Jorge Rodriguez

CYBERSEC CONTRACT AUDIT REPORT
XDEF Finance - CTDSEC.com

Introduction

During January of 2021, XDEF Finance engaged CTDSec to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. XDEF Finance provided CTDSec with access to their code repository and whitepaper.

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bugfree status. The audit documentation is for discussion purposes only.

I always recommend having a bug bounty program opened to detect future bugs.

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the XDEF Finance contract followed by issue reporting, along with mitigation and remediation instructions outlined in this report. The following code files are considered in-scope for the review:
Github: https://github.com/combinefinance

- ERC20UpgradeSafe.sol
- ERC677Token.sol
- LPToken.sol
- TokenPriceOracle.sol
- TvlOracle.sol
- XdefToken.sol
- XdefTokenMonetaryPolicy.sol
- XdefTokenOrchestrator.sol
- TokenPool.sol
- TokenGeyser.sol
- IStaking.sol

Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

Correctness of the protocol implementation [Result OK]

User funds are secure on the blockchain and cannot be transferred without user permission [Result OK]

Vulnerabilities within each component as well as secure interaction between the network components [Result OK]

Correctly passing requests to the network core [Result OK]

Data privacy, data leaking, and information integrity [Result OK]

Susceptible to reentrancy attack [Result OK]

Key management implementation: secure private key storage and proper management of encryption and signing keys [Result OK]

Handling large volumes of network traffic [Result OK]

Resistance to DDoS and similar attacks [SOLVED BY DEV TEAM]

Aligning incentives with the rest of the network [Result OK]

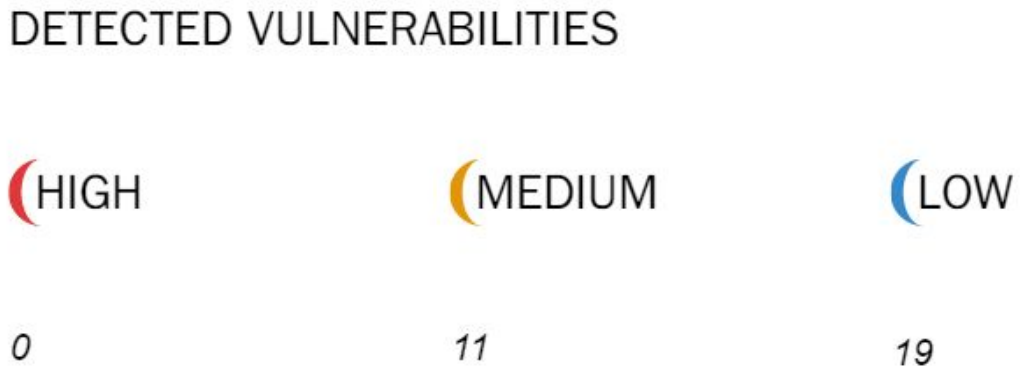
Any attack that impacts funds, such as draining or manipulating of funds [Result OK]

Mismanagement of funds via transactions [Result OK]

Inappropriate permissions and excess authority [Result OK]

Special token issuance model [Result OK]

Vulnerabilities



ISSUES

MEDIUM SWC-128

Loop over unbounded data structure.

Gas consumption in function "applyCharity" in contract "XdefTokenMonetaryPolicy" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Locations

```
2189 }
2190
2191 for (uint256 i = 0; i < charityRecipients.length; i++) {
2192 address recipient = charityRecipients[i];
2193 uint256 recipientPercent = supplyDelta < 0 ? charityPercentOnContraction[recipient]
```

MEDIUM SWC-128

Loop over unbounded data structure.

Gas consumption in function "rebase" in contract "XdefTokenOrchestrator" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Locations

```
2457  policy.rebase();
2458
2459  for (uint i = 0; i < transactionEnabled.length; i++) {
2460  // Transaction storage t = transactions[i];
2461  if (transactionEnabled[i]) {
```

MEDIUM SWC-128

Implicit loop over unbounded data structure.

Gas consumption in function "rebase" in contract "XdefTokenOrchestrator" depends on the size of data structures that may grow unboundedly. The highlighted statement involves copying the array "transactionData[i]" from "storage" to "memory". When copying arrays from "storage" to "memory" the Solidity compiler emits an implicit loop.If the array grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Locations

```
2460  // Transaction storage t = transactions[i];
2461  if (transactionEnabled[i]) {
2462  bool result = externalCall(transactionDestination[i], transactionData[i]);
2463  if (!result) {
2464  emit TransactionFailed(transactionDestination[i], i, transactionData[i]);
```

MEDIUM SWC-128

Implicit loop over unbounded data structure.

Gas consumption in function "removeTransaction" in contract "XdefTokenOrchestrator" depends on the size of data structures that may grow unboundedly. The highlighted assignment overwrites or deletes a state variable that contains an array. When assigning to or deleting storage arrays, the Solidity compiler emits an implicit clearing loop.If the array grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Locations

```
2496  transactionEnabled[index] = transactionEnabled[transactionEnabled.length - 1];
2497  transactionDestination[index] = transactionDestination[transactionEnabled.length - 1];
2498  transactionData[index] = transactionData[transactionEnabled.length - 1];
2499  }
```

MEDIUM SWC-000

Function could be marked as external.

The function definition of "setXdefToken" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Locations

MEDIUM SWC-000

Function could be marked as external.

The function definition of "burnShares" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Locations

```
1783  }
1784
1785  function burnShares(address recipient, uint256 amount)
1786  public
1787  {
1788  require(msg.sender == monetaryPolicy, "forbidden");
1789  require(_shareBalances[recipient] >= amount, "amount");
1790  _shareBalances[recipient] = _shareBalances[recipient].sub(amount);
1791  _totalShares = _totalShares.sub(amount);
1792  }
1793
1794  function initialize()
```

MEDIUM SWC-000

Function could be marked as external.

The function definition of "sharesOf" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Locations

```
1767  }
1768
1769  function sharesOf(address user)
1770  public
1771  view
1772  returns (uint256)
1773  {
1774  return _shareBalances[user];
1775  }
1776
1777  function mintShares(address recipient, uint256 amount)
```

MEDIUM SWC-000

Function could be marked as external.

The function definition of "setRebasesPaused" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Locations

```
1698  }
1699
1700  function setRebasesPaused(bool _rebasesPaused)
1701  public
1702  onlyOwner
1703  {
1704  rebasesPaused = _rebasesPaused;
1705  }
1706
1707  /**
```

MEDIUM SWC-000

Function could be marked as external.

The function definition of "setTransferPauseExempt" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Locations

```
1687  }
1688
1689  function setTransferPauseExempt(address user, bool exempt)
1690  public
1691  onlyOwner
1692  {
1693  if (exempt) {
1694  transferPauseExemptList[user] = true;
1695  } else {
1696  delete transferPauseExemptList[user];
1697  }
1698  }
1699
1700  function setRebasesPaused(bool _rebasesPaused)
```

MEDIUM SWC-000

Function could be marked as external.

The function definition of "transferOwnership" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

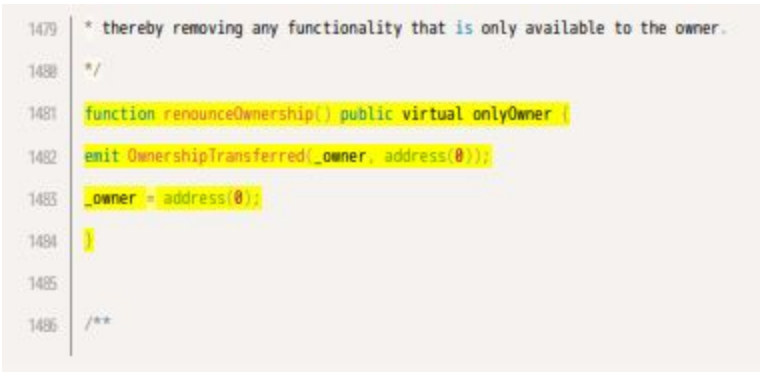
Locations

MEDIUM SWC-000

Function could be marked as external.

The function definition of "renounceOwnership" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Locations



LOW - SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""=0.4.24<0.7.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Locations

LOW - SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Locations

LOW - SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.0"". It is recommended to specify a fixed

compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Locations

LOW - SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Locations

LOW - SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.2"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Locations

LOW - SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.16"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

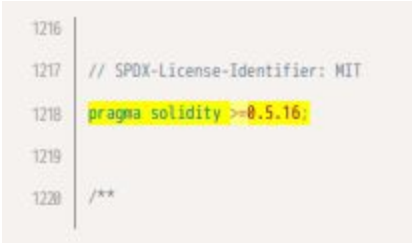
Locations

LOW - SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.16"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Locations



LOW - SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.0"".. It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Locations

LOW - SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Locations

```
2355 function inRebaseWindow() public view returns (bool) {
2356 return (
2357 now.mod(minRebaseTimeIntervalSec) >= rebaseWindowOffsetSec &&
2358 now.mod(minRebaseTimeIntervalSec) < (rebaseWindowOffsetSec.add(rebaseWindowLengthSec))
2359 );
2360 }
```

LOW - SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Locations

```
2109 function rebase() external {
2110 require(msg.sender == orchestrator, "you are not the orchestrator");
2111 require(inRebaseWindow(), "the rebase window is closed");
2112
2113 // This comparison also ensures there is no reentrancy.
```

LOW - SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Locations

```
2112
2113  // This comparison also ensures there is no reentrancy.
2114  require(lastRebaseTimestampSec.add(minRebaseTimeIntervalSec) < now, "cannot rebase yet");
2115
2116  // Snap the rebase time to the start of this window.
```

LOW - SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Locations

```
244  */
245  function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
246  require(b <= a, errorMessage);
247  uint256 c = a - b;
```

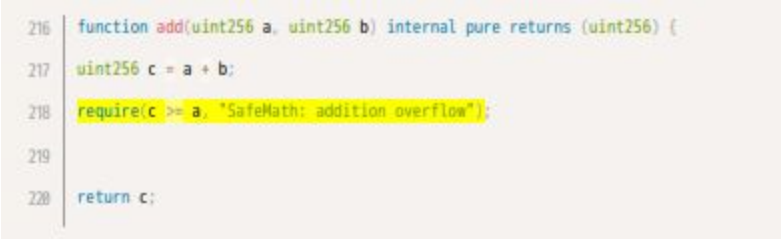
LOW - SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment

variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Locations

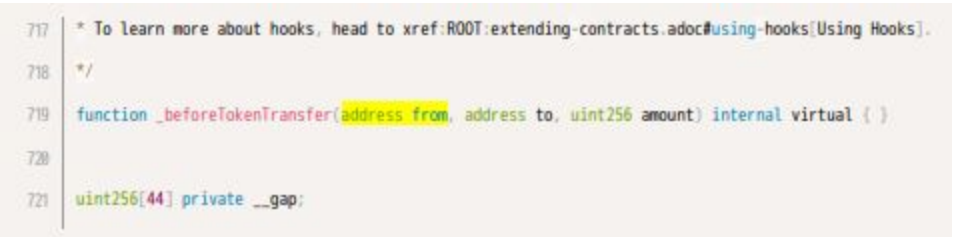


LOW - SWC-131

Unused function parameter "from".

The value of the function parameter "from" for the function "_beforeTokenTransfer" of contract "ERC20UpgradeSafe" does not seem to be used anywhere in "_beforeTokenTransfer".

Locations



LOW - SWC-131

Unused function parameter "to".

The value of the function parameter "to" for the function "_beforeTokenTransfer" of contract "ERC20UpgradeSafe" does not seem to be used anywhere in "_beforeTokenTransfer".

Locations

```
717 * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
718 */
719 function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
720
721 uint256[44] private __gap;
```

LOW - SWC-131

Unused function parameter "amount".

The value of the function parameter "amount" for the function "_beforeTokenTransfer" of contract "ERC20UpgradeSafe" does not seem to be used anywhere in "_beforeTokenTransfer".

Locations

```
717 * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
718 */
719 function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
720
721 uint256[44] private __gap;
```

LOW - SWC-131

Unused local variable "_timestamp".

The local variable "_timestamp" is declared within the function "getData" of contract "TvlOracle" but its value does not seem to be used anywhere in "getData".

Locations

```
1405 {
1406 bool _didGet;
1407 uint _timestamp;
1408 uint256 _value;
```

LOW - SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Locations

```
313 ERC20 token = ERC20(_tokenA);
314
315 require(token.allowance(msg.sender,address(this)) >= _amountA, "ERC20: Allowance not enough");
316 require(token.balanceOf(msg.sender) >= _amountA, "ERC20: Balance is not enough");
```

```
232
233 // UniswapRouterV2 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D
234 contract limitOrder is Ownable,ReentrancyGuard {
235
236 using SafeMath for uint256;
237
238 /*********************************
239 * EVENTS ************************
240 *********************************/
241 event NewTrade(uint256 _tradeID, address _tradeOwner, address _tokenA, uint256 _amountA, address _tokenB, uint256 _amountB);
242 event TradeCanceled(uint256 _tradeID);
243 event TradeExecuted(uint256 _tradeID, address _tradeOwner, address _tradeExecutor);
244
245
246 /*********************************
247 * CONSTANTS *********************
248 *********************************/
249
250 /*********************************
251 * VARIABLES *********************
252 *********************************/
253 enum TradeStatus { NA, ACTIVE, CLOSED, CANCELED }
254
```

```
255 struct Trade {
256 uint256 ID;
257 address tradeOwner;
258 address tokenA;
259 uint256 amountA;
260 address tokenB;
261 uint256 amountB;
262 uint256 creationDate;
263 uint256 closeDate;
264 TradeStatus status; // 0 Not available, 1 Active, 2 Closed, 3 Canceled
265 }
266
267 Trade[] public trades;
268
269 IUniswapV2Router01 public UniswapRouter;
270 address public UniswapRouterAddress;
271
272 uint256 executionFee = 100; // 1%
273
274 /******************************
275 * MODIFIER ********************
276 ******************************/
277
```

```
278 modifier isTradeOwner(uint256 _tradeID, address _tradeOwner) {
279 require(trades[_tradeID].tradeOwner == _tradeOwner, "TRADE: you are not the owner");
280 _;
281 }
282
283 modifier isNotTradeOwner(uint256 _tradeID, address _tradeOwner) {
284 require(trades[_tradeID].tradeOwner != _tradeOwner, "TRADE: you are the owner");
285 _;
286 }
287
288 modifier isTradeStatus(uint256 _tradeID, TradeStatus _tradeStatus) {
289 require(trades[_tradeID].status == _tradeStatus, "TRADE: status is not valid");
290 _;
291 }
292
293 /*******************************
294 * COSTRUCTOR ******************
295 *******************************/
296 constructor(address _router) public {
297 UniswapRouter = IUniswapV2Router01(_router);
298 UniswapRouterAddress = _router;
299 }
300
```

```
301 /*******************************
302 * FUNCTION *********************
303 *******************************/
304
305 function _createTrade(address _tradeOwner, address _tokenA, uint256 _amountA, address _tokenB, uint256 _amountB) private {
306 trades.push(Trade(0,_tradeOwner, _tokenA, _amountA, _tokenB, _amountB, block.timestamp, 0,TradeStatus.ACTIVE));
307 uint tradeID = trades.length - 1;
308 trades[tradeID].ID = tradeID;
309 emit NewTrade(tradeID, _tradeOwner, _tokenA, _amountA, _tokenB, _amountB);
310 }
311
312 function createTrade(address _tokenA, uint256 _amountA, address _tokenB, uint256 _amountB) external nonReentrant() {
313 ERC20 token = ERC20(_tokenA);
314
315 require(token.allowance(msg.sender,address(this)) >= _amountA, "ERC20: Allowance not enough");
316 require(token.balanceOf(msg.sender) >= _amountA, "ERC20: Balance is not enough");
317
318 uint256 initialAmount = token.balanceOf(address(this));
319 token.transferFrom(msg.sender, address(this), _amountA); // sposto i fondi all'interno dello smart contract
320 uint256 exactAmountOffered = token.balanceOf(address(this)).sub(initialAmount);
321
322 _createTrade(msg.sender, _tokenA, exactAmountOffered, _tokenB, _amountB);
323 }
```

```
324
325 function _transferToken(address _to, uint256 _amount, address _token) private {
326 if (_token == address(0)) {
327 payable(_to).transfer(_amount);
328 } else {
329 ERC20 token = ERC20(_token);
330 token.transfer(_to, _amount);
331 }
332 }
333
334 function _cancelTrade(uint256 _tradeID) private {
335 trades[_tradeID].status = TradeStatus.CANCELED;
336 trades[_tradeID].closeDate = block.timestamp;
337
338 address _to = trades[_tradeID].tradeOwner;
339 address _token = trades[_tradeID].tokenA;
340 uint256 _amount = trades[_tradeID].amountA;
```

```
341
342 // send back the funds to the Owner
343 _transferToken(_to, _amount, _token);
344 emit TradeCanceled(_tradeID);
345 }
346
347 function cancelTrade(uint256 _tradeID) isTradeOwner(_tradeID, msg.sender) isTradeStatus(_tradeID, TradeStatus.ACTIVE) external nonReentrant() {
348 _cancelTrade(_tradeID);
349 }
350
351 function revertTrade(uint256 _tradeID) isTradeStatus(_tradeID, TradeStatus.ACTIVE) onlyOwner() external nonReentrant() {
352 _cancelTrade(_tradeID);
353 }
354
355 function verifyTrade(uint256 _tradeID) isTradeStatus(_tradeID, TradeStatus.ACTIVE) external view returns(bool) {
356
357 address _tokenA = trades[_tradeID].tokenA;
358 uint256 _amountA = trades[_tradeID].amountA;
359 address _tokenB = trades[_tradeID].tokenB;
360 uint256 _amountB = trades[_tradeID].amountB;
361
362 address[] memory path = new address[](2);
363 path[0] = _tokenA;
364 path[1] = _tokenB;
365
366 uint[] memory amounts = UniswapRouter.getAmountsOut(_amountA, path);
```

```
368 // return true if profitable for Trade Owner point of view
369 if (amounts[1] >= _amountB) {
370 return(true);
371 } else {
372 return(false);
373 }
374 }
375
376 function verifyTradeWithFee(uint256 _tradeID) isTradeStatus(_tradeID, TradeStatus.ACTIVE) external view returns(bool) {
377
378 address _tokenA = trades[_tradeID].tokenA;
379
380 // take execution fee into account
381 uint256 _amountA = trades[_tradeID].amountA;
382 uint256 _feeAmount = _amountA.mul(executionFee).div(10000);
383 _amountA = _amountA.sub(_feeAmount);
384
385 address _tokenB = trades[_tradeID].tokenB;
386 uint256 _amountB = trades[_tradeID].amountB;
387
388 address[] memory path = new address[](2);
389 path[0] = _tokenA;
390 path[1] = _tokenB;
391
392 uint[] memory amounts = UniswapRouter.getAmountsOut(_amountA, path);
393
```

```
392 uint[] memory amounts = UniswapRouter.getAmountsOut(_amountA, path);
393
394 // return true if profitable for Trade Owner point of view
395 if (amounts[1] >= _amountB) {
396 return(true);
397 } else {
398 return(false);
399 }
400 }
401
402 // Trade Execution - everybody can trigger it and earn reward in case of profitable trade
403 function executeTradeWithFee(uint256 _tradeID) isTradeStatus(_tradeID, TradeStatus.ACTIVE) nonReentrant() external returns(bool) {
```

```
404
405 address _tokenA = trades[_tradeID].tokenA;
406
407 uint256 _amountA = trades[_tradeID].amountA;
408 uint256 _feeAmount = _amountA.mul(executionFee).div(10000);
409 _amountA = _amountA.sub(_feeAmount);
410
411 address _tokenB = trades[_tradeID].tokenB;
412 uint256 _amountB = trades[_tradeID].amountB;
413
414 address _tradeOwner = trades[_tradeID].tradeOwner;
415
416 address[] memory path = new address[](2);
417 path[0] = _tokenA;
418 path[1] = _tokenB;
419
420 uint[] memory amounts = UniswapRouter.getAmountsOut(_amountA, path);
421
422 // if return amount is at least what requested, then proceed with swap
423 if (amounts[1] >= _amountB) {
424 // profitable trade
425 // allowance for UniswapRouterV2 for the amount requested
426 ERC20 token = ERC20(_tokenA);
427 token.approve(UniswapRouterAddress, _amountA);
428
429 amounts = UniswapRouter.swapExactTokensForTokens(_amountA,_amountB, path, _tradeOwner, block.timestamp + 1);
430 // execution done and correct also for "inclusive transaction fee" token life RFI
431 require(amounts[1] >= _amountB, "TRADE: return tokens are below expectation");
432
```

```
433 // send fee to the trade executor
434 _transferToken(msg.sender, _feeAmount, _tokenA);
435
436 //TODO Mint HFS for TradeOwner and Trade Executor. It will add when HFS Token will be avaiable
437
438 // update the data structure
439 trades[_tradeID].closeDate = block.timestamp;
440 trades[_tradeID].status = TradeStatus.CLOSED;
441
442 emit TradeExecuted(_tradeID, _tradeOwner, msg.sender);
443 return(true);
444 } else {
445 return(false);
446 }
447 }
448
449 /*****************************
450 * GETTER *********************
451 *****************************/
452
453 // GETTER - current Blockchain Time
454 function currentTime() public view returns(uint256) {
455 return(block.timestamp);
456 }
457
```

```
458 // GETTER - numbers of deals
459 function totalTrades() public view returns(uint256) {
460 return(trades.length);
461 }
462
463 }
464
465 interface IUniswapV2Router01 {
```

LOW - SWC-134

Call with hardcoded gas amount.

The highlighted function call forwards a fixed amount of gas. This is discouraged as the gas cost of EVM instructions may change in the future, which could break this contract's assumptions. If this was done to prevent reentrancy attacks, consider alternative methods such as the checks-effects-interactions pattern or reentrancy locks instead.

Locations

```
273 require(token.allowance(msg.sender,address(this)) >= _amountOffered, "ERC20: Allowance not enough");
274 require(token.balanceOf(msg.sender) >= _amountOffered, "ERC20: Balance is not enough");
275 token.transferFrom(msg.sender, address(this), _amountOffered); // sposto i fondi all'interno dello smart contract
276 }
277 _createDeal(msg.sender, _tokenAddressOffer, _amountOffered, _tokenAddressRequest, _amountRequest);
```

Summary of the Audit

After an exhaustive review with the development team, both security and development fixes have been applied that have improved the performance and integrity of the contract. After reviewing the modified version of the contract it is safe to deploy.