The Vault Challenge

General program description:

Write a Vault program that functions like a dead man's switch. It should allow the Deployer to deposit some large amount of tokens. It should then ask the Deployer to press the switch. If the Deployer presses the button, the contract should return the funds to them. If the Deployer does not press the switch then it transfers the funds to the Attacher.

There are 3 levels to this program. **You can only submit for one level**. Payouts are denoted for each level – you will only receive payout for the level of program you are submitting. (i.e. submit level 2 get $40 USDC)

We recommend building level 1 first and then moving on to level 2. If you are having trouble with level 2 you can fall back on your level 1 solution for submission.

Building a level 3 solution for each of the Hack Challenges is a great way to grow your Reach project portfolio.

Plagiarism Warning

Plagiarism weakens the community and does nothing to improve your web3 development skills. Submissions that are plagiarized will not be accepted.

These programs attempt to get you thinking about building code solutions from word problems. These should be your own work written from scratch.

If you are stuck, lost, or confused, reach out to us on Discord for assistance.

Program Requirements:

- ➔ Level 1 (**$20 USDC**)
    - ◆ Start with `reach init`
        - This will create your index.rsh and index.mjs files
    - ◆ Use 2 Participants for The Vault
        - The Deployer (Alice) should `pay` the contract a large amount, like an inheritance
            - ○ Fund this `newTestAccount` with at least 5,000 network tokens (ALGO or ETH)
        - Bob should send a Boolean value to the backend to accept terms to the Vault
            - ○ Create a function in the frontend and `return true` to the backend
                - ◆ This is us automating a "yes" answer here

        - Then publish this Bool
  - Create a constant value for a Countdown Timer
    - This should be declared in your backend rsh file
    - Show this to each Alice and Bob's frontend
      - This function should be part of both participants Shared functions
      - Be sure to inherit the Shared functions in each Participant Interact Interface
  - Create a function that returns Alice's action for the switch
    - This should be a frontend function that you call from the backend
    - Use Math.random to create a random number between 0-1
      - 0 returns false to the backend
      - 1 returns true to the backend
      - This is us automating this action for Alice
    - Display this choice as a string to the console
      - 0 prints "I'm not here"
      - 1 prints "I'm still here"
    - Use this value to transfer funds in the backend
      - true will transfer funds to Alice
      - false will transfer funds to Bob
  - Display status messages to the console
    - Show the balance of both participants before the contract
    - Display your fixed time after you set it
      - This will require you to pass that time from the backend to the frontend
      - This will also require you to parseInt
        - Why?
    - Display both account balances at the end of the program
  - This should be done in 2 files, index.rsh and index.mjs

- Level 2 (**$40 USDC**)
  - Instead of forcing the responses – make the program interactive.
    - Allow Alice to input her response
    - Ask Bob to accept the terms
      - exit the program if he does not
  - Create a timeout
    - Timeout if Alice doesn't push the button by your deadline
  - Use the Countdown Timer for The Vault
    - This should be done in your backend .rsh file
    - Use lastConsensusTime and add your Countdown Timer to create a constant for a fixed time some point in the future
      - Read this error about dealing in network seconds
    - Then count from lastConsensusTime() until your fixed time.

        - This should be a condition in the while loop below
    - Use a while loop to continue the execution if Alice is “still here”
        - Read about while loops in Reach
        - This runs indefinitely until your timer counts down to zero
            - If your timer reaches “zero” and Alice is “still here”, transfer the balance of the contract to Alice
    - Display status messages to the console
        - Timeout message for each participant
        - Account balances
        - Outcome of each round
        - Final outcome
    - This should be done in 2 files, index.rsh and index.mjs

- Level 3 (**$80 USDC**)
    - Include functionality from previous levels
        - You may have to change some of the functions used when switching to Testnet
        - You can use either launchToken or a custom Testnet Token
            - You don’t need to hit the faucet until you have >5,000 Testnet ALGOs
    - Allow for multiple Bobs (with API)
        - Distribute equal amounts of the contract to each Bob when appropriate
    - Build out a front end GUI
        - Use the framework of your choice
        - It will contain as many files as you need to complete the task

- [ ] **Checklist**
    - [ ] **Does your program function like the General Program Description?**
    - [ ] **Did you provide solutions for all of the problems on your program level?**
    - [ ] **Are you submitting for just one level?**
    - [ ] **My submission is my original work**