Armors Labs

LambSwap

Smart Contract Audit

- LambSwap Audit Summary
- LambSwap Audit
  - Document information
    - Audit results
    - Audited target file
  - Vulnerability analysis
    - Vulnerability distribution
    - Summary of audit results
    - Contract file
    - Analysis of audit results
      - Re-Entrancy
      - Arithmetic Over/Under Flows
      - Unexpected Blockchain Currency
      - Delegatecall
      - Default Visibilities
      - Entropy Illusion
      - External Contract Referencing
      - Unsolved TODO comments
      - Short Address/Parameter Attack
      - Unchecked CALL Return Values
      - Race Conditions / Front Running
      - Denial Of Service (DOS)
      - Block Timestamp Manipulation
      - Constructors with Care
      - Unintialised Storage Pointers
      - Floating Points and Numerical Precision
      - tx.origin Authentication

LambSwap Audit Summary

Project name : LambSwap Contract

Project address : None

Code1 URL: https://github.com/lambswap/uniswap-v2-core

Code1 Commit : dbdef8ce737fd1a8a570461f0d14deb1462921d5

Code2 URL: https://github.com/lambswap/uniswap-v2-periphery

Code2 Commit : f81fb15ffc7cca8a917bd0cf0cb7d34e96931636

Code3 URL: https://github.com/lambswap/liquidity-staker

Code3 Commit : 8ad651bb5bac1c4a4221e6cfe757dd86a706dc7e

Code4 URL: https://github.com/lambswap/gov

Code4 Commit : 2e927c412f14f60f77c05f7e989e808438a1649d

Code5 URL: https://github.com/lambswap/bridge

Code5 Commit : 3d523728357442ca849b583d7632ed5c31ccc4cf

Projct target : LambSwap Contract Audit

Blockchain : Huobi ECO Chain（Heco）

Test result : PASSED

Audit Info

Audit NO : 0X202104020008

Audit Team : Armors Labs

Audit Proofreading: https://armors.io/#project-cases

LambSwap Audit

The LambSwap team asked us to review and audit their LambSwap contract. We looked at the code and now publish our results.

Here is our assessment and recommendations, in order of importance.

Document information

Name	Auditor	Version	Date
LambSwap Audit	Rock ,Hosea, Rushairer	1.0.0	2021-04-02

Audit results

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the LambSwap contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 18 months from the date of output.

(Statement: Armors Labs reports only on facts that have occurred or existed before this report is issued and assumes corresponding responsibilities. Armors Labs is not able to determine the security of its smart contracts and is not responsible for any subsequent or existing facts after this report is issued. The security audit analysis and other content of this report are only based on the documents and information provided by the information provider to Armors Labs at the time of issuance of this report (" information provided " for short). Armors Labs postulates that the information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered, deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused.)

Audited target file

file	md5
./uniswap-v2-periphery/test/RouterEventEmitter.sol	bb6d14d6c3ceaed70c1dae947354710a
./uniswap-v2-periphery/test/WETH9.sol	c60afbcd54ec9da3b3908e2db56c0d12
./uniswap-v2-periphery/test/DeflatingERC20.sol	7476ba791f9afb6f870f10560972e0a0
./uniswap-v2-periphery/test/ERC20.sol	3c84c6f456884e55b360a8450d0f0446
./uniswap-v2-periphery/UniswapV2Migrator.sol	2771c36e1b0dbff52fd0d33124f259d7
./uniswap-v2-periphery/libraries/UniswapV2LiquidityMathLibrary.sol	ff4345b5db079741e28bc593d0356893
./uniswap-v2-periphery/libraries/SafeMath.sol	9f77127a0e2c55c2058453327e84bde3
./uniswap-v2-periphery/libraries/UniswapV2OracleLibrary.sol	6375842b01002cf7b8f7899b01c5e8c8
./uniswap-v2-periphery/libraries/UniswapV2Library.sol	baacc4e29cf296e1f6106918eb5178c0
./uniswap-v2-periphery/examples/ExampleComputeLiquidityValue.sol	8630765210922a33990d47c45a55bbe1
./uniswap-v2-periphery/examples/ExampleFlashSwap.sol	614bf65e008878f8b5175ed977e6eb68
./uniswap-v2-periphery/examples/ExampleOracleSimple.sol	53b250317c7e59cac44708abd7405ce0
./uniswap-v2-periphery/examples/ExampleSwapToPrice.sol	978e6a841d1f58c0d6e40d8e45f333c6
./uniswap-v2-periphery/examples/ExampleSlidingWindowOracle.sol	26398fbcf959fac32d155b8456eeceb4
./uniswap-v2-periphery/TestToken.sol	8cfcaeb439da3ce116003c9674e8cfe6
./uniswap-v2-periphery/UniswapV2Router01.sol	c39e8c34a9007c632081c9bcf44323d1
./uniswap-v2-periphery/UniswapV2Router02.sol	12a8c57909541a65738ecced73a40885
./uniswap-v2-periphery/interfaces/V1/IUniswapV1Factory.sol	abe9343b20a93919250e0dba78bfd9c1
./uniswap-v2-periphery/interfaces/V1/IUniswapV1Exchange.sol	5ec7fe77a73b5ace5fc1fbae6b87f34e
./uniswap-v2-periphery/interfaces/IUniswapV2Migrator.sol	f626f6b55e8b5ffbda4f64e9409e5614

