

# Ox Exchange v4

Date	December 2020
Lead Auditor	Steve Marx
Co-auditors	Nicholas Ward

## 1 Executive Summary

This report presents the results of our engagement with Ox Labs to review version 4 of the Ox Exchange smart contracts.


The review was conducted by Steve Marx and Nicholas Ward over the course of four person-weeks between November 30th and December 11th, 2020.

## 2 Scope

Our review focused on the commit hash [475b608338561a1dce3199bfb9fb59ee9372149b](#) and was limited to the protocol v4 smart contracts ( `contracts/zero-ex` in the linked repository). Notably, the scope excluded all transformer contracts, staking contracts, and any existing live deployments of the protocol contracts. The complete list of files in scope can be found in the [Appendix](#).

## 3 System Overview

Ox provides [detailed documentation](#) of the system. In brief:

-  The exchange proxy is an efficient upgradeability mechanism that allows different features to be deployed, upgraded, and rolled back on a

function-by-function basis.

- The functionality behind this proxy enables traders (market makers and takers) to swap arbitrary ERC20 tokens at prices they set and to interact with 1st-party and 3rd-party automated liquidity providers (such as Uniswap or Ox's PLP).
- Market makers interact with the system by offering cryptographically signed trades.
- Takers accept a market maker's trade by submitting it to the exchange, either directly or as a metatransaction through a relayer.

## 4 Security Specification

This section describes, **from a security perspective**, the expected behavior of the system under audit. It is not a substitute for documentation. The purpose of this section is to identify specific security properties that were validated by the audit team.

### 4.1 Actors

The relevant actors are listed below with their respective abilities:

- **Market makers** provide liquidity to the system. They cryptographically sign *orders*, which offer to trade a quantity of a *maker token* for a different quantity of a *taker token*. Makers can offer *limit orders* and *RFQ orders*. These are broadly similar, but the latter is restricted in terms of what transaction origin can take the trade.
- **Takers** accept the trades offered by market makers. They submit these trades to a smart contract, which settles the trades by transferring tokens between the taker and maker. Takers can also trade with external entities via functions like `sellToUniswap()`, which directly trades with the Uniswap automated liquidity protocol.
- **Relayers** The exchange supports metatransactions, where a third-party *relayer* submits a transaction on behalf of a trader.
- The **system owner** has special privileges, such as deploying new features. In the deployed configuration, the owner is a multisig wallet that implements a *time lock* to prevent sudden unexpected changes to the system.



## 4.2 Trust Model

In any system, it's important to identify what trust is expected/required between various actors. For this audit, we established the following trust model:

- The exchange owner is a multisig wallet controlled by Ox Labs. This wallet and the associated governance system are out of scope for this audit, but it is expected that traders do not need to trust Ox Labs. Changes are voted on by the community and then bound to a significant time lock, which gives users a chance to see changes coming before they are applied.
- Traders (both makers and takers) should not need to trust each other. If a maker signs an order, they expect that funds can only be transferred away from them under the terms of that order. Similarly, a taker only pays when the trade happens as specified in the order. No one should be able to cause a trader to participate in a trade they didn't agree to.

## 4.3 Security Properties

The following is a non-exhaustive list of security properties that were verified in this audit:

- Orders should not be able to be overfilled (filled beyond the taker or maker amounts specified).
- Only orders properly signed by their makers should be fillable.
- A given metatransaction should only be executed once.
- Canceled or expired orders should not be filled.
- Calls to external systems (e.g. Uniswap, PLP, and transformers) should be guarded such that only outcomes acceptable to the taker can occur.
- *Protocol fees* are taken when limit orders are filled. These fees are then sent to a staking contract, which is out of scope for this audit.

# 5 Recommendations



## 5.1 Remove the unused

MetaTransactionFeature.\_executeMetaTransaction  
( )

### function

#### Description

This function appears to be unused:

**code/contracts/zero-**

**ex/contracts/src/features/MetaTransactionsFeature.sol:L190-L214**

```
/// @dev Execute a meta-transaction via `sender`. Privileged variant.
///      Only callable from within.
/// @param sender Who is executing the meta-transaction.
/// @param mtx The meta-transaction.
/// @param signature The signature by `mtx.signer`.
/// @return returnResult The ABI-encoded result of the underlying call.
function _executeMetaTransaction(
    address sender,
    MetaTransactionData memory mtx,
    LibSignature.Signature memory signature
)
    public
    payable
    override
    onlySelf
    returns (bytes memory returnResult)
{
    ExecuteState memory state;
    state.sender = sender;
    state.mtx = mtx;
    state.hash = getMetaTransactionHash(mtx);
    state.signature = signature;

    return _executeMetaTransactionPrivate(state);
}
```

#### Recommendation

Unless there's a justification for its existence, this unused function should be removed.



