ABDK CONSULTING

SMART CONTRACT AUDIT

Aktionariat

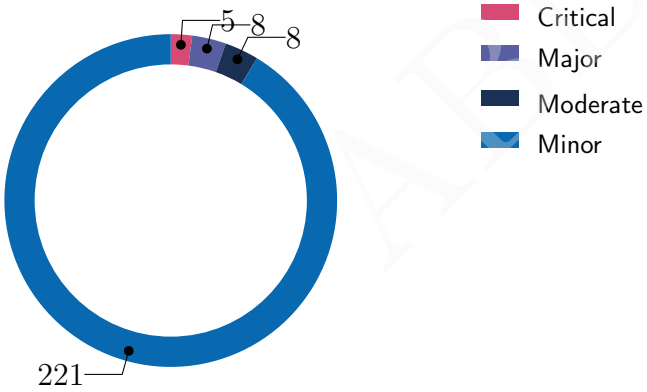
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
24th February 2022

We've been asked to review the 30 files in a Github repository. We found 5 critical, 8 major, and a few less important issues. All identified critical issues have been fixed or otherwise addressed in collaboration with the client.



Findings

ID	Severity	Category	Status
CVF-1	Minor	Unclear behavior	Info
CVF-2	Minor	Suboptimal	Info
CVF-3	Minor	Documentation	Fixed
CVF-4	Minor	Bad datatype	Fixed
CVF-5	Minor	Documentation	Info
CVF-6	Minor	Suboptimal	Fixed
CVF-7	Minor	Unclear behavior	Info
CVF-8	Minor	Bad datatype	Fixed
CVF-9	Minor	Procedural	Fixed
CVF-10	Minor	Bad naming	Fixed
CVF-11	Minor	Procedural	Info
CVF-12	Minor	Procedural	Info
CVF-13	Minor	Documentation	Fixed
CVF-14	Moderate	Bad datatype	Info
CVF-15	Minor	Procedural	Fixed
CVF-16	Minor	Suboptimal	Fixed
CVF-17	Minor	Suboptimal	Info
CVF-18	Minor	Suboptimal	Info
CVF-19	Minor	Suboptimal	Fixed
CVF-20	Minor	Suboptimal	Info
CVF-21	Minor	Suboptimal	Fixed
CVF-22	Critical	Flaw	Fixed
CVF-23	Minor	Suboptimal	Fixed
CVF-24	Minor	Suboptimal	Fixed
CVF-25	Minor	Suboptimal	Info
CVF-26	Moderate	Flaw	Info
CVF-27	Moderate	Flaw	Fixed

ID	Severity	Category	Status
CVF-28	Minor	Suboptimal	Info
CVF-29	Minor	Suboptimal	Info
CVF-30	Minor	Suboptimal	Info
CVF-31	Minor	Suboptimal	Info
CVF-32	Minor	Suboptimal	Info
CVF-33	Minor	Suboptimal	Info
CVF-34	Minor	Suboptimal	Info
CVF-35	Minor	Suboptimal	Fixed
CVF-36	Minor	Bad naming	Info
CVF-37	Minor	Procedural	Fixed
CVF-38	Minor	Bad datatype	Fixed
CVF-39	Moderate	Unclear behavior	Info
CVF-40	Minor	Suboptimal	Info
CVF-41	Moderate	Flaw	Fixed
CVF-42	Minor	Procedural	Fixed
CVF-43	Minor	Suboptimal	Info
CVF-44	Minor	Suboptimal	Info
CVF-45	Minor	Bad datatype	Fixed
CVF-46	Minor	Procedural	Info
CVF-47	Minor	Bad datatype	Fixed
CVF-48	Minor	Bad datatype	Fixed
CVF-49	Minor	Bad datatype	Fixed
CVF-50	Minor	Suboptimal	Fixed
CVF-51	Minor	Procedural	Fixed
CVF-52	Minor	Suboptimal	Fixed
CVF-53	Minor	Bad datatype	Fixed
CVF-54	Minor	Unclear behavior	Info
CVF-55	Minor	Bad datatype	Fixed
CVF-56	Minor	Bad datatype	Fixed
CVF-57	Minor	Bad datatype	Fixed

ID	Severity	Category	Status
CVF-58	Minor	Bad datatype	Fixed
CVF-59	Minor	Bad naming	Info
CVF-60	Minor	Bad naming	Fixed
CVF-61	Minor	Bad datatype	Fixed
CVF-62	Minor	Bad datatype	Fixed
CVF-63	Minor	Suboptimal	Fixed
CVF-64	Minor	Suboptimal	Info
CVF-65	Minor	Procedural	Fixed
CVF-66	Minor	Unclear behavior	Info
CVF-67	Minor	Unclear behavior	Info
CVF-68	Minor	Unclear behavior	Info
CVF-69	Minor	Unclear behavior	Info
CVF-70	Minor	Unclear behavior	Fixed
CVF-71	Critical	Flaw	Fixed
CVF-72	Minor	Documentation	Info
CVF-73	Minor	Bad datatype	Fixed
CVF-74	Minor	Suboptimal	Info
CVF-75	Minor	Bad datatype	Fixed
CVF-76	Minor	Documentation	Fixed
CVF-77	Minor	Bad datatype	Fixed
CVF-78	Minor	Bad datatype	Fixed
CVF-79	Minor	Bad naming	Info
CVF-80	Minor	Suboptimal	Info
CVF-81	Minor	Suboptimal	Info
CVF-82	Minor	Bad datatype	Info
CVF-83	Minor	Suboptimal	Fixed
CVF-84	Minor	Procedural	Fixed
CVF-85	Minor	Bad naming	Info
CVF-86	Minor	Procedural	Fixed
CVF-87	Minor	Procedural	Fixed

ID	Severity	Category	Status
CVF-88	Minor	Suboptimal	Info
CVF-89	Minor	Bad datatype	Fixed
CVF-90	Minor	Suboptimal	Fixed
CVF-91	Minor	Suboptimal	Info
CVF-92	Minor	Suboptimal	Info
CVF-93	Minor	Suboptimal	Info
CVF-94	Minor	Suboptimal	Info
CVF-95	Minor	Unclear behavior	Info
CVF-96	Minor	Bad naming	Info
CVF-97	Minor	Bad naming	Fixed
CVF-98	Minor	Procedural	Fixed
CVF-99	Minor	Procedural	Fixed
CVF-100	Minor	Bad datatype	Fixed
CVF-101	Minor	Procedural	Info
CVF-102	Minor	Bad datatype	Fixed
CVF-103	Minor	Suboptimal	Fixed
CVF-104	Minor	Procedural	Info
CVF-105	Minor	Bad datatype	Fixed
CVF-106	Minor	Bad datatype	Fixed
CVF-107	Minor	Procedural	Fixed
CVF-108	Minor	Bad datatype	Fixed
CVF-109	Minor	Suboptimal	Fixed
CVF-110	Minor	Suboptimal	Fixed
CVF-111	Minor	Bad datatype	Info
CVF-112	Minor	Bad naming	Info
CVF-113	Minor	Bad datatype	Fixed
CVF-114	Minor	Unclear behavior	Info
CVF-115	Minor	Bad datatype	Fixed
CVF-116	Minor	Bad datatype	Info
CVF-117	Minor	Suboptimal	Fixed

ID	Severity	Category	Status
CVF-118	Minor	Suboptimal	Fixed
CVF-119	Minor	Procedural	Info
CVF-120	Minor	Bad datatype	Fixed
CVF-121	Minor	Bad datatype	Fixed
CVF-122	Minor	Bad datatype	Fixed
CVF-123	Minor	Suboptimal	Fixed
CVF-124	Minor	Bad datatype	Fixed
CVF-125	Minor	Suboptimal	Fixed
CVF-126	Minor	Procedural	Fixed
CVF-127	Major	Flaw	Fixed
CVF-128	Minor	Bad datatype	Fixed
CVF-129	Major	Flaw	Info
CVF-130	Minor	Procedural	Fixed
CVF-131	Minor	Bad datatype	Fixed
CVF-132	Minor	Procedural	Fixed
CVF-133	Minor	Documentation	Fixed
CVF-134	Minor	Documentation	Fixed
CVF-135	Minor	Bad datatype	Fixed
CVF-136	Minor	Bad naming	Info
CVF-137	Minor	Procedural	Fixed
CVF-138	Minor	Bad datatype	Fixed
CVF-139	Minor	Bad datatype	Fixed
CVF-140	Minor	Bad naming	Fixed
CVF-141	Minor	Bad datatype	Fixed
CVF-142	Minor	Procedural	Info
CVF-143	Minor	Documentation	Fixed
CVF-144	Minor	Documentation	Fixed
CVF-145	Minor	Procedural	Fixed
CVF-146	Minor	Procedural	Info
CVF-147	Minor	Suboptimal	Info

ID	Severity	Category	Status
CVF-148	Moderate	Overflow/Underflow	Info
CVF-149	Major	Flaw	Info
CVF-150	Major	Suboptimal	Info
CVF-151	Major	Procedural	Info
CVF-152	Minor	Bad datatype	Fixed
CVF-153	Minor	Unclear behavior	Fixed
CVF-154	Minor	Suboptimal	Fixed
CVF-155	Minor	Procedural	Fixed
CVF-156	Minor	Suboptimal	Fixed
CVF-157	Minor	Documentation	Fixed
CVF-158	Minor	Suboptimal	Fixed
CVF-159	Minor	Bad datatype	Fixed
CVF-160	Minor	Suboptimal	Info
CVF-161	Minor	Suboptimal	Info
CVF-162	Minor	Suboptimal	Info
CVF-163	Minor	Suboptimal	Info
CVF-164	Minor	Suboptimal	Fixed
CVF-165	Moderate	Flaw	Info
CVF-166	Minor	Readability	Fixed
CVF-167	Minor	Suboptimal	Fixed
CVF-168	Critical	Flaw	Fixed
CVF-169	Minor	Suboptimal	Fixed
CVF-170	Minor	Readability	Info
CVF-171	Minor	Suboptimal	Info
CVF-172	Minor	Suboptimal	Fixed
CVF-173	Moderate	Flaw	Info
CVF-174	Minor	Suboptimal	Fixed
CVF-175	Major	Flaw	Info
CVF-176	Minor	Suboptimal	Info
CVF-177	Minor	Suboptimal	Fixed

ID	Severity	Category	Status
CVF-178	Critical	Flaw	Fixed
CVF-179	Minor	Suboptimal	Fixed
CVF-180	Minor	Suboptimal	Fixed
CVF-181	Minor	Procedural	Info
CVF-182	Minor	Procedural	Fixed
CVF-183	Minor	Bad naming	Info
CVF-184	Minor	Suboptimal	Info
CVF-185	Minor	Bad naming	Info
CVF-186	Minor	Readability	Fixed
CVF-187	Minor	Documentation	Info
CVF-188	Minor	Suboptimal	Fixed
CVF-189	Minor	Documentation	Fixed
CVF-190	Minor	Suboptimal	Fixed
CVF-191	Minor	Procedural	Fixed
CVF-192	Minor	Procedural	Fixed
CVF-193	Minor	Procedural	Fixed
CVF-194	Minor	Suboptimal	Fixed
CVF-195	Critical	Flaw	Fixed
CVF-196	Minor	Suboptimal	Fixed
CVF-197	Minor	Suboptimal	Fixed
CVF-198	Minor	Suboptimal	Info
CVF-199	Minor	Procedural	Fixed
CVF-200	Minor	Bad naming	Fixed
CVF-201	Minor	Bad datatype	Fixed
CVF-202	Minor	Bad datatype	Info
CVF-203	Minor	Procedural	Info
CVF-204	Minor	Procedural	Info
CVF-205	Minor	Bad datatype	Info
CVF-206	Minor	Bad datatype	Fixed
CVF-207	Minor	Procedural	Fixed