file	md5
./uniswap-v2-periphery/interfaces/IWETH.sol	270f2c1a1f22475259d5a2f6a524f72e
./uniswap-v2-periphery/interfaces/IUniswapV2Router01.sol	95d8729b2c698dc663322167b3c085fe
./uniswap-v2-periphery/interfaces/IUniswapV2Router02.sol	3fd36426274a5b35f7e702f87ba08a8e
./uniswap-v2-periphery/interfaces/IERC20.sol	3d2b948a9266ee66b6e739de97c7e6e9
./bridge/mLAMB.sol	50bce2ed6ca1af81fdf0fdaef841cbf8
./uniswap-v2-core/test/ERC20.sol	5ad37a09a9d83a1255b7b71bc650a86a
./uniswap-v2-core/libraries/SafeMath.sol	a65356c329efeffba75210664f5df3cd
./uniswap-v2-core/libraries/UQ112x112.sol	2a703244e9f645c6fb297a983ce7c9c4
./uniswap-v2-core/libraries/Math.sol	bee6c1959728d5d42c14151076cebd83
./uniswap-v2-core/UniswapV2ERC20.sol	283962ba341ad9858ef1bab62b63a00c
./uniswap-v2-core/calHash.sol	a69c90812089b0547ea06405c04ebf68
./uniswap-v2-core/UniswapV2Pair.sol	4b1317cac959bb5acc258a9e1199ca7a
./uniswap-v2-core/interfaces/IUniswapV2ERC20.sol	752c725168e1a5b73a8209897f18c5b9
./uniswap-v2-core/interfaces/IUniswapV2Pair.sol	3fa31c3860f2b9585c3a98c64850829e
./uniswap-v2-core/interfaces/IUniswapV2Callee.sol	30a4aed15b3fa3ce5588118b7cecfa7f
./uniswap-v2-core/interfaces/IERC20.sol	3d2b948a9266ee66b6e739de97c7e6e9
./uniswap-v2-core/interfaces/IUniswapV2Factory.sol	ec39b5372f4f4d52101b4da561e2aee6
./uniswap-v2-core/UniswapV2Factory.sol	bbd1d6d8cec14355db9a75d10b0d6802
./liquidity-staker/test/TestERC20.sol	9b5af907146aa7d95018cee69b7a77b4
./liquidity-staker/RewardsDistributionRecipient.sol	9e0dbeaf0074757c4ce3c1caa799663b
./liquidity-staker/StakingRewards.sol	f130dfaed41b3dfaa9fcf7bf985046eb
./liquidity-staker/TestToken.sol	fa5f241f4c677d9e66762b9fb322e681
./liquidity-staker/StakingRewardsFactory.sol	878048711430631b55055bab1ccef043
./liquidity-staker/interfaces/IStakingRewards.sol	cb4631035c61f6d7d5b382984a2af45c
./gov/GOAT.sol	09f5e9d5e3e110a454229069ab727e69

Vulnerability analysis

Vulnerability distribution

vulnerability level	number
Critical severity	0
High severity	0

vulnerability level	number
Medium severity	0
Low severity	0

Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Arithmetic Over/Under Flows	safe
Unexpected Blockchain Currency	safe
Delegatecall	safe
Default Visibilities	safe
Entropy Illusion	safe
External Contract Referencing	safe
Short Address/Parameter Attack	safe
Unchecked CALL Return Values	safe
Race Conditions / Front Running	safe
Denial Of Service (DOS)	safe
Block Timestamp Manipulation	safe
Constructors with Care	safe
Unintialised Storage Pointers	safe
Floating Points and Numerical Precision	safe
tx.origin Authentication	safe

