**Intro to fuzz testing**

**Fuzz testing**

Generally, fuzz testing, also known as fuzzing, is an automated software testing technique that involves injecting invalid, malformed, or unexpected inputs into a system to identify software defects and vulnerabilities. This method helps in revealing issues that may lead to crashes, security breaches, or performance problems. Fuzz testing operates by feeding a program with large volumes of random data (referred to as "fuzz") to observe how the system handles such inputs. If the system crashes or exhibits abnormal behavior, it indicates a potential vulnerability or defect that needs to be addressed.

How can we apply this in Foundry?

Let's find out by testing the fact that fulfillRandomWords can only be called after the upkeep is performed.

Open RaffleTest.t.sol and add the following:

import {VRFCoordinatorV2\_5Mock} from "chainlink/src/v0.8/vrf/mocks/VRFCoordinatorV2\_5Mock.sol"; in the import section.

function testFulfillRandomWordsCanOnlyBeCalledAfterPerformUpkeep()

public

raffleEntredAndTimePassed

{

// Arrange

// Act / Assert

vm.expectRevert("nonexistent request");

// vm.mockCall could be used here...

VRFCoordinatorV2\_5Mock(vrfCoordinator).fulfillRandomWords(

0,

address(raffle)

);

}

So we define the function and use the modifier we created in the previous lesson to make PLAYER enter the raffle and set block.timestamp into the future. We use the expectRevert because we expect the next call to revert with the "nonexistent request" message. How do we know that? Simple, inside the VRFCoordinatorV2Mock we can see the following code:

function fulfillRandomWords(uint256 \_requestId, address \_consumer) external nonReentrant {

fulfillRandomWordsWithOverride(\_requestId, \_consumer, new uint256[](0));

}

/\*\*

\* @notice fulfillRandomWordsWithOverride allows the user to pass in their own random words.

\*

\* @param \_requestId the request to fulfill

\* @param \_consumer the VRF randomness consumer to send the result to

\* @param \_words user-provided random words

\*/

function fulfillRandomWordsWithOverride(uint256 \_requestId, address \_consumer, uint256[] memory \_words) public {

uint256 startGas = gasleft();

if (s\_requests[\_requestId].subId == 0) {

revert("nonexistent request");

}

If the requestId is not registered, then the if (s\_requests[\_requestId].subId == 0) check would revert using the desired message.

Moving on, we called vm.expectRevert then we called fulfillRandomWords with an invalid requestId, i.e. requestId = 0. But why only 0, what if it works with other numbers? How can we test the same thing with 1000 different inputs to make sure that this is more relevant?

Here comes Foundry fuzzing:

function testFulfillRandomWordsCanOnlyBeCalledAfterPerformUpkeep(uint256 randomRequestId)

public

raffleEntredAndTimePassed

{

// Arrange

// Act / Assert

vm.expectRevert("nonexistent request");

// vm.mockCall could be used here...

VRFCoordinatorV2\_5Mock(vrfCoordinator).fulfillRandomWords(

randomRequestId,

address(raffle)

);

}

If we specify an input parameter in the test function declaration, Foundry will provide random values wherever we use that parameter inside our test function.

This was just a small taste. Foundry fuzzing has an enormous testing capability. We will discuss more about them in the next sections.