Introduction

Smart contract programming languages, similar to those used in conventional software development, facilitate developers' transition to smart contract development. Due to reasons such as newly discovered vulnerabilities and continuous iteration of requirements, the issue of smart contract maintenance has increasingly gained attention. However, recent incidents have shown that existing smart contract maintenance technologies cannot adequately handle data state changes, resulting in operational complexity and risks to user financial security. Therefore, it is crucial to study methods to ensure the correct maintenance of stateful smart contracts. A significant aspect is the capability to deploy new versions of smart contracts and ensure consistency in the smart contract state before and after the maintenance. This study mainly focuses on evaluating the effort needed to maintain stateful smart contracts (i.e., deploying new smart contracts and state migration between old and new versions) in the context of the Ethereum blockchain and the commonly used Solidity programming language.

In this study, there are two main tasks and several sub-tasks. Please execute the tasks in the given order, measure the time and gas cost required for each task, and answer the questions on the provided answer sheet. Make sure to read through the task descriptions in this document before starting.

We provide a pre-configured VM (username: study_user, password: 12345) with all necessary tools and an answer sheet installed. You will find all relevant files in the "Smart Contract Maintenance " directory, which is easily accessible from the Desktop link. You can access the local Remix IDE through the web browser at http://localhost:8080, either by using the browser bookmarks or the desktop icon. In this way, you can test the functionalities of the smart contract.

Starting Questionnaire:

1、Please answer the following questions with either yes or no.
Q1: Have you ever conducted research in the field of Blockchain?
Q2: Did you write Solidity code in the last two weeks?
Q3: Have you previously worked on a production-grade Solidity-based Ethereum contract?

2、Please answer the following questions by rating on a scale from 1 to 7, where 1 means you are not familiar at all (e.g., you have heard of the topic but have no understanding), and 7 means that you are most familiar (e.g., you have expert knowledge and extensive experience in the field).
Q4: How familiar are you with the Ethereum Blockchain in particular?
Q5: How familiar are you with the Solidity programming language?
Q6: How familiar are you with Solidity contract maintenance techniques?

Task 1: Convert Contracts to Follow Delegatecall-based Proxy Pattern

In this task, you will convert a smart contract to follow *the delegatecall-based proxy pattern.* Unlike software, smart contracts' code cannot be changed once they are deployed. However, for various reasons (e.g., bug fixes and new feature implementation), there has been a great demand for making

smart contracts upgradable, and the most popular method is *the delegatecall-based proxy pattern.*

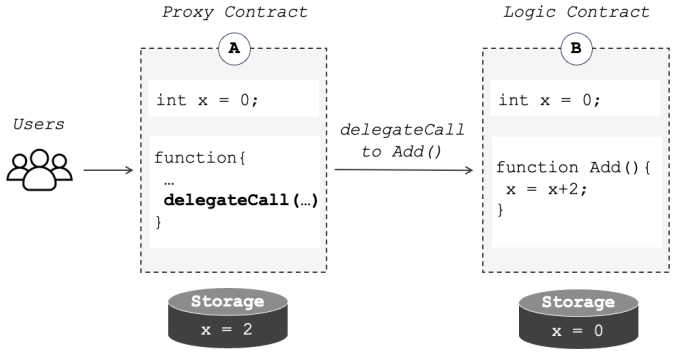

Figure 1: The delegatecall-based proxy pattern.

The delegatecall-based proxy pattern splits one contract into two: a proxy contract with address *A* and a logic contract with address *B*. The proxy contract holds the storage state as well as a changeable pointer to the logic contract. As shown in Figure 1, the proxy contract redirects the users' call to the logic contract by using `delegatecall` and executes the actual `Add()` in the proxy's context, resulting in an update of variable *x* in the proxy.

Your task is to convert an existing contract into this delegatecall-based proxy pattern style. If you are not familiar with it, you first need to understand the complexities of using this pattern and how it works.

Finally, you will use an automated tool, *SmartUpdater*, to convert the contract with hyperproxy-based upgradable framework. In this framework, the contract separates state and logic by following a delegatecall-based proxy pattern. These contracts are further divided into multiple sub-state and sub-logic contracts to minimize overall deployment and upgrade gas costs. Additionally, it introduces a hyperproxy as the immutable entry point to coordinate these sub-contracts, ensuring that any updates to the state or logic are transparent to external interactions.

Step 1 – Manual Conversion

Begin by navigating to the task-1/1-manual directory and opening the `XRT.sol` file within the Remix IDE. Your primary task involves implementing the delegatecall-based proxy pattern for the XRT contract, with its source code located in `XRT.sol`. After implementing the proxy pattern, proceed to test the new contract by deploying it directly from the Remix IDE.

Please note that no additional guidance will be provided to simulate varying levels of prior knowledge. You are encouraged to utilize any available resources to complete this task. To ensure the task is completed within the allotted study time, please set a timer and limit your work to a maximum of 3 hours.

Step 2 – Automatic Conversion with SmartUpdater

Navigate to the task-1/2-SmartUpdater directory. You can open the terminal and run the SmartUpdater deployment script:

```
$ python smartupdater_deploy.py --help

usage: smartupdater_deploy.py [-h] <contract_source>

SmartUpdater Command Line Interface for Contract Conversion


positional arguments:
  contract_source      Path to the Solidity contract source file


optional arguments:
  -h, --help           Show this help message and exit
```

You need to provide the path of the contract, for example:

```
Example usage:

  $ python smartupdater_deploy.py ./source.sol
```

SmartUpdater will then automatically convert the given contract to follow the hyperproxy-based upgradable framework.