ID	Severity	Category	Status
CVF-208	Minor	Suboptimal	Fixed
CVF-209	Minor	Documentation	Fixed
CVF-210	Minor	Bad datatype	Fixed
CVF-211	Minor	Readability	Info
CVF-212	Minor	Bad datatype	Fixed
CVF-213	Minor	Bad datatype	Fixed
CVF-214	Minor	Suboptimal	Fixed
CVF-215	Minor	Suboptimal	Fixed
CVF-216	Minor	Bad naming	Fixed
CVF-217	Minor	Procedural	Fixed
CVF-218	Minor	Suboptimal	Info
CVF-219	Minor	Procedural	Info
CVF-220	Major	Flaw	Fixed
CVF-221	Major	Bad datatype	Fixed
CVF-222	Minor	Unclear behavior	Info
CVF-223	Minor	Procedural	Fixed
CVF-224	Minor	Procedural	Fixed
CVF-225	Minor	Bad datatype	Fixed
CVF-226	Minor	Documentation	Fixed
CVF-227	Minor	Procedural	Fixed
CVF-228	Minor	Documentation	Fixed
CVF-229	Minor	Procedural	Fixed
CVF-230	Minor	Bad datatype	Fixed
CVF-231	Minor	Bad datatype	Fixed
CVF-232	Minor	Procedural	Fixed
CVF-233	Minor	Bad datatype	Fixed
CVF-234	Minor	Documentation	Fixed
CVF-235	Minor	Procedural	Fixed
CVF-236	Minor	Bad datatype	Fixed
CVF-237	Minor	Bad datatype	Fixed

ID	Severity	Category	Status
CVF-238	Minor	Unclear behavior	Fixed
CVF-239	Minor	Procedural	Info
CVF-240	Minor	Procedural	Info
CVF-241	Minor	Procedural	Info
CVF-242	Minor	Procedural	Info

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	February 11, 2022	D. Khovratovich	Initial Draft
0.2	February 11, 2022	D. Khovratovich	Minor revision
1.0	February 11, 2022	D. Khovratovich	Release
1.1	February 24, 2022	D. Khovratovich	CVF-129 Critical category downgraded
2.0	February 24, 2022	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.
We have reviewed the contracts at the commit:

- brokerbot/Brokerbot.sol
- brokerbot/IBrokerbot.sol
- brokerbot/IUniswapV3.sol
- brokerbot/LicensedBrokerbot.sol
- brokerbot/PaymentHub.sol
- draggable/ERC20Draggable.sol
- draggable/IDraggable.sol
- draggable/IOffer.sol
- draggable/IOfferFactory.sol
- draggable/Offer.sol
- draggable/OfferFactory.sol
- ERC20/ERC20Allowlistable.sol
- ERC20/ERC20Flaggable.sol
- ERC20/ERC20Named.sol
- ERC20/IERC20.sol
- ERC20/IERC677Receiver.sol
- multisig/MultiSigCloneFactory.sol
- multisig/MultiSigWallet.sol
- multisig/Nonce.sol
- multisig/RLPEncode.sol
- recovery/ERC20Recoverable.sol
- recovery/IRecoverable.sol
- recovery/IRecoveryHub.sol
- recovery/RecoveryHub.sol

- shares/AllowlistDraggableShares.sol
- shares/AllowlistShares.sol
- shares/DraggableShares.sol
- shares/Shares.sol
- utils/Address.sol
- utils/Ownable.sol

2.1 About ABDK

ABDK Consulting, established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function. The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

2.2 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication. After fixing the indicated issues the smart contracts should be re-audited.

2.3 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and and used technologies, as well as on what the client is expecting from the audit. In current audit we use:

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places and that their visibility scopes and access levels are relevant. At this phase we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and is done properly. At this phase we understand user roles and permissions, as well as what assets the system ought to protect.

- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

3 Detailed Results

3.1 CVF-1

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** Nonce.sol

Description What is the reason to use 100 here? 64 would be more logical.
Client Comment 100 allows for more pending transactions being executed out of order than 64, while still guarding against unreasonable jumps.

Listing 1:

```
32 uint256 public constant MAX_INCREASE = 100;
```

3.2 CVF-2

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Nonce.sol

Description This field basically stored two values: the maximum nonce ever used and the bitmap of the 128 nonces prior to that nonce. Such format doesn't allow representing the situation when no nonces were ever used.
Recommendation Consider storing the lowest free nonce and the bitmap of 128 nonces prior to the nonce before the lowest free nonce. So, in case the lowest free nonce is 0, which is the initial state, the bitmap would store flags for the nonces from -129 to -2. Of cause negative nonces could not be used, but this doesn't prevent storing flags for them. Such approach would allow using all the nonces from 0 to 2^128 - 1.
Client Comment Not worth changing given the risk of introducing bugs.

Listing 2:

```
34 uint256 private compound;
```

3.3 CVF-3

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** Nonce.sol

Recommendation The comment is not accurate as the minimum value that could be returned by this function is 129.

Listing 3:

```
41 * The next recommended nonce, which is the highest nonce ever
   ↪ used plus one.
```

3.4 CVF-4

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Nonce.sol

Description In most cases, the "uint128" type is used for nonces, but not in the return type of this functions.
Recommendation Consider using "uint128" as the return type.

Listing 4:

```
43 function nextNonce() external view returns (uint256){
```

3.5 CVF-5

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** Nonce.sol

Recommendation This comment is not accurate, as the nonce 128 may never be used, even when it fits into the range [nextNonce() - 129, nextNonce() - 1].
Client Comment I think it would actually be possible to use 128 as a nonce even though the first recommended nonce is 129.

Listing 5:

```
50 * For the nonces in the interval [nextNonce() - 129, nextNonce()
   ↪ - 1], this is true for the nonces that have not been used
   ↪ yet.
```

3.6 CVF-6

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Nonce.sol

Description This is equivalent to: revert ("used");

Listing 6:

```
70 require(false, "used");
```

3.7 CVF-7

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** Nonce.sol

Description This function reverts in case max < nonce.
Recommendation Consider returning false in such a case.
Client Comment We can live with the function reverting here.

Listing 7:

```
90 function isValidLowNonce(uint128 max, uint128 reg, uint256 nonce
   ↪ ) private pure returns (bool){
```

3.8 CVF-8

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Nonce.sol

Description In most cases, the "uint128" type is used for nonces, but not for the "nonce" argument here.
Recommendation Consider using "uint128" for the nonce argument.

Listing 8:

```
90 function isValidLowNonce(uint128 max, uint128 reg, uint256 nonce
   ↪ ) private pure returns (bool){
```

3.9 CVF-9

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** MultiSigWallet.sol

Description This variable is used without being initialized.
Recommendation Consider explicitly initializing to 0.

Listing 9:

```
19 uint16 public signerCount;
```

3.10 CVF-10

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** MultiSigWallet.sol

Recommendation The name is confusing. Usually a cosignature is a signature complementary to the given one, i.e. someone else signature. This number is either the number of required cosignatures +1 , or the total number of signatures, and should be renamed accordingly.

Listing 10:

```
24 uint8 cosignaturesNeeded
```

3.11 CVF-11

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** MultiSigWallet.sol

Description This event don't include the full transaction data and ether value.
Recommendation Consider including this information.
Client Comment Transaction data can be extracted off-chain.

Listing 11:

```
27 event Transacted(
```

3.12 CVF-12

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** MultiSigWallet.sol

Recommendation This parameter should be indexed.
Client Comment Indexed costs more gas and we didn't need it so far

Listing 12:

```
29 bytes4 selector, // selected operation
```

3.13 CVF-13

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** MultiSigWallet.sol

Description This comment is confusing, as the gas price is not used here.
Recommendation Consider rephrasing that the gas price field is used to embed the contract ID into an encoded transaction.

Listing 13:

```
34 // We use the gas price to get a unique id into our transactions
   ↪ .
```

3.14 CVF-14

- **Severity** Moderate
- **Status** Info
- **Category** Bad datatype
- **Source** MultiSigWallet.sol

Description Due to the birthday paradox, only 2^16 contracts are needed to have an Id collsion with probability >0.5 This is not a big number and can be achieved in practice.
Recommendation Consider using at least 64 bits for an ID.
Client Comment Can be checked externally and a new contract deployed in case of a collision. Also, practical attack surface is limited.

Listing 14:

```
36 // same id, but it practically rules out that someone
   ↪ accidentally creates two
```

3.15 CVF-15

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** MultiSigWallet.sol

Recommendation it is a good practice to put a comment into an empty block to explain why the block is empty.

Listing 15:

```
47 receive() external payable {
   }
```

3.16 CVF-16

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** MultiSigWallet.sol

Recommendation Consider emitting an event inside this function with msg.sender and msg.value as the parameters. This would make it easier to track incoming ether transfers to the wallet.

Listing 16:

```
47 receive() external payable {
```

3.17 CVF-17

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** MultiSigWallet.sol

Recommendation It would be more efficient to accept a single array of structs with three fields rather than three parallel arrays. Such approach would also make the length check unnecessary.
Client Comment Maybe in future versions.

Listing 17:

```
54   uint8[] calldata v, bytes32[] calldata r, bytes32[] calldata s
       ↪ ) external view returns (address[] memory) {

67 function execute(uint128 nonce, address to, uint value, bytes
     ↪ calldata data, uint8[] calldata v, bytes32[] calldata r,
     ↪ bytes32[] calldata s) external returns (bytes memory) {

119 function verifySignatures(bytes32 transactionHash, uint8[]
      ↪ calldata v, bytes32[] calldata r, bytes32[] calldata s)
```

3.18 CVF-18

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** MultiSigWallet.sol

Description Actually, there is no reliable way to tell whether the transaction actually succeeded or not, as the transaction itself could revert with the Test passed. Reverting." message.
Recommendation Consider implementing a function that tries executing the transaction, reverts its effects and returns a boolean execution status plus the data returned by the transaction. The following gist illustrated the idea: https://gist.github.com/3sGgpQ8H/bd94f6e31c10895d9e01891cde34b5e0
Client Comment Prefer not using assembly if not necessary for readability.

Listing 18:

```
62 function checkExecution(address to, uint value, bytes calldata
   ↪ data) external {
```

3.19 CVF-19

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** MultiSigWallet.sol

Description This is equivalent to: revert ("Test passed. Reverting.");

Listing 19:

```
64 require(false, "Test passed. Reverting.");
```

3.20 CVF-20

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** MultiSigWallet.sol

Recommendation A more efficient way would be: bytes4 selector; assembly { selector := mload (add (data, 0x20)) }
Client Comment Compiler complains: "Call data elements cannot be accessed directly. Use ".offset" and ".length" to access the calldata offset and length of this array and then use "calldatacopy"."

Listing 20:

```
80 return bytes4(data[0]) | (bytes4(data[1]) >> 8) | (bytes4(data
   ↪ [2]) >> 16) | (bytes4(data[3]) >> 24);
```

3.21 CVF-21

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** MultiSigWallet.sol

Description This function is very inefficient.
Recommendation Consider using the following approach: function toBytes (uint256 x) public pure returns (bytes memory result) { uint l = 0; uint xx = x; if (x >= 0x100000000000000000000000000000000) { x >>= 128; l += 16; } if (x >= 0x10000000000000000) { x >>= 64; l += 8; } if (x >= 0x100000000) { x >>= 32; l += 4; } if (x >= 0x10000) { x >>= 16; l += 2; } if (x >= 0x100) { x >>= 8; l += 1; } if (x > 0x0) { l += 1; } assembly { result := mload (0x40) mstore (0x40, add (result, add (l, 0x20))) mstore (add (result, l), xx) mstore (result, l) } }

Listing 21:

```
84 function toBytes(uint number) internal pure returns (bytes
    ↪ memory){
```

3.22 CVF-22

- **Severity** Critical
- **Status** Fixed
- **Category** Flaw
- **Source** MultiSigWallet.sol

Description This loop will never end in case number >= 2^248.

