Project 5: "Smart" Contracts

This project is due on **Wednesday, Dec 8** at **6 p.m.** and counts for 8% of your course grade. Late submissions will be penalized by 10% plus an additional 10% every 5 hours until received. Late work will not be accepted after 24 hours past the deadline. If you have a conflict due to travel, interviews, etc., please plan accordingly and turn in your project early.

This is a group project; you will work in **teams of two** and submit one project per team. Please find a partner as soon as possible. If have trouble forming a team, post to Piazza's partner search forum.

The code and other answers your group submits must be entirely your own work, and you are bound by the Honor Code. You may consult with other students about the conceptualization of the project and the meaning of the questions, but you may not look at any part of someone else's solution or collaborate with anyone outside your group. You may consult published references, provided that you appropriately cite them (e.g., with program comments), as you would in an academic paper.

Solutions must be submitted electronically via **Github Classroom** (`https://classroom.github.com/a/buYR1cTi`), following the submission checklist below.

Introduction

In this project, you will write Solidity code to exploit a vulnerable smart contract in order to steal (testnet) funds from it.

Objectives:

- Implement and deploy a basic Ethereum smart contract in Solidity
- Understand basic smart contract vulnerabilities
- See how vulnerabilities can lead to stolen funds.

Part 0. Setup

You'll need to familiarize yourself with a few new tools for interacting with the Ethereum Blockchain. We'll be using **Ropsten Testnet** Ethereum, which is exactly like the main Ethereum, but the Ether is free and not exchangeable. **You won't need to buy anything, or use real money** at any point in this project (though all the techniques can and have been used on the main net as well).

Note: using this exploit to steal real money on main Ethereum is *theft*. In addition to being a crime, there are also real Ethereum contracts that *appear* vulnerable, but in reality steal attacker's coins!

Install MetaMask

MetaMask is a Chrome/Firefox/Brave extension that acts as an Ethereum wallet and will store your (testnet) Ether currency. Install it in your browser of choice:
`https://metamask.io/download.html`
You'll need to create/save a password that lets you open your wallet. Be sure to keep the password safe!

Once installed, go get some Ropsten Testnet Ether from a Ropsten faucet: e.g. `https://faucet.dimensions.network/` or `https://faucet.metamask.io/`. This should despoit 1-5 Ether into your MetaMask account. Make sure you have the Ropsten Test Network selected (instead of Main Ethereum Network) in your MetaMask account to verify the balance.

Use MyEtherWallet

MyEtherWallet is used to interact with existing and deploy new smart contracts. Go to `https://www.myetherwallet.com/` and make a wallet. You don't need to create a new wallet, since you connect to MetaMask instead. When you get to the creating a wallet step, click on "Access My Wallet", and then select "MEW CX" (we'll connect with MetaMask). Accept the terms and click the "Access My Wallet" button. This should bring up a MetaMask Connect Request notification, where you can click Connect.

Interact with a contract / send a transaction Click on the Contract / Interact with Contract:
`https://www.myetherwallet.com/interface/interact-with-contract`

Enter a contract address: `0x36A540E3A78084962B75E25877CfACf8846Be018`
and copy/paste the ABI/JSON at `https://ecen4133.org/static/vuln-abi.json` into the ABI/JSON Interface field. Click Continue.

Deposit Ether in contract This contract lets you deposit and withdraw Ether, and tracks the balance that you've entered. Try using the deposit function to deposit a small amount of Ether (e.g. 0.05 ETH). Enter the value and click Write, then confirm the transaction in MetaMask. This creates a transaction, that you can click on in MetaMask to view on Etherscan.io (an Ethereum

block/contract/transaction explorer website). You can see all the transactions that have interacted with this contract here:
`https://ropsten.etherscan.io/address/0x36A540E3A78084962B75E25877CfACf8846Be018`

How much ETH does this contract have in its balance?

Read from the contract Now select "balances" and enter your MetaMask account address (e.g. `0x5a7A84f39f5F9E653b9ee54DC1dB0BfF34EF2b10`). Click Read. The value returned will be in *Wei*, which is 10^{-18} Ether, the smallest unit of ETH.

Notice that reading a contract does not require any confirmation from MetaMask, and does not create a transaction. Why not?

Make your own contract with Remix

Remix is an online IDE for Ethereum smart contracts. You can write contracts in Solidity, a high-level language that compiles down to Ethereum VM Bytecode (what miners use to execute contracts). Go to `https://remix.ethereum.org/`.

Select Solidity, and New File, and give it a name. You can write your own source, or copy in the above vulnerable contract (and modify it, if you want!) from your Github Classroom repo (`vuln.sol`). Click on Compile.

Deploy your contract to the testnet Remix can deploy your contract for you, if you select the "Injected Web3" Environment. This will require you to confirm with MetaMask (again, ensure you are on the Ropsten Testnet!).

You can also use MyEtherWallet to deploy a new contract (Contract / Deploy Contract). Paste in your Bytecode and ABI/JSON copied from Remix, and give it a name. Click Sign Transaction, and confirm with MetaMask, and soon you'll be able to see (and interact) with your contract!

What to submit Part 0 is just for introduction, no need to submit anything for this part.

Part 1. Vulnerable Contracts

In this part, you'll investigate how to steal Ethereum from vulnerable smart contracts. We have setup a vulnerable smart contract on the Ropsten testnet that you have permission to steal funds from. However, using this technique on other contracts you do not have the same permission for outside this class is a *crime*. Remember that just because you *can* do something technical, doesn't mean that you *should*!

We've deployed our contract to address `0x36A540E3A78084962B75E25877CfACf8846Be018`. The source is available in your Github starter code as `vuln.sol`.
The contract has two functions: `deposit` and `withdraw` that let you send and receive money from the contract. On the surface, it appears that an address will only be able to withdraw what that address originally deposited. But this contract is vulnerable to a bug that lets you extract more if you're clever!

In this part, you'll write and use a contract that steals funds from the Vuln contract. Your goal is to make a contract that includes a `payable` function, that interacts with the Vuln contract to steal funds from it. Your contract should let you pay it a small amount (e.g. 0.1 ETH), and then later let you extract a greater amount (e.g. 0.2 ETH). If we look at the (internal) transactions between your contract and the Vuln contract, we should see that yours sends (deposits) less than it gets back (withdraws) from the Vuln contract.

Note : This is a class-wide contract, and only has so many testnet coins in it. As such, please limit your stealing to a humble amount, so that others can also enjoy the thrill of stealing as well. We suggest trying to deposit/steal small amounts such as 0.1 ETH (100 finney) at a time. You don't need to steal all the coins to prove that you can, but if you end up stealing too much, you can always `deposit()` it back.

Feel free to deploy your own copy of the Vuln contract (and fund it with your own testnet coins) to practice attacking before you try to attack the main Vuln contract.

What to submit

1. Your stealing contract, `attack.sol`, that you used to extract more funds than you deposited.
2. A plaintext file, `attack.txt`, that lists the deployed address of your attack.sol contract (e.g. `0x5b95c5aff4b...217e`)
3. Upload the source to your contract's etherscan.io page via the **Contract** tab on the page. (The autograder will use this to check that the deployed address matches your `attack.sol` source)

Submission Checklist

Submit your code via Github Classroom: `https://classroom.github.com/a/buYR1cTi`

Part 1

Include your stealing contract **`attack.sol`**, and a plaintext file `attack.txt` with its deployed address.
Make sure to add the source of your contract to your contract's etherscan.io page via the **Contract** tab **/ Verify and Publish** function. Any license you choose is fine.