Contract file

```
.
├── bridge
│   └── mLAMB.sol
├── gov
│   └── GOAT.sol
├── liquidity-staker
│   ├── RewardsDistributionRecipient.sol
│   ├── StakingRewards.sol
│   ├── StakingRewardsFactory.sol
│   ├── TestToken.sol
│   ├── interfaces
│   │   └── IStakingRewards.sol
│   └── test
│       └── TestERC20.sol
├── uniswap-v2-core
│   ├── UniswapV2ERC20.sol
│   ├── UniswapV2Factory.sol
│   ├── UniswapV2Pair.sol
│   ├── calHash.sol
│   ├── interfaces
│   │   ├── IERC20.sol
│   │   ├── IUniswapV2Callee.sol
│   │   ├── IUniswapV2ERC20.sol
│   │   ├── IUniswapV2Factory.sol
│   │   └── IUniswapV2Pair.sol
│   ├── libraries
│   │   ├── Math.sol
│   │   ├── SafeMath.sol
│   │   └── UQ112x112.sol
│   └── test
│       └── ERC20.sol
└── uniswap-v2-periphery
    ├── TestToken.sol
    ├── UniswapV2Migrator.sol
    ├── UniswapV2Router01.sol
    ├── UniswapV2Router02.sol
    ├── examples
    │   ├── ExampleComputeLiquidityValue.sol
    │   ├── ExampleFlashSwap.sol
    │   ├── ExampleOracleSimple.sol
    │   ├── ExampleSlidingWindowOracle.sol
    │   ├── ExampleSwapToPrice.sol
    │   └── README.md
    ├── interfaces
    │   ├── IERC20.sol
    │   ├── IUniswapV2Migrator.sol
    │   ├── IUniswapV2Router01.sol
    │   ├── IUniswapV2Router02.sol
    │   ├── IWETH.sol
    │   └── V1
    │       ├── IUniswapV1Exchange.sol
    │       └── IUniswapV1Factory.sol
    ├── libraries
    │   ├── SafeMath.sol
    │   ├── UniswapV2Library.sol
    │   ├── UniswapV2LiquidityMathLibrary.sol
    │   └── UniswapV2OracleLibrary.sol
    └── test
        ├── DeflatingERC20.sol
        ├── ERC20.sol
        ├── RouterEventEmitter.sol
        └── WETH9.sol

15 directories, 46 files
```

Analysis of audit results

Re-Entrancy

- **Description:**
  One of the features of smart contracts is the ability to call and utilise code of other external contracts. Contracts also typically handle Blockchain Currency, and as such often send Blockchain Currency to various external user addresses. The operation of calling external contracts, or sending Blockchain Currency to an address, requires the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the contract to execute further code (i.e. through a fallback function) , including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Arithmetic Over/Under Flows

- **Description:**
  The Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unexpected Blockchain Currency

- **Description:**
  Typically when Blockchain Currency is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where Blockchain Currency can exist in a contract without having executed any code. Contracts which rely on code execution for every Blockchain Currency sent to the contract can be vulnerable to attacks where Blockchain Currency is forcibly sent to a contract.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Delegatecall

- **Description:**
  The CALL and DELEGATECALL opcodes are useful in allowing developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Default Visibilities

- **Description:**
  Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whBlockchain Currency a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs. Functions default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devestating vulernabilities in smart contracts as will be discussed in this section.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Entropy Illusion

- **Description:**
  All transactions on the blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no rand() function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

External Contract Referencing

- **Description:**
  One of the benefits of the global computer is the ability to re-use code and interact with contracts already deployed on the network. As a result, a large number of contracts reference external contracts and in general operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Unsolved TODO comments

- **Description:**
  Check for Unsolved TODO comments
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Short Address/Parameter Attack

- **Description:**
  This attack is not specifically performed on Solidity contracts themselves but on third party applications that may interact with them. I add this attack for completeness and to be aware of how parameters can be manipulated in contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unchecked CALL Return Values

- **Description:**
  There a number of ways of performing external calls in solidity. Sending Blockchain Currency to external accounts is commonly performed via the transfer() method. However, the send() function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The call() and send() functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (intialised by call() or send()) fails, rather the call() or send() will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Race Conditions / Front Running

- **Description:**
  The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds of race conditions that can occur on the blockchain. There is a variety of good posts on this subject, a few are: Wiki - Safety, DASP - Front-Running and the Consensus - Smart Contract Best Practices.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Denial Of Service (DOS)

