiated by himself. \nThis is especially relevant for payments, where it is better to let users withdraw funds rather than push funds to them automatically (this also reduces the chance of problems with the gas limit).

File(s) Affected

contracts/libraries/Pausable.sol #49-49
contracts/vault/VaultFlashLoans.sol #70-70 #112-112 #116-116 #123-123

Examples

```
46       * @dev Returns true if the contract is paused, and false otherwise.
47       */
48      function paused() public view virtual returns (bool) {
49          return _paused;
50      }
51
52      /**
```

Recommendation

Avoid combining multiple calls in a single transaction, especially when calls are executed as part of a loop Always assume that external calls can fail Implement the contract logic to handle failed calls

5. DoS With Failed Call

Low risk Security Analyzer

External calls can fail accidentally or deliberately, which can cause a DoS condition in the contract. To minimize the damage caused by such failures, it is better to isolate each external call into its transaction that the recipient can initiate. Namely, the state change of the recipient should be initiated by himself. \nThis is especially relevant for payments, where it is better to let users withdraw funds rather than push funds to them automatically (this also reduces the chance of problems with the gas limit).

File(s) Affected

contracts/SyncSwapRouter.sol #91-91 #133-141 #320-322

Examples

```
88              }
89          }
90
91          liquidity = IPool(pool).mint(data, msg.sender, callback, callbackData);
92
93          if (liquidity < minLiquidity) {
94              revert NotEnoughLiquidityMinted();
```

Recommendation

Avoid combining multiple calls in a single transaction, especially when calls are executed as part of a loop Always assume that external calls can fail Implement the contract logic to handle failed calls

6. DoS With Failed Call in contracts/abstract/Multicall.sol

Low risk Security Analyzer

External calls can fail accidentally or deliberately, which can cause a DoS condition in the contract. To minimize the damage caused by such failures, it is better to isolate each external call into its transaction that the recipient can initiate. Namely, the state change of the recipient should be initiated by himself. \nThis is especially relevant for payments, where it is better to let users withdraw funds rather than push funds to them automatically (this also reduces the chance of problems with the gas limit).

File(s) Affected

contracts/abstract/Multicall.sol #28-28

Examples

```
25
26              // cannot realistically overflow on human timescales
27              unchecked {
28                  ++i;
29              }
30          }
31      }
```

Recommendation

Avoid combining multiple calls in a single transaction, especially when calls are executed as part of a loop Always assume that external calls can fail Implement the contract logic to handle failed calls

7. Unused Return Value in contracts/pool/classic/SyncSwapClassicPoolFactory.sol

Low risk Security Analyzer

Either the return value of an external call is not stored in a local or state variable, or the return value is declared but never used in the function body.

File(s) Affected

contracts/pool/classic/SyncSwapClassicPoolFactory.sol #16-30

Examples

```
15
16      function _createPool(address token0, address token1) internal override returns (address pool) {
17          // Perform sanity checks.
18          IERC20(token0).balanceOf(address(this));
19          IERC20(token1).balanceOf(address(this));
20
21          bytes memory deployData = abi.encode(token0, token1);
```

Recommendation

Ensure the return value of external function calls is used. Remove or comment out the unused return function parameters.

8. Unused Return Value in contracts/SyncSwapRouter.sol

Low risk Security Analyzer

Either the return value of an external call is not stored in a local or state variable, or the return value is declared but never used in the function body.

File(s) Affected

contracts/SyncSwapRouter.sol #55-66 #295-349

Examples

```
55      function _transferFromSender(address token, address to, uint amount) private {
56          if (token == NATIVE_ETH) {
57              // Deposit ETH to the vault.
58              IVault(vault).deposit{value: amount}(token, to);
59          } else {
60              // Transfer tokens to the vault.
61              TransferHelper.safeTransferFrom(token, msg.sender, vault, amount);
```

Recommendation

Ensure the return value of external function calls is used. Remove or comment out the unused return function parameters.