## 5.2 Remove stale comments ✓ Fixed

### Resolution

This is fixed in [OxProject/protocol@437a3b0](#).

### Description

There are several comments throughout the codebase that applied to an earlier version of the code and are no longer accurate.

### Examples

- `FeeCollector.sol`

**code/contracts/zero-ex/contracts/src/external/FeeCollector.sol:L60**

```
// Leave 1 wei behind to avoid expensive zero-->non-zero state change.
```

- `UniswapFeature.sol`

**code/contracts/zero-ex/contracts/src/features/UniswapFeature.sol:L400-L401**

```
// Cap the gas limit to prvent all gas being consumed  
// if the token reverts.
```

### Recommendation

Stale comments should be removed or updated.

## 5.3 `UniswapFeature` : Revert if `msg.value` is non-zero but user is not selling ether ✓ Fixed

### Resolution



This is fixed in [0xProject/protocol@437a3b0](https://github.com/0xProject/protocol@437a3b0).

## Description

If a user calls `UniswapFeature.sellToUniswap()` and is attempting to sell ether for any token, the function reverts if `msg.value != sellAmount`. However, there is no check on `msg.value` if the user is attempting to sell any other token. This could lead to stuck funds if a user erroneously supplies ether.

Below is the check made when the caller is selling ether for other tokens:

**code/contracts/zero-ex/contracts/src/features/UniswapFeature.sol:L179-L181**

```
if iszero(eq(callvalue(), sellAmount)) {  
    revert(0, 0)  
}
```

## Recommendation

Add a check that `msg.value == 0` when the user is not selling ether.

# 6 Findings

Each issue has an assigned severity:

- **Minor** issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- **Medium** issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- **Major** issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- **Critical** issues are directly exploitable security vulnerabilities that need to be fixed.



## 6.1 Ether temporarily held during transactions can be stolen via reentrancy

Major

✓ Fixed

### Resolution

This is addressed in [OxProject/protocol@437a3b0](#) by transferring exactly `msg.value` in `sellToLiquidityProvider()`. This adequately protects against this specific vulnerability.

The client team decided to leave the accounting in `MetaTransactionsFeature` as-is due to the complexity/expense of tracking ether consumption more strictly.

### Description

The exchange proxy typically holds no ether balance, but it can temporarily hold a balance *during* a transaction. This balance is vulnerable to theft if the following conditions are met:

1. No check at the end of the transaction reverts if ether goes missing,
2. reentrancy is possible during the transaction, and
3. a mechanism exists to spend ether held by the exchange proxy.

We found one example where these conditions are met, but it's possible that more exist.

### Example

`MetaTransactionsFeature.executeMetaTransaction()` accepts ether, which is used to pay protocol fees. It's possible for less than the full amount in `msg.value` to be consumed, which is why the function uses the `refundsAttachedEth` modifier to return any remaining ether to the caller:

**code/contracts/zero-ex/contracts/src/features/MetaTransactionsFeature.sol:L98-L106**



```

/// @dev Refunds up to `msg.value` leftover ETH at the end of the call.
modifier refundsAttachedEth() {
    _;
    uint256 remainingBalance =
        LibSafeMathV06.min256(msg.value, address(this).balance);
    if (remainingBalance > 0) {
        msg.sender.transfer(remainingBalance);
    }
}

```

Notice that this modifier just returns the remaining ether balance (up to `msg.value`). It does not check for a specific amount of remaining ether. This meets condition (1) above.

It's impossible to reenter the system with a second metatransaction because `executeMetaTransaction()` uses the modifier `nonReentrant`, but there's nothing preventing reentrancy via a different feature. We can achieve reentrancy by trading a token that uses callbacks (e.g. ERC777's hooks) during transfers. This meets condition (2).

To find a full exploit, we also need a way to extract the ether held by the exchange proxy. `LiquidityProviderFeature.sellToLiquidityProvider()` provides such a mechanism. By passing `ETH_TOKEN_ADDRESS` as the `inputToken` and an address in the attacker's control as the `provider`, an attacker can transfer out any ether held by the exchange proxy. Note that `sellToLiquidityProvider()` can transfer any amount of ether, not limited to the amount sent via `msg.value`:

**code/contracts/zero-ex/contracts/src/features/LiquidityProviderFeature.sol:L114-L115**

```

if (inputToken == ETH_TOKEN_ADDRESS) {
    provider.transfer(sellAmount);
}

```

This meets condition (3).