Listing 22:

```
87 while (number >= temp){
```

3.23 CVF-23

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** MultiSigWallet.sol

Recommendation As this value is constant, it would be more efficient to just hardcode its value: all[2] = bytes ("\x82\x52\x08"); and not pass the value through the "RLPEncode.encodeBytes" function.

Listing 23:

```
106 all[2] = toBytes(21000); // gas limit
```

3.24 CVF-24

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** MultiSigWallet.sol

Recommendation As the size of an encoded address is known, it would be more efficient to just hardcode the RLP prefix: all[3] = abi.encodePacked (bytes1 (0x94), to); and not pass this value through the "RLPEncode.encodeBytes" function.

Listing 24:

```
107 all[3] = abi.encodePacked(to);
```

3.25 CVF-25

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** MultiSigWallet.sol

Recommendation The value assigned here is just an empty bytes array, no need to call the "toBytes" function.
Client Comment Considered for future improvement.

Listing 25:

```
111 all[7] = toBytes(0);
```

3.26 CVF-26

- **Severity** Moderate
- **Status** Info
- **Category** Flaw
- **Source** MultiSigWallet.sol

Description There is not check that the lengths of the "v", "r", and "s" arrays are the same. In case the "v" or "s" array is longer than the "r" array, extra elements are silently ignored.
Recommendation Consider adding appropriate checks.
Client Comment Caller can do that verification off-chain before calling.

Listing 26:

```
119 function verifySignatures(bytes32 transactionHash, uint8[]
    ↪ calldata v, bytes32[] calldata r, bytes32[] calldata s)
```

3.27 CVF-27

- **Severity** Moderate
- **Status** Fixed
- **Category** Flaw
- **Source** MultiSigWallet.sol

Description On an invalid signature, the "ecrecover" function returns zero address, so anybody may "sign" anything on behalf of zero address.
Recommendation Consider adding a "require" statement to check that the "signer" address is not zero.

Listing 27:

```
123 address signer = ecrecover(transactionHash, v[i], r[i], s[i]);
```

3.28 CVF-28

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** MultiSigWallet.sol

Description The second check probably should be done only for the signer with the smallest 'cosignaturesNeeded' number.
Client Comment No, if one signer needs 2 signatures and there are two other signers that need 3, we still need 3 signatures, so the maximum should be checked for safety. Sometimes, this will deny valid signatures, but it is the task of the caller to make sure to not include unnecessary signatories.

Listing 28:

```
125 require(cosignaturesNeeded > 0 && cosignaturesNeeded <= r.length
    ↪ , "cosigner error");
```

3.29 CVF-29

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** MultiSigWallet.sol

Description This check has O(n^2) complexity. It could be reduced to O(n) by requiring that the signs in the arrays are sorted by the signer address.
Recommendation In such case it would be enough to check that found [i + 1] > found [i] for every "i".
Client Comment Considered for future improvements.

Listing 29:

```
128 requireNoDuplicates(found);
```

3.30 CVF-30

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** MultiSigWallet.sol

Description This function implement a straightforward approach with O(n^2) complexity. More efficient approaches exists. However, the complexity could be reduced to O(n) by sorting the signatures.
Recommendation See the comment for the previous function.
Client Comment Considered for future improvement.

Listing 30:

```
132 function requireNoDuplicates(address[] memory found) private
    ↪ pure {
```

3.31 CVF-31

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** MultiSigWallet.sol

Description This check is redundant. It is still possible to reduce the number of signers below the minimum number of cosigners, effectively making it impossible to execute anything. A good lock protection would be to require that the signed being modified doesn't participate in signing the "setSigner" transaction.
Client Comment Not sure if we want to restrict the ability to change ones own signatory power. Maybe we should say: if the signer participates, power can only be improved (i.e. number of signers reduced, but not to zero).

Listing 31:

```
145 require(signerCount > 0, "signer count 0");
```

3.32 CVF-32

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** MultiSigWallet.sol

Recommendation These functions should probably log an event.
Client Comment Actually, two SignerChanged events are emitted.

Listing 32:

```
148 function migrate(address destination) external {

152 function migrate(address source, address destination) external
    ↪ authorized {
```

3.33 CVF-33

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** MultiSigWallet.sol

Description This check is redundant. It is still possible to add a dead address as a singer, or add an address of a not yet deployed contract, and then deploy the contract.
Recommendation Consider removing this check.
Client Comment It does not prevent everything that can go wrong, but at least it prevents one of the most common errors.

Listing 33:

```
163 require(!Address.isContract(signer), "signer cannot be a
    ↪ contract");
```

3.34 CVF-34

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Shares.sol

Description Writing terms into the storage could consume lots of gas.
Recommendation Consider just logging them in an event, as the terms are most probably not needed on-chain.
Client Comment Legal requirement to have it as a permanent field in the register.

Listing 34:

```
50 string public terms;
```

3.35 CVF-35

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Shares.sol

Description This initialization is misleading, as the real initialization takes place in the constructor.
Recommendation Consider leaving the variable uninitialized here.

Listing 35:

```
52 uint256 public totalShares = 0; // total number of shares, maybe
    ↪ not all tokenized
```

3.36 CVF-36

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** Shares.sol

Recommendation Events are usually named via nouns, such as "TokenInvalidation" or just "InvalidTokens".
Client Comment Transacted, NamedChanged, OwnershipTransferred are common events that follow the same pattern.

Listing 36:

```
56 event TokensDeclaredInvalid(address indexed holder, uint256
   ↪ amount, string message);
```

3.37 CVF-37

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Shares.sol

Recommendation Theae function should emit some events.

Listing 37:

```
73 function setTerms(string memory _terms) external onlyOwner {

83 function setTotalShares(uint256 _newTotalShares) external
   ↪ onlyOwner() {
```

3.38 CVF-38

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Shares.sol

Recommendation The argument type should be "IERC20".

Listing 38:

```
98 function setCustomClaimCollateral(address collateral, uint256
   ↪ rate) external onlyOwner() {
```

3.39 CVF-39

- **Severity** Moderate
- **Status** Info
- **Category** Unclear behavior
- **Source** Shares.sol

Description Without immobilizing or destroying the shares this function doesn't make any sense, as the invalidated shares could still be sold or transferred, mixed with valid shares and thus effectively made valid again. as it wouldn't be possible to distinguish invalid shard from valid ones.
Recommendation Consider immobilizing the shares or destroying them.
Client Comment This is a deliberate choice. Legally, this function can only be called after a court found that the tokens have been irrevocably lost (them being stolen does not suffice) so we can safely assume that they cannot be moved any more.

Listing 39:

```
111 * This function is purely declarative. It does not technically
    ↪ immobilize the affected tokens as
* that would give the issuer too much power.
```

3.40 CVF-40

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Shares.sol

Description This check could easily be bypassed by invoking the function several times.
Recommendation Consider removing this check.
Client Comment This should guard against obvious errors, not increase security.

Listing 40:

```
116 require(amount <= holderBalance, "amount too high");
```

3.41 CVF-41

- **Severity** Moderate
- **Status** Fixed
- **Category** Flaw
- **Source** Shares.sol

Description The returned value is ignored.
Recommendation Consider revering when false is returned.

Listing 41:

```
135 IERC677Receiver(callee).onTokenTransfer(shareholder, amount,
    ↪ data);
```

3.42 CVF-42

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Shares.sol

Recommendation This check should be performed inside the "_transfer" function.

Listing 42:

```
162 require(_amount <= balanceOf(msg.sender), "balance");
```

3.43 CVF-43

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Shares.sol

Recommendation It would be more efficient to just do: _burn (msg.sender, _amount);
Client Comment See comment, there are legal considerations to formally return the tokens to the issuer first.

Listing 43:

```
163 _transfer(msg.sender, address(this), _amount);
_burn(address(this), _amount);
```

3.44 CVF-44

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** DraggableShares.sol

Description Keeping the terms in the storage could consume lots of gas.
Recommendation Consider storing them in the contract bytecode (hardcode at compile time, as Solidity don't support immutable variables of non-atomic types) or just logging them in an event once, as the terms are not usually needed on-chain.
Client Comment Legal requirement to keep it as a field in the register.

Listing 44:

```
42 string public terms;
```

3.45 CVF-45

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** DraggableShares.sol

Recommendation The type of this argument should be "IERC20".

Listing 45:

```
46 address _wrappedToken,
```

3.46 CVF-46

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** DraggableShares.sol

Description There are no range checks for these arguments.
Recommendation Consider adding appropriate checks.
Client Comment Ok with us.

Listing 46:

```
47 uint256 _quorumBps,
   uint256 _votePeriodSeconds,
```

3.47 CVF-47

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** DraggableShares.sol

Recommendation The type of this argument should be "IRecoveryHub".

Listing 47:

```
49 address _recoveryHub,
```

3.48 CVF-48

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** DraggableShares.sol

Recommendation The type of this argument should be "IOfferFactory".

Listing 48:

```
50 address _offerFactory,
```

3.49 CVF-49

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** AllowlistShares.sol

Recommendation The type of this argument should be "IRecoveryHub".

Listing 49:

```
41 address _recoveryHub,
```

3.50 CVF-50

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** AllowlistShares.sol

Description This assignment is redundant, as the "Shares" constructor has already stored the terms.

Listing 50:

```
47 terms = _terms; // to update the terms, migrate to a new
   ↪ contract. That way it is ensured that the terms can only
   ↪ be updated when the quorom agrees.
```

3.51 CVF-51

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** AllowlistDraggableShares.sol

Description This import is not used.
Recommendation Consider removing it.

Listing 51:

```
32 import "./DraggableShares.sol";
```

3.52 CVF-52

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** AllowlistDraggableShares.sol

Description Keeping the terms in the storage could consume lots of gas.
Recommendation Consider storing them in the contract bytecode (hardcode at compile time, as Solidity don't support immutable variables of non-atomic types) or just logging them in an event once, as the terms are not usually needed on-chain.

Listing 52:

```
36 string public terms;
```

3.53 CVF-53

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** AllowlistDraggableShares.sol

Recommendation The type of this argument should be "IERC20".

Listing 53:

```
40 address _wrappedToken,
```

3.54 CVF-54

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** AllowlistDraggableShares.sol

Description There are no range checks for these arguments.
Recommendation Consider adding appropriate checks.
Client Comment Ok with us.

Listing 54:

```
41 uint256 _quorum,
   uint256 _votePeriod,
```

3.55 CVF-55

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** AllowlistDraggableShares.sol

Recommendation The type of this argument should be "IRecoveryHub".

Listing 55:

```
43 address _recoveryHub,
```

3.56 CVF-56

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** AllowlistDraggableShares.sol

Recommendation The type of this argument should be "IOfferFactory".

Listing 56:

```
44 address _offerFactory,
```

3.57 CVF-57

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** RecoveryHub.sol

Recommendation The type of this field should be "IERC20".

Listing 57:

```
41 address currencyUsed; // The currency (XCHF) can be updated, we
   ↪ record the currency used for every request
```

3.58 CVF-58

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** RecoveryHub.sol

Recommendation The type of the first key should be "IRecoverable".

Listing 58:

```
44 mapping(address => mapping (address => Claim)) public claims; //
   ↪  there can be at most one claim per token and claimed
   ↪ address
```

3.59 CVF-59

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** RecoveryHub.sol

Recommendation Events are usually named via nouns, such as "Claim", "ClaimClearing", "ClaimDeletion" etc.
Client Comment Transacted, NamedChanged, OwnershipTransferred are common events that follow the same pattern.