9. Default Function Visibility in contracts/pool/BasePoolFactory.sol

Low risk Security Analyzer

Functions that do not have a function visibility type specified are public by default, Which can lead to a vulnerability if a developer forgets to set the visibility. A malicious user can make unauthorized or unintended state changes.

File(s) Affected

contracts/pool/BasePoolFactory.sol #64-65

Examples

```
64      function _createPool(address tokenA, address tokenB) internal virtual returns (address) {
65      }
```

Recommendation

Functions can be specified as being external, public, internal or private. It is recommended to make a conscious decision on which visibility type is appropriate for a function, which can dramatically reduce the attack surface of a contract system.

10. Default Function Visibility in contracts/libraries/Ownable.sol

Low risk Security Analyzer

Functions that do not have a function visibility type specified are public by default, Which can lead to a vulnerability if a developer forgets to set the visibility. A malicious user can make unauthorized or unintended state changes.

File(s) Affected

contracts/libraries/Ownable.sol #67-70

Examples

```
67      function transferOwnership(address newOwner) public virtual onlyOwner {
68          require(newOwner != address(0), "Ownable: new owner is the zero address");
69          _transferOwnership(newOwner);
70      }
```

Recommendation

Functions can be specified as being external, public, internal or private. It is recommended to make a conscious decision on which visibility type is appropriate for a function, which can dramatically reduce the attack surface of a contract system.

11. Missing Input Validation in contracts/libraries/ERC20Permit2.sol

Low risk Security Analyzer

In solidity, there is no "null" equivalent. So we should know every default value for different value types. Meanwhile, we should validate the value passed to state variable. So, for example, when we assign an address type, we should check the value to ensure it's not a zero address.

File(s) Affected

contracts/libraries/ERC20Permit2.sol #70-73 #92-106

Examples

```
68      }
69
70      function _approve(address _owner, address _spender, uint _amount) private {
71          allowance[_owner][_spender] = _amount;
72          emit Approval(_owner, _spender, _amount);
73      }
74
```

Recommendation

Validate input value.

12. Missing Input Validation in contracts/vault/SyncSwapVault.sol

Low risk Security Analyzer

In solidity, there is no "null" equivalent. So we should know every default value for different value types. Meanwhile, we should validate the value passed to state variable. So, for example, when we assign an address type, we should check the value to ensure it's not a zero address.

File(s) Affected

contracts/vault/SyncSwapVault.sol #22-24 #44-74 #91-125

Examples

```
20      mapping(address => uint) public override reserves; // token -> reserve
21
22      constructor(address _wETH) VaultFlashLoans(msg.sender) {
23          wETH = _wETH;
24      }
25
26      receive() external payable {
```

Recommendation

Validate input value.

13. Missing Input Validation in contracts/master/SyncSwapPoolMaster.sol

Low risk Security Analyzer

In solidity, there is no "null" equivalent. So we should know every default value for different value types. Meanwhile, we should validate the value passed to state variable. So, for example, when we assign an address type, we should check the value to ensure it's not a zero address.

File(s) Affected

contracts/master/SyncSwapPoolMaster.sol #100-103 #115-141

Examples

```
98      // Factories
99
100     function setFactoryWhitelisted(address factory, bool whitelisted) external override onlyOwner {
101         isFactoryWhitelisted[factory] = whitelisted;
102         emit SetFactoryWhitelisted(factory, whitelisted);
103     }
104
```

Recommendation

Validate input value.

14. Missing Input Validation in contracts/master/SyncSwapFeeRecipient.sol

Low risk Security Analyzer

In solidity, there is no "null" equivalent. So we should know every default value for different value types. Meanwhile, we should validate the value passed to state variable. So, for example, when we assign an address type, we should check the value to ensure it's not a zero address.

File(s) Affected

contracts/master/SyncSwapFeeRecipient.sol #38-40 #55-79 #120-139

Examples

```
36      event SetFeeRegistry(address indexed feeRegistry);
37
38      constructor(address _feeRegistry) {
39          feeRegistry = _feeRegistry;
40      }
41
42      function feeTokensLength(uint epoch) external view returns (uint) {
```

Recommendation

Validate input value.

