Report

v. 1.0

Customer

Uniswap

Smart Contract Audit

Permit 2

16th November 2022

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	16.11.22	A. Zveryanskaya	Initial Draft
0.2	16.11.22	A. Zveryanskaya	Minor revision
1.0	16.11.22	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) 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.

The Uniswap Protocol is an open-source protocol for providing liquidity and trading ERC20 tokens on Ethereum. It eliminates trusted intermediaries and unnecessary forms of rent extraction, allowing for safe, accessible, and efficient exchange activity. The protocol is non-upgradable and designed to be censorship resistant.

3 Project scope

We were asked to review:

- Original Repository
- Fix Repository

Files:

/			
	AllowanceTransfer.sol	EIP712.sol	Permit2.sol
	PermitErrors.sol	SignatureTransfer.sol	
interfaces/			
	IAllowanceTransfer.sol	IDAIPermit.sol	IERC1271.sol
	ISignatureTransfer.sol		
libraries/			
	Allowance.sol	Permit2Lib.sol	PermitHash.sol
	SignatureVerification.sol		

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code 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 as well as 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 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 if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure 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.

5 Our findings

We found 1 critical, 12 major, and a few less important issues. All identified Critical issues have been fixed. We have checked that the Major issues that have not been fixed, do not pose a threat (see client's comments in each case)

Issues

Active
0

Severity
Critical

Fixed
1

Severity	Active	Fixed
Major	2	10
Moderate	0	7
Minor	9	30

Fixed 48 out of 59 issues

6 Critical Issues

CVF-26. FIXED

- **Category** Overflow/Underflow
- **Source** Permit2Lib.sol

Description Unchecked overflow is possible when converting “amount” to “uint160”.

Recommendation Consider using a checked conversion.

Client Comment *This is patched. We now safecast in the Permit2Lib.*

```
if (!success) PERMIT2.transferFrom(address(token), from, to, uint160
    ↪ (amount));
```

```
            amount: uint160(amount),
```

7 Major Issues

CVF-3. FIXED

- **Category** Documentation
- **Source** SignatureTransfer.sol

Description The logic, that sends funds to the spender in case the destination address is zero, is not documented. This logic looks like an unnecessary overcomplication.

Recommendation Consider either documenting this logic clearly or removing it.

Client Comment *This logic has been removed.*

```
address recipient = to == address(0) ? permit.spender : to;
```

CVF-4. FIXED

- **Category** Unclear behavior
- **Source** SignatureTransfer.sol

Description The logic of handling zero destination address is different for single and batch transfers.

Recommendation Consider applying the same logic for all transfers.

Client Comment *This logic has been removed.*

```
address recipient = to == address(0) ? permit.spender : to;
```

```
        ERC20(permit.tokens[i]).safeTransferFrom(owner,
           ↪ toAmountPairs[i].to, requestedAmount);
```

CVF-10. FIXED

- **Category** Flaw
- **Source** PermitHash.sol

Description A separator string between the witmess type name and the witness type doesn't guarantee unambiguity, as a witness type name may contain " witness)" as a substring.

Recommendation Consider ensuring that "witnesTypeName" contains only alphanumeric characters.

Client Comment *This is now one parameter.*

```
abi.encodePacked(_PERMIT_TRANSFER_FROM_WITNESS_TYPEHASH_STUB,
   ↪ witnessTypeName, " witness)", witnessType)
```

```
    _PERMIT_BATCH_WITNESS_TRANSFER_FROM_TYPEHASH_STUB,
       ↪ witnessTypeName, " witness)", witnessType
```

CVF-13. FIXED

- **Category** Unclear behavior
- **Source** AllowanceTransfer.sol

Description This will throw in case the batch is empty.

Recommendation Consider doing nothing or reverting with a meaningful error in such a case.

Client Comment *Removed the logic where we only check one nonce in the batched case so this should be resolved.*

```
PackedAllowance storage allowed = allowance[owner][permitData.tokens
   ↪ [0]][permitData.spender];
```

CVF-17. FIXED

- **Category** Documentation
- **Source** AllowanceTransfer.sol

Description The logic that treats type(uint160).max allowance and unlimited is not documented and looks like an unnecessary overcomplication.