Listing 59:

```
47 event ClaimMade(address indexed token, address indexed
   ↪ lostAddress, address indexed claimant, uint256 balance);
   event ClaimCleared(address indexed token, address indexed
   ↪ lostAddress, uint256 collateral);
   event ClaimDeleted(address indexed token, address indexed
   ↪ lostAddress, address indexed claimant, uint256 collateral)
   ↪ ;
50 event ClaimResolved(address indexed token, address indexed
   ↪ lostAddress, address indexed claimant, uint256 collateral)
   ↪ ;
```

3.60 CVF-60

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** RecoveryHub.sol

Recommendation The name should be "isRecoverable" for consistency with the setter name.

Listing 60:

```
60 function isRecoveryEnabled(address target) public view returns (
   ↪ bool) {
```

3.61 CVF-61

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** RecoveryHub.sol

Recommendation The type of the "token" argument should be "IRecoverable".

Listing 61:

```
 79 function declareLost(address token, address collateralType,
      ↪ address lostAddress) external {

105 function getClaimant(address token, address lostAddress)
      ↪ external view returns (address) {

109 function getCollateral(address token, address lostAddress)
      ↪ external view returns (uint256) {

113 function getCollateralType(address token, address lostAddress)
      ↪ external view returns (address) {

117 function getTimeStamp(address token, address lostAddress)
      ↪ external view returns (uint256) {

129 function clearClaimFromUser(address token) external {

133 function clearClaim(address token, address holder) private {

148 function recover(address token, address lostAddress) external {
```

3.62 CVF-62

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** RecoveryHub.sol

Recommendation The type of the "collateralType" argument should be "IERC20".

Listing 62:

```
 79 function declareLost(address token, address collateralType,
      ↪ address lostAddress) external {
```

3.63 CVF-63

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RecoveryHub.sol

Description The variable is redundant, as access to "msg.sender" is cheaper than to a local variable.

Listing 63:

```
83 address claimant = msg.sender;
```

3.64 CVF-64

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** RecoveryHub.sol

Description This formula treats the collateral rate as an integer number, thus fractional rates are not allowed. This is very limiting.
Recommendation Consider supporting fractional collateral rates.

Listing 64:

```
85 uint256 collateral = balance * collateralRate;
```

3.65 CVF-65

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** RecoveryHub.sol

Recommendation This check should be two lines above the current position.

Listing 65:

```
87 require(balance > 0, "empty");
```

3.66 CVF-66

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** RecoveryHub.sol

Description This function silently returns the zero address for a non-existing claim.
Recommendation Consider reverting in such a case.
Client Comment Ok with us

Listing 66:

```
105 function getClaimant(address token, address lostAddress)
      ↪ external view returns (address) {
```

3.67 CVF-67

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** RecoveryHub.sol

Description This function silently returns zero for a non-existing claim.
Recommendation Consider reverting in such a case.
Client Comment Ok with us.

Listing 67:

```
109 function getCollateral(address token, address lostAddress)
      ↪ external view returns (uint256) {
```

3.68 CVF-68

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** RecoveryHub.sol

Description This function silently returns the zero address for a non-existing claim.
Recommendation Consider reverting in such a case.
Client Comment Ok with us.

Listing 68:

```
113 function getCollateralType(address token, address lostAddress)
      ↪ external view returns (address) {
```

3.69 CVF-69

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** RecoveryHub.sol

Description This function silently returns zero for a non-existing claim.
Recommendation Consider reverting in such a case.
Client Comment Ok with us.

Listing 69:

```
117 function getTimeStamp(address token, address lostAddress)
    ↪ external view returns (uint256) {
```

3.70 CVF-70

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** RecoveryHub.sol

Recommendation The function should revert in case this condition is false. Otherwise it would be hard to tell whether a claim was cleared successfully or not.

Listing 70:

```
135 if (claim.collateral != 0) {
```

3.71 CVF-71

- **Severity** Critical
- **Status** Fixed
- **Category** Flaw
- **Source** RecoveryHub.sol

Description This tries to "return" the collateral to this smart contract rather than to the lost address.
Recommendation Should be: currency.transfer (holder, claim.collateral)
Client Comment Fortunately correct in the previous version used by clients.

Listing 71:

```
139 require(currency.transfer(address(this), claim.collateral), "
    ↪ could not return collateral");
```

3.72 CVF-72

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** IRecoverable.sol

Description The number format of the returned value is unclear.
Recommendation Consider documenting.
Client Comment Is documented in ERC20Recoverable.

Listing 72:

```
12 function getCollateralRate(address collateral) public view
   ↪ virtual returns(uint256);
```

3.73 CVF-73

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** IRecoverable.sol

Recommendation The argument type should be "IERC20".

Listing 73:

```
12 function getCollateralRate(address collateral) public view
   ↪ virtual returns(uint256);
```

3.74 CVF-74

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** ERC20Recoverable.sol

Description The claim deleter would need to pay for gas when deleting a claim, and the claimant would take the whole collateral back. This could be used to attack the claim deleter.
Recommendation Consider accumulating some gas in filled storage slots when a claim is made, and then using this gas to refund claim deletion. The same accumulated gas could be used to refund gas costs for claim recovery and claim clearing.
Client Comment As far as I know, it costs more gas to create data than to delete it again. Therefore, the attack is more costly than the defense.

Listing 74:

```
44 * Furthermore, if "getClaimDeleter" is defined in the subclass,
   ↪ the returned address is allowed to
* delete claims, returning the collateral. This can help to
   ↪ prevent obvious cases of abuse of the claim
* function, e.g. cases of front-running.
```

3.75 CVF-75

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** ERC20Recoverable.sol

Recommendation The type of this variable should be "IERC20".

Listing 75:

```
54 address public customCollateralAddress;
```

3.76 CVF-76

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** ERC20Recoverable.sol

Description The number format of this variable is unclear.
Recommendation Consider documenting.

Listing 76:

```
55 uint256 public customCollateralRate;
```

3.77 CVF-77

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** ERC20Recoverable.sol

Recommendation The argument type should be "IRecoveryHub".

Listing 77:

```
59 constructor(address recoveryHub){
```

3.78 CVF-78

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** ERC20Recoverable.sol

Recommendation This value should be a named constant.

Listing 78:

```
82 return 180 days;
```

3.79 CVF-79

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** Ownable.sol

Recommendation Events are usually named via nouns, such as "OwnershipTransfer".
Client Comment Kept for compatibility with OpenZeppelin.

Listing 79:

```
28 event OwnershipTransferred(address indexed previousOwner,
   ↪ address indexed newOwner);
```

3.80 CVF-80

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Ownable.sol

Description The "previousOwner" parameter is redundant, as the previous owner address could be derived from the previous event.
Client Comment Kept for compatibility with OpenZeppelin.

Listing 80:

```
28 event OwnershipTransferred(address indexed previousOwner,
   ↪ address indexed newOwner);
```

3.81 CVF-81

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Ownable.sol

Description This event is emitted even if nothing actually changed.
Client Comment Transfer of 0 tokens also emits an emit even if nothing is transferred.

Listing 81:

```
43 emit OwnershipTransferred(owner, newOwner);
```

3.82 CVF-82

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** Address.sol

Recommendation This value should be a named constant initialized with a keccak256 expression.
Client Comment Considered for future improvement.

Listing 82:

```
30 bytes32 accountHash = 0
   ↪ xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d
   ↪
85a470;
```

3.83 CVF-83

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Address.sol

Description This wraps the "returndata" value into an ABI-encoded calldata.
Recommendation Consider reverting with raw "returndata" using an assembly block: assembly{ revert (add (returndata, 0x20), mload (returndata)) }

Listing 83:

```
43 revert(string(returndata));
```

3.84 CVF-84

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** MultiSigCloneFactory.sol

Recommendation This variable should be declared as immutable.

Listing 84:

```
13 address public multiSigImplementation;
```

3.85 CVF-85

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** MultiSigCloneFactory.sol

Recommendation Events are usually named via nouns, such as "ContractCreation" or "NewContract".
Client Comment Affects backend.

Listing 85:

```
15 event ContractCreated(address contractAddress, string typeName);
```

3.86 CVF-86

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** MultiSigCloneFactory.sol

Recommendation The "contractAddress" parameter should be indexed.

Listing 86:

```
15 event ContractCreated(address contractAddress, string typeName);
```

3.87 CVF-87

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** MultiSigCloneFactory.sol

Recommendation The "typeName" parameter should probably be indexed.

Listing 87:

```
15 event ContractCreated(address contractAddress, string typeName);
```

3.88 CVF-88

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** MultiSigCloneFactory.sol

Recommendation The "typeName" parameter is redundant as its value is always the same.
Client Comment Might use that event for different purposes in future.

Listing 88:

```
15 event ContractCreated(address contractAddress, string typeName);
```

3.89 CVF-89

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** MultiSigCloneFactory.sol

Recommendation The return type should be "MultiSigWallet".

Listing 89:

```
21 function predict(bytes32 salt) external view returns (address) {

25 function create(address owner, bytes32 salt) external returns (
   ↪ address) {
```

3.90 CVF-90

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** ERC20Allowlistable.sol

Description Calling the "setFlag" function three times in a row is suboptimal.
Recommendation Consider implementing an efficient function to batch-set multiple flags at once.
Client Comment Calling 'setFlag' more directly when context allows.

Listing 90:

```
83 setFlag(account, FLAG_INDEX_ALLOWLIST, typeNumber ==
   ↪ TYPE_ALLOWLISTED);
   setFlag(account, FLAG_INDEX_FORBIDDEN, typeNumber ==
   ↪ TYPE_FORBIDDEN);
   setFlag(account, FLAG_INDEX_POWERLIST, typeNumber ==
   ↪ TYPE_POWERLISTED);
```

3.91 CVF-91

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** ERC20Allowlistable.sol

Description Calling the "hasFlaginternal" function twice in a row is suboptimal.
Recommendation Consider implementing an efficient function to check several flags at once.
Client Comment Considered for future improvement.

Listing 91:

```
99 return hasFlagInternal(account, FLAG_INDEX_ALLOWLIST) ||
   ↪ hasFlagInternal(account, FLAG_INDEX_POWERLIST);
```

3.92 CVF-92

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** ERC20Allowlistable.sol

Description The "powerlisted" flag is checked twice, one inside the "canReceiveFromAnyone" function and another time inside the "isPowerlisted" function.
Recommendation Consider refactoring the code to check this flag only once.
Client Comment Note that it is checked for two different addresses.

Listing 92:

```
129 if (canReceiveFromAnyone(to)){

135   if (isPowerlisted(from)){
```

3.93 CVF-93

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** ERC20Allowlistable.sol

Description The "allowlisted" flag is checked twice: once inside the "canReceiveFromAnyone" function and another time directly.
Recommendation Consider refactoring the code to check this flag only once.
Client Comment Considered for future improvement.

Listing 93:

```
129 if (canReceiveFromAnyone(to)){

140   else if (hasFlagInternal(from, FLAG_INDEX_ALLOWLIST) ||
        ↪ isForbidden(from)){
```

3.94 CVF-94

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** ERC20Allowlistable.sol

Description The "forbidden" flag is checked twice: once inside the "isForbidden" function and another time directly.
Recommendation Consider refactoring the code to check this flag only once.
Client Comment Considered for future improvement.

Listing 94:

```
131 } else if (isForbidden(to)){

140   else if (hasFlagInternal(from, FLAG_INDEX_ALLOWLIST) ||
        ↪ isForbidden(from)){
```

3.95 CVF-95

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** ERC20Allowlistable.sol

Description So, by performing a transfer to a free address, a powerlisted address not only allows the destination address to receive transfers from anyone, but also forbids the destination address to perform transfers to other free addresses. What if the owner of the destination address doesn't want to loose the ability to perform transfers to free addresses?
Recommendation Consider allowing an address owner ot opt-out from automatic allowlisting.
Client Comment It is always possible to convert freely transferrable tokens into allowlisted tokens by sending them to an according address. Solution is to manually remove the flag again or to not use the powerlist functionality.