The full steps to exploit this vulnerability are as follows:

1. A maker/attacker signs a trade where one of the tokens will invoke a callback during the trade.



A taker signs a metatransaction to take this trade.



3. A relayer sends in the metatransaction, providing more ether than is necessary to pay the protocol fee. (It's unclear how likely this situation is.)
4. During the token callback, the attacker invokes `LiquidityProviderFeature.sellToLiquidityProvider()` to transfer the excess ether to their account.
5. The metatransaction feature returns the remaining ether balance, which is now zero.

## Recommendation

In general, we recommend using strict accounting of ether throughout the system. If there's ever a temporary balance, it should be accurately resolved at the end of the transaction, after any potential reentrancy opportunities.

For the example we specifically found, we recommend doing strict accounting in the metatransactions feature. This means features called via a metatransaction would need to return how much ether was consumed. The metatransactions feature could then refund exactly `msg.value - <consumed ether>`. The transaction should be reverted if this fails because it means ether went missing during the transaction.

We also recommend limiting `sellToLiquidityProvider()` to only transfer up to `msg.value`. This is a form of defense in depth in case other vectors for a similar attack exist.

## 6.2 UniswapFeature : Non-static call to

`ERC20.allowance()` Minor ✓ Fixed

### Resolution

This is fixed in [0xProject/protocol@437a3b0](https://github.com/0xProject/protocol@437a3b0).

## Description



In the case where a token is possibly “greedy” (consumes all gas on failure), `UniswapFeature` makes a call to the token's `allowance()` function to check

whether the user has provided a token allowance to the protocol proxy or to the `AllowanceTarget`. This call is made using `call()`, potentially allowing state-changing operations to take place before control of the execution returns to `UniswapFeature`.

## code/contracts/zero-ex/contracts/src/features/UniswapFeature.sol:L373-L377

```
// `token.allowance()`
mstore(0xB00, ALLOWANCE_CALL_SELECTOR_32)
mstore(0xB04, caller())
mstore(0xB24, address())
let success := call(gas(), token, 0, 0xB00, 0x44, 0xC00, 0x20)
```

## Recommendation

Replace the `call()` with a `staticcall()`.

## 6.3 UniswapFeature : Unchecked returndatasize in low-level external calls Minor ✓ Fixed

### Resolution

This is fixed in [0xProject/protocol@437a3b0](https://github.com/0xProject/protocol@437a3b0).

## Description

`UniswapFeature` makes a number of external calls from low-level assembly code. Two of these calls rely on the `CALL` opcode to copy the returndata to memory without checking that the call returned the expected amount of data. Because the `CALL` opcode does not zero memory if the call returns less data than expected, this can lead to usage of dirty memory under the assumption that it is data returned from the most recent call.

## Examples



Call to `UniswapV2Pair.getReserves()`

## code/contracts/zero-ex/contracts/src/features/UniswapFeature.sol:L201-L205

```
// Call pair.getReserves(), store the results at `0xC00`
mstore(0xB00, UNISWAP_PAIR_RESERVES_CALL_SELECTOR_32)
if iszero(staticcall(gas(), pair, 0xB00, 0x4, 0xC00, 0x40)) {
    bubbleRevert()
}
```

- Call to `ERC20.allowance()`

## code/contracts/zero-ex/contracts/src/features/UniswapFeature.sol:L372-L377

```
// Check if we have enough direct allowance by calling
// `token.allowance()`
mstore(0xB00, ALLOWANCE_CALL_SELECTOR_32)
mstore(0xB04, caller())
mstore(0xB24, address())
let success := call(gas(), token, 0, 0xB00, 0x44, 0xC00, 0x20)
```

## Recommendation

Instead of providing a memory range for `call()` to write returndata to, explicitly check `returndatasize()` after the call is made and then copy the data into memory using `returndatacopy()`.

```
if lt(returndatasize(), EXPECTED_SIZE) {
    revert(0, 0)
}
returndatacopy(0xC00, 0x00, EXPECTED_SIZE)
```

## 6.4 Rollback functionality can lead to untested combinations

Minor

Acknowledged

### Resolution



from the client team:

Just like our migrations, we batch our rollbacks by release, which enforces rolling back to known good configurations.

The documentation now includes an [emergency playbook](#) that describes how rollbacks should be done.

## Description

`SimpleFunctionRegistry` maps individual function selectors to implementation contracts. As features are newly deployed or upgraded, functions are registered in logical groups after a timelock enforced by the owning multisig wallet. This gives users time to evaluate upcoming changes and stop using the contract if they don't like the changes.

