Week 3: Hardhat. Unit Testing, Mutation testing, and Static Analysis

Learning Objectives

- Learn how to set up and use the hardhat environment
- Learn how to test every possible state change in Ethereum
- Learn how to fork the mainnet and test against it
- Learn how and why to use solhint, eslint, and prettier and why those tools are important
- Learn and use static analysis
- Learn what mutation testing is, why it is important, and how to use it

Suggested Materials

- Intro to and installing hardhat
- RareSkills: A thorough tutorial of unit testing
- Mainnet forking
- Account Impersonation
- Verifying hardhat code on Etherscan
- Static analysis
  - Slither
- Look at OpenZeppelin for inspiration of what a good test suite looks like: https://github.com/OpenZeppelin/openzeppelin-contracts/tree/master/test/token/ERC20
- Mutation Testing
  - Wikipedia page on mutation testing: https://en.wikipedia.org/wiki/Mutation_testing
  - Solidity mutation testing tool: https://github.com/JoranHonig/vertigo

What is due at the end of the week

- [ ] Old code cleanup

    - [ ] Clean up your old assignments from the previous weeks with solhint and prettier. You might need to adjust the maximum line length if the formatting looks funny. **From now on you should use these tools by default.**
        - [ ] Token Sale with sell back.
        - [ ] Token Exchange
        - [ ] ERC721 with staking (the three contract ecosystem)
    - [ ] Run slither on those assignments. Did it find any errors or were they false positives? Copy and paste the output of slither and add comments to say whether the detection is a false positive or an error the tool found for you.
- [ ] Add unit tests to the Sell Back and Token Exchange assignment
    - [ ] You should have 100% branch and line coverage
    - [ ] Run mutation testing. Did you discover any faulty tests? Mutation tests take a while to run, use the `--sample-ratio` argument set to 0.1 to see if the workflow is working. Set hardhat parallel to the number of cores on your machine. Some shells do not preserve the output. Explicitly save the output to a file using the `--output` argument. This tool is buggy with typescript tests. Convert your tests to javascript if you used typescript
- [ ] Ethernaut
    - [ ] 0 and 1 . You can run in the environment they use or you can copy and paste the code to a local hardhat project and create tests there. These exercises are quite simple. This is just to get you started.

Addendum

From now on, you should be using solhint, prettier, and slither on every project going forward. Not all of them will require unit tests, but production apps should be 100% covered (line and branch) and you should run mutation testing on top of that.