Listing 95:

```
136 // it is not allowlisted, but we can make it so
    setType(to, TYPE_ALLOWLISTED);
```

3.96 CVF-96

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** Offer.sol

Recommendation Events are usually named via nouns, such as "VotesChange" or just "Votes", "OfferCreation" or just "Offer", "OfferEnd" or just "End".
Client Comment Affects backend.

Listing 96:

```
57 event VotesChanged(uint256 newYesVotes, uint256 newNoVotes);
   event OfferCreated(address indexed buyer, address token, uint256
   ↪  pricePerShare, address currency);
   event OfferEnded(address indexed buyer, bool success, string
   ↪ message);
```

3.97 CVF-97

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** Offer.sol

Description The work "new" in the parameter named is redundant, as there are no "old" parameters.

Listing 97:

```
57 event VotesChanged(uint256 newYesVotes, uint256 newNoVotes);
```

3.98 CVF-98

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Offer.sol

Recommendation The parameters "token" and "currency" should be indexed.

Listing 98:

```
58 event OfferCreated(address indexed buyer, address token, uint256
   ↪  pricePerShare, address currency);
```

3.99 CVF-99

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Offer.sol

Description The parameter "success" should be indexed.
Recommendation Alternatively, consider defining separate events for successful and unsuccessful vote endings.

Listing 99:

```
59 event OfferEnded(address indexed buyer, bool success, string
   ↪ message);
```

3.100 CVF-100

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Offer.sol

Recommendation The type of this argument should be "IDraggable".

Listing 100:

```
64 address _token,
```

3.101 CVF-101

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** Offer.sol

Description There are now range checks for these arguments.
Recommendation Consider adding appropriate checks.
Client Comment Ok with us.

Listing 101:

```
65 uint256 _price,

67 uint256 _quorum,
   uint256 _votePeriod
```

3.102 CVF-102

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Offer.sol

Recommendation The type of this argument should be "IERC20".

Listing 102:

```
66 address _currency,
```

3.103 CVF-103

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Offer.sol

Description The conversions to "address" are redundant as "_token" and "_currency" are already "address".

Listing 103:

```
80 emit OfferCreated(_buyer, address(_token), _price, address(
   ↪ _currency));
```

3.104 CVF-104

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** Offer.sol

Description Hardcoding addresses is discouraged as it makes it harder to test contracts.
Recommendation Consider passing the license fee recipient address as a constructor argument and storing in an immutable variable.
Client Comment Considered for future improvement.

Listing 104:

```
83 payable(0x29Fe8914e76da5cE2d90De98a64d0055f199d06D).transfer(3
   ↪ ether);
```

3.105 CVF-105

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Offer.sol

Recommendation The argument type should be "IOffer".

Listing 105:

```
86 function makeCompetingOffer(address betterOffer) external
     ↪ override {
```

3.106 CVF-106

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Offer.sol

Recommendation The "30 days" value should be a named constant.

Listing 106:

```
98 return block.timestamp > voteEnd + 30 days; // buyer has thirty
     ↪ days to complete acquisition after voting ends
```

3.107 CVF-107

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Offer.sol

Description The "quorum" value is not necessary 75%.
Recommendation Consider not using a fixed number in the comments.

Listing 107:

```
138 // is it already clear that 75% will vote yes even though the
      ↪ vote is not over yet?

148 // is it already clear that 25% will vote no even though the
      ↪ vote is not over yet?
```

3.108 CVF-108

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Offer.sol

Recommendation The "10000" value should be a named constant.

Listing 108:

```
139 return yesVotes * 10000 >= quorum * IDraggable(token).
      ↪ totalVotingTokens();

142 return yesVotes * 10000 >= quorum * (yesVotes + noVotes);

150 return (supply - noVotes) * 10000 < quorum * supply;

153 return 10000 * yesVotes < quorum * (yesVotes + noVotes);
```

3.109 CVF-109

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Offer.sol

Description Converting "token" to "IDraggable" is redundant, as "token" is already "IDraggable".

Listing 109:

```
139     return yesVotes * 10000 >= quorum * IDraggable(token).
          ↪ totalVotingTokens();

221     IDraggable(token).notifyVoted(msg.sender);

223 update(previousVote, newVote, IDraggable(token).votingPower(msg.
      ↪ sender));

236 IDraggable(token).notifyOfferEnded();
```

3.110 CVF-110

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Offer.sol

Recommendation These lines could be simplified using the "+=" and "-=" operators.

Listing 110:

```
169 noVotes = noVotes - votes_;

171 yesVotes = yesVotes - votes_;

174 noVotes = noVotes + votes_;

176 yesVotes = yesVotes + votes_;
```

3.111 CVF-111

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** ERC20Draggable.sol

Recommendation The type of this variable could be more specific.
Client Comment No, could be any address.

Listing 111:

```
67 address private oracle;
```

3.112 CVF-112

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** ERC20Draggable.sol

Recommendation Events are usually named via nouns, such as "MigrationSuccess" or just "Migration".
Client Comment Affects backend.

Listing 112:

```
69 event MigrationSucceeded(address newContractAddress, uint256
   ↪ yesVotes, uint256 oracleVotes, uint256 totalVotingPower);
```

3.113 CVF-113

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** ERC20Draggable.sol

Recommendation The type of this argument should be "IERC20".

Listing 113:

```
72 address _wrappedToken,
```

3.114 CVF-114

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** ERC20Draggable.sol

Description There is no range check for this argument.
Recommendation Consider adding an appropriate check.
Client Comment Ok with us

Listing 114:

```
73 uint256 _quorum,
```

3.115 CVF-115

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** ERC20Draggable.sol

Recommendation The type of this argument should be "IOfferFactory".

Listing 115:

```
75 address _offerFactory,
```

3.116 CVF-116

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** ERC20Draggable.sol

Recommendation The type of this argument could be more specific.
Client Comment Can be oracle contract, multisig wallet or EOA (general is a trusted entity). See 112.

Listing 116:

```
76 address _oracle
```

3.117 CVF-117

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** ERC20Draggable.sol

Description The "wrapped.name()" expression is calculated in two places.
Recommendation Consider calculating at one place before the conditional operator.

Listing 117:

```
117 return string(abi.encodePacked(wrapped.name(), " SHA"));

119 return string(abi.encodePacked(wrapped.name(), " (Wrapped)"));
```

3.118 CVF-118

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** ERC20Draggable.sol

Recommendation It would be more efficiently to do: IShares(address(wrapped)).burn (isBinding() ? amount : amount * unwrapConversionFactor);

Listing 118:

```
159 uint256 factor = isBinding() ? 1 : unwrapConversionFactor;
160 IShares(address(wrapped)).burn(amount * factor);
```

3.119 CVF-119

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** ERC20Draggable.sol

Description There are no range checks for this argument.
Recommendation Consider adding appropriate checks.
Client Comment Ok with us.

Listing 119:

```
165 uint256 pricePerShare,
```

3.120 CVF-120

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** ERC20Draggable.sol

Recommendation The type of this argument should be "IERC20".

Listing 120:

```
166 address currency
```

3.121 CVF-121

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** ERC20Draggable.sol

Recommendation The type of the "newOffer" variable should be "IOffer".

Listing 121:

```
169 address newOffer = factory.create{value: msg.value}(
```

3.122 CVF-122

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** ERC20Draggable.sol

Recommendation The type of the "currency" argument should be "IERC20".

Listing 122:

```
178 function drag(address buyer, address currency) external override
    ↪ {
```

3.123 CVF-123

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** ERC20Draggable.sol

Description When this condition is false, the function silently does nothing.
Recommendation Consider reverting in such a case.

Listing 123:

```
185 if (msg.sender == address(offer)) {
```

3.124 CVF-124

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** ERC20Draggable.sol

Recommendation The type of the "newWrapped" argument should be "IERC20".

Listing 124:

```
190 function replaceWrapped(address newWrapped, address
    ↪ oldWrappedDestination) internal {
```

3.125 CVF-125

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** ERC20Draggable.sol

Description Here, the "wrapped" value is read from the storage just after being written into it.
Recommendation Consider reusing the just written value.

Listing 125:

```
196 deactivate(wrapped.balanceOf(address(this)) / totalSupply());
```

3.126 CVF-126

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** ERC20Draggable.sol

Description The arguments of these two functions have the same meaning but different names.
Recommendation Consider using the same names for the arguments with the same meaning.

Listing 126:

```
208 function migrateWithExternalApproval(address target, uint256
    ↪ externalSupportingVotes) external {

222 function migrate(address successor, uint256 additionalVotes)
    ↪ internal {
```

3.127 CVF-127

- **Severity** Major
- **Status** Fixed
- **Category** Flaw
- **Source** ERC20Draggable.sol

Recommendation Should be "<=".

Listing 127:

```
225 require(yesVotes < totalVotes, "votes");
```

3.128 CVF-128

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** ERC20Draggable.sol

Recommendation The denominator "10000" should be a named constant.

Listing 128:

```
227 require(yesVotes * 10000 >= totalVotes * quorum, "quorum");
```

3.129 CVF-129

- **Severity** Major
- **Status** Info
- **Category** Flaw
- **Source** ERC20Draggable.sol

Description This function could be called by anyone, thus a malicious actor may mark anybody as voted.
Client Comment Only thing an attacker can do is making the next transfer slightly more expensive as this is double-checked in the offer.

Listing 129:

```
244 function notifyVoted(address voter) external override {
```

3.130 CVF-130

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** ERC20Draggable.sol

Recommendation This interface should be moved to a separate file named "IShares.sol".

Listing 130:

```
265 abstract contract IShares {
```

3.131 CVF-131

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Brokerbot.sol

Recommendation The type of these variables should be "IERC20".

Listing 131:

```
38 address public immutable base;  // ERC-20 currency
   address public immutable token; // ERC-20 share token
```

3.132 CVF-132

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Brokerbot.sol

Description Hardcoded addresses make it harder to test contracts.
Recommendation Consider passing the fee recipient address as a constructor argument and storing in an immutable variable.

Listing 132:

```
41 address public constant COPYRIGHT = 0
     ↪ x29Fe8914e76da5cE2d90De98a64d0055f199d06D; // Aktionariat
     ↪ AG
```

3.133 CVF-133

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** Brokerbot.sol

Description The number format of this variable is unclear.
Recommendation Consider documenting.

Listing 133:

```
43 uint256 private price; // current offer price, without drift
```

3.134 CVF-134

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** Brokerbot.sol

Description The comment doesn't clarify anything.
Recommendation Consider either removing it or (preferably) making it more verbose.

Listing 134:

```
44 uint256 public increment; // increment
```

3.135 CVF-135

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Brokerbot.sol

Recommendation The "who" and "base" parameters should be indexed.

Listing 135:

```
62 event Trade(address indexed token, address who, bytes ref, int
   ↪ amount, address base, uint totPrice, uint fee, uint
   ↪ newprice);
```

3.136 CVF-136

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** Brokerbot.sol

Recommendation Events are usually named via nouns, such as "PaymentHubChange" or just "PaymentHub".
Client Comment Affects backend.

Listing 136:

```
63 event ChangePaymentHub(address newPaymentHub, address who);
```

3.137 CVF-137

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Brokerbot.sol

Recommendation The "newPaymentHub" parameter or even both parameters should be indexed.

Listing 137:

```
63 event ChangePaymentHub(address newPaymentHub, address who);
```

3.138 CVF-138

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Brokerbot.sol

Recommendation The type of the "token" and "base" parameters should be "IERC20".

Listing 138:

```
62 event Trade(address indexed token, address who, bytes ref, int
   ↪ amount, address base, uint totPrice, uint fee, uint
   ↪ newprice);
```

3.139 CVF-139

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Brokerbot.sol

Recommendation These arguments should have type "IERC20".

Listing 139:

```
66 address _shareToken,

69 address _baseCurrency,
```

3.140 CVF-140

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** Brokerbot.sol

Description The names of these arguments differ from the names of the corresponding storage variables.
Recommendation Consider using consistent naming.

Listing 140:

```
66 address _shareToken,

69 address _baseCurrency,
```

3.141 CVF-141

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Brokerbot.sol

Recommendation The type of these arguments should be "IERC20".

Listing 141:

```
66 address _shareToken,

69 address _baseCurrency,
```

3.142 CVF-142

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** Brokerbot.sol

Description There are no range checks for these arguments.
Recommendation Consider adding appropriate checks.
Client Comment Ok with us.

Listing 142:

```
67 uint256 _price,
   uint256 _increment,
```

3.143 CVF-143

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** Brokerbot.sol

Description The word "new" in the parameter name is redundant, as there are no other payment hub parameters.

Listing 143:

```
63 event ChangePaymentHub(address newPaymentHub, address who);
```

3.144 CVF-144

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** Brokerbot.sol

Description The word "new" in the argument names is redundant.

Listing 144:

```
82 function setPrice(uint256 newPrice, uint256 newIncrement)
     ↪ external onlyOwner {

92 function setDrift(uint256 secondsPerStep, int256
     ↪ newDriftIncrement) external onlyOwner {

293 function setEnabled(bool newBuyingEnabled, bool
     ↪ newSellingEnabled) external onlyOwner() {
```

3.145 CVF-145

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Brokerbot.sol

Recommendation These functions should emit some events.

Listing 145:

```
82  function setPrice(uint256 newPrice, uint256 newIncrement)
      ↪ external onlyOwner {

92  function setDrift(uint256 secondsPerStep, int256
      ↪ newDriftIncrement) external onlyOwner {

289 function setSettings(uint256 settings_) external onlyOwner() {

293 function setEnabled(bool newBuyingEnabled, bool
      ↪ newSellingEnabled) external onlyOwner() {
```

3.146 CVF-146

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** Brokerbot.sol

Description There are no range checks for the arguments of these functions.
Recommendation Consider adding appropriate checks.
Client Comment Ok with us.

Listing 146:

```
82  function setPrice(uint256 newPrice, uint256 newIncrement)
      ↪ external onlyOwner {

92  function setDrift(uint256 secondsPerStep, int256
      ↪ newDriftIncrement) external onlyOwner {

98  function anchorPrice(uint256 currentPrice) private {
```

3.147 CVF-147

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Brokerbot.sol

Description This formula makes the price to be a step function of time.
Recommendation It could be turned into a continuous function by calculating as: passed * driftIncrement / timeToDrift.
Client Comment Behavior is intended. Enables us to give users an exact quote that stays valid for some time.

Listing 147:

```
114 int256 drifted = int256(passed / timeToDrift) * driftIncrement;
```

3.148 CVF-148

- **Severity** Moderate
- **Status** Info
- **Category** Overflow/Underflow
- **Source** Brokerbot.sol

Description Overflow is possible during conversion to "int256".
Client Comment Numbers never get that high.

Listing 148:

```
114     int256 drifted = int256(passed / timeToDrift) *
          ↪ driftIncrement;
        int256 driftedPrice = int256(price) + drifted;

140 emit Trade(token, from, ref, int256(shares), base, costs, 0,
      ↪ getPrice());

231 emit Trade(token, recipient, ref, -int256(amount), base,
      ↪ totPrice, fee, getPrice());
```

3.149 CVF-149

- **Severity** Major
- **Status** Info
- **Category** Flaw
- **Source** Brokerbot.sol

Description The returned value is ignored.
Client Comment The brokerbot only supports tokens that revert on failure. Check is unnecessary. Added documentation.

Listing 149:

```
130     IERC20(base).transfer(from, paid - costs);

132 IERC20(token).transfer(from, shares);

150 IERC20(token).transfer(buyer, shares);

226     baseToken.transfer(COPYRIGHT, fee);

229     baseToken.transfer(recipient, totPrice - fee);

281 IERC20(ercAddress).transfer(to, amount);
```

3.150 CVF-150

- **Severity** Major
- **Status** Info
- **Category** Suboptimal
- **Source** Brokerbot.sol

Recommendation It would be more efficient to pass a single array of structs with three fields, rather than three parallel arrays. Such approach would also make length checks unnecessary.
Client Comment Considered for future improvement.

Listing 150:

```
153 function notifyTrades(address[] calldata buyers, uint256[]
    ↪ calldata shares, bytes[] calldata ref) external onlyOwner
    ↪ {

159 function notifyTradesAndTransfer(address[] calldata buyers,
    ↪ uint256[] calldata shares, bytes[] calldata ref) external
    ↪ onlyOwner {
```

3.151 CVF-151

- **Severity** Major
- **Status** Info
- **Category** Procedural
- **Source** Brokerbot.sol

Description There are no length checks for the passed arrays, so in case the "shares"' or "ref" array is longer than the "buyers" array, extra elements are silently ignored.
Recommendation Consider adding appropriate length check.
Client Comment Considered for future improvement.

Listing 151:

```
153 function notifyTrades(address[] calldata buyers, uint256[]
    ↪ calldata shares, bytes[] calldata ref) external onlyOwner
    ↪ {

159 function notifyTradesAndTransfer(address[] calldata buyers,
    ↪ uint256[] calldata shares, bytes[] calldata ref) external
    ↪ onlyOwner {
```

3.152 CVF-152

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Brokerbot.sol

Recommendation The type of the "_token" argument should be "IERC20".

Listing 152:

```
168 function processIncoming(address token_, address from, uint256
    ↪ amount, bytes calldata ref) public payable returns (
    ↪ uint256) {
```

3.153 CVF-153

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** Brokerbot.sol

Description Is it really possible that msg.sender == base but, at the same time, token_ == token? If this is not possible, then the second part of the condition ("msg.sender == base") is redundant.

Listing 153:

```
169 require(msg.sender == token_ || msg.sender == base || msg.sender
    ↪  == paymenthub, "invalid calle");
```

3.154 CVF-154

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Brokerbot.sol

Description This is equivalent to: revert ("invalid token");

Listing 154:

```
175 require(false, "invalid token");
```

3.155 CVF-155

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Brokerbot.sol

Description This code is unreachable.
Recommendation Consider removing it.

Listing 155:

```
176 return 0;
```

```
213     return true;
```

3.156 CVF-156

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Brokerbot.sol

Description These functions are redundant as the "settings" variable is public.

Listing 156:

```
191 function buyingEnabled() external view returns (bool){
```

```
195 function sellingEnabled() external view returns (bool){
```

3.157 CVF-157

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** Brokerbot.sol

Description The format of the “ref‘“ data is unclear.
Recommendation Consider documenting here or giving a reference to the documentation.

Listing 157:

```
203 function isDirectSale(bytes calldata ref) internal pure returns
    ↪ (bool) {
```

3.158 CVF-158

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Brokerbot.sol

Description This is equivalent to: revert ("unknown ret");

Listing 158:

```
212 require(false, "unknown ref");
```

3.159 CVF-159

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Brokerbot.sol

Recommendation The value “10000” should be a named constant.

Listing 159:

```
236 return totPrice * LICENSEFEEBPS / 10000;
```

3.160 CVF-160

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Brokerbot.sol

Recommendation This function could be simplified as: return (getPrice() - (shares + 1) * increment / 2) * shares;
Client Comment Prefer better code readability.

Listing 160:

```
239 function getSellPrice(uint256 shares) public view returns (
    ↪ uint256) {
```

3.161 CVF-161

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Brokerbot.sol

Recommendation This function could be simplified as: return shares == 0 ? 0 : (getPrice() + (shares - 1) * increment / 2) * shares;
Client Comment Prefer better code readability.

Listing 161:

```
243 e
```

3.162 CVF-162

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Brokerbot.sol

Recommendation This function could be simplified as: return shares == 0 ? 0 : (lowest + (shares - 1) * increment / 2) * shares;
Client Comment Prefer better code readability.

Listing 162:

```
247 function getPrice(uint256 lowest, uint256 shares) internal view
    ↪ returns (uint256){
```

3.163 CVF-163

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Brokerbot.sol

Description This function basically solves a quadratic equation via the bisection method, which is very inefficient.
Recommendation Consider solving analytically or using Newton method.
Client Comment Considered for future improvement.

Listing 163:

```
256 function getShares(uint256 money) public view returns (uint256)
    ↪ {
```

3.164 CVF-164

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Brokerbot.sol

Description Using the "transfer" function is discouraged as it allows the recipient to spend only a fixed amount of gas, but operation gas costs could change in the future.
Recommendation Consider using the "call" function instead.

Listing 164:

```
273 payable(msg.sender).transfer(amount); // return change
```

3.165 CVF-165

- **Severity** Moderate
- **Status** Info
- **Category** Flaw
- **Source** Brokerbot.sol

Description The returned value is ignored here.

Listing 165:

```
277 IERC20(erc20).approve(who, amount);
```

3.166 CVF-166

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** Brokerbot.sol

Description The error message is not accurate.
Recommendation Consider changing to "not authorized" or "not owner nor hub".

Listing 166:

```
303 require(owner == msg.sender || paymenthub == msg.sender, "not
    ↪ owner");
```

3.167 CVF-167

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RLPEncode.sol

Description The variable "m" is redundant.
Recommendation Consider using "inputBytes" instead.

Listing 167:

```
57 let m := mload(0x40)
```

3.168 CVF-168

- **Severity** Critical
- **Status** Fixed
- **Category** Flaw
- **Source** RLPEncode.sol

Description Free memory is not guaranteed to be zeroed.
Recommendation Consider explicitly setting the first 20 bytes of the newly allocated memory block to zero like this: bytes memory inputBytes; assembly { inputBytes := mload (0x40) mstore (0x40, add (inputBytes, 0x34)) mstore (add (inputBytes, 20), x) mstore (inputBytes, 0x14) }
Client Comment Removed method, was actually never used.

Listing 168:

```
58 mstore(add(m, 20), xor(0
    ↪ x1400000000000000000000000000000000000000, self))
```

3.169 CVF-169

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RLPEncode.sol

Description Using "xor" here is confusing.
Recommendation Consider using "or" instead.

Listing 169:

```
58 mstore(add(m, 20), xor(0
   ↪ x1400000000000000000000000000000000000000, self))
```

3.170 CVF-170

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** RLPEncode.sol

Recommendation "bytes1 (uint8 (len + offset))" would be more readable.
Client Comment Considered for future improvement.

Listing 170:

```
109 encoded[0] = bytes32(len + offset)[31];
```

```
119 encoded[0] = bytes32(lenLen + offset + 55)[31];
```

```
121     encoded[i] = bytes32((len / (256**(lenLen-i))) % 256)[31];
```

3.171 CVF-171

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** RLPEncode.sol

Description The code inside this branch is very inefficient.
Recommendation Consider using the appriach imilar to the one suggested for the "toBinary" function.

Listing 171:

```
110 } else {
```

3.172 CVF-172

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RLPEncode.sol

Recommendation "while (len >= i) {" would be more readable and more efficient.

Listing 172:

```
113 while (len / i != 0) {
```

3.173 CVF-173

- **Severity** Moderate
- **Status** Info
- **Category** Flaw
- **Source** RLPEncode.sol

Description This condition will eventually revert in case len >= 2^248.
Client Comment Will never be that long.

Listing 173:

```
113 while (len / i != 0) {
```

3.174 CVF-174

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RLPEncode.sol

Recommendation Shift would be more efficient.