15. Shadowing using Local Variables in contracts/vault/SyncSwapVault.sol and other 1 file

Low risk Security Analyzer

Detection of shadowing using local variables.

For example: ```solidity pragma solidity ^0.4.24;

contract Bug { uint owner;

```
function sensitive_function(address owner) public {
    // ...require(owner == msg.sender);
}

function alternate_sensitive_function() public {
    address owner = msg.sender;
    // ...require(owner == msg.sender);
}
```

} ``` `sensitive_function.owner` **shadows** `Bug.owner`. **As a result, the use of owner in** `sensitive_function` might be incorrect.

File(s) Affected

contracts/vault/SyncSwapVault.sol #33-33
contracts/libraries/Ownable.sol #41-43

Examples

```
33      function balanceOf(address token, address owner) external view override returns (uint balance) {
```

Recommendation

Rename the local variables that shadow another component.

16. Difficult-to-Understand Magic Number in contracts/libraries/ERC20Permit2.sol and other 1 file

Low risk Security Analyzer

Magic numbers are difficult to read and review and may lead to false positives or vulnerabilities.

File(s) Affected

contracts/libraries/ERC20Permit2.sol #94-94
contracts/pool/classic/SyncSwapClassicPool.sol #348-348 #351-351 #519-519 #522-522

Examples

```
94          if (_allowed != type(uint).max) {
```

Recommendation

Magic numbers should not be used.

17. Difficult-to-Understand Magic Number in contracts/master/SyncSwapFeeManager.sol

Low risk Security Analyzer

Magic numbers are difficult to read and review and may lead to false positives or vulnerabilities.

File(s) Affected

contracts/master/SyncSwapFeeManager.sol #16-16 #46-46 #50-50

Examples

```
16      uint24 private constant ZERO_CUSTOM_FEE = type(uint24).max;
```

Recommendation

Magic numbers should not be used.

18. Difficult-to-Understand Magic Number in contracts/SyncSwapRouter.sol

Low risk Security Analyzer

Magic numbers are difficult to read and review and may lead to false positives or vulnerabilities.

File(s) Affected

contracts/SyncSwapRouter.sol #394-394

Examples

```
394             TransferHelper.safeApprove(token, stakingPool, type(uint).max);
```

Recommendation

Magic numbers should not be used.

19. Difficult-to-Understand Magic Number in contracts/libraries/Math.sol

Low risk Security Analyzer

Magic numbers are difficult to read and review and may lead to false positives or vulnerabilities.

File(s) Affected

contracts/libraries/Math.sol #59-59

Examples

```
59              z := shr(18, mul(z, add(shr(r, x), 65536))) // A `mul()` is saved from starting `z` at 181.
```

Recommendation

Magic numbers should not be used.

20. Reentrancy Vulnerability in contracts/vault/SyncSwapVault.sol

Low risk Security Analyzer

A state variable is changed after a contract calls another contract function. The target contract can callback and reenter before the state variable is updated. This may lead to an unexpected result.

For example: `solidity function withdrawBalance(){ // send userBalance[msg.sender] Ether to msg.sender // if mgs.sender is a contract, it will call its fallback function if( ! (msg.sender.call.value(userBalance[msg.sender])() ) ){ throw; } userBalance[msg.sender] = 0; }` `msg.sender` can reenter the withdrawBalance function and withdraw all ether in the contract.

File(s) Affected

contracts/vault/SyncSwapVault.sol #155-179 #185-218 #220-232

Examples

```
155     function withdraw(address token, address to, uint amount) external override nonReentrant {
156         if (token == NATIVE_ETH) {
157             // Send native ETH to recipient.
158             TransferHelper.safeTransferETH(to, amount);
159         } else {
160             if (token == wETH) {
161                 // Ensure the same `reserves` and `balances` as native ETH.
```

Recommendation

Apply the `check-effects-interactions pattern`.