Recommendation Consider either clearly documenting this logic or removing it.

Client Comment *Resolved.*

```
if (maxAmount != type(uint160).max) {
```

CVF-19. FIXED

- **Category** Suboptimal
- **Source** AllowanceTransfer.sol

Description The expression "allowances[msg.sender]" is calculated on every loop iteration.

Recommendation Consider calculating once before the loop.

Client Comment *Resolved by saving the msg.sender value outside of the loop.*

```
allowance[msg.sender][approvals[i].token][approvals[i].spender].
   ↪ amount = 0;
```

CVF-21. INFO

- **Category** Suboptimal
- **Source** EIP712.sol

Description The contract is named "EIP712" while the hash is calculated from the name "Permit2", which make the implementation non flexible.

Recommendation Consider passing the real contract's name as a constructor argument, hashing it in the constructor and storing in an immutable variable.

Client Comment *Opted to not make this change as it is less readable. This change would require having abstract functions in both allowance and signature transfer.*

```
bytes32 private constant _HASHED_NAME = keccak256("Permit2");
```

CVF-24. INFO

- **Category** Suboptimal
- **Source** Permit2Lib.sol

Description Hardcoding mainnet addresses is a bad practice, as it makes it harder to test the code.

Recommendation Consider passing the Permit2 address as an argument to each function that needs it. Alternatively, turn this library into an extract contract, accept the Permit2 address as a constructor argument and store it in an internal variable.

Client Comment *Opted to not make this change as it would either be more difficult from an integrators perspective or require turning the lib into a contract.*

```
22 Permit2 internal constant PERMIT2 = Permit2(address(0
   ↪ xCe71065D4017F316EC606Fe4422e11eB2c47c246)); // TODO
```

CVF-27. FIXED

- **Category** Flaw
- **Source** Permit2Lib.sol

Description Any return data size other than 32 should be treated as unsuccess, as it breakes the function contract.

```
84 and(iszero(iszero(mload(0))), gt(returndatasize(), 31)),
```

CVF-35. FIXED

- **Category** Suboptimal
- **Source** ISignatureTransfer.sol

Description These fields are redundant as their values always equal to “msg.sender”.

```
21 address spender;
```

```
41 address spender;
```

CVF-43. FIXED

- **Category** Suboptimal
- **Source** ISignatureTransfer.sol

Description Only a destination address and an amount are specified for each transfer, so the corresponding token address is supposed to be taken from the permit batch. This assume a separate permit item for each transfer, which is suboptimal in case amounts of the same token are transferred to several recipients.

Recommendation Consider passing a separate “tokens” argument, to allow the number of permit items and the number of transfer to differ.

Client Comment *You are right in that we did not edit this to allow for different number of tokens specified. And you are correct that we do not handle the case where 1 permit on a token can release N transfers. It is still 1 to 1. This fix makes it possible to essentially not do a transfer on a token that was permitted if the spender does not need the permitted token, meaning that the number of transfers is allowed to differ from the number of permits BUT an owner still must permit a token N times if the spender wants to do N transfers of that token. In this case, if we allowed this, we would have to check at each iteration of the tokens array that that token address exists in the permitted tokens array and I’m not sure there is an efficient way to do that. So I think we opted for maybe less optimal calldata specification for more optimal runtime/gas.*

```
ToAmountPair[] calldata ToAmountPairs,
```

```
ToAmountPair[] calldata ToAmountPairs,
```

CVF-50. FIXED

- **Category** Suboptimal
- **Source** IAllowanceTransfer.sol

Recommendation The “token” and “spender” parameters should be indexed.

```
event InvalidateNonces(address indexed owner, uint32 indexed toNonce, address token, address spender);
```

8 Moderate Issues

CVF-9. FIXED

- **Category** Suboptimal
- **Source** PermitHash.sol

Description The ‘encodePacked‘ function is not injective and thus may produce colliding outputs for distinct inputs, thus yielding a hash collision.

Recommendation Consider using regular ‘encode‘ here.

