MythX

REPORT 600F351BD79442001266D314

Created	Mon Jan 25 2021 21:16:11 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	contact@rubicon.finance

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
ee2cdb55-3f09-4f13-b2ac-1665a9583728	C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol	43

Analysis ee2cdb55-3f09-4f13-b2ac-1665a9583728 MythX

Started	Mon Jan 25 2021 21:16:18 GMT+0000 (Coordinated Universal Time)
Finished	Mon Jan 25 2021 22:02:41 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	C:\Users\Benjamin Hughes\Workspace\Rubicon\Rubicon_protocol\Contracts\RubiconMarket.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	35	8

ISSUES

MEDIUM Function could be marked as external.

SWC-000

The function definition of "setOwner" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
20  }
21
22  function setOwner(address owner_) public auth {
23  owner = owner_;
24  emit LogSetOwner(owner);
25  }
26
27  modifier auth {
```

MEDIUM Function could be marked as external.

SWC-000

The function definition of "totalSupply" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
120 /// @notice ERC-20 interface as derived from EIP-20
121 contract ERC20 {
122 function totalSupply() public view returns (uint256);
123
124 function balanceOf(address guy) public view returns (uint256);
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "balanceOf" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
122 function totalSupply() public view returns (uint256);
123
124 function balanceOf(address guy) public view returns (uint256);
125
126 function allowance(address src, address guy) public view returns (uint256);
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "allowance" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
124 function balanceOf(address guy) public view returns (uint256);
125
126 function allowance(address src, address guy) public view returns (uint256);
127
128 function approve(address guy, uint256 wad) public returns (bool);
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "approve" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
126 function allowance(address src, address guy) public view returns (uint256);
127
128 function approve(address guy, uint256 wad) public returns (bool);
129
130 function transfer(address dst, uint256 wad) public returns (bool);
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "transfer" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
128 function approve(address guy, uint256 wad) public returns (bool);
129
130 function transfer(address dst, uint256 wad) public returns (bool);
131
132 function transferFrom(
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "transferFrom" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
130 function transfer(address dst, uint256 wad) public returns (bool);
131
132 function transferFrom(
133 address src,
134 address dst,
135 uint256 wad
136 ) public returns (bool);
137 }
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "getOffer" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
275 }
276
277 function getOffer(uint256 id)
278 public
279 view
280 returns (
281 uint256,
282 ERC20,
283 uint256,
284 ERC20
285 )
286 {
287 OfferInfo memory offer = offers[id];
288 return (offer.pay_amt, offer.pay_gem, offer.buy_amt, offer.buy_gem);
289 }
290
291 /// @notice Below are the main public entrypoints
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "bump" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
291 /// @notice Below are the main public entrypoints
292
293 function bump(bytes32 id_) public can_buy(uint256(id_)) {
294 uint256 id = uint256(id_);
295 emit LogBump(
296 id_,
297 keccak256(abi.encodePacked(offers[id].pay_gem, offers[id].buy_gem)),
298 offers[id].owner,
299 offers[id].pay_gem,
300 offers[id].buy_gem,
301 uint128(offers[id].pay_amt),
302 uint128(offers[id].buy_amt),
303 offers[id].timestamp
304 );
305 }
306
307 /// @notice Accept a given `quantity` of an offer. Transfers funds from caller/taker to offer maker, and from market to caller/taker.
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "kill" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
416 }
417
418 function kill(bytes32 id) public {
419 require(cancel(uint256(id)));
420 }
421
422 function make(
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "make" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
420 }
421
422 function make(
423 ERC20 pay_gem,
424 ERC20 buy_gem,
425 uint128 pay_amt,
426 uint128 buy_amt
427 ) public returns (bytes32 id) {
428 return bytes32(offer(pay_amt, pay_gem, buy_amt, buy_gem));
429 }
430
431 /// @notice Key function to make a new offer. Takes funds from the caller into market escrow.
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "take" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
469 }
470
471 function take(bytes32 id, uint128 maxTakeAmount) public {
472 require(buy(uint256(id), maxTakeAmount));
473 }
474
475 function _next_id() internal returns (uint256) {
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "stop" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
522 }
523
524 function stop() public auth {
525 stopped = true;
526 }
527 }
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "make" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
616 // ---- Public entrypoints ---- //
617
618 function make(
619 ERC20 pay_gem,
620 ERC20 buy_gem,
621 uint128 pay_amt,
622 uint128 buy_amt
623 ) public returns (bytes32) {
624 return bytes32(offer(pay_amt, pay_gem, buy_amt, buy_gem));
625 }
626
627 function take(bytes32 id, uint128 maxTakeAmount) public {
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "kill" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
629 }
630
631 function kill(bytes32 id) public {
632 require(cancel(uint256(id)));
633 }
634
635 // Routing function to make a trade where the user is sending Native ETH
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "offerInETH" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
634
635 // Routing function to make a trade where the user is sending Native ETH
636 function offerInETH(
637 uint256 buy_amt, //taker (ask) buy how much
638 ERC20 buy_gem //taker (ask) buy which token
639 ) public payable returns (uint256) {
640 require(!locked, "Reentrancy attempt");
641
642 IWETH(WETHAddress).deposit.value(msg.value)();
643 // IWETH(WETHAddress).approve(address(this), msg.value);
644 IWETH(WETHAddress).transfer(msg.sender, msg.value);
645
646 ERC20 WETH = ERC20(WETHAddress);
647
648 // Not sure which route to use here...
649 //Push Normal Order with WETH
650 // function (uint256,ERC20,uint256,ERC20) returns (uint256) fn = matchingEnabled ? _offeru : super.offer;
651 // return fn(msg.value, WETH, buy_amt, buy_gem);
652
653 if (matchingEnabled) {
654 return _matcho(msg.value, WETH, buy_amt, buy_gem, 0, true);
655 }
656 return super.offer(msg.value, WETH, buy_amt, buy_gem);
657 }
658
659 function buyInETH(uint256 id) public payable can_buy(id) returns (bool) {
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "buyInETH" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
657 }
658
659 function buyInETH(uint256 id) public payable can_buy(id) returns (bool) {
660 require(!locked, "Reentrancy attempt");
661 ERC20 WETH = ERC20(WETHAddress);
662 require(offers[id].buy_gem == WETH, "offer you buy must be in WETH");
663 IWETH(WETHAddress).deposit.value(msg.value)();
664 IWETH(WETHAddress).transfer(msg.sender, msg.value);
665
666 super.buy(id, msg.value);
667 }
668
669 // Make a new offer. Takes funds from the caller into market escrow.
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "offer" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
694
695 // Make a new offer. Takes funds from the caller into market escrow.
696 function offer(
697 uint256 pay_amt, //maker (ask) sell how much
698 ERC20 pay_gem, //maker (ask) sell which token
699 uint256 buy_amt, //maker (ask) buy how much
700 ERC20 buy_gem, //maker (ask) buy which token
701 uint256 pos //position to insert offer, 0 should be used if unknown
702 ) public can_offer returns (uint256) {
703 return offer(pay_amt, pay_gem, buy_amt, buy_gem, pos, true);
704 }
705
706 function offer(
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "insert" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
753 //insert offer into the sorted list
754 //keepers need to use this function
755 function insert(
756 uint256 id, //maker (ask) id
757 uint256 pos //position to insert into
758 ) public returns (bool) {
759 require(!locked, "Reentrancy attempt");
760 require(!isOfferSorted(id)); //make sure offers[id] is not yet sorted
761 require(isActive(id)); //make sure offers[id] is active
762
763 _hide(id); //remove offer from unsorted offers list
764 _sort(id, pos); //put offer into the sorted offers list
765 emit LogInsert(msg.sender, id);
766 return true;
767 }
768
769 //deletes _rank [id]
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "del_rank" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
769 //deletes _rank [id]
770 // Function should be called by keepers.
771 function del_rank(uint256 id) public returns (bool) {
772 require(!locked, "Reentrancy attempt");
773 require(
774 !isActive(id) &&
775 _rank[id].delb != 0 &&
776 _rank[id].delb < block.number - 10
777 );
778 delete _rank[id];
779 emit LogDelete(msg.sender, id);
780 return true;
781 }
782
783 //set the minimum sell amount for a token
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "setMinSell" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
786 // cost more gas to accept the offer, than the value
787 // of tokens received.
788 function setMinSell(
789 ERC20 pay_gem, //token to assign minimum sell amount to
790 uint256 dust //maker (ask) minimum sell amount
791 ) public auth note returns (bool) {
792 _dust[address(pay_gem)] = dust;
793 emit LogMinSell(address(pay_gem), dust);
794 return true;
795 }
796
797 //returns the minimum sell amount for an offer
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "getMinSell" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
796
797 //returns the minimum sell amount for an offer
798 function getMinSell(
799 ERC20 pay_gem //token for which minimum sell amount is queried
800 ) public view returns (uint256) {
801 return _dust[address(pay_gem)];
802 }
803
804 //set buy functionality enabled/disabled
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "setBuyEnabled" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
803
804 //set buy functionality enabled/disabled
805 function setBuyEnabled(bool buyEnabled_) public auth returns (bool) {
806 buyEnabled = buyEnabled_;
807 emit LogBuyEnabled(buyEnabled);
808 return true;
809 }
810
811 //set matching enabled/disabled
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "setMatchingEnabled" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
816 // If matchingEnabled is false then RubiconMarket is reverted to ExpiringMarket,
817 // and matching is not done, and sorted lists are disabled.
818 function setMatchingEnabled(bool matchingEnabled_)
819 public
820 auth
821 returns (bool)
822 {
823 matchingEnabled = matchingEnabled_;
824 emit LogMatchingEnabled(matchingEnabled);
825 return true;
826 }
827
828 //return the best offer for a token pair
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "getBetterOffer" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
849 // the next higher priced one if its a bid offer
850 // and in both cases the older one if they're equal.
851 function getBetterOffer(uint256 id) public view returns (uint256) {
852 return _rank[id].next;
853 }
854
855 //return the amount of better offers for a token pair
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "getOfferCount" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
854
855 //return the amount of better offers for a token pair
856 function getOfferCount(ERC20 sell_gem, ERC20 buy_gem)
857 public
858 view
859 returns (uint256)
860 {
861 return _span[address(sell_gem)][address(buy_gem)];
862 }
863
864 //get the first unsorted offer that was inserted by a contract
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "getFirstUnsortedOffer" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
867 // Keepers can calculate the insertion position offchain and pass it to the insert() function to insert
868 // the unsorted offer into the sorted list. Unsorted offers will not be matched, but can be bought with buy().
869 function getFirstUnsortedOffer() public view returns (uint256) {
870 return _head;
871 }
872
873 //get the next unsorted offer
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "getNextUnsortedOffer" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
873 //get the next unsorted offer
874 // Can be used to cycle through all the unsorted offers.
875 function getNextUnsortedOffer(uint256 id) public view returns (uint256) {
876 return _near[id];
877 }
878
879 function isOfferSorted(uint256 id) public view returns (bool) {
```

MEDIUM Function could be marked as external.

SWC-000 The function definition of "sellAllAmount" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
885 }
886
887 function sellAllAmount(
888 ERC20 pay_gem,
889 uint256 pay_amt,
890 ERC20 buy_gem,
891 uint256 min_fill_amount
892 ) public returns (uint256 fill_amt) {
893 require(!locked, "Reentrancy attempt");
894 uint256 offerId;
895 while (pay_amt > 0) {
896 //while there is amount to sell
897 offerId = getBestOffer(buy_gem, pay_gem); //Get the best offer for the token pair
898 require(offerId != 0); //Fails if there are not more offers
899
900 // There is a chance that pay_amt is smaller than 1 wei of the other token
901 if (
902 pay_amt * 1 ether <
903 wdiv(offers[offerId].buy_amt, offers[offerId].pay_amt)
904 ) {
905 break; //We consider that all amount is sold
906 }
907 if (pay_amt >= offers[offerId].buy_amt) {
908 //If amount to sell is higher or equal than current offer amount to buy
909 fill_amt = add(fill_amt, offers[offerId].pay_amt); //Add amount bought to acumulator
910 pay_amt = sub(pay_amt, offers[offerId].buy_amt); //Decrease amount to sell
911 take(bytes32(offerId), uint128(offers[offerId].pay_amt)); //We take the whole offer
912 } else {
913 // if lower
914 uint256 baux =
915 rmul(
916 pay_amt * 10**9,
917 rdiv(offers[offerId].pay_amt, offers[offerId].buy_amt)
918 ) / 10**9;
919 fill_amt = add(fill_amt, baux); //Add amount bought to acumulator
920 take(bytes32(offerId), uint128(baux)); //We take the portion of the offer that we need
921 pay_amt = 0; //All amount is sold
922 }
923 }
924 require(fill_amt >= min_fill_amount);
925 }
926
927 function buyAllAmount(
```

MEDIUM Function could be marked as external.

SWC-000 The function definition of "buyAllAmount" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
925 }
926
927 function buyAllAmount(
928 ERC20 buy_gem,
929 uint256 buy_amt,
930 ERC20 pay_gem,
931 uint256 max_fill_amount
932 ) public returns (uint256 fill_amt) {
933 require(!locked, "Reentrancy attempt");
934 uint256 offerId;
935 while (buy_amt > 0) {
936 //Meanwhile there is amount to buy
937 offerId = getBestOffer(buy_gem, pay_gem); //Get the best offer for the token pair
938 require(offerId != 0);
939
940 // There is a chance that buy_amt is smaller than 1 wei of the other token
941 if (
942 buy_amt * 1 ether <
943 wdiv(offers[offerId].pay_amt, offers[offerId].buy_amt)
944 ) {
945 break; //We consider that all amount is sold
946 }
947 if (buy_amt >= offers[offerId].pay_amt) {
948 //If amount to buy is higher or equal than current offer amount to sell
949 fill_amt = add(fill_amt, offers[offerId].buy_amt); //Add amount sold to acumulator
950 buy_amt = sub(buy_amt, offers[offerId].pay_amt); //Decrease amount to buy
951 take(bytes32(offerId), uint128(offers[offerId].pay_amt)); //We take the whole offer
952 } else {
953 //if lower
954 fill_amt = add(
955 fill_amt,
956 rmul(
957 buy_amt * 10**9,
958 rdiv(offers[offerId].buy_amt, offers[offerId].pay_amt)
959 ) / 10**9
960 ); //Add amount sold to acumulator
961 take(bytes32(offerId), uint128(buy_amt)); //We take the portion of the offer that we need
962 buy_amt = 0; //All amount is bought
963 }
964 }
965 require(fill_amt <= max_fill_amount);
966 }
967
968 function getBuyAmount(
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "getBuyAmount" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
966 }
967
968 function getBuyAmount(
969 ERC20 buy_gem,
970 ERC20 pay_gem,
971 uint256 pay_amt
972 ) public view returns (uint256 fill_amt) {
973 uint256 offerId = getBestOffer(buy_gem, pay_gem); //Get best offer for the token pair
974 while (pay_amt > offers[offerId].buy_amt) {
975 fill_amt = add(fill_amt, offers[offerId].pay_amt); //Add amount to buy accumulator
976 pay_amt = sub(pay_amt, offers[offerId].buy_amt); //Decrease amount to pay
977 if (pay_amt > 0) {
978 //If we still need more offers
979 offerId = getWorseOffer(offerId); //We look for the next best offer
980 require(offerId != 0); //Fails if there are not enough offers to complete
981 }
982 }
983 fill_amt = add(
984 fill_amt,
985 rmul(
986 pay_amt * 10**9,
987 rdiv(offers[offerId].pay_amt, offers[offerId].buy_amt)
988 ) / 10**9
989 ); //Add proportional amount of last offer to buy accumulator
990 }
991
992 function getPayAmount(
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "getPayAmount" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
990  }
991
992  function getPayAmount(
993  ERC20 pay_gem,
994  ERC20 buy_gem,
995  uint256 buy_amt
996  ) public view returns (uint256 fill_amt) {
997  uint256 offerId = getBestOffer(buy_gem, pay_gem); //Get best offer for the token pair
998  while (buy_amt > offers[offerId].pay_amt) {
999  fill_amt = add(fill_amt, offers[offerId].buy_amt); //Add amount to pay accumulator
1000 buy_amt = sub(buy_amt, offers[offerId].pay_amt); //Decrease amount to buy
1001 if (buy_amt > 0) {
1002 //If we still need more offers
1003 offerId = getWorseOffer(offerId); //We look for the next best offer
1004 require(offerId != 0); //Fails if there are not enough offers to complete
1005 }
1006 }
1007 fill_amt = add(
1008 fill_amt,
1009 rmul(
1010 buy_amt * 10**9,
1011 rdiv(offers[offerId].buy_amt, offers[offerId].pay_amt)
1012 ) / 10**9
1013 ); //Add proportional amount of last offer to pay accumulator
1014 }
1015
1016 // ---- Internal Functions ---- //
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "setFeeBPS" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
1287 }
1288
1289 function setFeeBPS(uint256 _newFeeBPS) public auth returns (bool) {
1290 feeBPS = _newFeeBPS;
1291 return true;
1292 }
1293
1294 function setAqueductDistributionLive(bool live) public auth returns (bool) {
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "setAqueductDistributionLive" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
1292 }
1293
1294 function setAqueductDistributionLive(bool live) public auth returns (bool) {
1295 AqueductDistributionLive = live;
1296 return true;
1297 }
1298
1299 function setAqueductAddress(address _Aqueduct) public auth returns (bool) {
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "setAqueductAddress" 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
1297 }
1298
1299 function setAqueductAddress(address _Aqueduct) public auth returns (bool) {
1300 AqueductAddress = _Aqueduct;
1301 return true;
1302 }
1303 }
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.12"". 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
2 /// @notice Please see the GNU General Public License for this code at https://github.com/RubiconDeFi/rubicon_protocol
3
4 pragma solidity ^0.5.12;
5
6 /// @notice DSAuth events for authentication schema
```

LOW

SWC-108

State variable visibility is not set.

It is best practice to set the visibility of state variables explicitly. The default visibility for "locked" is internal. Other possible visibility settings are public and private.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
218 mapping(uint256 => OfferInfo) public offers;
219
220 bool locked;
221
222 /// @notice This parameter provides the ability for a protocol fee on taker trades
```

LOW

SWC-108

State variable visibility is not set.

It is best practice to set the visibility of state variables explicitly. The default visibility for "_head" is internal. Other possible visibility settings are public and private.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
581 mapping(address => uint256) public _dust; //minimum sell amount for a token to avoid dust offers
582 mapping(uint256 => uint256) public _near; //next unsorted offer id
583 uint256 _head; //first unsorted offer id
584 uint256 public dustId; // id of the latest offer marked as dust
585 address public AqueductAddress;
```

LOW

SWC-110

An assertion violation was triggered.

It is possible to cause an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
83
84 function wdiv(uint256 x, uint256 y) internal pure returns (uint256 z) {
85 z = add(mul(x, WAD), y / 2) / y;
86 }
```

LOW

SWC-119

Function parameter shadows a state variable.

The function parameter "close_time" in contract "RubiconMarket" shadows the state variable with the same name "close_time" in contract "ExpiringMarket".

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
592
593 constructor(
594 uint64 close_time,
595 // address aqueduct,
596 bool RBCNDist,
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
774 !isActive(id) &&
775 _rank[id].delb != 0 &&
776 _rank[id].delb < block.number - 10
777 );
778 delete _rank[id];
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. 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.

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
1254
1255 _span[pay_gem][buy_gem]--;
1256 _rank[id].delb = block.number; //mark _rank[id] for deletion
1257 return true;
1258 }
```

LOW

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).

SWC-123

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
454 offers[id] = info;
455
456 require(pay_gem.transferFrom(msg.sender, address(this), pay_amt));
457
458 emit LogItemUpdate(id);
```

Source file

C:\Users\Benjamin Hughes\workspace\rubicon\rubicon_protocol\contracts\RubiconMarket.sol

Locations

```
567 /// @notice This contract is based on the original open-source work done by OasisDEX under the Apache License 2.0
568 /// @dev This contract inherits the key trading functionality from SimpleMarket
569 contract RubiconMarket is MatchingEvents, ExpiringMarket, DSNote {
570 bool public buyEnabled = true; //buy enabled
571 bool public matchingEnabled = true; //true: enable matching,
572 //false: revert to expiring market
573 struct sortInfo {
574 uint256 next; //points to id of next higher offer
575 uint256 prev; //points to id of previous lower offer
576 uint256 delb; //the blocknumber where this entry was marked for delete
577 }
578 mapping(uint256 => sortInfo) public _rank; //doubly linked lists of sorted offer ids
579 mapping(address => mapping(address => uint256)) public _best; //id of the highest offer for a token pair
580 mapping(address => mapping(address => uint256)) public _span; //number of offers stored for token pair in sorted orderbook
581 mapping(address => uint256) public _dust; //minimum sell amount for a token to avoid dust offers
582 mapping(uint256 => uint256) public _near; //next unsorted offer id
583 uint256 _head; //first unsorted offer id
584 uint256 public dustId; // id of the latest offer marked as dust
585 address public AqueductAddress;
586 bool public AqueductDistributionLive;
587
588 //TODO: for Mainnnet deployment, WETH address will be hard coded as below
589 // address public WETHAddress = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;
590 /// @dev Below is Kovan WETH Address
591 address public WETHAddress; //= 0x772c16c1dD9cC51fe601B6bA8c8B2feF074528f1;
592
593 constructor(
594 uint64 close_time,
595 // address aqueduct,
596 bool RBCNDist,
597 address _feeTo,
598 address WETH
599 ) public ExpiringMarket(close_time) SimpleMarket(_feeTo) {
600 // AqueductAddress = aqueduct;
601 AqueductDistributionLive = RBCNDist;
602 /*For Testing Only:*/
603 WETHAddress = WETH;
604 }
605
606 // After close, anyone can cancel an offer
607 modifier can_cancel(uint256 id) {
608 require(isActive(id), "Offer was deleted or taken, or never existed.");
609 require(
610 isClosed() || msg.sender == getOwner(id) || id == dustId,
611 "Offer can not be cancelled because user is not owner, and market is open, and offer sells required amount of tokens."
```

```
);
_;
}

// ---- Public entrypoints ---- //

function make(
ERC20 pay_gem,
ERC20 buy_gem,
uint128 pay_amt,
uint128 buy_amt
) public returns (bytes32) {
return bytes32(offer(pay_amt, pay_gem, buy_amt, buy_gem));
}

function take(bytes32 id, uint128 maxTakeAmount) public {
require(buy(uint256(id), maxTakeAmount));
}

function kill(bytes32 id) public {
require(cancel(uint256(id)));
}

// Routing function to make a trade where the user is sending Native ETH
function offerInETH(
uint256 buy_amt, //taker (ask) buy how much
ERC20 buy_gem //taker (ask) buy which token
) public payable returns (uint256) {
require(!locked, "Reentrancy attempt");

IWETH(WETHAddress).deposit.value(msg.value)();
// IWETH(WETHAddress).approve(address(this), msg.value);
IWETH(WETHAddress).transfer(msg.sender, msg.value);

ERC20 WETH = ERC20(WETHAddress);

// Not sure which route to use here...
//Push Normal Order with WETH
// function (uint256,ERC20,uint256,ERC20) returns (uint256) fn = matchingEnabled ? _offeru : super.offer;
// return fn(msg.value, WETH, buy_amt, buy_gem);

if (matchingEnabled) {
return _matcho(msg.value, WETH, buy_amt, buy_gem, 0, true);
}
return super.offer(msg.value, WETH, buy_amt, buy_gem);
}

function buyInETH(uint256 id) public payable can_buy(id) returns (bool) {
require(!locked, "Reentrancy attempt");
ERC20 WETH = ERC20(WETHAddress);
require(offers[id].buy_gem == WETH, "offer you buy must be in WETH");
IWETH(WETHAddress).deposit.value(msg.value)();
IWETH(WETHAddress).transfer(msg.sender, msg.value);

super.buy(id, msg.value);
}

// Make a new offer. Takes funds from the caller into market escrow.
//
// If matching is enabled:
// * creates new offer without putting it in
// the sorted list.
// * available to authorized contracts only!
```

```
// * keepers should call insert(id,pos)
// to put offer in the sorted list.
//
// If matching is disabled:
// * calls expiring market's offer().
// * available to everyone without authorization.
// * no sorting is done.
//
function offer(
uint256 pay_amt, //maker (ask) sell how much
ERC20 pay_gem, //maker (ask) sell which token
uint256 buy_amt, //taker (ask) buy how much
ERC20 buy_gem //taker (ask) buy which token
) public returns (uint256) {
require(!locked, "Reentrancy attempt");
function(uint256, ERC20, uint256, ERC20) returns (uint256) fn =
matchingEnabled ? _offeru : super.offer;
return fn(pay_amt, pay_gem, buy_amt, buy_gem);
}

// Make a new offer. Takes funds from the caller into market escrow.
function offer(
uint256 pay_amt, //maker (ask) sell how much
ERC20 pay_gem, //maker (ask) sell which token
uint256 buy_amt, //maker (ask) buy how much
ERC20 buy_gem, //maker (ask) buy which token
uint256 pos //position to insert offer, 0 should be used if unknown
) public can_offer returns (uint256) {
return offer(pay_amt, pay_gem, buy_amt, buy_gem, pos, true);
}

function offer(
uint256 pay_amt, //maker (ask) sell how much
ERC20 pay_gem, //maker (ask) sell which token
uint256 buy_amt, //maker (ask) buy how much
ERC20 buy_gem, //maker (ask) buy which token
uint256 pos, //position to insert offer, 0 should be used if unknown
bool matching //match "close enough" orders?
) public can_offer returns (uint256) {
require(!locked, "Reentrancy attempt");
require(_dust[address(pay_gem)] <= pay_amt);

if (matchingEnabled) {
return _matcho(pay_amt, pay_gem, buy_amt, buy_gem, pos, matching);
}
return super.offer(pay_amt, pay_gem, buy_amt, buy_gem);
}

//Transfers funds from caller to offer maker, and from market to caller.
function buy(uint256 id, uint256 amount) public can_buy(id) returns (bool) {
require(!locked, "Reentrancy attempt");

//RBCN distribution on the trade
if (AqueductDistributionLive) {
IAqueduct(AqueductAddress).distributeToMakerAndTaker(
getOwner(id),
msg.sender
);
}
function(uint256, uint256) returns (bool) fn =
matchingEnabled ? _buys : super.buy; //<conditional> ? <if-true> : <if-false> --- Offers with matching enabled that get matched? are routed via _matcho into this buy

return fn(id, amount);
```

```
}

// Cancel an offer. Refunds offer maker.
function cancel(uint256 id) public can_cancel(id) returns (bool success) {
require(!locked, "Reentrancy attempt");
if (matchingEnabled) {
if (isOfferSorted(id)) {
require(_unsort(id));
} else {
require(_hide(id));
}
}
return super.cancel(id); //delete the offer.
}

//insert offer into the sorted list
//keepers need to use this function
function insert(
uint256 id, //maker (ask) id
uint256 pos //position to insert into
) public returns (bool) {
require(!locked, "Reentrancy attempt");
require(!isOfferSorted(id)); //make sure offers[id] is not yet sorted
require(isActive(id)); //make sure offers[id] is active

_hide(id); //remove offer from unsorted offers list
_sort(id, pos); //put offer into the sorted offers list
emit LogInsert(msg.sender, id);
return true;
}

//deletes _rank [id]
// Function should be called by keepers.
function del_rank(uint256 id) public returns (bool) {
require(!locked, "Reentrancy attempt");
require(
!isActive(id) &&
_rank[id].delb != 0 &&
_rank[id].delb < block.number - 10
);
delete _rank[id];
emit LogDelete(msg.sender, id);
return true;
}

//set the minimum sell amount for a token
// Function is used to avoid "dust offers" that have
// very small amount of tokens to sell, and it would
// cost more gas to accept the offer, than the value
// of tokens received.
function setMinSell(
ERC20 pay_gem, //token to assign minimum sell amount to
uint256 dust //maker (ask) minimum sell amount
) public auth note returns (bool) {
_dust[address(pay_gem)] = dust;
emit LogMinSell(address(pay_gem), dust);
return true;
}

//returns the minimum sell amount for an offer
function getMinSell(
ERC20 pay_gem //token for which minimum sell amount is queried
) public view returns (uint256) {
```

```
return _dust[address(pay_gem)];
}

//set buy functionality enabled/disabled
function setBuyEnabled(bool buyEnabled_) public auth returns (bool) {
buyEnabled = buyEnabled_;
emit LogBuyEnabled(buyEnabled);
return true;
}

//set matching enabled/disabled
// If matchingEnabled true(default), then inserted offers are matched.
// Except the ones inserted by contracts, because those end up
// in the unsorted list of offers, that must be later sorted by
// keepers using insert().
// If matchingEnabled is false then RubiconMarket is reverted to ExpiringMarket,
// and matching is not done, and sorted lists are disabled.
function setMatchingEnabled(bool matchingEnabled_)
public
auth
returns (bool)
{
matchingEnabled = matchingEnabled_;
emit LogMatchingEnabled(matchingEnabled);
return true;
}

//return the best offer for a token pair
// the best offer is the lowest one if it's an ask,
// and highest one if it's a bid offer
function getBestOffer(ERC20 sell_gem, ERC20 buy_gem)
public
view
returns (uint256)
{
return _best[address(sell_gem)][address(buy_gem)];
}

//return the next worse offer in the sorted list
// the worse offer is the higher one if its an ask,
// a lower one if its a bid offer,
// and in both cases the newer one if they're equal.
function getWorseOffer(uint256 id) public view returns (uint256) {
return _rank[id].prev;
}

//return the next better offer in the sorted list
// the better offer is in the lower priced one if its an ask,
// the next higher priced one if its a bid offer
// and in both cases the older one if they're equal.
function getBetterOffer(uint256 id) public view returns (uint256) {
return _rank[id].next;
}

//return the amount of better offers for a token pair
function getOfferCount(ERC20 sell_gem, ERC20 buy_gem)
public
view
returns (uint256)
{
return _span[address(sell_gem)][address(buy_gem)];
}

```

```
//get the first unsorted offer that was inserted by a contract
// Contracts can't calculate the insertion position of their offer because it is not an O(1) operation.
// Their offers get put in the unsorted list of offers.
// Keepers can calculate the insertion position offchain and pass it to the insert() function to insert
// the unsorted offer into the sorted list. Unsorted offers will not be matched, but can be bought with buy().
function getFirstUnsortedOffer() public view returns (uint256) {
return _head;
}

//get the next unsorted offer
// Can be used to cycle through all the unsorted offers.
function getNextUnsortedOffer(uint256 id) public view returns (uint256) {
return _near[id];
}

function isOfferSorted(uint256 id) public view returns (bool) {
return
_rank[id].next != 0 ||
_rank[id].prev != 0 ||
_best[address(offers[id].pay_gem)][address(offers[id].buy_gem)] ==
id;
}

function sellAllAmount(
ERC20 pay_gem,
uint256 pay_amt,
ERC20 buy_gem,
uint256 min_fill_amount
) public returns (uint256 fill_amt) {
require(!locked, "Reentrancy attempt");
uint256 offerId;
while (pay_amt > 0) {
//while there is amount to sell
offerId = getBestOffer(buy_gem, pay_gem); //Get the best offer for the token pair
require(offerId != 0); //Fails if there are not more offers

// There is a chance that pay_amt is smaller than 1 wei of the other token
if (
pay_amt * 1 ether <
wdiv(offers[offerId].buy_amt, offers[offerId].pay_amt)
) {
break; //We consider that all amount is sold
}
if (pay_amt >= offers[offerId].buy_amt) {
//If amount to sell is higher or equal than current offer amount to buy
fill_amt = add(fill_amt, offers[offerId].pay_amt); //Add amount bought to acumulator
pay_amt = sub(pay_amt, offers[offerId].buy_amt); //Decrease amount to sell
take(bytes32(offerId), uint128(offers[offerId].pay_amt)); //We take the whole offer
} else {
// if lower
uint256 baux =
rmul(
pay_amt * 10**9,
rdiv(offers[offerId].pay_amt, offers[offerId].buy_amt)
) / 10**9;
fill_amt = add(fill_amt, baux); //Add amount bought to acumulator
take(bytes32(offerId), uint128(baux)); //We take the portion of the offer that we need
pay_amt = 0; //All amount is sold
}
}
require(fill_amt >= min_fill_amount);
}

```

```
function buyAllAmount(
ERC20 buy_gem,
uint256 buy_amt,
ERC20 pay_gem,
uint256 max_fill_amount
) public returns (uint256 fill_amt) {
require(!locked, "Reentrancy attempt");
uint256 offerId;
while (buy_amt > 0) {
//Meanwhile there is amount to buy
offerId = getBestOffer(buy_gem, pay_gem); //Get the best offer for the token pair
require(offerId != 0);

// There is a chance that buy_amt is smaller than 1 wei of the other token
if (
buy_amt * 1 ether <
wdiv(offers[offerId].pay_amt, offers[offerId].buy_amt)
) {
break; //We consider that all amount is sold
}
if (buy_amt >= offers[offerId].pay_amt) {
//If amount to buy is higher or equal than current offer amount to sell
fill_amt = add(fill_amt, offers[offerId].buy_amt); //Add amount sold to acumulator
buy_amt = sub(buy_amt, offers[offerId].pay_amt); //Decrease amount to buy
take(bytes32(offerId), uint128(offers[offerId].pay_amt)); //We take the whole offer
} else {
//if lower
fill_amt = add(
fill_amt,
rmul(
buy_amt * 10**9,
rdiv(offers[offerId].buy_amt, offers[offerId].pay_amt)
) / 10**9
); //Add amount sold to acumulator
take(bytes32(offerId), uint128(buy_amt)); //We take the portion of the offer that we need
buy_amt = 0; //All amount is bought
}
}
require(fill_amt <= max_fill_amount);
}

function getBuyAmount(
ERC20 buy_gem,
ERC20 pay_gem,
uint256 pay_amt
) public view returns (uint256 fill_amt) {
uint256 offerId = getBestOffer(buy_gem, pay_gem); //Get best offer for the token pair
while (pay_amt > offers[offerId].buy_amt) {
fill_amt = add(fill_amt, offers[offerId].pay_amt); //Add amount to buy accumulator
pay_amt = sub(pay_amt, offers[offerId].buy_amt); //Decrease amount to pay
if (pay_amt > 0) {
//If we still need more offers
offerId = getWorseOffer(offerId); //We look for the next best offer
require(offerId != 0); //Fails if there are not enough offers to complete
}
}
fill_amt = add(
fill_amt,
rmul(
pay_amt * 10**9,
rdiv(offers[offerId].pay_amt, offers[offerId].buy_amt)
) / 10**9
); //Add proportional amount of last offer to buy accumulator
```

```
}

function getPayAmount(
ERC20 pay_gem,
ERC20 buy_gem,
uint256 buy_amt
) public view returns (uint256 fill_amt) {
uint256 offerId = getBestOffer(buy_gem, pay_gem); //Get best offer for the token pair
while (buy_amt > offers[offerId].pay_amt) {
fill_amt = add(fill_amt, offers[offerId].buy_amt); //Add amount to pay accumulator
buy_amt = sub(buy_amt, offers[offerId].pay_amt); //Decrease amount to buy
if (buy_amt > 0) {
//If we still need more offers
offerId = getWorseOffer(offerId); //We look for the next best offer
require(offerId != 0); //Fails if there are not enough offers to complete
}
}
fill_amt = add(
fill_amt,
rmul(
buy_amt * 10**9,
rdiv(offers[offerId].buy_amt, offers[offerId].pay_amt)
) / 10**9
); //Add proportional amount of last offer to pay accumulator
}

// ---- Internal Functions ---- //

function _buys(uint256 id, uint256 amount) internal returns (bool) {
require(buyEnabled);
if (amount == offers[id].pay_amt) {
if (isOfferSorted(id)) {
//offers[id] must be removed from sorted list because all of it is bought
_unsort(id);
} else {
_hide(id);
}
}

require(super.buy(id, amount));

// If offer has become dust during buy, we cancel it
if (
isActive(id) &&
offers[id].pay_amt < _dust[address(offers[id].pay_gem)]
) {
dustId = id; //enable current msg.sender to call cancel(id)
cancel(id);
}
return true;
}

//find the id of the next higher offer after offers[id]
function _find(uint256 id) internal view returns (uint256) {
require(id > 0);

address buy_gem = address(offers[id].buy_gem);
address pay_gem = address(offers[id].pay_gem);
uint256 top = _best[pay_gem][buy_gem];
uint256 old_top = 0;

// Find the larger-than-id order whose successor is less-than-id.
while (top != 0 && _isPricedLtOrEq(id, top)) {
```

```
old_top = top;
top = _rank[top].prev;
}
return old_top;
}

//find the id of the next higher offer after offers[id]
function _findpos(uint256 id, uint256 pos) internal view returns (uint256) {
require(id > 0);

// Look for an active order.
while (pos != 0 && !isActive(pos)) {
pos = _rank[pos].prev;
}

if (pos == 0) {
//if we got to the end of list without a single active offer
return _find(id);
} else {
// if we did find a nearby active offer
// Walk the order book down from there...
if (_isPricedLtOrEq(id, pos)) {
uint256 old_pos;

// Guaranteed to run at least once because of
// the prior if statements.
while (pos != 0 && _isPricedLtOrEq(id, pos)) {
old_pos = pos;
pos = _rank[pos].prev;
}
return old_pos;

// ...or walk it up.
} else {
while (pos != 0 && !_isPricedLtOrEq(id, pos)) {
pos = _rank[pos].next;
}
return pos;
}
}
}

//return true if offers[low] priced less than or equal to offers[high]
function _isPricedLtOrEq(
uint256 low, //lower priced offer's id
uint256 high //higher priced offer's id
) internal view returns (bool) {
return
mul(offers[low].buy_amt, offers[high].pay_amt) >=
mul(offers[high].buy_amt, offers[low].pay_amt);
}

//these variables are global only because of solidity local variable limit

//match offers with taker offer, and execute token transactions
function _matcho(
uint256 t_pay_amt, //taker sell how much
ERC20 t_pay_gem, //taker sell which token
uint256 t_buy_amt, //taker buy how much
ERC20 t_buy_gem, //taker buy which token
uint256 pos, //position id
bool rounding //match "close enough" orders?
) internal returns (uint256 id) {
```

```
uint256 best_maker_id; //highest maker id
uint256 t_buy_amt_old; //taker buy how much saved
uint256 m_buy_amt; //maker offer wants to buy this much token
uint256 m_pay_amt; //maker offer wants to sell this much token

// there is at least one offer stored for token pair
while (_best[address(t_buy_gem)][address(t_pay_gem)] > 0) {
best_maker_id = _best[address(t_buy_gem)][address(t_pay_gem)];
m_buy_amt = offers[best_maker_id].buy_amt;
m_pay_amt = offers[best_maker_id].pay_amt;

// Ugly hack to work around rounding errors. Based on the idea that
// the furthest the amounts can stray from their "true" values is 1.
// Ergo the worst case has t_pay_amt and m_pay_amt at +1 away from
// their "correct" values and m_buy_amt and t_buy_amt at -1.
// Since (c - 1) * (d - 1) > (a + 1) * (b + 1) is equivalent to
// c * d > a * b + a + b + c + d, we write...
if (
mul(m_buy_amt, t_buy_amt) >
mul(t_pay_amt, m_pay_amt) +
(
rounding
? m_buy_amt + t_buy_amt + t_pay_amt + m_pay_amt
: 0
)
) {
break;
}
// ^ The `rounding` parameter is a compromise borne of a couple days
// of discussion.
buy(best_maker_id, min(m_pay_amt, t_buy_amt));
t_buy_amt_old = t_buy_amt;
t_buy_amt = sub(t_buy_amt, min(m_pay_amt, t_buy_amt));
t_pay_amt = mul(t_buy_amt, t_pay_amt) / t_buy_amt_old;

if (t_pay_amt == 0 || t_buy_amt == 0) {
break;
}
}

if (
t_buy_amt > 0 &&
t_pay_amt > 0 &&
t_pay_amt >= _dust[address(t_pay_gem)]
) {
//new offer should be created
id = super.offer(t_pay_amt, t_pay_gem, t_buy_amt, t_buy_gem);
//insert offer into the sorted list
_sort(id, pos);
}
}

// Make a new offer without putting it in the sorted list.
// Takes funds from the caller into market escrow.
// Keepers should call insert(id,pos) to put offer in the sorted list.
function _offeru(
uint256 pay_amt, //maker (ask) sell how much
ERC20 pay_gem, //maker (ask) sell which token
uint256 buy_amt, //maker (ask) buy how much
ERC20 buy_gem //maker (ask) buy which token
) internal returns (uint256 id) {
require(_dust[address(pay_gem)] <= pay_amt);
id = super.offer(pay_amt, pay_gem, buy_amt, buy_gem);
```

```
_near[id] = _head;
_head = id;
emit LogUnsortedOffer(id);
}

//put offer into the sorted list
function _sort(
uint256 id, //maker (ask) id
uint256 pos //position to insert into
) internal {
require(isActive(id));

ERC20 buy_gem = offers[id].buy_gem;
ERC20 pay_gem = offers[id].pay_gem;
uint256 prev_id; //maker (ask) id

pos = pos == 0 ||
offers[pos].pay_gem != pay_gem ||
offers[pos].buy_gem != buy_gem ||
!isOfferSorted(pos)
? _find(id)
: _findpos(id, pos);

if (pos != 0) {
//offers[id] is not the highest offer
//requirement below is satisfied by statements above
//require(_isPricedLtOrEq(id, pos));
prev_id = _rank[pos].prev;
_rank[pos].prev = id;
_rank[id].next = pos;
} else {
//offers[id] is the highest offer
prev_id = _best[address(pay_gem)][address(buy_gem)];
_best[address(pay_gem)][address(buy_gem)] = id;
}

if (prev_id != 0) {
//if lower offer does exist
//requirement below is satisfied by statements above
//require(!_isPricedLtOrEq(id, prev_id));
_rank[prev_id].next = id;
_rank[id].prev = prev_id;
}

_span[address(pay_gem)][address(buy_gem)]++;
emit LogSortedOffer(id);
}

// Remove offer from the sorted list (does not cancel offer)
function _unsort(
uint256 id //id of maker (ask) offer to remove from sorted list
) internal returns (bool) {
address buy_gem = address(offers[id].buy_gem);
address pay_gem = address(offers[id].pay_gem);
require(_span[pay_gem][buy_gem] > 0);

require(
_rank[id].delb == 0 && //assert id is in the sorted list
isOfferSorted(id)
);

if (id != _best[pay_gem][buy_gem]) {
// offers[id] is not the highest offer
```

```
require(_rank[_rank[id].next].prev == id);
_rank[_rank[id].next].prev = _rank[id].prev;
} else {
//offers[id] is the highest offer
_best[pay_gem][buy_gem] = _rank[id].prev;
}

if (_rank[id].prev != 0) {
//offers[id] is not the lowest offer
require(_rank[_rank[id].prev].next == id);
_rank[_rank[id].prev].next = _rank[id].next;
}

_span[pay_gem][buy_gem]--;
_rank[id].delb = block.number; //mark _rank[id] for deletion
return true;
}

//Hide offer from the unsorted order book (does not cancel offer)
function _hide(
uint256 id //id of maker offer to remove from unsorted list
) internal returns (bool) {
uint256 uid = _head; //id of an offer in unsorted offers list
uint256 pre = uid; //id of previous offer in unsorted offers list

require(!isOfferSorted(id)); //make sure offer id is not in sorted offers list

if (_head == id) {
//check if offer is first offer in unsorted offers list
_head = _near[id]; //set head to new first unsorted offer
_near[id] = 0; //delete order from unsorted order list
return true;
}
while (uid > 0 && uid != id) {
//find offer in unsorted order list
pre = uid;
uid = _near[uid];
}
if (uid != id) {
//did not find offer id in unsorted offers list
return false;
}
_near[pre] = _near[id]; //set previous unsorted offer to point to offer after offer id
_near[id] = 0; //delete order from unsorted order list
return true;
}

function setFeeBPS(uint256 _newFeeBPS) public auth returns (bool) {
feeBPS = _newFeeBPS;
return true;
}

function setAqueductDistributionLive(bool live) public auth returns (bool) {
AqueductDistributionLive = live;
return true;
}

function setAqueductAddress(address _Aqueduct) public auth returns (bool) {
AqueductAddress = _Aqueduct;
return true;
}
}
```

```
1305
    interface IWETH {
```