Informational (19)

1. Uninitialized Local Variables in contracts/pool/classic/SyncSwapClassicPool.sol

Informational Security Analyzer

A local variable is either never initialized or is initialized only under certain conditions, while the variable will be used regardless of its initialization. As a result, the default zero value is used, which is not desired.

File(s) Affected

contracts/pool/classic/SyncSwapClassicPool.sol #98-98 #171-171 #231-231 #314-314

Examples

```
98          ICallback.BaseMintCallbackParams memory params;
```

Recommendation

Initialize the local variable to a reasonable value. Explicitly setting it to zero if it is meant to be initialized to zero.

2. Uninitialized Local Variables in contracts/vault/VaultFlashLoans.sol

Informational Security Analyzer

A local variable is either never initialized or is initialized only under certain conditions, while the variable will be used regardless of its initialization. As a result, the default zero value is used, which is not desired.

File(s) Affected

contracts/vault/VaultFlashLoans.sol #100-100

Examples

```
100             uint i;
```

Recommendation

Initialize the local variable to a reasonable value. Explicitly setting it to zero if it is meant to be initialized to zero.

3. Uninitialized Local Variables in contracts/libraries/StableMath.sol

Informational Security Analyzer

A local variable is either never initialized or is initialized only under certain conditions, while the variable will be used regardless of its initialization. As a result, the default zero value is used, which is not desired.

File(s) Affected

contracts/libraries/StableMath.sol #28-28 #54-54

Examples

```
28          for (uint i; i < 256; ) {
```

Recommendation

Initialize the local variable to a reasonable value. Explicitly setting it to zero if it is meant to be initialized to zero.

4. Uninitialized Local Variables in contracts/pool/stable/SyncSwapStablePool.sol

Informational Security Analyzer

A local variable is either never initialized or is initialized only under certain conditions, while the variable will be used regardless of its initialization. As a result, the default zero value is used, which is not desired.

File(s) Affected

contracts/pool/stable/SyncSwapStablePool.sol #113-113 #186-186 #246-246 #329-329

Examples

```
113         ICallback.BaseMintCallbackParams memory params;
```

Recommendation

Initialize the local variable to a reasonable value. Explicitly setting it to zero if it is meant to be initialized to zero.

5. Uninitialized Local Variables in contracts/SyncSwapRouter.sol

Informational Security Analyzer

A local variable is either never initialized or is initialized only under certain conditions, while the variable will be used regardless of its initialization. As a result, the default zero value is used, which is not desired.

File(s) Affected

contracts/SyncSwapRouter.sol #79-79 #128-128 #173-173 #301-301 #302-302 #303-303

Examples

```
79         TokenInput memory input;
```

Recommendation

Initialize the local variable to a reasonable value. Explicitly setting it to zero if it is meant to be initialized to zero.

6. Uninitialized Local Variables in contracts/master/SyncSwapFeeRecipient.sol

Informational Security Analyzer

A local variable is either never initialized or is initialized only under certain conditions, while the variable will be used regardless of its initialization. As a result, the default zero value is used, which is not desired.

File(s) Affected

contracts/master/SyncSwapFeeRecipient.sol #89-89 #125-125

Examples

```
89         for (uint i; i < n; ) {
```

Recommendation

Initialize the local variable to a reasonable value. Explicitly setting it to zero if it is meant to be initialized to zero.

7. Tips for Gas Optimisation in contracts/SyncSwapRouter.sol

Informational Security Analyzer

Reading and writing state variables multiple times will consume more gas.

File(s) Affected

contracts/SyncSwapRouter.sol #55-66

Examples

```
55      function _transferFromSender(address token, address to, uint amount) private {
56          if (token == NATIVE_ETH) {
57              // Deposit ETH to the vault.
   •••
64              IVault(vault).deposit(token, to);
65          }
66      }
```

Recommendation

Use local variables instead of state variables.