Client Comment *encodePacked is standard for EIP712 but this was updated with using nested structs in the typehash.*

49
```
keccak256(abi.encodePacked(permit.tokens)),
```

51
```
keccak256(abi.encodePacked(permit.amounts)),
keccak256(abi.encodePacked(permit.expirations)),
```

76
```
keccak256(abi.encodePacked(permit.tokens)),
```

78
```
keccak256(abi.encodePacked(permit.signedAmounts)),
```

117
```
keccak256(abi.encodePacked(permit.tokens)),
```

119
```
keccak256(abi.encodePacked(permit.signedAmounts)),
```

CVF-12. FIXED

- **Category** Unclear behavior
- **Source** AllowanceTransfer.sol

Description There is no check to ensure that “permitData.tokens”, “permitData.amounts”, and “permitData.expirations” arrays are of the same length. In case, the other arrays are longer than “permitData.tokens” the remaining elements are silently ignored.

Recommendation Consider adding appropriate checks.

Client Comment *This is resolved as we move to nested structs.*

46
```
function permitBatch(address owner, PermitBatch calldata permitData,
   ↪  bytes calldata signature) external {
```

CVF-14. FIXED

- **Category** Unclear behavior
- **Source** AllowanceTransfer.sol

Description The nonce is validated and updated only for the first permit in a batch. This is error-prone and doesn't guarantee transaction ordering. As the corresponding mapping slots for all the permits are anyway read and written, it wouldn't consume much additional gas to check and update nonces for all the permits.

Recommendation Consider checking that the nonces of all the affected permits don't exceed the passed nonce, and at least for one permit the nonce equals to the passed nonce. Then set the nonces for all the affected permits to the passed nonce plus one. This would make the logic much more reasonable.

Client Comment *This is resolved by using nested structs. We now require a nonce to be passed for every batch permitted token.*

```
_validatePermit(allowed.nonce, permitData.nonce, permitData.
   ↪ sigDeadline);
```

```
        allowed.updateAmountAndExpiration(permitData.amounts[i],
           ↪ permitData.expirations[i]);
```

CVF-15. FIXED

- **Category** Flaw
- **Source** AllowanceTransfer.sol

Description The "Approval" event is not emitted for the first permit in a batch.

```
emit Approval(owner, permitData.tokens[i], permitData.spender,
   ↪ permitData.amounts[i]);
```

CVF-20. FIXED

- **Category** Overflow/Underflow
- **Source** AllowanceTransfer.sol

Description Overflow is indeed possible. One just needs to invalidate 65535 nonces 65538 times.

Recommendation Consider using a checked addition here.

Client Comment *Resolved as we no longer add an amount. The argument is now the new desired nonce. We still limit the number of nonces that can be invalidated.*

```
127 // Overflow is impossible on human timescales.
    newNonce = allowance[msg.sender][token][spender].nonce +=
        ↪ amountToInvalidate;
```

CVF-30. FIXED

- **Category** Overflow/Underflow
- **Source** Allowance.sol

Description Overflow is possible here.

Recommendation Consider using a checked addition.

Client Comment *Resolved. We opted to make nonces wider, so it would take a very large amount of transactions to overflow.*

```
18 storedNonce = nonce + 1;
```

CVF-60. FIXED

- **Category** Suboptimal
- **Source** IAllowanceTransfer.sol

Description The particular set of nonces invalidated by a call to this function depends on the current nonce, which makes front-run attacks possible.

Recommendation Consider explicitly specifying the nonce to invalidate up to.

```
121 function invalidateNonces(address token, address spender, uint32
        ↪ amountToInvalidate)
```

9 Minor Issues

CVF-1. FIXED

- **Category** Procedural
- **Source** SignatureTransfer.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as “^0.8.0”. Also relevant for: PermitHash.sol, Permit2.sol, AllowanceTransfer.sol, EIP712.sol, PermitErrors.sol, Permit2Lib.sol, SignatureVerification.sol, Allowance.sol, IERC1271.sol, IDAIPermit.sol, ISignatureTransfer.sol, IAllowanceTransfer.sol.

Client Comment *Added carets to interfaces and libraries.*

```
pragma solidity 0.8.17;
```

CVF-2. FIXED

- **Category** Documentation
- **Source** SignatureTransfer.sol

Description The semantics of the keys and values in this mapping is unclear.