Once deployed, however, any function can individually be rolled back *without* a timelock to any previous version of that function. Users are given no warning, functions can be rolled back to any previous implementation (regardless of how old), and the per-function granularity means that the configuration after rollback may be a never-before-seen combination of functions.

The combinatorics makes it impossible for a user (or auditor) to be comfortable with all the possible outcomes of rollbacks. If there are `n` versions each of `m` functions, there are  $n^m$  combinations that could be in effect at any moment. Some functions depend on other `onlySelf` functions, so the behavior of those combinations is not at all obvious.

This presents a trust problem for users.

## Recommendation

Rollback makes sense as a way to rapidly recover from a bad deployment, but we recommend limiting its scope. The following ideas are in preferred order (our favorite first):

- Disallow rollback altogether except to an implementation of `address(0)`. This way broken functionality can be immediately *disabled*, but no old version of a function can be reinstated.



- Limit rollback by number of versions, e.g. only allowing rollback to the immediately previous version of a function.
- Limit rollback by time, e.g. only allowing rollback to versions in the past  $n$  weeks.

## Appendix 1 - Files in Scope

This audit covered the following files:

File Name	SHA-1 hash
ZeroExOptimized.sol	e7e11491de1aece81c55caa1368f0d27c3a80633
ZeroEx.sol	ee3599a8120bf4e0d23dfd051ca19103145a74d6
lZeroEx.sol	faa7bb1a3e210ec44be4e659a614808641384a60
features/INativeOrdersFeature.sol	0a9418aadddf4bd53356f562d7df0de3e6da6c6e
features/LiquidityProviderFeature.sol	63d02651ae4bdc9d4a0a3c1c45cf1ed4f715b9fe
features/IMetaTransactionsFeature.sol	4cae4935a15ccbc1be1b1566ec498de161f9db88
features/ILiquidityProviderFeature.sol	a394372fee2c38140574771b935e7e85e3202b55
features/IBootstrapFeature.sol	4ada28f1cf166a45c6816a4a8c8343e542af7dbb
features/OwnableFeature.sol	b7c0184113283fe9d931a760725c8b308da5c4ba
features/IOwnableFeature.sol	2e84fc1ab8130cfaabe2b9e9df75ff15cf406f9e
features/BootstrapFeature.sol	65cf0c9cf041d6ffef455e7ec627857e740019d3



File Name	SHA-1 hash
features/IUniswapFeature.sol	8b0884c62215679ccd8820ad660beb4da2cff9d7
features/UniswapFeature.sol	c57720d6bb301512b0e2634ff8bb08cf85b2f4be
features/SignatureValidatorFeature.sol	103d1d2e9f87d099bff0c45b92b5b5b5a9906acd
features/IFeature.sol	b0941335bf210546cc0c539cd5081469122c81ed
features/SimpleFunctionRegistryFeature.sol	b9bf0b338aba19a6de727598ee6e689971671d8a
features/MetaTransactionsFeature.sol	9acf4c2893bbe048be2134a1ba03dd44f3491df4
features/ITokenSpenderFeature.sol	45bf89b19de3094a98a5bc80af7538c65ccd613e
features/NativeOrdersFeature.sol	53da6f5812dea148f32fec018cddd8555cb0703d
features/TransformERC20Feature.sol	dcadde7b91e198f902a71af7f8f458a3039bddbd
features/TokenSpenderFeature.sol	b246f4f0a32f3faf3c4cf4db4780a94b9876e60e
features/ISimpleFunctionRegistryFeature.sol	1855f16802fbf273e4e0fca41a34c6eae28a2449
features/ITransformERC20Feature.sol	93cc9efdd7be831b5061201a4414e736ba074ac2
features/ISignatureValidatorFeature.sol	c59a306588413d3c99045d48a762c11429f56eb2
features/libs/LibNativeOrder.sol	bf2b3834ffbf107b79885670862ae17419dafa84
 features/libs/LibSignature.sol	39754070b5fca1e05cd642b9a8509acbe64f3f85