8. Tips for Gas Optimisation in contracts/pool/stable/SyncSwapStablePool.sol

Informational Security Analyzer

Reading and writing state variables multiple times will consume more gas.

File(s) Affected

contracts/pool/stable/SyncSwapStablePool.sol #107-176 #180-235 #240-319 #323-387 #407-413 #415-418 #421-438 #480-485 #487-492 #494-525 #527-561 #563-579

Examples

```
107     function mint(
108         bytes calldata _data,
109         address _sender,
   •••
174
175         return params.liquidity;
176     }
```

Recommendation

Use local variables instead of state variables.

9. Tips for Gas Optimisation in contracts/master/SyncSwapFeeManager.sol

Informational Security Analyzer

Reading and writing state variables multiple times will consume more gas.

File(s) Affected

contracts/master/SyncSwapFeeManager.sol #40-51

Examples

```
40      constructor(address _feeRecipient) {
41          feeRecipient = _feeRecipient;
42
   •••
49          defaultSwapFee[2] = 40; // 0.04%.
50          defaultProtocolFee[2] = 50000; // 50%.
51      }
```

Recommendation

Use local variables instead of state variables.

10. Tips for Gas Optimisation in contracts/pool/classic/SyncSwapClassicPool.sol

Informational Security Analyzer

Reading and writing state variables multiple times will consume more gas.

File(s) Affected

contracts/pool/classic/SyncSwapClassicPool.sol #92-161 #165-220 #225-304 #308-372 #392-398 #400-403 #406-423 #465-470 #472-477 #479-498 #500-516

Examples

```
92      function mint(
93          bytes calldata _data,
94          address _sender,
    ...
159
160           return params.liquidity;
161       }
```

Recommendation

Use local variables instead of state variables.

11. Tips for Gas Optimisation in contracts/master/ForwarderRegistry.sol

Informational Security Analyzer

Reading and writing state variables multiple times will consume more gas.

File(s) Affected

contracts/master/ForwarderRegistry.sol #26-30

Examples

```
26      function removeForwarder(address forwarder) external onlyOwner {
27          require(_isForwarder[forwarder], "NOT_FORWARDER");
28          delete _isForwarder[forwarder];
29          emit RemoveForwarder(forwarder);
30      }
```

Recommendation

Use local variables instead of state variables.

12. Tips for Gas Optimisation in contracts/master/SyncSwapFeeRecipient.sol

Informational Security Analyzer

Reading and writing state variables multiple times will consume more gas.

File(s) Affected

contracts/master/SyncSwapFeeRecipient.sol #120-139

Examples

```
120     function removeFeeDistributor(address distributor, bool updateArray) external onlyOwner {
121         require(isFeeDistributor[distributor], "Not set");
122         delete isFeeDistributor[distributor];
    ...
137         }
138         emit RemoveFeeDistributor(distributor);
139     }
```

Recommendation

Use local variables instead of state variables.

13. Tips for Gas Optimisation in contracts/vault/SyncSwapVault.sol

Informational Security Analyzer

Reading and writing state variables multiple times will consume more gas.

File(s) Affected

contracts/vault/SyncSwapVault.sol #44-74 #76-88 #91-125 #129-143 #147-153 #155-179 #185-218 #220-232

Examples

```
44      function deposit(address token, address to) public payable override nonReentrant returns (uint amoun
45          if (token == NATIVE_ETH) {
46              // Use `msg.value` as amount for native ETH.
    •••
72              balances[token][to] += amount;
73          }
74      }
```

Recommendation

Use local variables instead of state variables.

14. Error-prone Assembly Usage in contracts/libraries/SignatureChecker.sol

Informational Security Analyzer

The use of assembly is error-prone and should be avoided.