Recommendation Consider documenting.

```
mapping(address => mapping(uint256 => uint256)) public nonceBitmap;
```

CVF-5. FIXED

- **Category** Suboptimal
- **Source** SignatureTransfer.sol

Description In solidity, types narrower than 256 bits are less efficient in many cases than 256-bit types.

Recommendation Consider using “uint256” for the returned values.

```
function bitmapPositions(uint256 nonce) private pure returns (
    ↪ uint248 wordPos, uint8 bitPos) {
```

ABDK

CVF-6. FIXED

- **Category** Suboptimal
- **Source** SignatureTransfer.sol

Recommendation The bitwise AND operator here is redundant, as conversion to “uint8” would drop the digher bits..

```
bitPos = uint8(nonce & 255);
```

CVF-7. FIXED

- **Category** Suboptimal
- **Source** SignatureTransfer.sol

Description The mapping slot address is calculated twice.

Recommendation Consider calculating once.

```
uint256 bitmap = nonceBitmap[from][wordPos];
```

```
nonceBitmap[from][wordPos] = bitmap | (1 << bitPos);
```

CVF-8. FIXED

- **Category** Suboptimal
- **Source** SignatureTransfer.sol

Recommendation This code could be optimized as: uint256 bit = 1 « bitPos; uint256 flipped = nonceBitmap[from][wordPos] ^= bit; if (flipped & bit == 0) revert InvalidNonce();

```
uint256 bitmap = nonceBitmap[from][wordPos];
```

```
if ((bitmap >> bitPos) & 1 == 1) revert InvalidNonce();
```

```
nonceBitmap[from][wordPos] = bitmap | (1 << bitPos);
```

CVF-11. FIXED

- **Category** Documentation
- **Source** Permit2.sol

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

```
contract Permit2 is SignatureTransfer, AllowanceTransfer {}
```

CVF-16. FIXED

- **Category** Suboptimal
- **Source** AllowanceTransfer.sol

Description The “allowed.amount” value is read from the storage twice.

Recommendation Consider reusing the already read value.

```
uint256 maxAmount = allowed.amount;
```

```
            allowed.amount -= amount;
```

CVF-18. FIXED

- **Category** Unclear behavior
- **Source** AllowanceTransfer.sol

Description This function should emit some event.

Client Comment *Resolved, a lockdown event was added.*

```
function lockdown(TokenSpenderPair[] calldata approvals) external {
```

CVF-22. FIXED

- **Category** Suboptimal
- **Source** PermitErrors.sol

Recommendation These errors could be made more useful by adding some parameters into them, such as the signature expiration time, expected nonce etc.

Client Comment *Somewhat resolved as we opted to use errors with params where we thought it might be more helpful.*

```
error SignatureExpired();
error InvalidNonce();
```

CVF-23. INFO

- **Category** Suboptimal
- **Source** Permit2Lib.sol

Description Solidity compiler is smart e enough to calculate constant hash expressions at compile time.

Recommendation Consider using a hash expression instead of a hardcoded hash value.

Client Comment *Opted to not make this change.*

```
bytes32 internal constant DAI_DOMAIN_SEPARATOR = 0
    ↪ xdbb8cf42e1ecb028be3f3dbc922e1d878b963f411dc388ced501601c60f7c6f7
    ↪ ;
```

CVF-28. FIXED

- **Category** Suboptimal
- **Source** SignatureVerification.sol

Recommendation This error could be made more useful by adding the actual recovered signer into it as a parameter.

Client Comment *Resolved, some errors with parameters were edited.*

```
error InvalidSigner();
```

CVF-29. FIXED

- **Category** Unclear behavior
- **Source** SignatureVerification.sol

Description There is no signature length check.

Recommendation Consider adding an explicit check to ensure that the signature length is 65 bytes.

Client Comment *Resolved, and we now support compact signatures.*

```
uint8 v = uint8(signature[64]);
```

CVF-31. INFO

- **Category** Suboptimal
- **Source** Allowance.sol

Description This code could read and write the same storage slot twice.

Recommendation Consider reading it once in an assembly block, updating and writing back.