- **Description:**
  This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap Blockchain Currency in these contracts forever, as was the case with the Second Parity MultiSig hack
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Block Timestamp Manipulation

- **Description:**
  Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Constructors with Care

- **Description:**
  Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unintialised Storage Pointers

- **Description:**
  The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately intialising variables.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Floating Points and Numerical Precision

- **Description:**
  As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.
- **Detection results:**

```
PASSED!
```

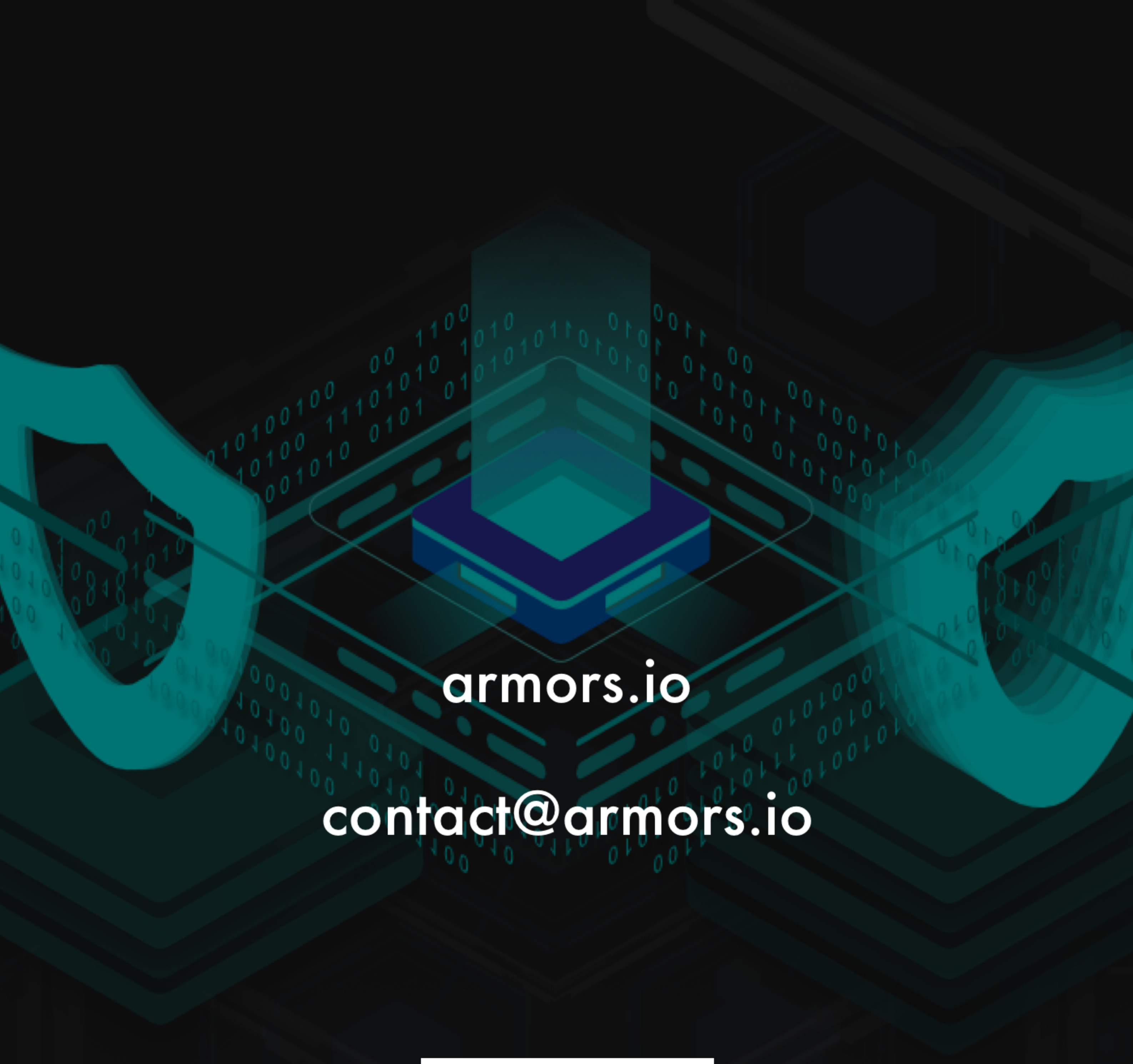
- **Security suggestion:**
  no.

tx.origin Authentication

- **Description:**
  Solidity has a global variable, tx.origin which traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts leaves the contract vulnerable to a phishing-like attack.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.



armors.io

contact@armors.io