Questionnaire

1、Please answer the following questions with either yes or no.
T1Q1: Have you previously used the delegatecall-based proxy pattern in a Solidity contract?
T1Q2: Have you previously used a different pattern to make a Solidity contract upgradable?

2、Please answer the following questions by rating on a scale from 1 to 7.
T1Q3: How familiar are you with the delegatecall-based proxy pattern? (1-not familiar, 7-most familiar)
T1Q4: How confident are you in the correctness of your conversion? (1-least confident, 7-most confident)
T1Q5: How difficult was the manual conversion? (1-easy, 7-most difficult)
T1Q6: How difficult was the automatic conversion with SmartUpdater? (1-easy, 7-most difficult)

3、Please report the time and gas cost you required for the step.
T1Q7: How much time did you need to manually convert the given contract using the delegatecall-based proxy pattern?
T1Q8: How much time did you need to convert the contract using SmartUpdater? (step 2)
T1Q9: How much gas does it take to deploy the given contract? (step 1)
T1Q10: How much gas does it take to deploy the converted contract? (step 1)
T1Q11: How much gas does it take to deploy the converted contract using SmartUpdater? (step 2)

Task 2: Complete the Contract Maintenance

During the state update process, it can be summarized in two main steps: 1) creating the new state/logic contracts according to the update requirements, and 2) initializing states to maintain

consistency between the old and new contracts. In this task, you will develop a new state contract using the smart contracts created in the previous task along with the provided state update requirements. Additionally, we will supply the state values from the old state contract, which you will need to transfer and recover in the new state contract you create. Finally, you will use the *SmartUpdater* to automate the maintenance process described above.

Step 1 – New Contract Creation and Deployment

Navigate to task-2/1-manual and open the Standard_contract.sol and require.pdf files. The Standard_contract.sol file contains the contracts configured with the delegatecall-based proxy pattern, as developed in the previous task, and serves as the baseline for your maintenance. The require.pdf file outlines the specific update requirements for this task. Your task is to implement the new contract by adjusting the existing code in Standard_contract.sol to meet the update specifications detailed in the require.pdf. This includes any necessary modifications or additions to the contract's functionality and structure. Subsequently, test the new contract by deploying it from within the Remix IDE.

Step 2 – State Recovery in the New Contract

Navigate to task-2/1-manual and open the data.xlsx file that contains the state values from the old contract. Your task is to read these values and integrate them effectively, ensuring that the state is correctly initialized in the new state contract.

Important Note: It is difficult to obtain the data of mapping-type states in XRT contract from the Ethereum directly, but the number of holders in the contract can be determined. Therefore, we will use the number of holders as the count of elements included in the mapping-type states and construct the corresponding data accordingly. Test these functions within the Remix IDE to verify that the state transfer is successful and that the new contract behaves as expected with the old data.

Step 3 – Automatic maintenance using SmartUpdater

Navigate to the task-2/2-SmartUpdater directory. You can open the terminal and run the SmartUpdater mintenance script:

```
$ python smartupdater_maintenance.py --help

usage:smartupdater_maintenance.py [-h] <contract_name> <requirement_source>

SmartUpdater Command Line Interface for Contract Maintenance


positional arguments:
  contract_name         Name ofthe Solidity contract
  requirement_source    Path to the requirement source file

optional arguments:
  -h, --help          Show this help message and exit
```

You need to provide the paths of the contract file and the requirement file , for example:

```
Example usage:

  $ python smartupdater_maintenance.py name ./requirement.txt
```

SmartUpdater will then automatically complete the maintenance process described above based on the requirement.

Questionnaire

1、Please answer the following questions.

T2Q1: How confident are you in the correctness of the new contract you created? (1-least confident, 7-most confident)

T2Q2: How confident are you in the integrity of the state recovery? (1-least confident, 7-most confident)

T2Q3: How confident are you in accurately retrieving state values from the old contract? (1-least confident, 7-most confident)

T2Q4: How difficult is it to create the new contract manually? (1-easy, 7-most difficult ; step 1)

T2Q5: How difficult is it to recover states in the new contract manually? (1-easy, 7-most difficult ; step 2)

T2Q6: How difficult was the maintenance process using SmartUpdater? (1-easy, 7-most difficult ; step 3)

T2Q7: Do you know how to get the values of the state with dynamic storage strategy (e.g., mapping-type state) from the Ethereum blockchain? If you know, please explain in detail how to get them.

2、Please report the time and gas cost you required for the step.

T2Q8: How much gas does it take to deploy the new contract manually? (step 1)

T2Q9: How much gas does it take to deploy the new contract using SmartUpdater? (step 2)

T2Q10: How much time did you need to manually create the new contract and recover states? (step 1 and step 2)

T2Q11: How much time did you need to complete maintenance process using SmartUpdater? (step 3)

Submitting the Results

To submit your results, please ensure that all edited files are saved and all questions in the answer.xlsx file are fully answered. By sending your results via email, you agree that your responses and any modifications to the source code will be analyzed and published anonymously as part of a scientific publication. Thank you for your valuable contribution to this study.