Client Comment *Opted to not make this change.*

```
allowed.expiration = expiration == 0 ? uint64(block.timestamp) : expiration;
allowed.amount = amount;
```

CVF-32. FIXED

- **Category** Bad naming
- **Source** IERC1271.sol

Description This interface should contain the magic value constant.

Client Comment *Resolved through documentation.*

```
interface IERC1271 {
```

CVF-33. FIXED

- **Category** Documentation
- **Source** IERC1271.sol

Description The returned value is not documented.

Recommendation Consider documenting.

Client Comment *Resolved through documentation.*

```
8 function isValidSignature(bytes32 hash, bytes memory signature)
    ↪ external view returns (bytes4 magicValue);
```

CVF-34. INFO

- **Category** Bad datatype
- **Source** ISignatureTransfer.sol

Recommendation The type of this field should be “IERC20”.

```
19 address token;
```

CVF-36. INFO

- **Category** Bad datatype
- **Source** ISignatureTransfer.sol

Recommendation The type of this field should be “IERC20[]”.

```
39 address[] tokens;
```

CVF-37. FIXED

- **Category** Suboptimal
- **Source** ISignatureTransfer.sol

Recommendation It would be more efficient to use a single array of structs with two fields, rather than two parallel arrays. This would also make a length check unnecessary.

```
39 address[] tokens;
```

```
43 uint256[] signedAmounts;
```

CVF-38. FIXED

- **Category** Documentation
- **Source** ISignatureTransfer.sol

Description This comment doesn't explain what this function actually does.

Recommendation Consider adding more details.

Client Comment *Resolved with documentation*

```
/// @notice A bitmap used for replay protection
```

CVF-39. FIXED

- **Category** Bad naming
- **Source** ISignatureTransfer.sol

Description The semantics of the arguments and the returned value are unclear.

Recommendation Consider giving descriptive names to the arguments and the returned value and/or describing in the documentation comment.

Client Comment *Resolved with documentation*

```
function nonceBitmap(address, uint256) external returns (uint256);
```

CVF-40. FIXED

- **Category** Suboptimal
- **Source** ISignatureTransfer.sol

Recommendation This function should be declared as "view".

```
function nonceBitmap(address, uint256) external returns (uint256);
```

CVF-41. FIXED

- **Category** Documentation
- **Source** ISignatureTransfer.sol

Description It is unclear what happens when the signed amount is bigger than the requested amount.

Recommendation Consider explaining.

Client Comment *Resolved with documentation*

```
/// @param requestedAmount The amount of tokens to transfer
```

```
/// @param requestedAmount The amount of tokens to transfer
```

```
    ToAmountPair[] calldata ToAmountPairs,
```

```
    ToAmountPair[] calldata ToAmountPairs,
```

CVF-42. FIXED

- **Category** Suboptimal
- **Source** ISignatureTransfer.sol

Description One of the arguments in each pair seems redundant.

Recommendation Consider leaving only one of them or clearly explaining why both are required.

Client Comment *Resolved with documentation*

```
string calldata witnessTypeName,
string calldata witnessType,
```

```
string calldata witnessTypeName,
string calldata witnessType,
```

CVF-44. FIXED

- **Category** Documentation
- **Source** ISignatureTransfer.sol

Description These arguments are not documented.

Recommendation Consider documenting.

Client Comment *Resolved with documentation*

```
ToAmountPair[] calldata ToAmountPairs,
```

```
ToAmountPair[] calldata ToAmountPairs,
```

CVF-45. FIXED

- **Category** Bad naming
- **Source** IAllowanceTransfer.sol

Description The name is confusing. It seems that it represents a particular transfer, rather than a contract that facilitate multiple transfers.

Recommendation Consider choosing a better name.

Client Comment *Resolved with documentation*

```
interface IAllowanceTransfer {
```

CVF-46. FIXED

- **Category** Procedural
- **Source** IAllowanceTransfer.sol

Recommendation These errors could be made more helpful by including additional parameters it them, such as the allowance expiration timestamp, the current and desired allowances, and the excessive invalidation amount.

