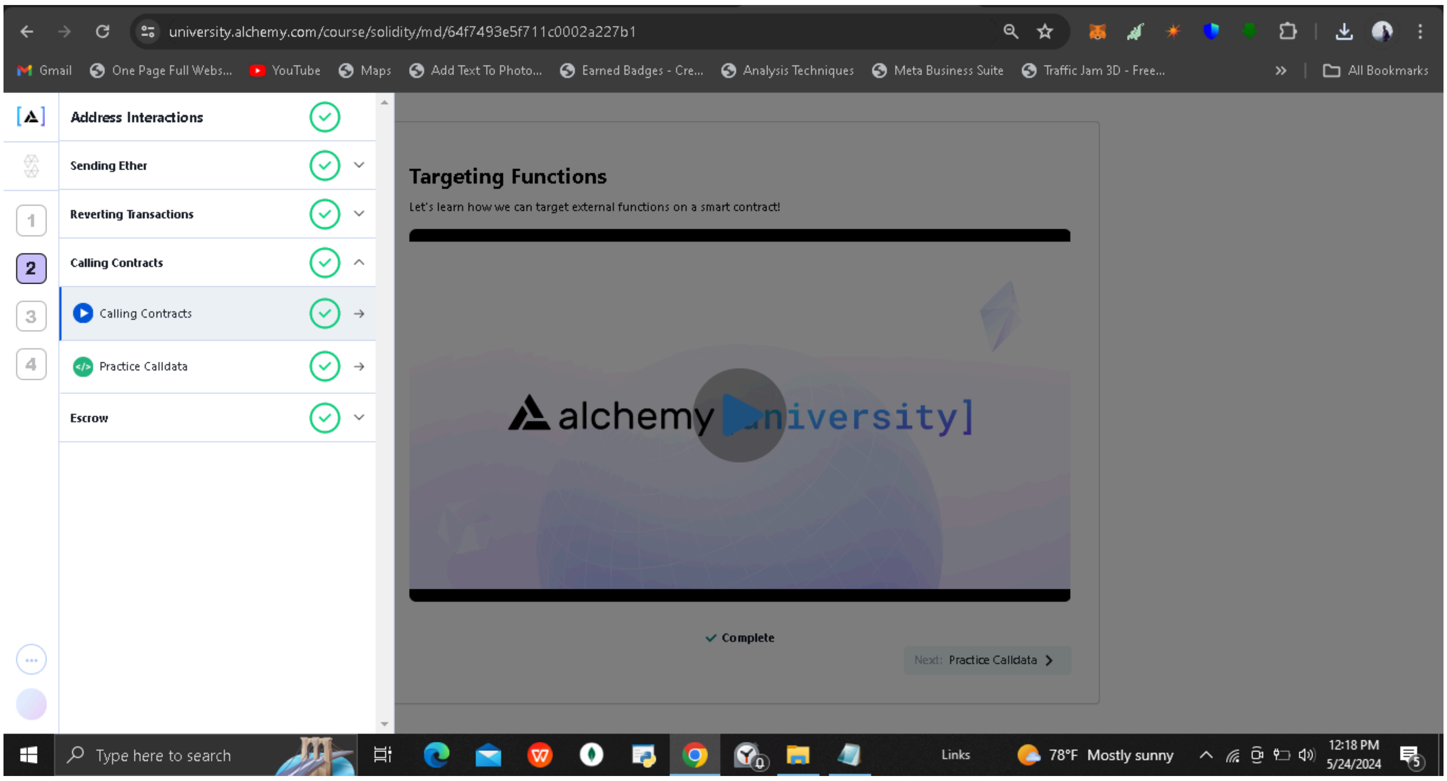
Lesson 2

Calling Contracts

- Call Function
- Signature
- With Signature
- Arbitrary Alert
- Fallback



Targeting Functions

Let's learn how we can target external functions on a smart contract!

- Call Function

The Sidekick contract you've provided serves as a mediator, allowing external callers to invoke functions on other contracts (referred to as "heroes") by relaying the function call and its data.

Let's break down the key components of your Sidekick contract:

1. **relay Function**:
   - This function is marked as external, meaning it can be called from outside the contract.
   - It takes two parameters:
     - hero: The address of the contract (the "hero") that the function call will be relayed to.

        - data: A bytes array containing the encoded function call and its arguments.
    - Inside the function, it invokes the call function on the hero contract, passing the provided data as the function call data.
    - It captures the success status of the call and ignores any returned data (denoted by _).
    - It then requires that the call was successful, reverting the transaction if it wasn't.
2. **Functionality**:
    - This contract essentially acts as a middleman or "sidekick" that relays function calls and data from external sources to other contracts.
    - By using the low-level call function, it forwards the provided data to the specified hero contract, executing any function specified in the data.
3. **Security Considerations**:
    - Using call to execute external code can be risky, as it opens the door to potential reentrancy attacks and unexpected behavior.
    - Ensure that the data being relayed is properly validated to prevent malicious or unintended function calls.
    - Carefully consider the permissions and access control mechanisms in place for invoking functions on the "hero" contracts.
4. **Usage**:
    - External callers can interact with the Sidekick contract by calling its relay function and specifying the target hero contract and the desired function call along with its arguments encoded in the data parameter.

