Week 6, 7, 8, and 8: Security

Learning Objectives

- Understand access control hacks
- Understand re-entrancy
- Understand unusual solidity behavior
- Understand arithmetic overflow

Suggested Materials

- Vulnerability Playlist (watch re-entrancy, arithmetic overflow, insecure randomness, forcefully send ether, contract with zero size, and denial of service. The rest are optional)
- Wolf Game Exploit
- Exploit contracts with selfdestruct
- Highly recommended reading: blog.sigmaprime.io/solidity-security.html
- Highly recommended reading: consensys.github.io/smart-contract-best-practices/attacks/
- Interesting reading: cmichel.io/how-to-become-a-smart-contract-auditor/
- Creating a smart contract from a smart contract

What is due at the End of the next two weeks

- ☐ Ethernaut (ethernaut.openzeppelin.com)
  Helpful resources
  Deleting Contracts | Solidity 0.8
  Send ETH | Solidity 0.8
  - ☐ Ethernaut 3 (insecure randomness)
  - ☐ Ethernaut 4 (solidity understanding see tx.origin)
  - ☐ Ethernaut 5 (arithmetic overflow/underflow)
  - ☐ Ethernaut 10 (Re-entrancy)
  - ☐ Ethernaut 11 (understanding interfaces)
  - ☐ Ethernaut 15 (understanding ERC20 protocol)
  - ☐ Ethernaut 17 (solidity understanding)

- [ ] Ethernaut 19 (arithmetic overflow/underflow)
    - [ ] Ethernaut 20 (solidity understanding)
    - [ ] Ethernaut 21 (understanding interfaces)

- [ ] Capture the Ether (capturetheether.com). The points assigned to the challenge are correlated with how difficult the problem is.
    - [ ] Guess the Number (warmup challenge)
    - [ ] Guess the random number (insecure randomness)
    - [ ] Guess the secret number (solidity understanding)
    - [ ] Guess the new number (insecure randomness)
    - [ ] Predict the future (solidity understanding)
    - [ ] Predict the block hash (solidity understanding, **Hint:**how does blockhash work?)
    - [ ] Token bank (re-entrancy, understanding token protocols)

        Hint: Most of this contract is an implementation of ERC223, it is recommended to understand at a high level how this standard works first so you don't have to learn the standard through reading all the code. You can use this as a starting point https://www.youtube.com/watch?v=7yKvh8esaQw and read the specification https://github.com/ethereum/eips/issues/223
    - [ ] Token sale (arithmetic overflow/underflow)
    - [ ] Token whale (arithmetic overflow/underflow)

- [ ] Damn Vulnerable Defi (damnvulnerabledefi.xyz)
    - [ ] Learn flash loans first

        Flash loans are a key part of hacking these. You can learn about flash loans with these resources, but the flash loans here are simple. A contract will give you ether or tokens if you return them in the same transaction. Otherwise, it reverts. https://www.youtube.com/watch?v=_GZHt-FVAQs (fast paced) https://www.youtube.com/watch?v=Aw7yvGFtOvI (more detailed introduction)

        Aave is most known for flash loans, but it isn't the only one. The overall principle is the same.

You can learn about the security considerations of flash loans in the EIP (https://eips.ethereum.org/EIPS/eip-3156)

- [ ] 1 Unstoppable (faulty business logic)
- [ ] 2 Naive Receiver (solidity understanding)
- [ ] 3 Truster (understanding imported libraries)
- [ ] 4 Side Entrance (re-entrancy)

- [ ] Solidity Riddles (private repo, please ask for access)
    - [ ] Overmint1 (re-entrancy)
    - [ ] Overmint2 (understanding ERC721)
    - [ ] Overmint3
    - [ ] Democracy (faulty business logic, token protocol)

For everything except solidity riddles, do a writeup in a markdown file and publish it on your github.

Hint: you can use static analysis tools like slither to detect issues. It will not always work however.

Where students commonly mess up

- Don't get stuck on one problem for days. You have 28 days and 27 problems. **If you get stuck, ask for a hint.**
- These problems will test you on subtleties of solidity. Make sure you thoroughly understand the specifications of the keywords at play.
- Try to solve problems by "theme." Once you know how to solve in one particular category, you will more easily be able to solve related ones
- If a problem imports a library, study that library closely and the assumptions that it makes
- Solidity prevents underflow and overflow by default in 0.8.0 or higher. Be sure to compile contracts with the version they specify if you do it in your own environment
- **Don't solve all the Ethernaut problems 3-21, only solve the problems we specify here.** We specifically skipped some of them because you probably don't have the prerequisite knowledge at this time.

- Although you can find the answers to most problems online, we strongly discourage you from doing that. It defeats the purpose of the exercises. Ask for a hint from us instead.
- For ethernaut, it may sometimes be easier to to copy the code into remix for some problems, but be sure the solidity version matches the openzeppelin version for the solidity version the problem uses.