Client Comment *Some custom error arguments were added. Specifically added for AllowanceExpired and InsufficientAllowance.*

```
error AllowanceExpired();
error InsufficientAllowance();
error ExcessiveInvalidation();
```

CVF-47. FIXED

- **Category** Suboptimal
- **Source** IAllowanceTransfer.sol

Description The interface is missing getter functions for current allowances, nonces, etc.

Recommendation Consider adding getters to the interface.

```
interface IAllowanceTransfer {
```

CVF-48. FIXED

- **Category** Bad naming
- **Source** IAllowanceTransfer.sol

Recommendation Events are usually named via n nouns, such as “Invalidations” or “NonceInvalidation”.

```
event InvalidateNonces(address indexed owner, uint32 indexed toNonce
    ↪ , address token, address spender);
```

CVF-49. FIXED

- **Category** Unclear behavior
- **Source** IAllowanceTransfer.sol

Description The “toNonce” parameter is numerical. Usually, numerical parameters are not indexed.

```
event InvalidateNonces(address indexed owner, uint32 indexed toNonce
    ↪ , address token, address spender);
```

CVF-51. INFO

- **Category** Bad datatype
- **Source** IAllowanceTransfer.sol

Recommendation The type of the “token” parameters should be “IERC20”.

```
13 event InvalidateNonces(address indexed owner, uint32 indexed toNonce, address token, address spender);
```

```
16 event Approval(address indexed owner, address indexed token, address indexed spender, uint160 amount);
```

CVF-52. FIXED

- **Category** Unclear behavior
- **Source** IAllowanceTransfer.sol

Recommendation This event should include the “expiration” parameter.

```
16 event Approval(address indexed owner, address indexed token, address indexed spender, uint160 amount);
```

CVF-53. INFO

- **Category** Bad datatype
- **Source** IAllowanceTransfer.sol

Recommendation The type of this field should be “IERC20”.

```
21 address token;
```

```
64 address token;
```

```
72 address token;
```

CVF-54. INFO

- **Category** Unclear behavior
- **Source** IAllowanceTransfer.sol

Description An allowance expiration timestamp uses a narrower type than a signature expiration timestamp. This is weird, as verifying a signature when allowance is already expired doesn't make sense.

Recommendation Consider using the same type for both timestamps.

```
27 uint64 expiration;
```

```
31 uint256 sigDeadline;
```

CVF-55. FIXED

- **Category** Documentation
- **Source** IAllowanceTransfer.sol

Description It is unclear, how unique a nonce should be. Should it be globally unique, or unique for a particular token owner, or what?

Recommendation Consider explaining. 32 bits is not that much for a unique thing.

Client Comment *Resolved with documentation.*

```
28 // a unique value for each signature
   uint32 nonce;
```

```
44 // a unique value for each signature
   uint32 nonce;
```

```
57 // a unique value for each signature
   uint32 nonce;
```

CVF-56. INFO

- **Category** Bad datatype
- **Source** IAllowanceTransfer.sol

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

```
37 address[] tokens;
```

CVF-57. FIXED

- **Category** Suboptimal
- **Source** IAllowanceTransfer.sol

Recommendation It would be more efficient to have a single array of structs with three fields, rather than three parallel arrays. This would also make length checks unnecessary.

Client Comment *Resolved with nested structs.*

```
address[] tokens;
```

```
uint160[] amounts;
```

```
uint64[] expirations;
```

CVF-58. INFO

- **Category** Bad datatype
- **Source** IAllowanceTransfer.sol

Recommendation The type of the "token" arguments should be "IERC20".

```
function approve(address token, address spender, uint160 amount,
   ↪ uint64 expiration) external;
```

```
function transferFrom(address token, address from, address to,
   ↪ uint160 amount) external;
```

```
function invalidateNonces(address token, address spender, uint32
   ↪ amountToInvalidate)
```

CVF-59. FIXED

- **Category** Documentation
- **Source** IAllowanceTransfer.sol

Description Some arguments are not documented.

Recommendation Consider documenting all the arguments.

Client Comment *Resolved with documentation.*

105
```
function transferFrom(address token, address from, address to,
   ↪ uint160 amount) external;
```

ABDK
Consulting

About us

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.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting