**Semester End Practical Exam!!**

**Name:** Dhavalkumar Vijaykumar Patel

**Class:** M.Sc. Cyber Security Sem-3

**Enrollment Number:** 032200300002034

**Subject:** Blockchain and Cryptocurrencies (CTMSCS SIII L1)

**Date:** 10 January 2024

**Q.1 Create Solidity Program for the Following Task.**

1. **Basic smart contract that prints “Hello, World!” when executed**

// SPDX-License-Identifier: MIT

pragma solidity >=0.6.12 <0.9.0;

contract HelloWorld {

  /\*\*

   \* @dev Prints Hello World string

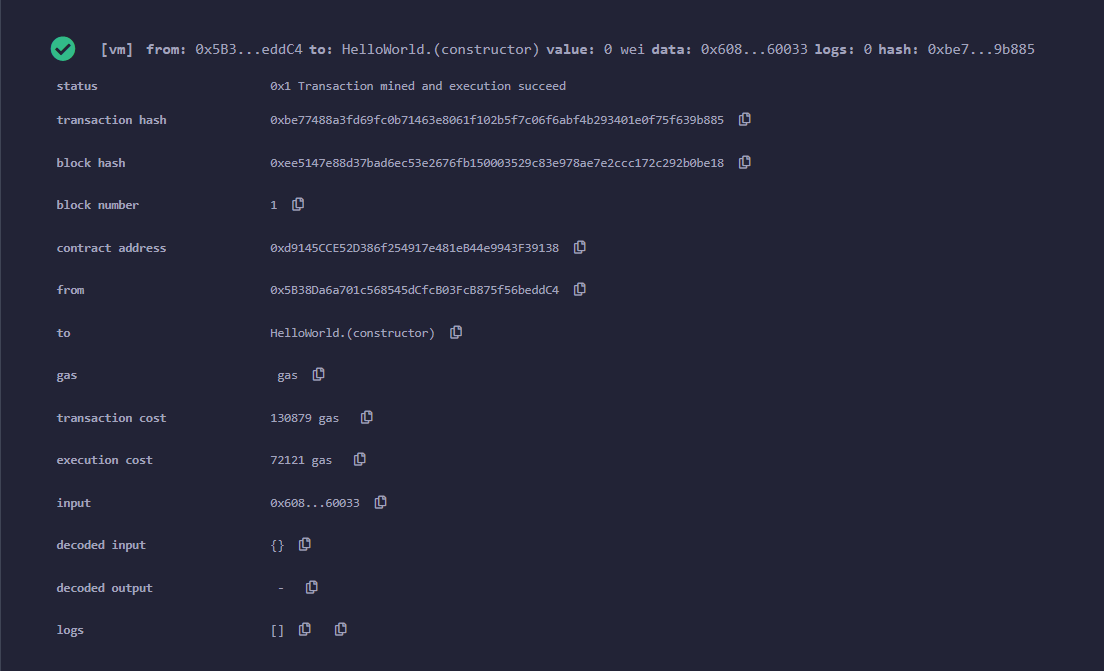
   \*/

  function print() public pure returns (string memory) {

    return "Hello World!";

  }

}



1. **Create a simple crowdfunding contract where participants can contribute fund and a project owner can withdraw the fund when goal is reached.**

// SPDX-License-Identifier: MIT

pragma solidity >=0.6.12 <0.9.0;

interface IERC20 {

    function transfer(address, uint) external returns (bool);

    function transferFrom(address, address, uint) external returns (bool);

}

contract CrowdFund {

    event Launch(

        uint id,

        address indexed creator,

        uint goal,

        uint32 startAt,

        uint32 endAt

    );

    event Cancel(uint id);

    event Pledge(uint indexed id, address indexed caller, uint amount);

    event Unpledge(uint indexed id, address indexed caller, uint amount);

    event Claim(uint id);

    event Refund(uint id, address indexed caller, uint amount);

    struct Campaign {

        // Creator of campaign

        address creator;

        // Amount of tokens to raise

        uint goal;

        // Total amount pledged

        uint pledged;

        // Timestamp of start of campaign

        uint32 startAt;

        // Timestamp of end of campaign

        uint32 endAt;

        // True if goal was reached and creator has claimed the tokens.

        bool claimed;

    }

    IERC20 public immutable token;

    // Total count of campaigns created.

    // It is also used to generate id for new campaigns.

    uint public count;

    // Mapping from id to Campaign

    mapping(uint => Campaign) public campaigns;

    // Mapping from campaign id => pledger => amount pledged

    mapping(uint => mapping(address => uint)) public pledgedAmount;

    constructor(address \_token) {

        token = IERC20(\_token);

    }

    function launch(uint \_goal, uint32 \_startAt, uint32 \_endAt) external {

        require(\_startAt >= block.timestamp, "start at < now");

        require(\_endAt >= \_startAt, "end at < start at");

        require(\_endAt <= block.timestamp + 90 days, "end at > max duration");

        count += 1;

        campaigns[count] = Campaign({

            creator: msg.sender,

            goal: \_goal,

            pledged: 0,

            startAt: \_startAt,

            endAt: \_endAt,

            claimed: false

        });

        emit Launch(count, msg.sender, \_goal, \_startAt, \_endAt);

    }

    function cancel(uint \_id) external {

        Campaign memory campaign = campaigns[\_id];

        require(campaign.creator == msg.sender, "not creator");

        require(block.timestamp < campaign.startAt, "started");

        delete campaigns[\_id];

        emit Cancel(\_id);

    }

    function pledge(uint \_id, uint \_amount) external {

        Campaign storage campaign = campaigns[\_id];

        require(block.timestamp >= campaign.startAt, "not started");

        require(block.timestamp <= campaign.endAt, "ended");

        campaign.pledged += \_amount;

        pledgedAmount[\_id][msg.sender] += \_amount;

        token.transferFrom(msg.sender, address(this), \_amount);

        emit Pledge(\_id, msg.sender, \_amount);

    }

    function unpledge(uint \_id, uint \_amount) external {

        Campaign storage campaign = campaigns[\_id];

        require(block.timestamp <= campaign.endAt, "ended");

        campaign.pledged -= \_amount;

        pledgedAmount[\_id][msg.sender] -= \_amount;

        token.transfer(msg.sender, \_amount);

        emit Unpledge(\_id, msg.sender, \_amount);

    }

    function claim(uint \_id) external {

        Campaign storage campaign = campaigns[\_id];

        require(campaign.creator == msg.sender, "not creator");

        require(block.timestamp > campaign.endAt, "not ended");

        require(campaign.pledged >= campaign.goal, "pledged < goal");

        require(!campaign.claimed, "claimed");

        campaign.claimed = true;

        token.transfer(campaign.creator, campaign.pledged);

        emit Claim(\_id);

    }

    function refund(uint \_id) external {

        Campaign memory campaign = campaigns[\_id];

        require(block.timestamp > campaign.endAt, "not ended");

        require(campaign.pledged < campaign.goal, "pledged >= goal");

        uint bal = pledgedAmount[\_id][msg.sender];

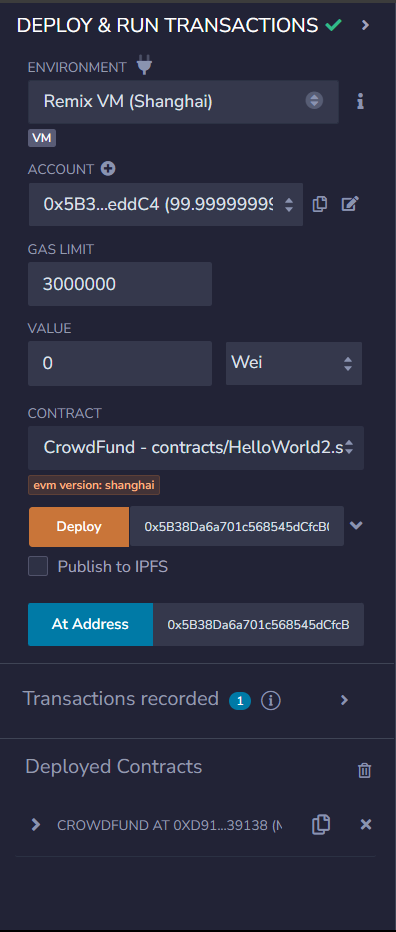
        pledgedAmount[\_id][msg.sender] = 0;

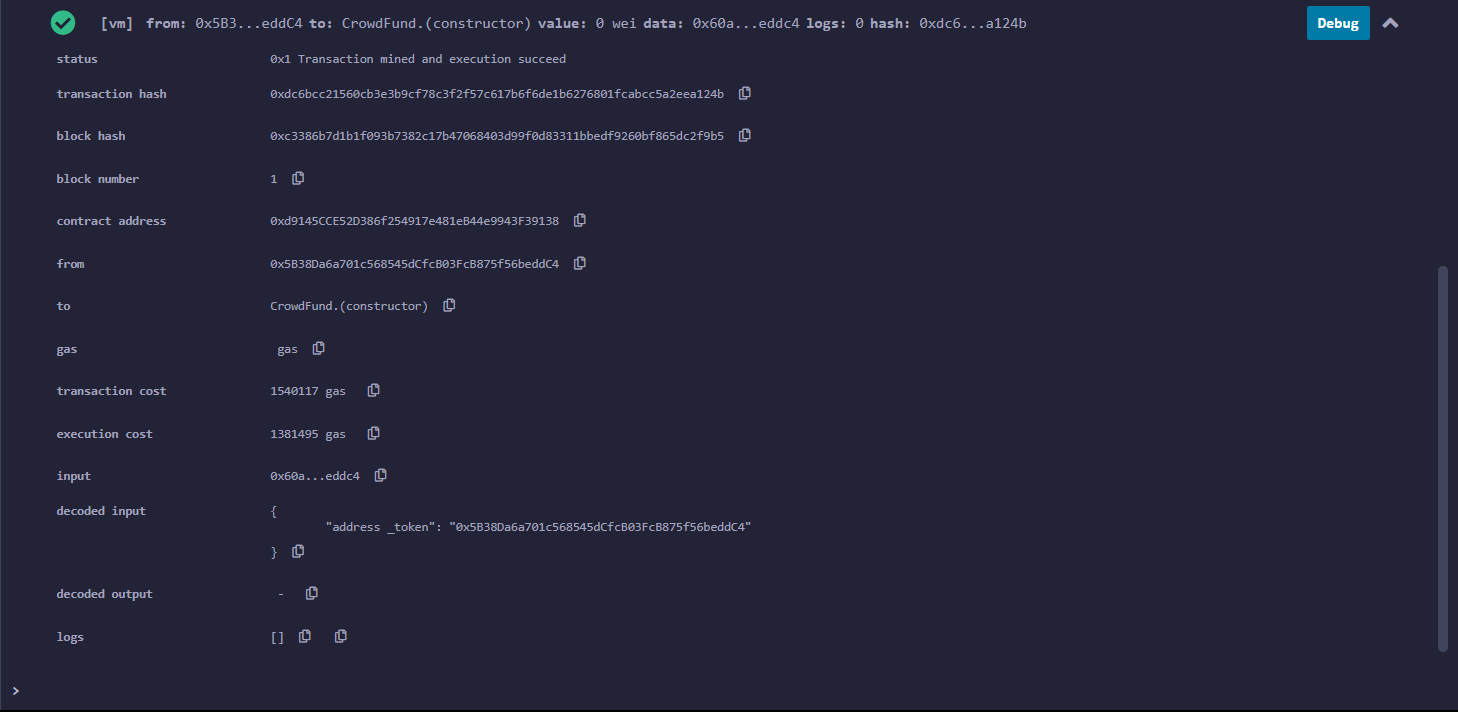
        token.transfer(msg.sender, bal);

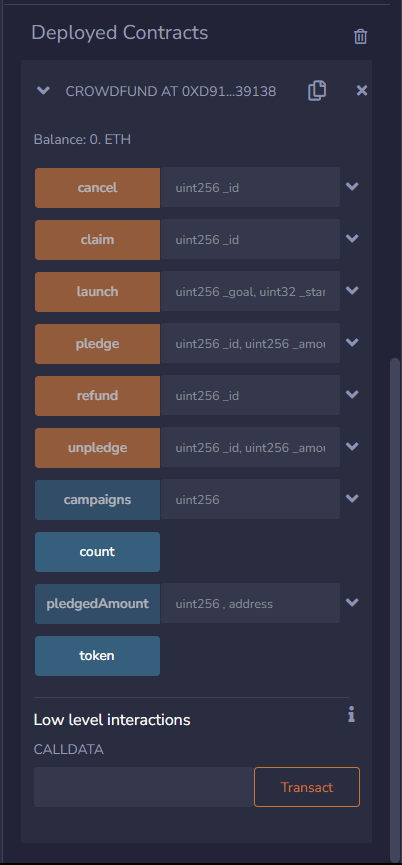
        emit Refund(\_id, msg.sender, bal);

    }

}

****

****

****

1. **Increment a simple voting smart contract web participant can work for different options.**

// SPDX-License-Identifier: MIT

pragma solidity >=0.6.12 <0.9.0;

contract SimpleAuction {

    // Parameters of the auction. Times are either

    // absolute unix timestamps (seconds since 1970-01-01)

    // or time periods in seconds.

    address payable public beneficiary;

    uint public auctionEndTime;

    // Current state of the auction.

    address public highestBidder;

    uint public highestBid;

    // Allowed withdrawals of previous bids

    mapping(address => uint) pendingReturns;

    // Set to true at the end, disallows any change.

    // By default initialized to `false`.

    bool ended;

    // Events that will be emitted on changes.

    event HighestBidIncreased(address bidder, uint amount);

    event AuctionEnded(address winner, uint amount);

    // Errors that describe failures.

    // The triple-slash comments are so-called natspec

    // comments. They will be shown when the user

    // is asked to confirm a transaction or

    // when an error is displayed.

    /// The auction has already ended.

    error AuctionAlreadyEnded();

    /// There is already a higher or equal bid.

    error BidNotHighEnough(uint highestBid);

    /// The auction has not ended yet.

    error AuctionNotYetEnded();

    /// The function auctionEnd has already been called.

    error AuctionEndAlreadyCalled();

    /// Create a simple auction with `biddingTime`

    /// seconds bidding time on behalf of the

    /// beneficiary address `beneficiaryAddress`.

    constructor(

        uint biddingTime,

        address payable beneficiaryAddress

    ) {

        beneficiary = beneficiaryAddress;

        auctionEndTime = block.timestamp + biddingTime;

    }

    /// Bid on the auction with the value sent

    /// together with this transaction.

    /// The value will only be refunded if the

    /// auction is not won.

    function bid() external payable {

        // No arguments are necessary, all

        // information is already part of

        // the transaction. The keyword payable

        // is required for the function to

        // be able to receive Ether.

        // Revert the call if the bidding

        // period is over.

        if (block.timestamp > auctionEndTime)

            revert AuctionAlreadyEnded();

        // If the bid is not higher, send the

        // Ether back (the revert statement

        // will revert all changes in this

        // function execution including

        // it having received the Ether).

        if (msg.value <= highestBid)

            revert BidNotHighEnough(highestBid);

        if (highestBid != 0) {

            // Sending back the Ether by simply using

            // highestBidder.send(highestBid) is a security risk

            // because it could execute an untrusted contract.

            // It is always safer to let the recipients

            // withdraw their Ether themselves.

            pendingReturns[highestBidder] += highestBid;

        }

        highestBidder = msg.sender;

        highestBid = msg.value;

        emit HighestBidIncreased(msg.sender, msg.value);

    }

    /// Withdraw a bid that was overbid.

    function withdraw() external returns (bool) {

        uint amount = pendingReturns[msg.sender];

        if (amount > 0) {

            // It is important to set this to zero because the recipient

            // can call this function again as part of the receiving call

            // before `send` returns.

            pendingReturns[msg.sender] = 0;

            // msg.sender is not of type `address payable` and must be

            // explicitly converted using `payable(msg.sender)` in order

            // use the member function `send()`.

            if (!payable(msg.sender).send(amount)) {

                // No need to call throw here, just reset the amount owing

                pendingReturns[msg.sender] = amount;

                return false;

            }

        }

        return true;

    }

    /// End the auction and send the highest bid

    /// to the beneficiary.

    function auctionEnd() external {

        // It is a good guideline to structure functions that interact

        // with other contracts (i.e. they call functions or send Ether)

        // into three phases:

        // 1. checking conditions

        // 2. performing actions (potentially changing conditions)

        // 3. interacting with other contracts

        // If these phases are mixed up, the other contract could call

        // back into the current contract and modify the state or cause

        // effects (ether payout) to be performed multiple times.

        // If functions called internally include interaction with external

        // contracts, they also have to be considered interaction with

        // external contracts.

        // 1. Conditions

        if (block.timestamp < auctionEndTime)

            revert AuctionNotYetEnded();

        if (ended)

            revert AuctionEndAlreadyCalled();

        // 2. Effects

        ended = true;

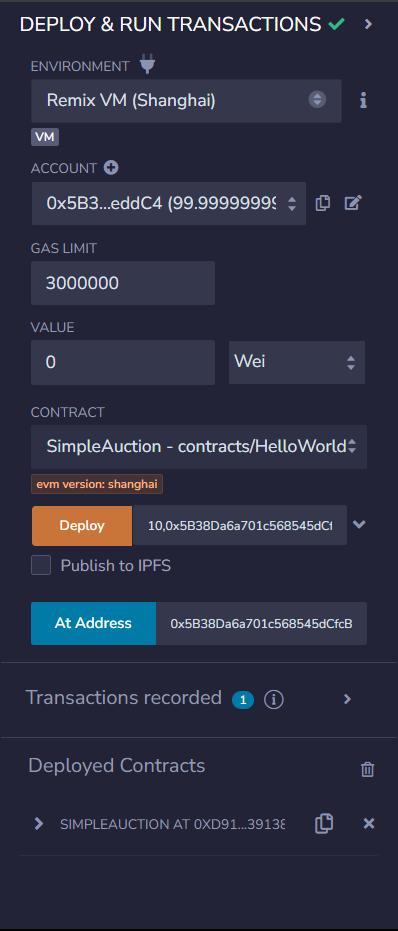
        emit AuctionEnded(highestBidder, highestBid);

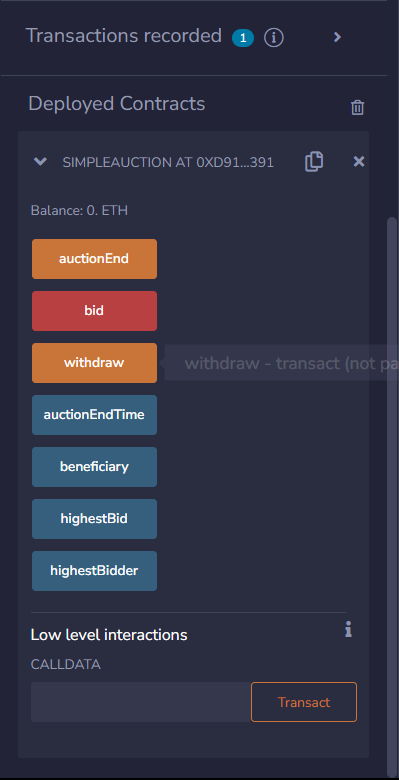
        // 3. Interaction

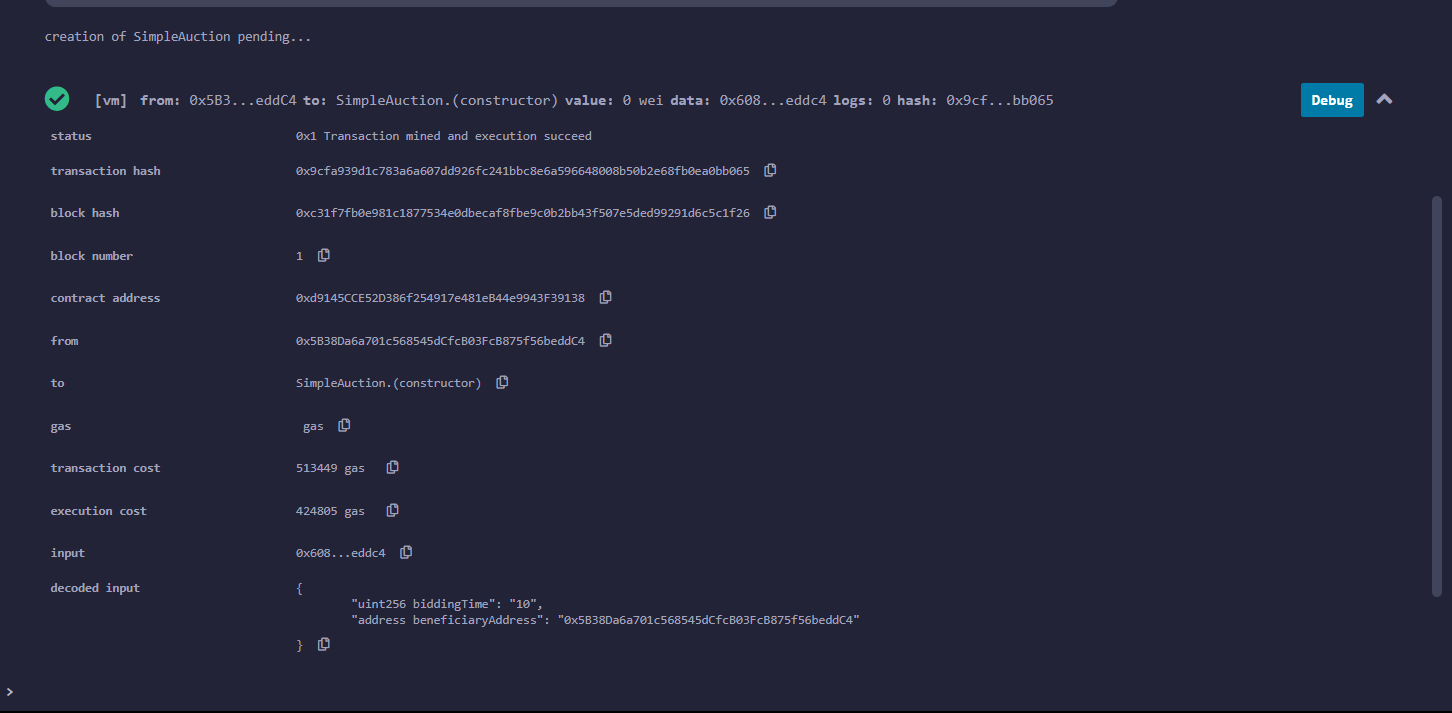
        beneficiary.transfer(highestBid);

    }

}

****

****

****

1. **Create a time-locked wallet where funds cannot be withdrawn until the specific time.**

// SPDX-License-Identifier: MIT

pragma solidity >=0.6.12 <0.9.0;

contract TimeLock {

    error NotOwnerError();

    error AlreadyQueuedError(bytes32 txId);

    error TimestampNotInRangeError(uint blockTimestamp, uint timestamp);

    error NotQueuedError(bytes32 txId);

    error TimestampNotPassedError(uint blockTimestmap, uint timestamp);

    error TimestampExpiredError(uint blockTimestamp, uint expiresAt);

    error TxFailedError();

    event Queue(

        bytes32 indexed txId,

        address indexed target,

        uint value,

        string func,

        bytes data,

        uint timestamp

    );

    event Execute(

        bytes32 indexed txId,

        address indexed target,

        uint value,

        string func,

        bytes data,

        uint timestamp

    );

    event Cancel(bytes32 indexed txId);

    uint public constant MIN\_DELAY = 10; // seconds

    uint public constant MAX\_DELAY = 1000; // seconds

    uint public constant GRACE\_PERIOD = 1000; // seconds

    address public owner;

    // tx id => queued

    mapping(bytes32 => bool) public queued;

    constructor() {

        owner = msg.sender;

    }

    modifier onlyOwner() {

        if (msg.sender != owner) {

            revert NotOwnerError();

        }

        \_;

    }

    receive() external payable {}

    function getTxId(

        address \_target,

        uint \_value,

        string calldata \_func,

        bytes calldata \_data,

        uint \_timestamp

    ) public pure returns (bytes32) {

        return keccak256(abi.encode(\_target, \_value, \_func, \_data, \_timestamp));

    }

    /\*\*

     \* @param \_target Address of contract or account to call

     \* @param \_value Amount of ETH to send

     \* @param \_func Function signature, for example "foo(address,uint256)"

     \* @param \_data ABI encoded data send.

     \* @param \_timestamp Timestamp after which the transaction can be executed.

     \*/

    function queue(

        address \_target,

        uint \_value,

        string calldata \_func,

        bytes calldata \_data,

        uint \_timestamp

    ) external onlyOwner returns (bytes32 txId) {

        txId = getTxId(\_target, \_value, \_func, \_data, \_timestamp);

        if (queued[txId]) {

            revert AlreadyQueuedError(txId);

        }

        // ---|------------|---------------|-------

        //  block    block + min     block + max

        if (

            \_timestamp < block.timestamp + MIN\_DELAY ||

            \_timestamp > block.timestamp + MAX\_DELAY

        ) {

            revert TimestampNotInRangeError(block.timestamp, \_timestamp);

        }

        queued[txId] = true;

        emit Queue(txId, \_target, \_value, \_func, \_data, \_timestamp);

    }

    function execute(

        address \_target,

        uint \_value,

        string calldata \_func,

        bytes calldata \_data,

        uint \_timestamp

    ) external payable onlyOwner returns (bytes memory) {

        bytes32 txId = getTxId(\_target, \_value, \_func, \_data, \_timestamp);

        if (!queued[txId]) {

            revert NotQueuedError(txId);

        }

        // ----|-------------------|-------

        //  timestamp    timestamp + grace period

        if (block.timestamp < \_timestamp) {

            revert TimestampNotPassedError(block.timestamp, \_timestamp);

        }

        if (block.timestamp > \_timestamp + GRACE\_PERIOD) {

            revert TimestampExpiredError(block.timestamp, \_timestamp + GRACE\_PERIOD);

        }

        queued[txId] = false;

        // prepare data

        bytes memory data;

        if (bytes(\_func).length > 0) {

            // data = func selector + \_data

            data = abi.encodePacked(bytes4(keccak256(bytes(\_func))), \_data);

        } else {

            // call fallback with data

            data = \_data;

        }

        // call target

        (bool ok, bytes memory res) = \_target.call{value: \_value}(data);

        if (!ok) {

            revert TxFailedError();

        }

        emit Execute(txId, \_target, \_value, \_func, \_data, \_timestamp);

        return res;

    }

    function cancel(bytes32 \_txId) external onlyOwner {

        if (!queued[\_txId]) {

            revert NotQueuedError(\_txId);

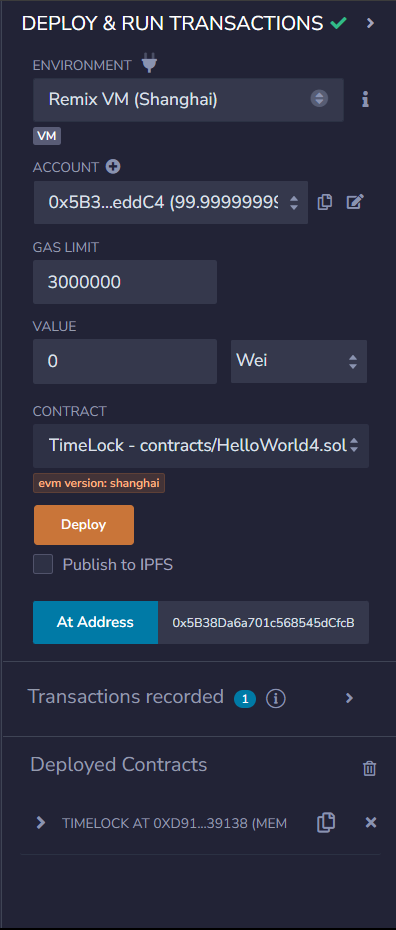
        }

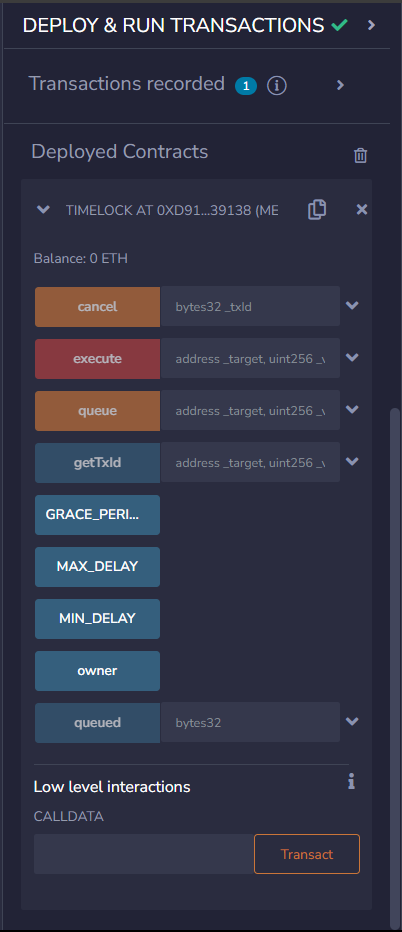
        queued[\_txId] = false;

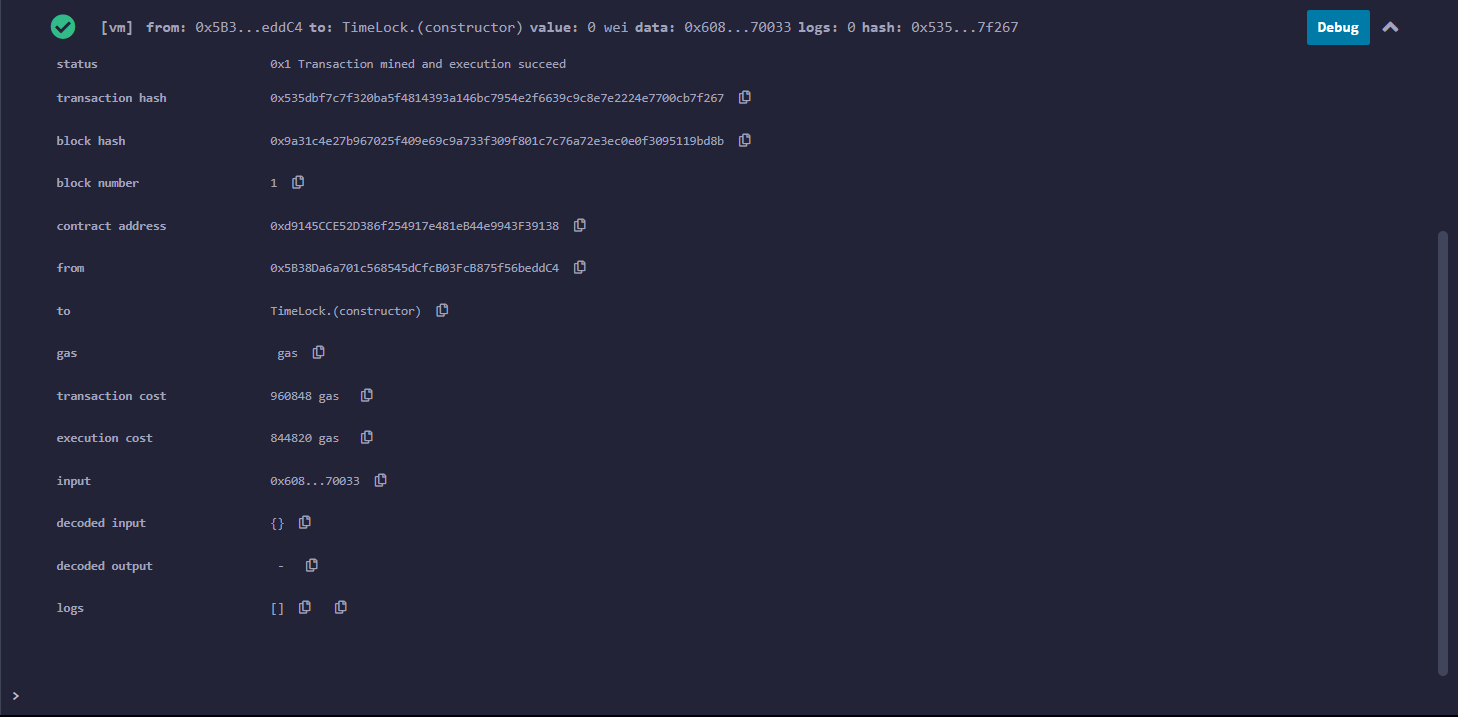
        emit Cancel(\_txId);

    }

}

****

****

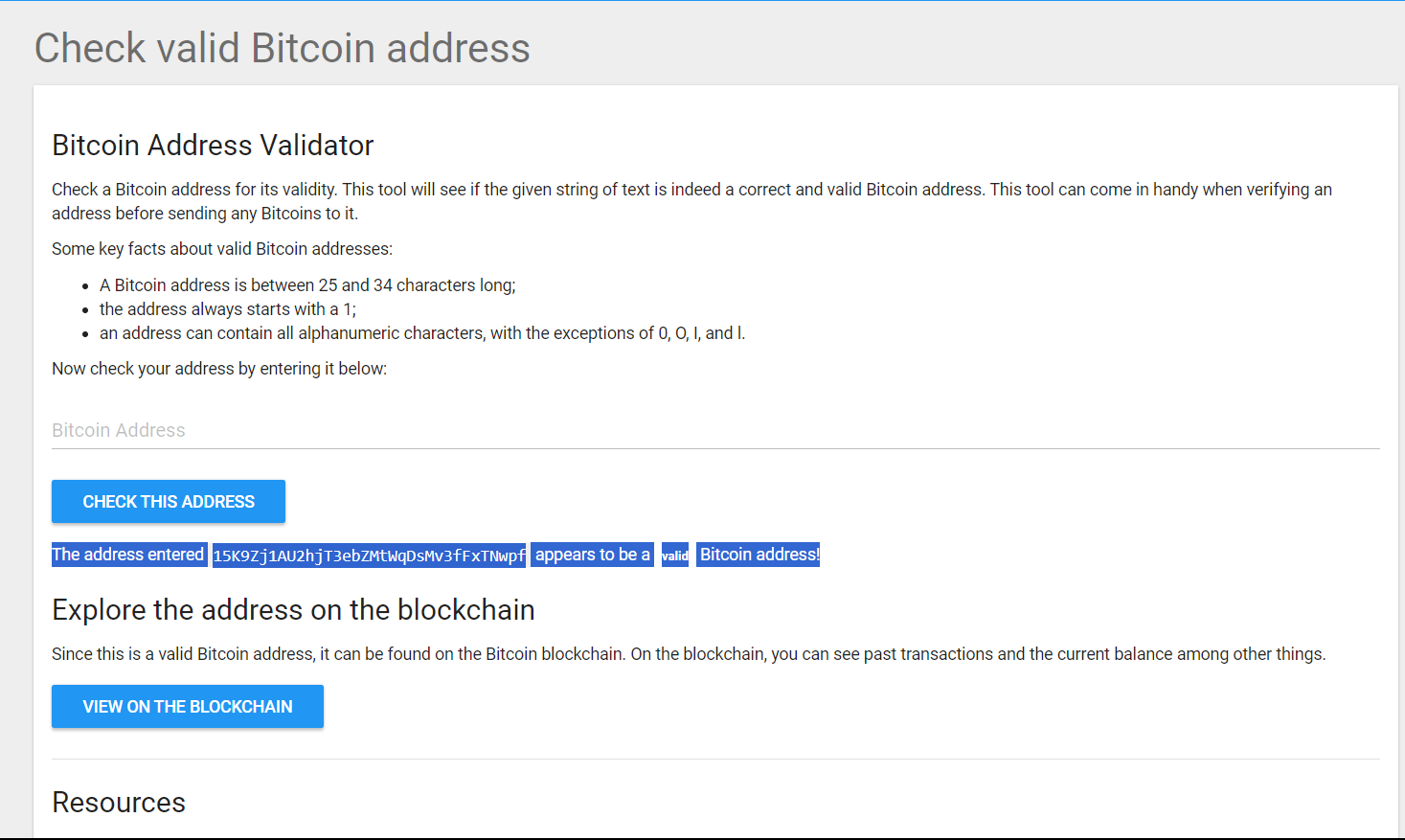
****

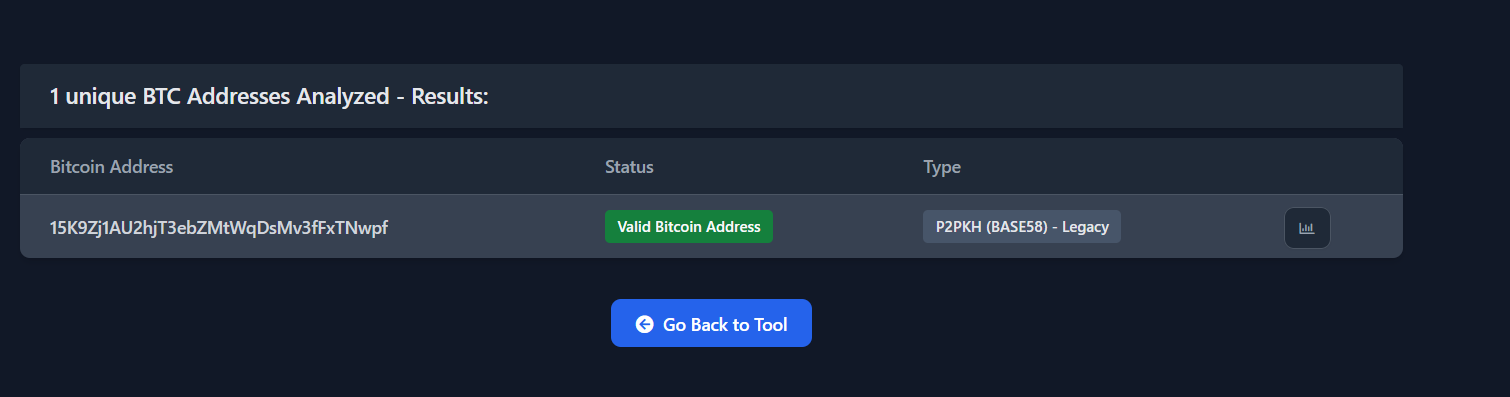
**Q.2 As an investigator, you came across a wallet address. Perform a following action for the observed wallet address.**

**Wallet address: 15K9Zj1AU2hjT3ebZMtWqDsMv3fFxTNwpf**

1. **Validate the address.**

Go to <https://thomas.vanhoutte.be/tools/validate-bitcoin-address.php> or https://awebanalysis.com/en/bitcoin-address-validate/ and enter the wallet address



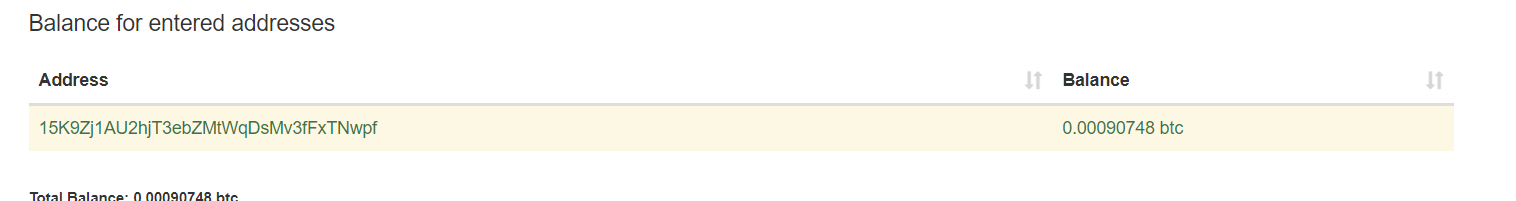


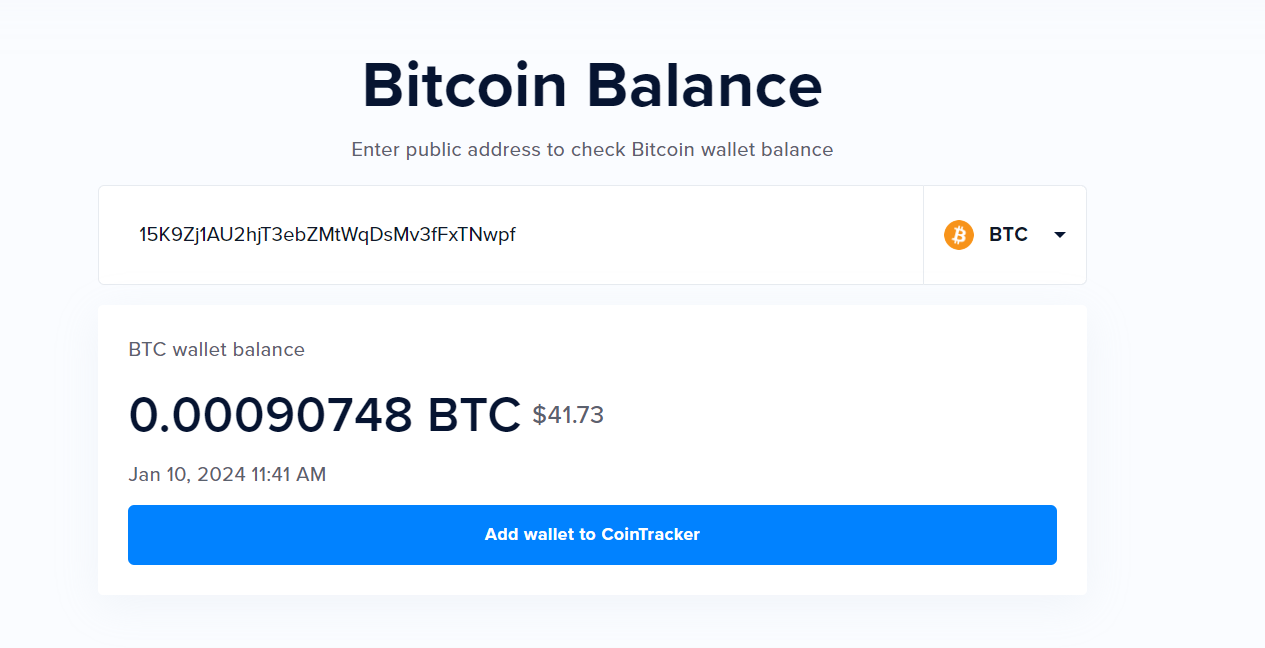
**“This is valid Wallet Address**

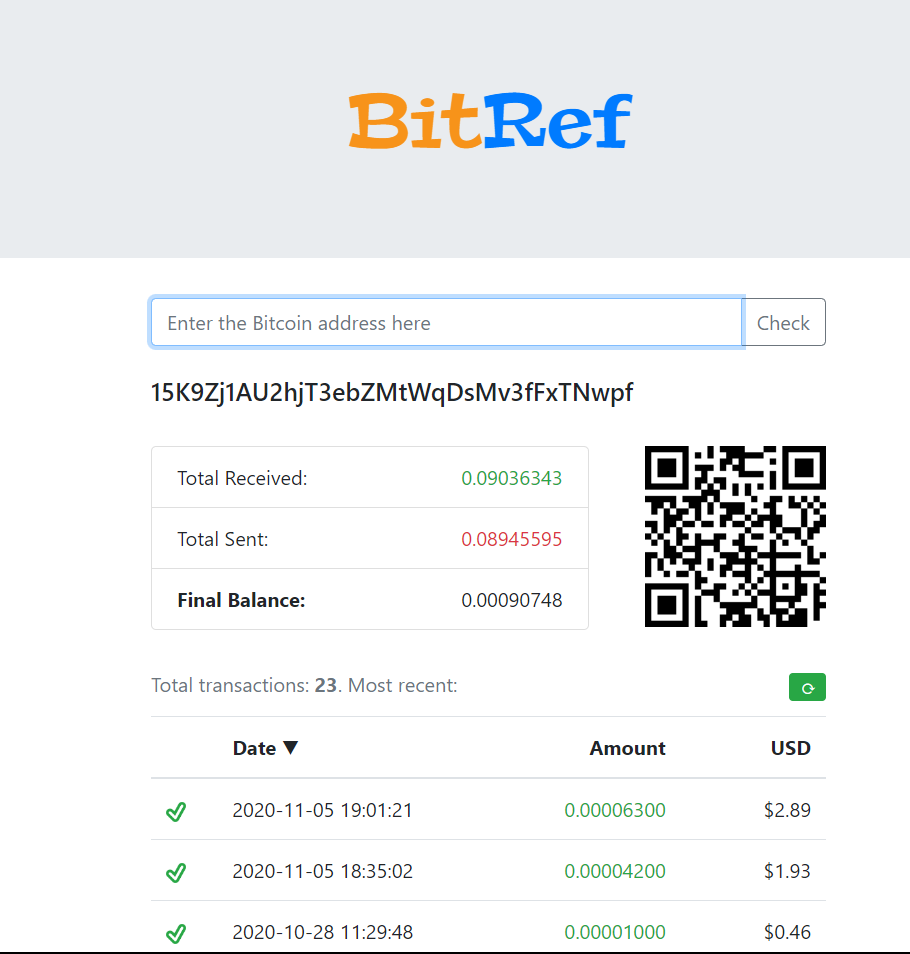
1. **Confirm the address balance.**

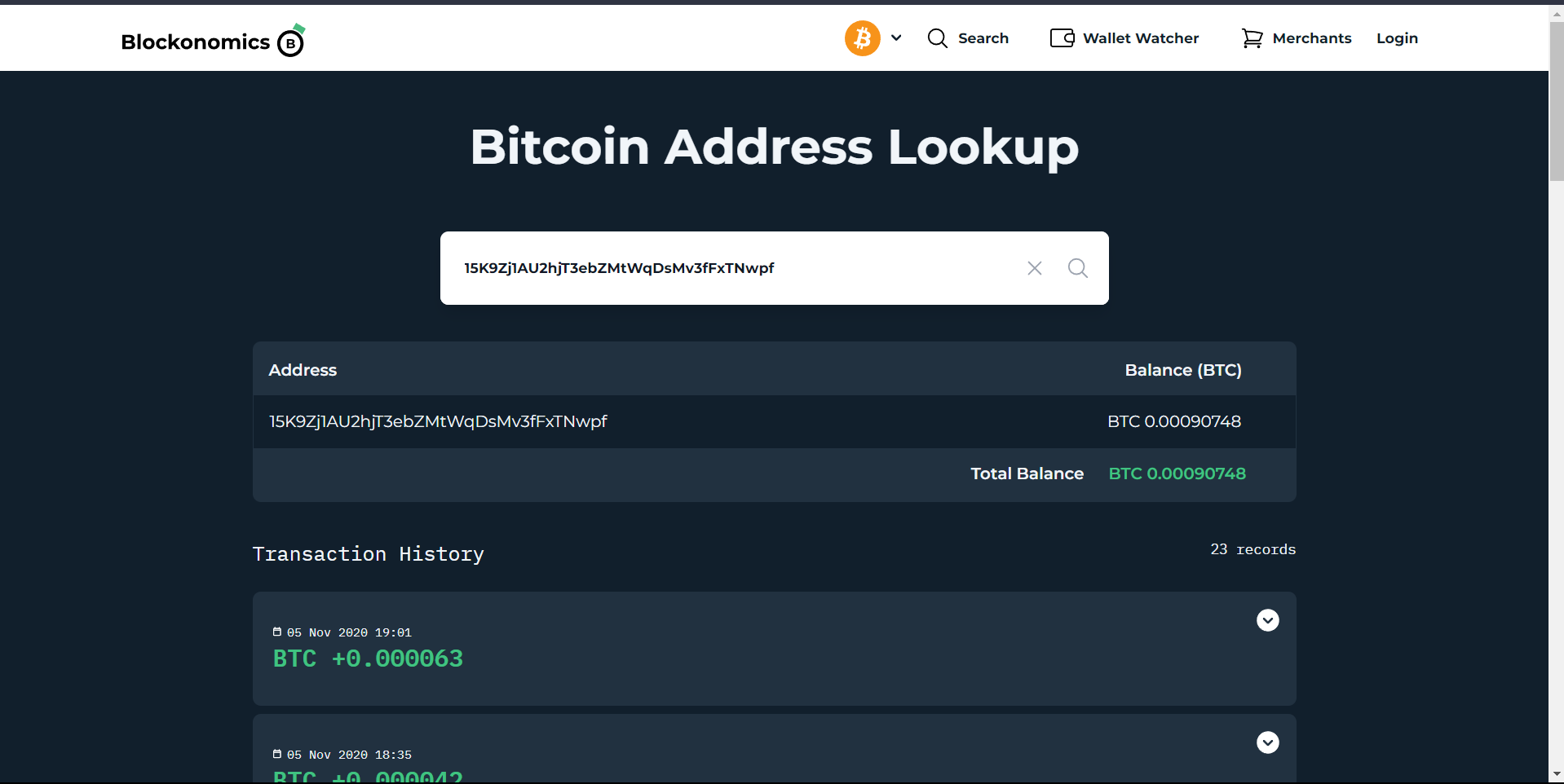
Go to any of this to Confirm

* + [**https://www.homebitcoin.com/easybalance/**](https://www.homebitcoin.com/easybalance/)
  + [**https://www.cointracker.io/wallet/bitcoin**](https://www.cointracker.io/wallet/bitcoin)
  + [**https://bitref.com/**](https://bitref.com/)
  + [**https://www.blockonomics.co/**](https://www.blockonomics.co/)



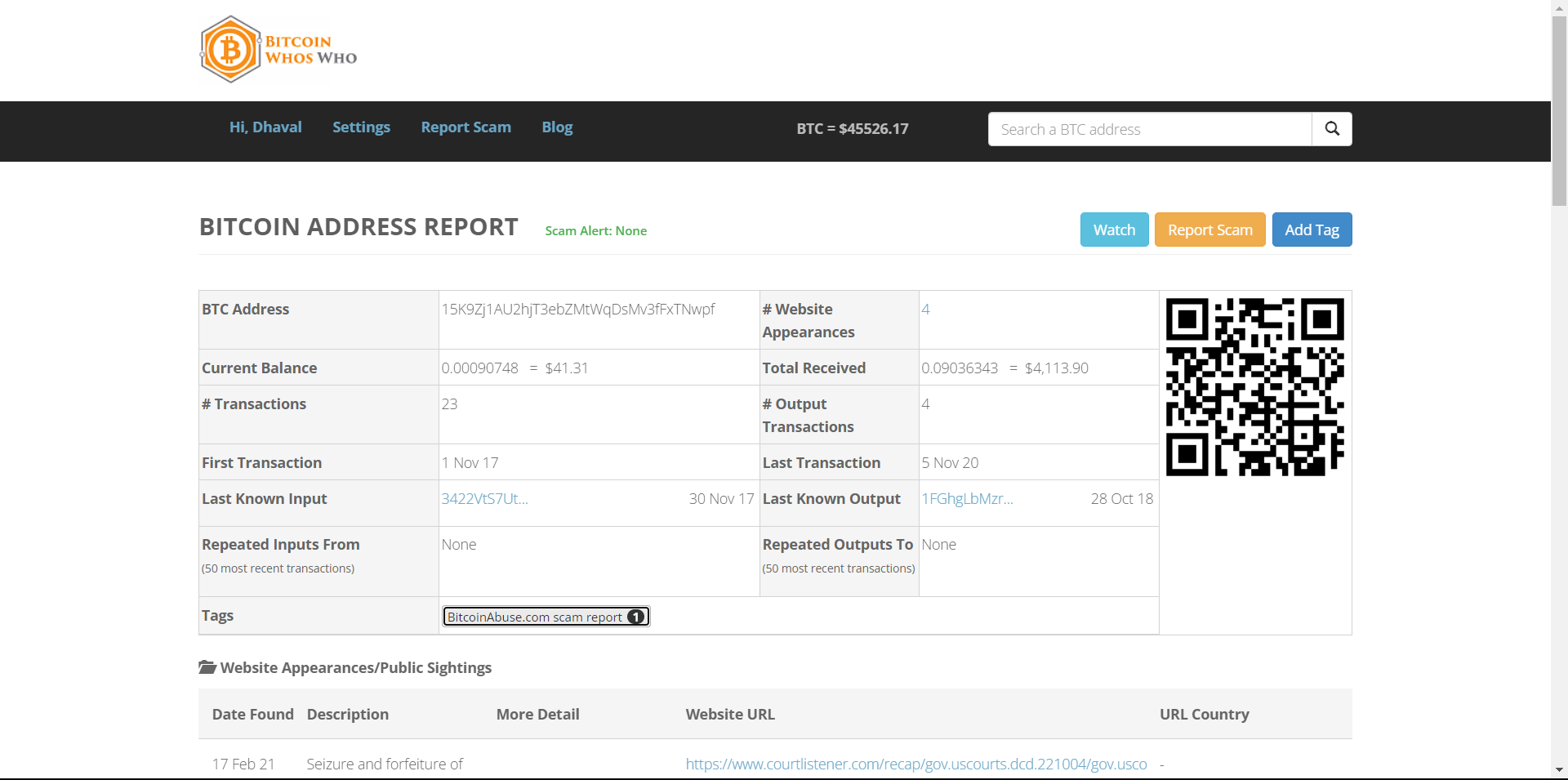


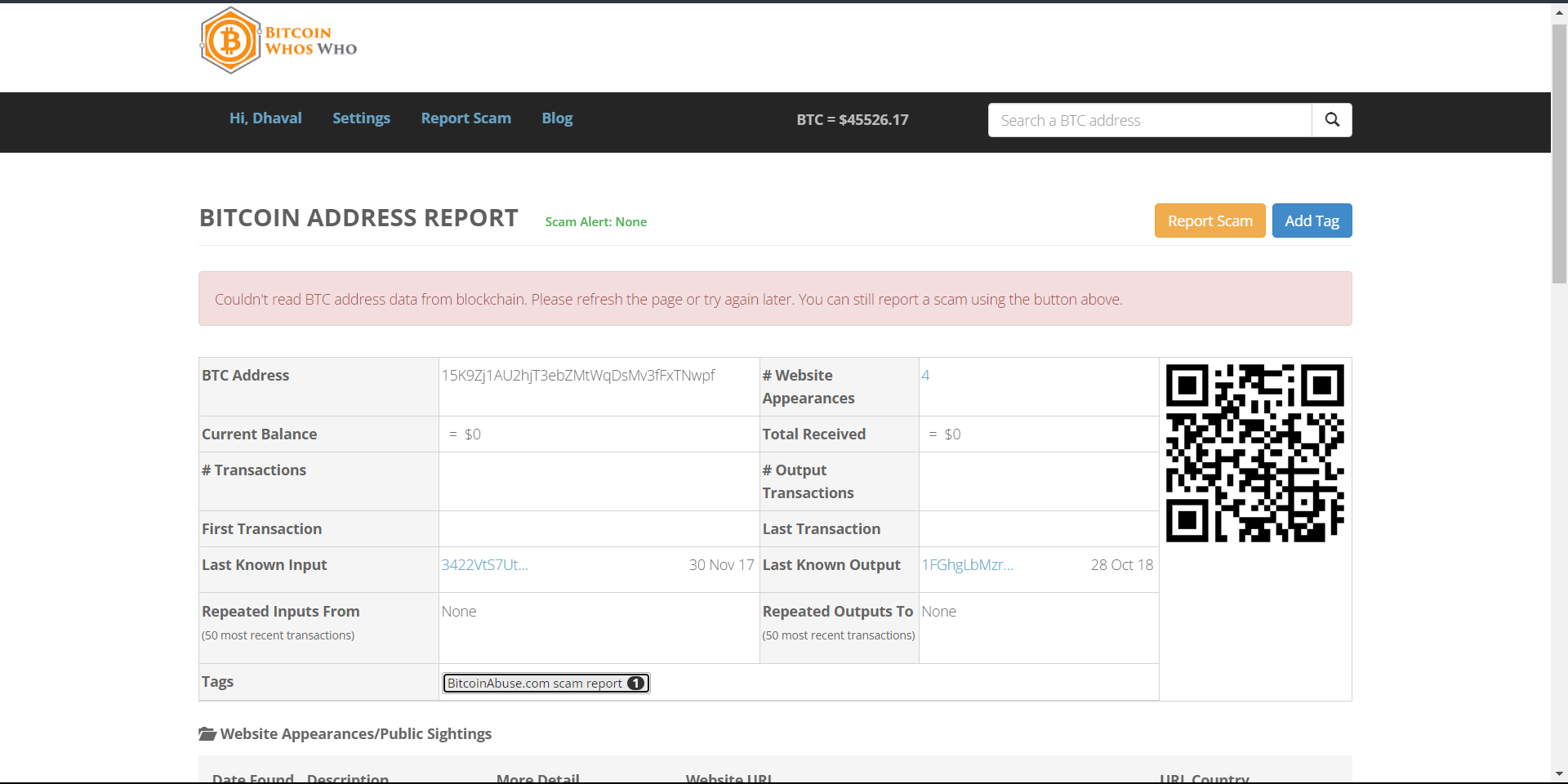


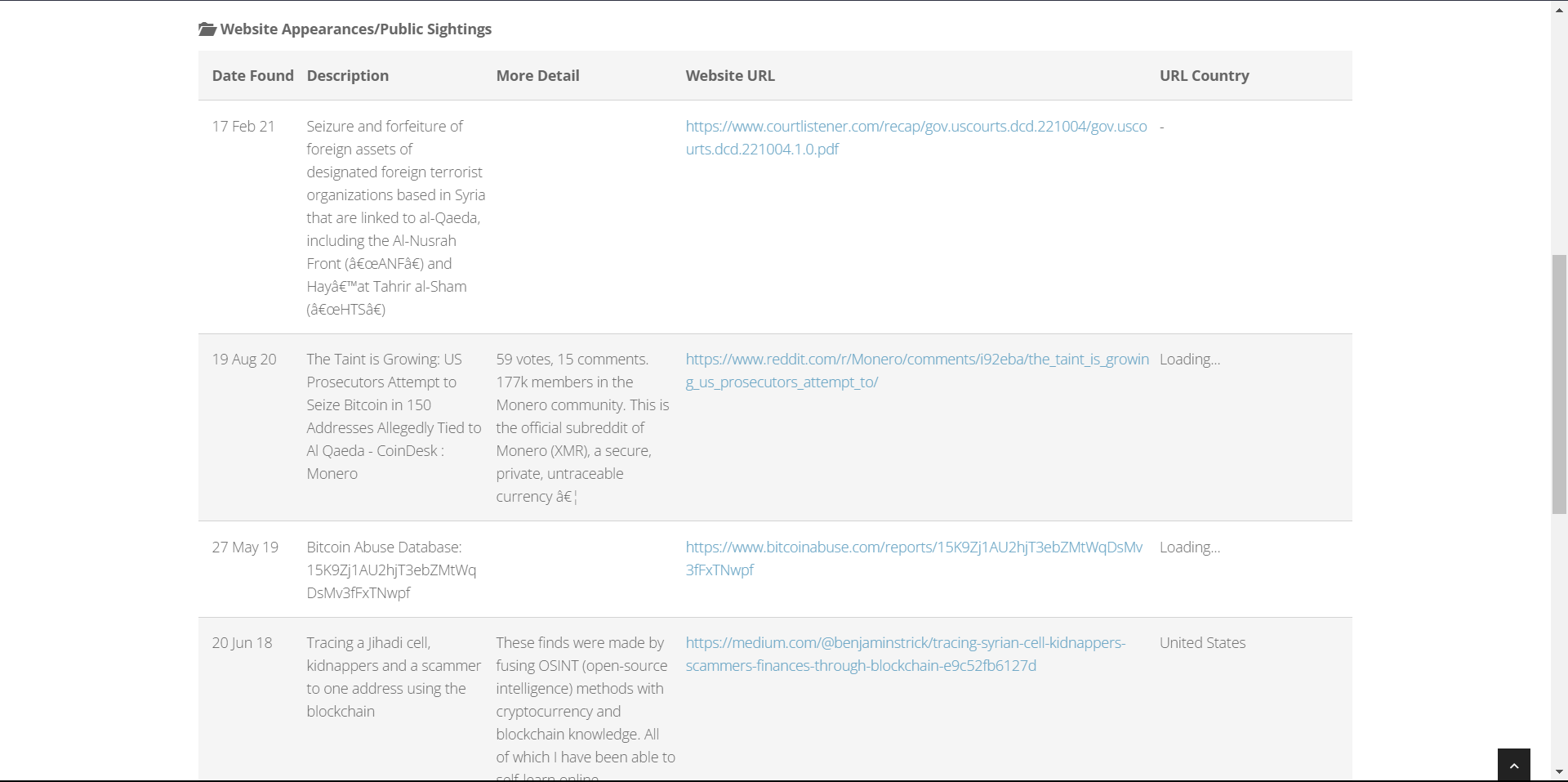


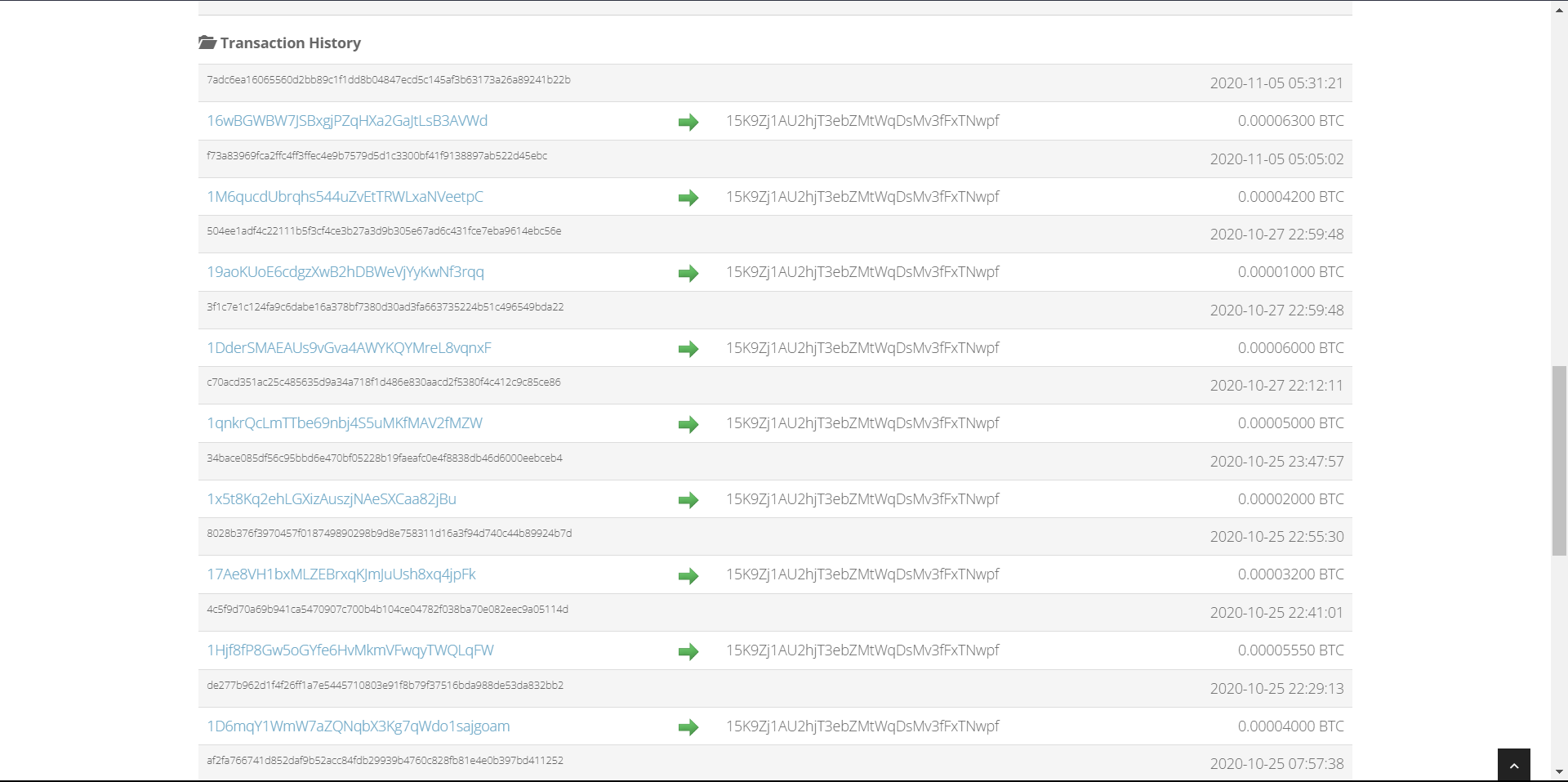
1. **Prepare report on Bitcoin address with should include SPAM/SCAM alert, Current balance, I/O transactions. IP address, Web Appearance, etc.**

Go to[**https://bitcoinwhoswho.com**](https://bitcoinwhoswho.com)

****

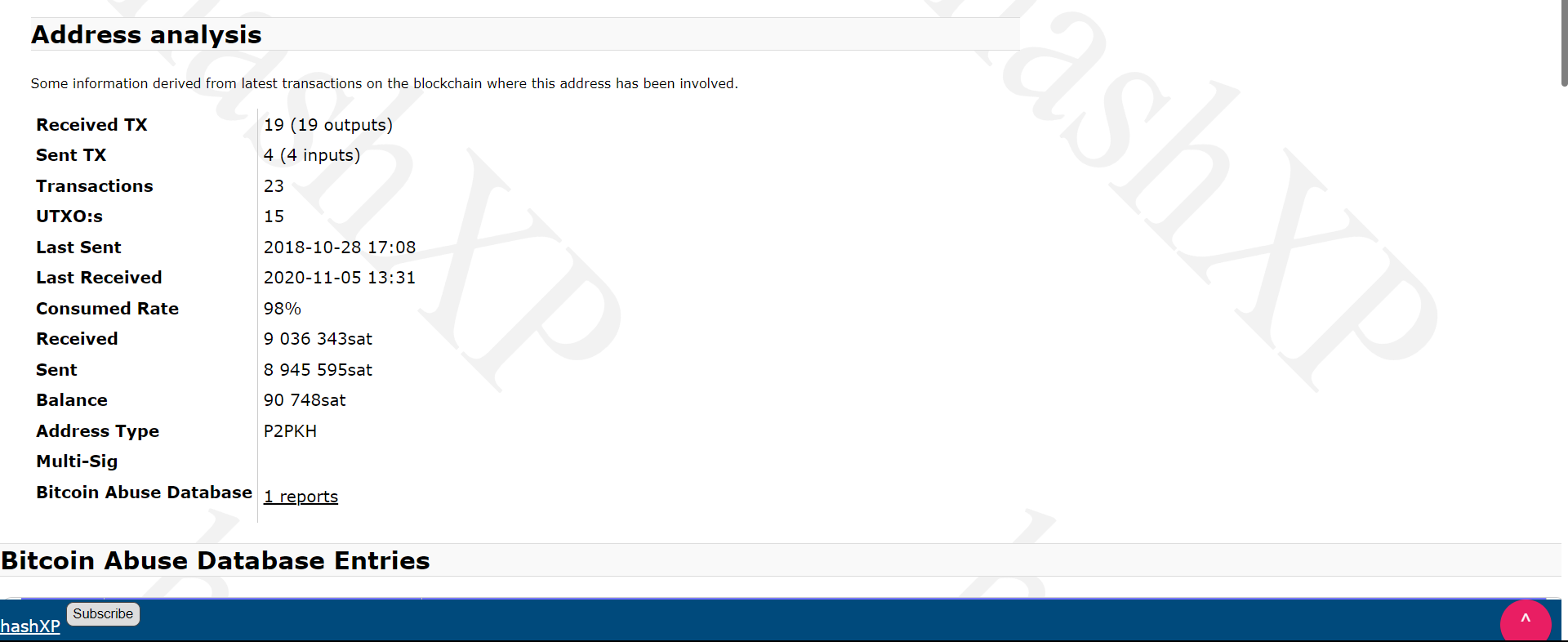
****

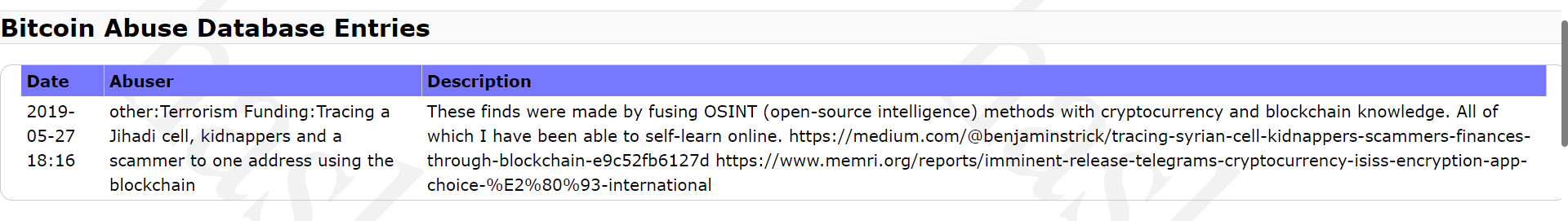
****

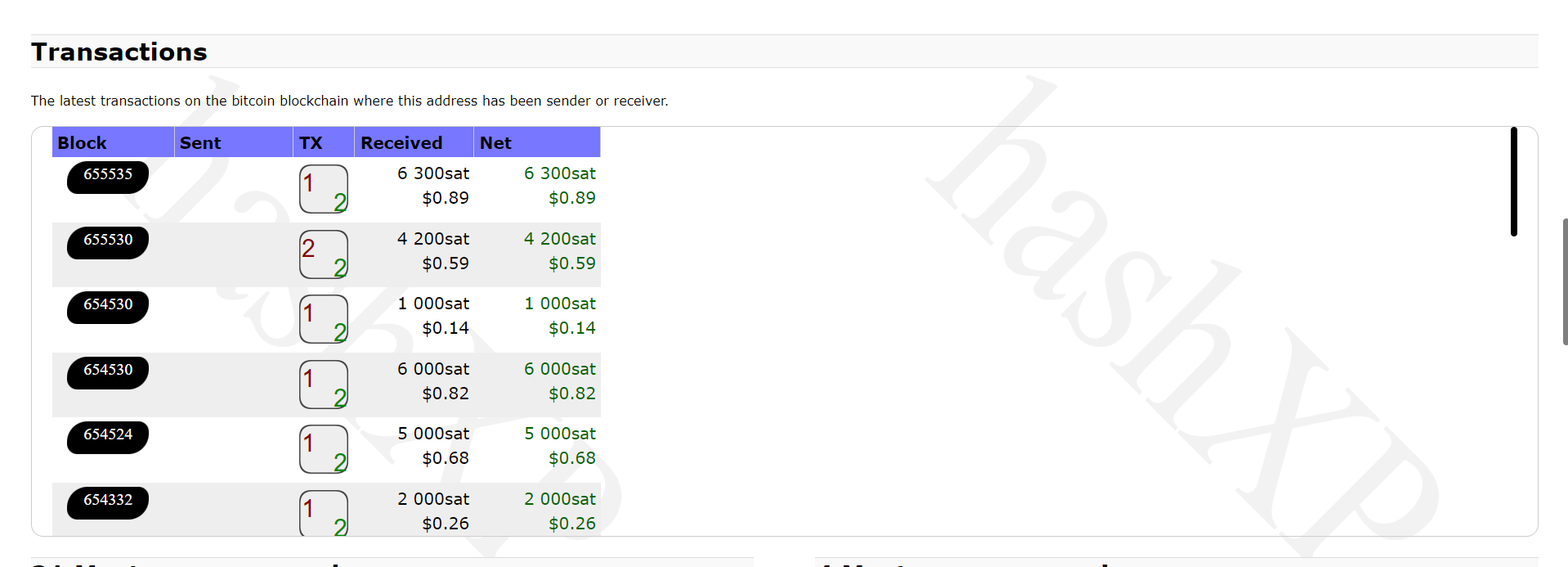
****

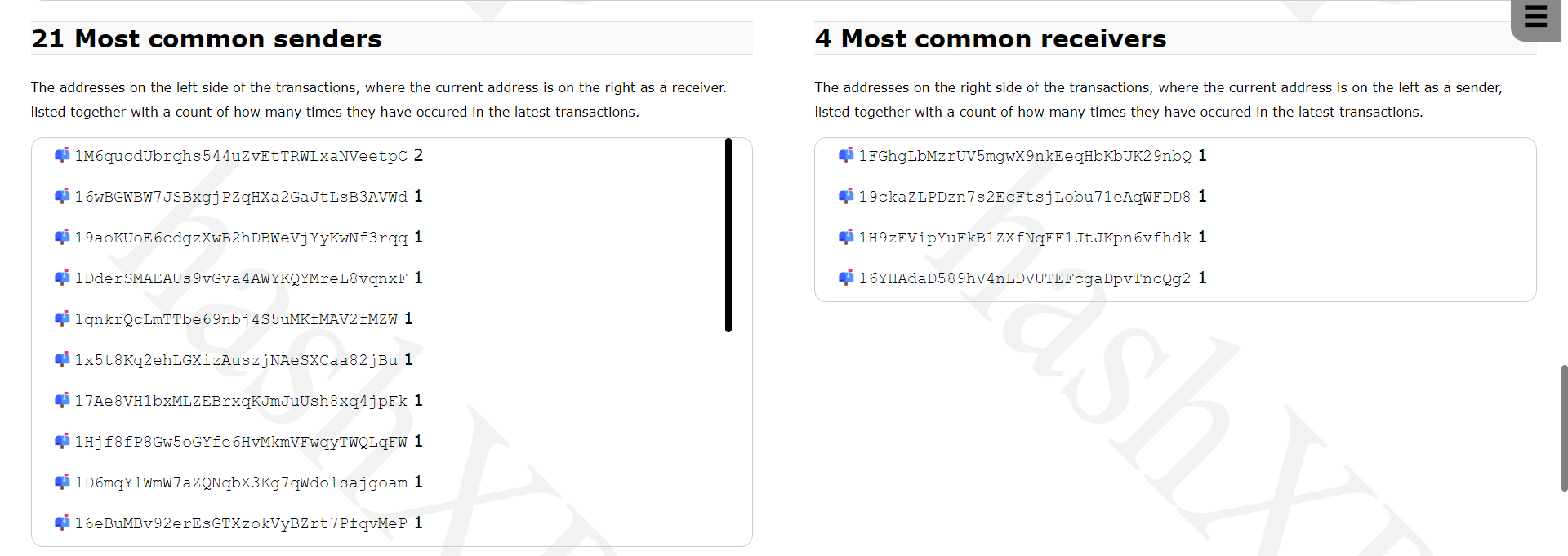
1. **In the prepared report, also include identified transactions, identified blocks, Mind blocks and exchange name related detail.**
   * GO to [**https://hashxp.org/**](https://hashxp.org/)

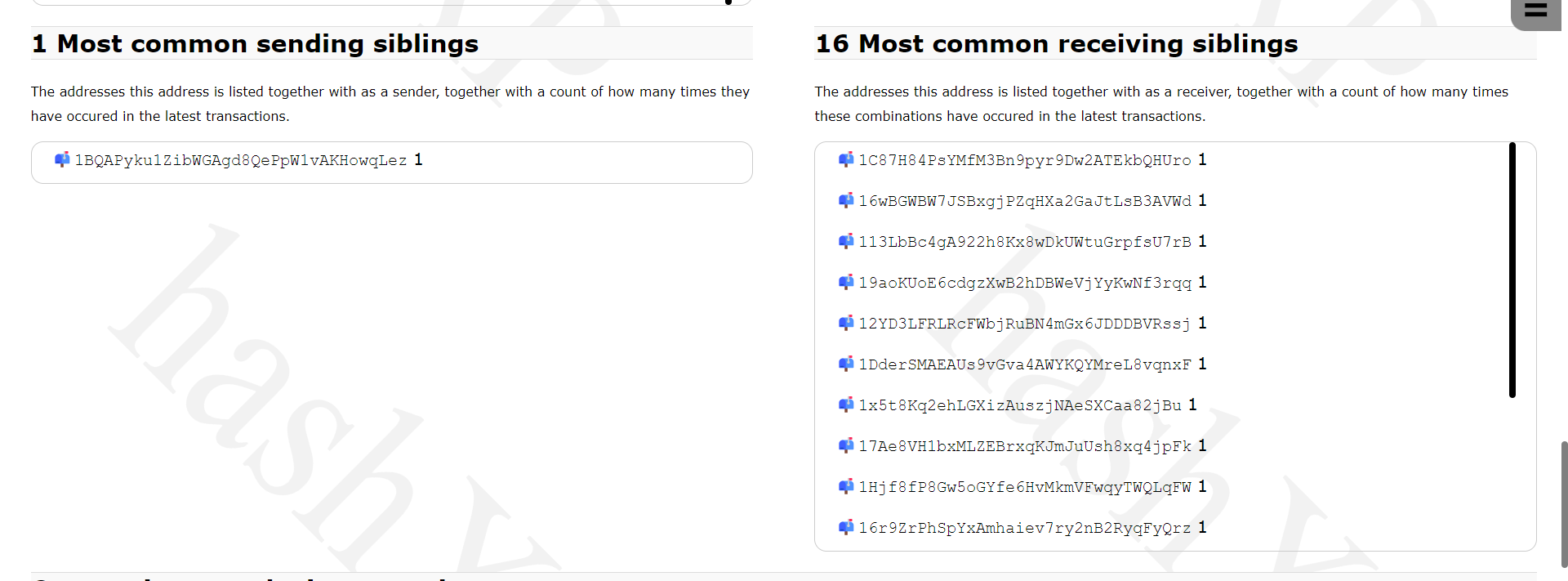


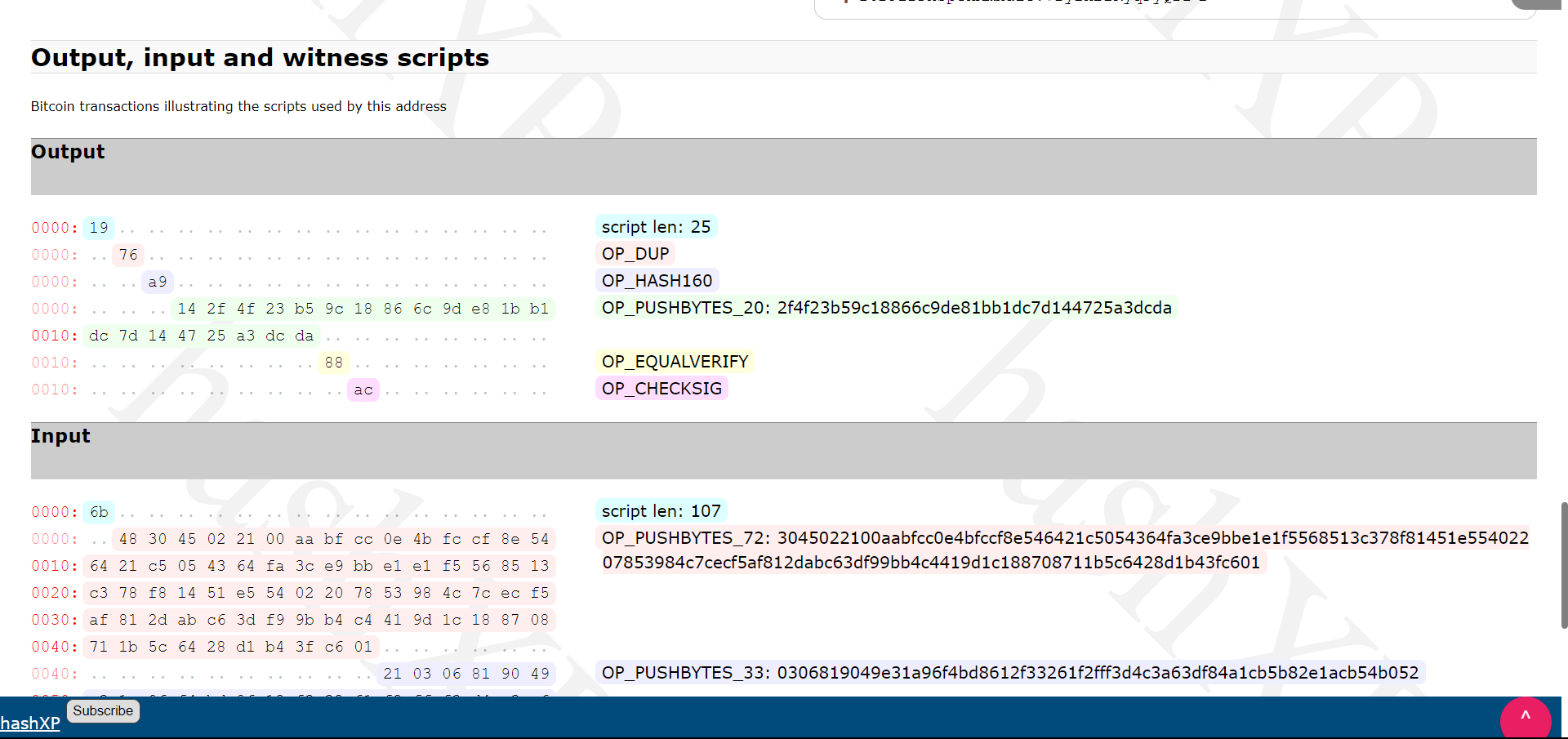


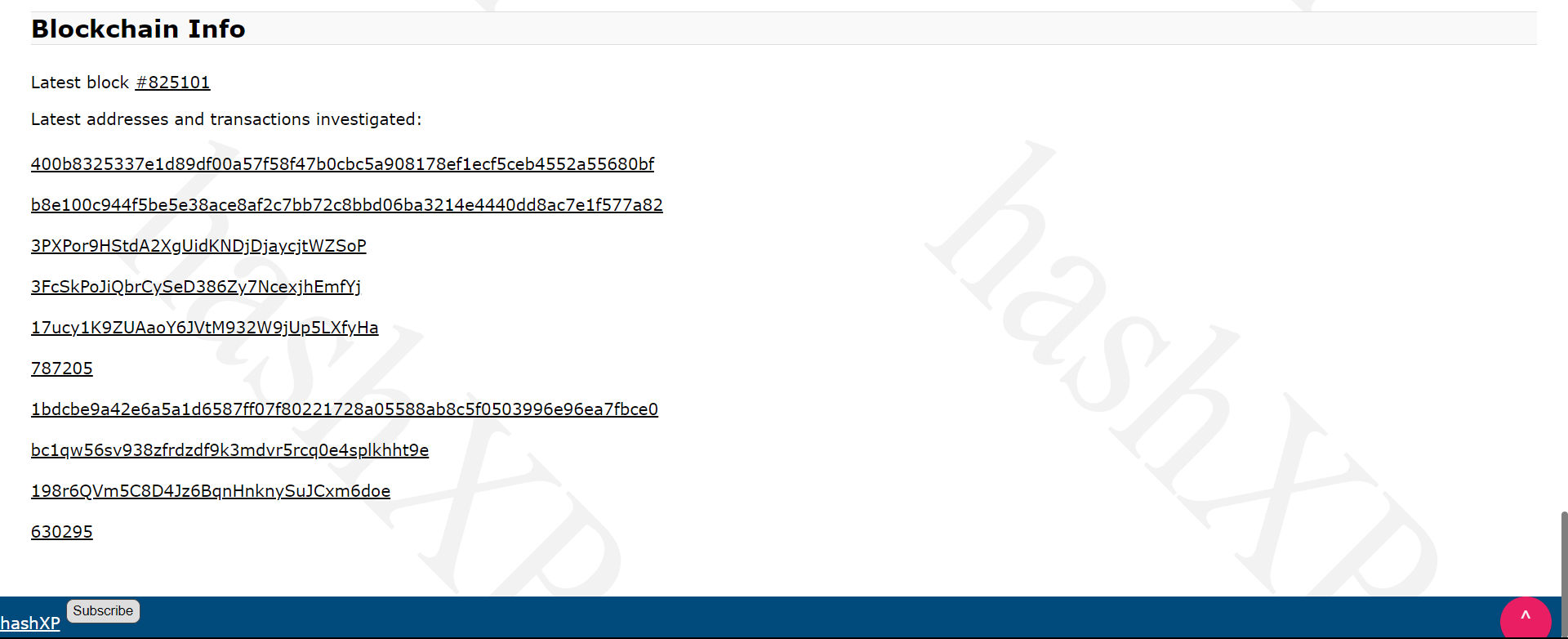












Go to Wallet explorer



Check on otx.me