File(s) Affected

contracts/libraries/SignatureChecker.sol #25-108

Examples

```
        returns (bool isValid)
    {
        /// @solidity memory-safe-assembly
        assembly {
            // Clean the upper 96 bits of `signer` in case they are dirty.
            for { signer := shr(96, shl(96, signer)) } signer {} {
                // Load the free memory pointer.
                // Simply using the free memory usually costs less if many slots are needed.
                let m := mload(0x40)

                let signatureLength := mload(signature)
                // If the signature is exactly 65 bytes in length.
                if iszero(xor(signatureLength, 65)) {
                    // Copy `r` and `s`.
                    mstore(add(m, 0x40), mload(add(signature, 0x20))) // `r`.
                    let s := mload(add(signature, 0x40))
                    mstore(add(m, 0x60), s)
                    // If `s` in lower half order, such that the signature is not malleable.
                    if iszero(gt(s, _MALLEABILITY_THRESHOLD)) {
                        mstore(m, hash)
                        // Compute `v` and store it in the memory.
                        mstore(add(m, 0x20), byte(0, mload(add(signature, 0x60))))
                        pop(
                            staticcall(
                                gas(), // Amount of gas left for the transaction.
                                0x01, // Address of `ecrecover`.
                                m, // Start of input.
                                0x80, // Size of input.
                                m, // Start of output.
                                0x20 // Size of output.
                            )
                        )
                        // `returndatasize()` will be `0x20` upon success, and `0x00` otherwise.
                        if mul(eq(mload(m), signer), returndatasize()) {
                            isValid := 1
                            break
                        }
                    }
                }

                // `bytes4(keccak256("isValidSignature(bytes32,bytes)"))`.
                let f := shl(224, 0x1626ba7e)
                // Write the abi-encoded calldata into memory, beginning with the function selector.
                mstore(m, f)
                mstore(add(m, 0x04), hash)
                mstore(add(m, 0x24), 0x40) // The offset of the `signature` in the calldata.
                {
                    let j := add(m, 0x44)
                    mstore(j, signatureLength) // The signature length.
                    // Copy the `signature` over.
                    for { let i := 0 } 1 {} {
                        i := add(i, 0x20)
                        mstore(add(j, i), mload(add(signature, i)))
                        if iszero(lt(i, signatureLength)) { break }
                    }
                }

```

```
                // forgefmt: disable-next-item
                isValid := and(
                    and(
                        // Whether the returndata is the magic value `0x1626ba7e` (left-aligned).
                        eq(mload(0x00), f),
                        // Whether the returndata is exactly 0x20 bytes (1 word) long.
                        eq(returndatasize(), 0x20)
                    ),
                    // Whether the staticcall does not revert.
                    // This must be placed at the end of the `and` clause,
                    // as the arguments are evaluated from right to left.
                    staticcall(
                        gas(), // Remaining gas.
                        signer, // The `signer` address.
                        m, // Offset of calldata in memory.
                        add(signatureLength, 0x64), // Length of calldata in memory.
                        0x00, // Offset of returndata.
                        0x20 // Length of returndata to write.
                    )
                )
                break
            }
        }
    }

}
```

Recommendation

We advise against using EVM assembly, as it is error-prone.

15. Unused Internal Functions in contracts/libraries/ReentrancyGuard.sol

Informational Security Analyzer

Presence of internal functions that are defined but never used in the contract. Such functions may introduce unnecessary gas consumption and make the code's review more difficult.

File(s) Affected

contracts/libraries/ReentrancyGuard.sol #74-76
contracts/libraries/Pausable.sol #84-87 #96-99

Examples

```
    function _reentrancyGuardEntered() internal view returns (bool) {
        return _status == _ENTERED;
    }
```

Recommendation

Remove unused functions to save gas and improve code readability.

16. Missing Mutability Specifier in contracts/pool/BasePoolFactory.sol

Informational Security Analyzer

The mutability specifiers are missing for variables that are assigned only once, either during the contract-level declaration or the execution of the constructor. This can lead to unnecessary gas consumption when utilizing these variables.

File(s) Affected

contracts/pool/BasePoolFactory.sol #12-12

Examples

```
    address public immutable master;
```

Recommendation

Add the mutability specifier for variables that are assigned only once. The constant keyword is recommended for the former case, and the immutable keyword is recommended for the latter case.