Overall, the Sidekick contract provides a flexible mechanism for relaying function calls and data between contracts but requires careful consideration of security implications and proper validation of input data.

- Signature

1. **Function Signature**:
    - The function signature for the alert function is "alert()". You compute its function selector using keccak256 hashing and then extract the first four bytes.
    - bytes4 is used to store the function selector.
2. **Function Call**:
    - The abi.encodeWithSelector function is used to encode the function call with its signature.
    - This encoded data is then used as the parameter for the call function, which invokes the alert function on the target contract (hero).
3. **Error Handling**:
    - The require statement ensures that the function call was successful. If the call fails (returns false), it reverts the transaction with an error message.

With these changes, the sendAlert function can properly invoke the alert function on the specified hero contract. Ensure that the alert function is defined in the target contract and handles the incoming call appropriately.

- With Signature

The Sidekick contract's sendAlert function is designed to invoke a specific function called alert on the target contract (hero) with two parameters: enemies of type uint256 and armed of type bool.

Your contract correctly uses abi.encodeWithSignature to encode the function call with its signature and parameters.

```
contract Sidekick {
    function sendAlert(address hero, uint enemies, bool armed) external {
        // Encode the function call with its signature and parameters
        bytes memory data = abi.encodeWithSignature("alert(uint256,bool)",
enemies, armed);

        // Call the function on the target contract
        (bool success, ) = hero.call(data);

        // Ensure the function call was successful
        require(success, "Function call failed");
    }
}
```

- The abi.encodeWithSignature function is used to encode the function call with its signature and parameters.
- In this case, the signature is "alert(uint256,bool)", representing a function named alert that takes a uint256 and a bool as parameters.

- The enemies and armed parameters are passed to abi.encodeWithSignature to include them in the encoded data.
- This encoded data is then used as the parameter for the call function, which invokes the alert function on the target contract (hero).
- The require statement ensures that the function call was successful. If the call fails (returns false), it reverts the transaction with an error message.

This function allows your Sidekick contract to interact with other contracts by invoking the alert function on them with specified parameters. Ensure that the alert function is defined in the target contract and handles the incoming call appropriately.

- ## Arbitrary Alert

The Sidekick contract's relay function acts as a generic relay mechanism, allowing any function call with arbitrary data to be forwarded to another contract (hero). Let's break down the key aspects of this function:

```
contract Sidekick {

  function relay(address hero, bytes memory data) external {

    // send all of the data as calldata to the hero

    (bool success,) = hero.call(data);

    require(success);

  }

}
```

- **relay Function**: This function takes two parameters:
  - hero: The address of the contract to which the function call will be relayed.
  - data: A bytes array containing the encoded function call and its arguments.
- **Function Invocation**: The function uses the low-level call method to invoke the function specified by the provided data on the hero contract. call forwards all supplied gas and ether to the target contract and executes its code in the context of the calling contract.
- **Success Check**: After the function call, the (bool success,) tuple receives the boolean success status of the call. If success is false, it means the function call failed. The require statement ensures that the call was successful. If the call fails, the transaction will revert with an error.

This relay function provides a flexible way to interact with other contracts, as it can relay any function call with arbitrary data. However, it's important to exercise caution when using such generic relay mechanisms, as they can expose your contract to potential security vulnerabilities if not used carefully. Always ensure that the data being relayed is properly formatted and validated to prevent unintended behavior or exploits.

- Fallback

The makeContact function in your Sidekick contract aims to invoke a function named greet on the target contract (hero) with a parameter value of 5. However, based on your comment, it seems you intend to trigger the fallback function of the hero contract.

To trigger the fallback function of the hero contract, you can simply send Ether along with the function call to invoke the fallback function. Here's how you can modify your makeContact function to achieve this:

```
contract Sidekick {

    function makeContact(address payable hero) external payable {

        // Trigger the hero's fallback function by sending Ether

        (bool success,) = hero.call{value: msg.value}("");

        require(success, "Fallback function call failed");

    }

}
```

- The makeContact function now accepts Ether (payable) and takes the hero address as a parameter.
- The function directly invokes the fallback function of the hero contract by using the call method and sending the received Ether along with the call ({value: msg.value}).
- The (bool success,) tuple captures the success status of the call.
- The require statement ensures that the fallback function call was successful. If the call fails, the transaction will revert with an error message.

With this modification, invoking the makeContact function will trigger the fallback function of the hero contract and transfer any accompanying Ether to it.