Listing 174:

```
115 i *= 256;
```

3.175 CVF-175

- **Severity** Major
- **Status** Info
- **Category** Flaw
- **Source** RLPEncode.sol

Description RLP doesn't support lengths wider than 8 bytes, while this code tries to encode them.
Recommendation Consider adding an explicit "require" statement to ensure the "lenLen" doesn't exceed 8.
Client Comment If the len field cannot be encoded in 8 bytes, the length of the data is much larger than if would fit into an ethereum block.

Listing 175:

```
119 encoded[0] = bytes32(lenLen + offset + 55)[31];
```

3.176 CVF-176

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** RLPEncode.sol

Recommendation Shift and binary "and" would be more efficient than division and modulo operators.

Listing 176:

```
121 encoded[i] = bytes32((len / (256**(lenLen-i))) % 256)[31];
```

3.177 CVF-177

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RLPEncode.sol

Description This function is very inefficient.
Recommendation Consider using the following approach: function toBytes (uint256 x) public pure returns (bytes memory result) { uint l = 0; uint xx = x; if (x >= 0x100000000000000000000000000000000) { x >>= 128; l += 16; } if (x >= 0x10000000000000000) { x >>= 64; l += 8; } if (x >= 0x100000000) { x >>= 32; l += 4; } if (x >= 0x10000) { x >>= 16; l += 2; } if (x >= 0x100) { x >>= 8; l += 1; } if (x > 0x0) { l += 1; } assembly { result := mload (0x40) mstore (0x40, add (result, add (l, 0x20))) mstore (add (result, l), xx) mstore (result, l) } }
Client Comment Removed function.

Listing 177:

```
132 function toBinary(uint _x) private pure returns (bytes memory) {
```

3.178 CVF-178

- **Severity** Critical
- **Status** Fixed
- **Category** Flaw
- **Source** RLPEncode.sol

Description This line reverts in case "len" is zero, i.e. in case "_len" is a factor of 32.
Recommendation Consider changing to: uint mask = type(uint).max » (len « 3);

Listing 178:

```
172 uint mask = 256 ** (32 - len) - 1;
```

3.179 CVF-179

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RLPEncode.sol

Description The "list[i].length" value is accessible as "item.length".

Listing 179:

```
211 flattenedPtr += _list[i].length;
```

3.180 CVF-180

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RLPEncode.sol

Recommendation This function could be simplified as: return abi.encodePacked (_preBytes, _postBytes);

Listing 180:

```
224 function concat(bytes memory _preBytes, bytes memory _postBytes)
    ↪ private pure returns (bytes memory) {
```

3.181 CVF-181

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** ERC20Named.sol

Description In the arguments list of the constructor and the "setName" function, a symbol goes the first and a name goes the second, however in the event parameters list the order is opposite. This is inconsistent and error-prone.
Recommendation Consider using the same order of values everywhere.
Client Comment Affects backend.

Listing 181:

```
13 constructor(string memory _symbol, string memory _name, uint8
    ↪ _decimals, address _admin) ERC20Flaggable(_decimals)
    ↪ Ownable(_admin) {

17 function setName(string memory _symbol, string memory _name)
    ↪ external onlyOwner {

24     emit NameChanged(_name, _symbol);
```

3.182 CVF-182

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** IERC20.sol

Description ERC-20 doesn't define such event, so it shouldn't be defined in a file named "IERC20.sol".
Recommendation Consider moving elsewhere.

Listing 182:

```
21 event NameChanged(string name, string symbol);
```

3.183 CVF-183

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IERC20.sol

Recommendation Events are usually named via nouns, such as "NameChange" or just "Name".
Client Comment Affects backend.

Listing 183:

```
21 event NameChanged(string name, string symbol);
```

3.184 CVF-184

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IERC20.sol

Description ERC-20 doesn't assume token name and symbol to be mutable. Changing them could not be handled properly by most applications.
Recommendation Consider making the name and the symbol immutable.
Client Comment Being able to change the name of a company and therefore also its shares is a legal requirement.

Listing 184:

```
21 event NameChanged(string name, string symbol);
```

3.185 CVF-185

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IERC20.sol

Description Unlike names of function arguments, names of event parameters are a part of the public API, so using names different from what is defined in the standard could cause compatibility problems.
Recommendation Consider using standard names for event parameters.
Client Comment We use same naming as OpenZeppelin.

Listing 185:

```
86 event Transfer(address indexed from, address indexed to, uint256
   ↪  value);

92 event Approval(address indexed owner, address indexed spender,
   ↪ uint256 value);
```

3.186 CVF-186

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** ERC20Flaggable.sol

Description Actually, the highest 32 bits are used for flags, and lower 224 bits are used for balances.
Recommendation Consider fixing the comment.

Listing 186:

```
55 mapping (address => uint256) private _balances; // lower 32 bits
   ↪  reserved for flags
```

3.187 CVF-187

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** ERC20Flaggable.sol

Description The order of the keys in unclear.
Recommendation Consider documenting.
Client Comment Same as OpenZeppelin and Consensys implementation.

Listing 187:

```
57 mapping (address => mapping (address => uint256)) private
    ↪ _allowances;
```

3.188 CVF-188

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** ERC20Flaggable.sol

Recommendation This could be simplified as: return uint224 (_balances [account]);

Listing 188:

```
78 return _balances[account] & BALANCES_MASK;
```

3.189 CVF-189

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** ERC20Flaggable.sol

Description The semantics of the returned value is unclear from the function name and signature.
Recommendation Consider documenting.

Listing 189:

```
85 function setFlag(address account, uint8 index, bool value)
    ↪ internal returns (bool) {
```

3.190 CVF-190

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** ERC20Flaggable.sol

Description Both functions called here calculate the flag mask and read the "_balances[account]" value.
Recommendation Consider optimizing like this: uint256 flagMask = 1 « index + 224; uint256 balance = _balances [account]; if ((balance & flagMask == flagMask) != value) { _balances [account] = balance ^ flagMask; return true; } else { return false; }

Listing 190:

```
86 if (hasFlagInternal(account, index) != value){
       toggleFlag(account, index);
```

3.191 CVF-191

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** ERC20Flaggable.sol

Recommendation It would be more conventional to return the previous flag value instead of the flag telling whether the new value is different from the old value or not.

Listing 191:

```
88 return true;

90 return false;
```

3.192 CVF-192

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** ERC20Flaggable.sol

Description This condition make the implementation not 100% compatible with the ERC-20 standard.
Recommendation Consider removing it.
Client Comment Use 2^255 as infinit allowance as documented in infiniteallowance.md.

Listing 192:

```
151 if (currentAllowance < (1 << 255)){
```

3.193 CVF-193

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** ERC20Flaggable.sol

Description Here any allowance above 2^255 - 1 is considered "infinity". More common is to only consider as "infinite" the 2^256 -1 allowance.
Recommendation Consider following the common approach.
Client Comment Use 2^255 as infinit allowance as documented in infiniteallowance.md.

Listing 193:

```
151 if (currentAllowance < (1 << 255)){
```

3.194 CVF-194

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** ERC20Flaggable.sol

Description This call emits the "Approval" event, which is not 100% compliant with the ERC-20 standard, as according to the standard, this event is emitted from the "approve" function.
Recommendation Consider refactoring the code to not emitting the "Approval" events on transfers.

Listing 194:

```
154 _approve(sender, msg.sender, currentAllowance - amount);
```

3.195 CVF-195

- **Severity** Critical
- **Status** Fixed
- **Category** Flaw
- **Source** ERC20Flaggable.sol

Description There is no check that the sender's balance is sufficient, and compiler-enforced underflow check won't help here, as it will treat flags as a part of the balance. So in case the sender has insufficient balance and some flags set, the balance will underflow.
Recommendation Consider adding an explicit "require" statement to check that the sender's balance is sufficient.

Listing 195:

```
175 _balances[sender] -= amount;

230 _balances[account] -= amount;
```

3.196 CVF-196

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** ERC20Flaggable.sol

Recommendation This code could be simplified as: return transfer (recipient, amount) && IERC677Receiver (recipient). onTokenTransfer (msg.sender, amount, data);

Listing 196:

```
182 bool success = transfer(recipient, amount);
    if (success){
        success = IERC677Receiver(recipient).onTokenTransfer(msg.
            ↪ sender, amount, data);
    }
    return success;
```

3.197 CVF-197

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** ERC20Flaggable.sol

Recommendation This check could be simplified as: require ‘((oldBalance ^ newBalance) » 224 == 0, "overflow‘’);

Listing 197:

```
211 require(newSettings == oldSettings, "overflow");
```

3.198 CVF-198

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** OfferFactory.sol

Recommendation Events are usually named via nouns, such as “OfferCreation” or “NewOffer”, or just “Offer”.
Client Comment Affects backend.

Listing 198:

```
35 event OfferCreated(address contractAddress, string typeName);
```

3.199 CVF-199

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** OfferFactory.sol

Recommendation This event should be moved to the "IOfferFactory" interface.

Listing 199:

```
35 event OfferCreated(address contractAddress, string typeName);
```

3.200 CVF-200

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** OfferFactory.sol

Description The function name is too generic.
Recommendation Consider renaming to "predictOfferAddress" or just "getOfferAddress" as this function could be used to obtain the address of an already created order.

Listing 200:

```
38 function predict(bytes32 salt, address buyer, address token,
   ↪ uint256 pricePerShare, address currency, uint256 quorum,
   ↪ uint256 votePeriod) external view returns (address) {
```

3.201 CVF-201

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** OfferFactory.sol

Recommendation The type of the "token" argument should be "IDraggable". The type of the "currency" argument should be "IERC20".

Listing 201:

```
38 function predict(bytes32 salt, address buyer, address token,
   ↪ uint256 pricePerShare, address currency, uint256 quorum,
   ↪ uint256 votePeriod) external view returns (address) {
```

3.202 CVF-202

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** OfferFactory.sol

Recommendation The return type should be "IOffer"'.
Client Comment As in the writen in the recommendation of the naming of the function, this is only to get the address not an Offer contract.

Listing 202:

```
38 function predict(bytes32 salt, address buyer, address token,
     ↪ uint256 pricePerShare, address currency, uint256 quorum,
     ↪ uint256 votePeriod) external view returns (address) {

45 function create(bytes32 salt, address buyer, uint256
     ↪ pricePerShare, address currency, uint256 quorum, uint256
     ↪ votePeriod) override external payable returns (address) {
```

3.203 CVF-203

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** OfferFactory.sol

Description There are no range checks for the arguments.
Recommendation Consider adding appropriate checks.
Client Comment Ok with us.

Listing 203:

```
38 function predict(bytes32 salt, address buyer, address token,
     ↪ uint256 pricePerShare, address currency, uint256 quorum,
     ↪ uint256 votePeriod) external view returns (address) {

45 function create(bytes32 salt, address buyer, uint256
     ↪ pricePerShare, address currency, uint256 quorum, uint256
     ↪ votePeriod) override external payable returns (address) {
```

3.204 CVF-204

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** OfferFactory.sol

Description Deploying a new instance of the "Offer" contract is suboptimal.
Recommendation Consider deploying a minimal proxy as described in EIP-1167: https://eips.ethereum.org/EIPS/eip-1167
Client Comment Costs of the offer should be paid by whoever initiated the offer and not the everyday transfers. Costs of contract are amortized after a few hundred proxy calls.

Listing 204:

```
46 Offer offer = new Offer{value: msg.value, salt: salt}(buyer, msg
   ↪ .sender, pricePerShare, currency, quorum, votePeriod);
```

3.205 CVF-205

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** PaymentHub.sol

Recommendation The type of this variable should be IWETH9.
Client Comment It's also type address in 'PeripheryImmutableState' from Uniswap.

Listing 205:

```
46 address public immutable weth;
```

3.206 CVF-206

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** PaymentHub.sol

Recommendation The type of this variable should be "IERC20".

Listing 206:

```
47 address public immutable currency;
```

3.207 CVF-207

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** PaymentHub.sol

Description Hardcoding addresses makes it harder to test contracts.
Recommendation Consider passing the addresses as constructor arguments and storing in immutable variables.