17. Inappropriate Solidity Naming Conventions in contracts/pool/classic/SyncSwapClassicPool.sol

Informational Security Analyzer

Solidity defines a naming convention that should be followed.

Rule exceptions

- Allow constant variable name/symbol/decimals to be lowercase (`ERC20`).
- Allow `_` at the beginning of the `mixed_case` match for private variables and unused parameters.

File(s) Affected

contracts/pool/classic/SyncSwapClassicPool.sol #27-27
contracts/pool/stable/SyncSwapStablePool.sol #30-30
contracts/libraries/ERC20Permit2.sol #34-34 #35-35

Examples

```
27      uint16 public constant override poolType = 1;
```

Recommendation

Follow the Solidity naming convention.

18. Variables with Similar Names in contracts/pool/stable/SyncSwapStablePool.sol and other 1 file

Informational Security Analyzer

Detect variables with names that are too similar.

File(s) Affected

contracts/pool/stable/SyncSwapStablePoolFactory.sol #18-18 #19-19
contracts/interfaces/pool/IPool.sol #43-43
contracts/pool/stable/SyncSwapStablePool.sol #43-43 #44-44 #59-59 #245-245 #328-328 #425-425 #435-435 #505-505 #538-538 #539-539 #569-569 #570-570
contracts/pool/classic/SyncSwapClassicPool.sol #111-111 #230-230 #313-313 #420-420

Examples

```
18          uint token0PrecisionMultiplier = 10 ** (18 - IERC20(token0).decimals());
```

Recommendation

Prevent variables from having similar names.

19. DoS with Block Gas Limit in contracts/SyncSwapRouter.sol

Informational Security Analyzer

When smart contracts are deployed, functions inside them are called, and the execution of these actions always requires a certain amount of gas based on how much computation is needed to complete them. The Ethereum network specifies a block gas limit, and the sum of all transactions included in a block can not exceed the threshold.\nProgramming patterns that are harmless in centralized applications can lead to Denial of Service conditions in smart contracts when the cost of executing a function exceeds the block gas limit. For example, modifying an array of unknown size that increases over time can lead to a Denial of Service condition.

File(s) Affected

contracts/abstract/Multicall.sol #9-31
contracts/libraries/StableMath.sol #16-41 #45-77
contracts/libraries/SignatureChecker.sol #25-108
contracts/SyncSwapRouter.sol #68-96 #116-156 #159-182 #295-349
contracts/master/SyncSwapFeeRecipient.sol #82-109 #120-139
contracts/vault/VaultFlashLoans.sol #86-149

Examples

```
    function multicall(bytes[] calldata data) public payable returns (bytes[] memory results) {
        results = new bytes[](data.length);

        for (uint i; i < data.length;) {
            (bool success, bytes memory result) = address(this).delegatecall(data[i]);

            if (!success) {
                // Next 5 lines from https://ethereum.stackexchange.com/a/83577
                if (result.length < 68) revert();
                assembly {
                    result := add(result, 0x04)
                }
                revert(abi.decode(result, (string)));
            }

            results[i] = result;

            // cannot realistically overflow on human timescales
            unchecked {
                ++i;
            }
        }
    }
}
```

Recommendation

Caution is advised when you expect to have large arrays that grow over time. Actions that require looping across the entire data structure should be avoided.\n If you absolutely must loop over an array of unknown size, then you should plan for it to take multiple blocks and potentially require multiple transactions.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without MetaTrust's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts MetaTrust to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. MetaTrust's position is that each company and individual are responsible for their own due diligence and continuous security. MetaTrust's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by MetaTrust is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS core-contracts Security Assessment AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, MetaTrust HEREBY DISCLAIMS

ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, MetaTrust SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, MetaTrust MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, MetaTrust PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER MetaTrust NOR ANY OF MetaTrust'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. MetaTrust WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT MetaTrust'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING core-contracts Security Assessment MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST MetaTrust WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF MetaTrust CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST MetaTrust WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.