File Name	SHA-1 hash
storage/LibMetaTransactionsStorage.sol	e63bf052efe3214b2486f76caee6e178c2855161
storage/LibTransformERC20Storage.sol	b3829af465d566b9e1baca6e905cf0bd2e0f4570
storage/LibSimpleFunctionRegistryStorage.sol	edc9005421bc9085ae84664a8185a62f7eef1c2b
storage/LibReentrancyGuardStorage.sol	8f68335f6bd36b15bc380b2e6dedf458e1035b11
storage/LibTokenSpenderStorage.sol	85cbb2b001867162e81506a4d1bd56f17b9372fc
storage/LibNativeOrdersStorage.sol	4c764e8bffd891bfc60d0321ac437635fe84bc3
storage/LibStorage.sol	0f112d17cf10569ebb1d4430692017ae9b8c64ae
storage/LibProxyStorage.sol	4f613dd79bd8d2eaff9c024c4d1526a9e84e8c9b
storage/LibOwnableStorage.sol	f669284be001bef9545e94d8c79fe4c9a66be898
fixins/FixinReentrancyGuard.sol	4ac7d89eb7808088e0d61abd10ddca269afafda7
fixins/FixinProtocolFees.sol	46970d1f231cdbb6f3a890d0f8d9652c69e65bfe
fixins/FixinTokenSpender.sol	cc5c619068c3c47076beac19886723b99c08e443
fixins/FixinEIP712.sol	0a9f38ed221270c8d17a461e3e63d007e74fe1eb
fixins/FixinCommon.sol	4a9fa7e6f12f22a095d963b24fc51979e1a4d026
internal/PermissionlessTransformerDeployer.sol	8fb02f2efda6d22e5408bdcee68b750f80cedf4e



File Name	SHA-1 hash
external/ILiquidityProviderSandbox.sol	70129318e68a1b505ad906bc35621cd00436ba7e
external/TransformerDeployer.sol	f730f39c0b8dcba296917ea3aa653497d70201df
external/LibFeeCollector.sol	019c152459e160cc0b3b438d0fc6a531fef7ee39
external/AllowanceTarget.sol	03cf3ab3b1bb0be08e50411a6369e3634d3701a6
external/LiquidityProviderSandbox.sol	be58a5d1807b08f1a113961c48e26698310def53
external/FlashWallet.sol	6df1faa76acf249602e08cdac9b87b7595820378
external/IAAllowanceTarget.sol	55e49c6c8ddfffd90b951052ac4f2102b31839dc
external/FeeCollectorController.sol	893b62b0d6ad86136540430e1ab1bea677af95ad
external/FeeCollector.sol	bc2c79e65d59219f20e51c11fdd6bd380d574d37
external/IFlashWallet.sol	c670f0cb9b6ae1ae8edddd1d70b037713fdb629
migrations/LibBootstrap.sol	27928b1e31e589d0586ea310ec4aa8b374517c3a
migrations/FullMigration.sol	ce79921725d5c137ed67b4d7c3d45c73e7ed03b8
migrations/InitialMigration.sol	69f2e365b10c3319de1599ddab9e81541f41da57
migrations/LibMigrate.sol	32fda4e20e5f2efff7b8556db06f4671531c7394
external/ILiquidityProvider.sol	c52b120596ca8b2e8d90939b5d46f72850d99b61





File Name	SHA-1 hash
vendor/v3/IERC20Bridge.sol	1c8c55e1a83d506eca75207a23b6b04528b3aa5f
vendor/v3/IExchange.sol	b672e592a178b7c0e861e75d2f2d614ab90dcdb1
vendor/v3/IGasToken.sol	daff2216f78162c74695a4e721867d8a1f840656
vendor/v3/IStaking.sol	e1321ee791effd3933b4ff3ae43d8a59dd2fe435

## Appendix 2 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.



**PURPOSE OF REPORTS** The Reports and the analysis described therein are intended solely for Clients and published with their consent. The scope of our

review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

**LINKS TO OTHER WEB SITES FROM THIS WEB SITE** You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

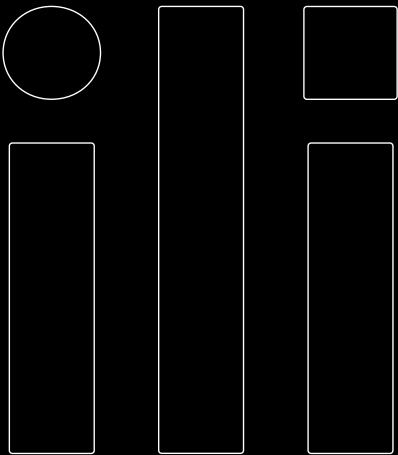
**TIMELINESS OF CONTENT** The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.



# Request a Security Review Today

Get in touch with our team to request a quote for a smart contract audit.

CONTACT US



- AUDITS
- FUZZING
- SCRIBBLE
- BLOG
- TOOLS
- RESEARCH
- ABOUT
- CONTACT
- CAREERS
- PRIVACY POLICY

Subscribe to Our Newsletter

Stay up-to-date on our latest offerings, tools, and the world of blockchain security.

Email\*

e-mail address