Listing 207:

```
49 IQuoter private constant UNISWAP_QUOTER = IQuoter(0
    ↪ xb27308f9F90D607463bb33eA1BeBb41C27CE5AB6);
50 ISwapRouter private constant UNISWAP_ROUTER = ISwapRouter(0
    ↪ xE592427A0AEce92De3Edee1F18E0157C05861564);
```

3.208 CVF-208

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** PaymentHub.sol

Recommendation The type of the "_aggregatorCHFUSD" and "_aggregatorETHUSD" arguments should be "AggregatorV3Interface".

Listing 208:

```
54 constructor(address _currency, address _aggregatorCHFUSD,
    ↪ address _aggregatorETHUSD) {
```

3.209 CVF-209

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** PaymentHub.sol

Description The number format of the returned value is unclear.
Recommendation Consider documenting.

Listing 209:

```
62  function getPriceInEther(uint256 amountOfXCHF) external returns
      ↪ (uint256) {

71  function getPriceInEther(uint256 amountOfXCHF, address brokerBot
      ↪ ) public returns (uint256) {

82  function getPriceInEtherFromOracle(uint256 amountOfXCHF) public
      ↪ view returns (uint256) {

89  function getPriceInUSD(uint256 amountOfCHF) public view returns
      ↪ (uint256) {

96  function getLatestPriceETHUSD() public view returns (int256) {

104 function getLatestPriceCHFUSD() public view returns (int) {
```

3.210 CVF-210

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** PaymentHub.sol

Recommendation The value "3000" should be a named constant.

Listing 210:

```
75  return UNISWAP_QUOTER.quoteExactOutputSingle(weth, currency,
      ↪ 3000, amountOfXCHF, 0);

116 weth, currency, 3000, recipient, block.timestamp, xchfamount,
      ↪ msg.value, 0);
```

3.211 CVF-211

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** PaymentHub.sol

Recommendation The fee value "3000" would be more readable when rendered as "0.003e6".
Client Comment Considered for future improvement.

Listing 211:

```
75  return UNISWAP_QUOTER.quoteExactOutputSingle(weth, currency,
        ↪ 3000, amountOfXCHF, 0);

116 weth, currency, 3000, recipient, block.timestamp, xchfamount,
        ↪ msg.value, 0);
```

3.212 CVF-212

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** PaymentHub.sol

Recommendation The denominator "10**8" should be a named constant.

Listing 212:

```
83  return getPriceInUSD(amountOfXCHF) * 10**8 / uint256(
        ↪ getLatestPriceETHUSD());

90  return (uint256(getLatestPriceCHFUSD()) * amountOfCHF) / 10**8;
```

3.213 CVF-213

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** PaymentHub.sol

Recommendation The denominator "10**8" could be rendered as "1e8".

Listing 213:

```
83  return getPriceInUSD(amountOfXCHF) * 10**8 / uint256(
        ↪ getLatestPriceETHUSD());

90  return (uint256(getLatestPriceCHFUSD()) * amountOfCHF) / 10**8;
```

3.214 CVF-214

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** PaymentHub.sol

Recommendation This expression effectively calculates the following formula: uint256(getLatestPriceCHFUSD()) * amountOfXCHF / 10**8 * 10**8 / uint256(getLatestPriceETHUSD()) which could be simplified as: uint256(getLatestPriceCHFUSD()) * amountOfXCHF / uint256(getLatestPriceETHUSD())

Listing 214:

```
83 return getPriceInUSD(amountOfXCHF) * 10**8 / uint256(
   ↪ getLatestPriceETHUSD());
```

3.215 CVF-215

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** PaymentHub.sol

Description Outer brackets are redundant.

Listing 215:

```
90 return (uint256(getLatestPriceCHFUSD()) * amountOfCHF) / 10**8;
```

3.216 CVF-216

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** PaymentHub.sol

Description Using the "int256" type together with its alias "int" makes the code harder to read.
Recommendation Consider using consistent type naming.

Listing 216:

```
96 function getLatestPriceETHUSD() public view returns (int256) {

104 function getLatestPriceCHFUSD() public view returns (int) {
```

3.217 CVF-217

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** PaymentHub.sol

Description Using the "transfer" function is discouraged as it allows only a fixed amount of gas to be spend by the callee, while operation gas costs may change in the future.
Recommendation Consider using the "call" function instead.

Listing 217:

```
125 payable(msg.sender).transfer(msg.value - amountIn); // return
    ↪ change

167     payable(msg.sender).transfer(msg.value - priceInEther); //
        ↪ return change
```

3.218 CVF-218

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** PaymentHub.sol

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays. This approach would also make the length check unnecessary.
Client Comment Maybe in future versions.

Listing 218:

```
129 function multiPay(address[] calldata recipients, uint256[]
    ↪ calldata amounts) external {

133 function multiPay(address token, address[] calldata recipients,
    ↪ uint256[] calldata amounts) public {

142 function multiPayAndNotify(address token, address[] calldata
    ↪ recipients, uint256[] calldata amounts, bytes calldata ref
    ↪ ) external {
```

3.219 CVF-219

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** PaymentHub.sol

Description There are no length checks for the array arguments, so in case the "amounts" array is longer than the "recipients" array, the extra elements will be silently ignored.
Recommendation Consider adding appropriate length check.
Client Comment Maybe in future versions.

Listing 219:

```
129 function multiPay(address[] calldata recipients, uint256[]
      ↪ calldata amounts) external {

133 function multiPay(address token, address[] calldata recipients,
      ↪ uint256[] calldata amounts) public {

142 function multiPayAndNotify(address token, address[] calldata
      ↪ recipients, uint256[] calldata amounts, bytes calldata ref
      ↪ ) external {
```

3.220 CVF-220

- **Severity** Major
- **Status** Fixed
- **Category** Flaw
- **Source** PaymentHub.sol

Description Returned value is ignored.
Client Comment Generally, we only work with tokens that revert on errors.

Listing 220:

```
135     IERC20(token).transferFrom(msg.sender, recipients[i],
          ↪ amounts[i]);

155 IERC20(token).transferFrom(msg.sender, recipient, amount);

188 IERC20(ercAddress).transfer(to, amount);
```

3.221 CVF-221

- **Severity** Major
- **Status** Fixed
- **Category** Bad datatype
- **Source** PaymentHub.sol

Recommendation There should be a constant for the "keep ether' setting bit, and this constant should be defined in the "'iBrokerbot" interface.

Listing 221:

```
180 return IBrokerbot(recipient).settings() & 0x4 == 0x4;
```

3.222 CVF-222

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** PaymentHub.sol

Recommendation This function is weird, as it allows anyone to take tokens sent to this contract. Common way to recover accidentally sent tokens is to allow the contract owner to sent them to somebody.
Client Comment There is no owner.

Listing 222:

```
187 function recover(address ercAddress, address to, uint256 amount)
    ↪ external {
```

3.223 CVF-223

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** PaymentHub.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

Listing 223:

```
193 receive() external payable {}
```

3.224 CVF-224

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** IBrokerbot.sol

Recommendation The constants describing the meaning of particular settings bits should be moved in this interface.

Listing 224:

```
4 interface IBrokerbot {

6   function settings() external view returns (uint256);
```

3.225 CVF-225

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** IBrokerbot.sol

Recommendation The type of the "token_" argument should be "IERC20".

Listing 225:

```
8 function processIncoming(address token_, address from, uint256
    ↪ amount, bytes calldata ref) external payable returns (
    ↪ uint256);
```

3.226 CVF-226

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IBrokerbot.sol

Description The semantics of the returned value is unclear.
Recommendation Consider documenting.

Listing 226:

```
8 function processIncoming(address token_, address from, uint256
    ↪ amount, bytes calldata ref) external payable returns (
    ↪ uint256);
```

3.227 CVF-227

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** IRecoveryHub.sol

Description The "I" prefix is commonly used for interface names, while here it is used for an abstract contract.
Recommendation Consider turning this abstract contract into an interface.

Listing 227:

```
4 abstract contract IRecoveryHub {
```

3.228 CVF-228

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IRecoveryHub.sol

Description The difference between these two functions is unclear.
Recommendation Consider documenting.

Listing 228:

```
8  function deleteClaim(address target) external virtual;

10 function clearClaimFromToken(address holder) external virtual;
```

3.229 CVF-229

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** IOfferFactory.sol

Description The "I" prefix is commonly used for interface names, while here it is used for the name of an abstract contract.
Recommendation Consider turning this abstract contract into an interface.

Listing 229:

```
4 abstract contract IOfferFactory {
```

3.230 CVF-230

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** IOfferFactory.sol

Recommendation The type of the "currency" argument should be "IERC20".

Listing 230:

```
6 bytes32 salt, address buyer, uint256 pricePerShare, address
    ↪ currency, uint256 quorum, uint256 votePeriod
```

3.231 CVF-231

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** IOfferFactory.sol

Recommendation The return type should be "IOffer".

Listing 231:

```
7 ) external payable virtual returns (address);
```

3.232 CVF-232

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** IOffer.sol

Description The "I" prefix is commonly used for interface names, while here it is used for the name of an abstract contract.
Recommendation Consider turning this abstract contract into an interface.

Listing 232:

```
4 abstract contract IOffer {
```

3.233 CVF-233

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** IOffer.sol

Recommendation The type of the "newOffer" argument should be "IOffer".

Listing 233:

```
5 function makeCompetingOffer(address newOffer) external virtual;
```

3.234 CVF-234

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IOffer.sol

Description The semantics of this function is unclear.
Recommendation Consider documenting.

Listing 234:

```
7 function notifyMoved(address from, address to, uint256 value)
    ↪ external virtual;
```

3.235 CVF-235

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** IDraggable.sol

Description The "I" prefix is commonly used for interface names, while here is it used for the name of an abstract contract.
Recommendation Consider turning this abstract contract into an interface for consistency.

Listing 235:

```
4 abstract contract IDraggable {
```

3.236 CVF-236

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** IDraggable.sol

Recommendation The return type could be more specific.
Client Comment No, could be any address.

Listing 236:

```
6 function getOracle() public virtual returns (address);
```

3.237 CVF-237

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** IDraggable.sol

Recommendation The type of the "currency" argument should be "IERC20".

Listing 237:

```
7 function drag(address buyer, address currency) external virtual;
```

3.238 CVF-238

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** IDraggable.sol

Description Some functions are defined as "public" while other declared as "external" without obvious reason.
Recommendation Consider using consistent access modifiers.

Listing 238:

```
6 function getOracle() public virtual returns (address);
  function drag(address buyer, address currency) external virtual;
```

3.239 CVF-239

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** IUniswapV3.sol

Recommendation This interface should be defined in a separate file named "IQuoter.sol".
Client Comment Kept for simplifaction.

Listing 239:

```
5 interface IQuoter {
```

3.240 CVF-240

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** IUniswapV3.sol

Description These interfaces are simplified versions of the original interfaces defined in Uniswap repo.
Recommendation Consider using the original versions as is for consistency and to reduce the code base.
Client Comment Kept for simplifaction.

Listing 240:

```
5  interface IQuoter {

18 interface ISwapRouter {
```

3.241 CVF-241

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** IUniswapV3.sol

Recommendation This interface should be defined in a separate file named "ISwapRouter.sol".
Client Comment Kept for simplifaction.

Listing 241:

```
18 interface ISwapRouter {
```

3.242 CVF-242

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** IUniswapV3.sol

Description In the Uniswap code this function is defined in the "IPeripheryPayments" interface, rather than in "ISwapRouter".
Recommendation Consider replicating this structure here as well, or just using the original Uniswap interfaces for consistency.
Client Comment Kept for simplifaction.

Listing 242:

```
32 function refundETH() external payable;
```