Truster

More and more lending pools are offering flashloans. In this case, a new pool has launched that is offering flashloans of DVT tokens for free.
The pool holds 1 million DVT tokens. You have nothing.
To pass this challenge, rescue all funds in the pool executing a single transaction. Deposit the funds into the designated recovery account.

The attack strategy

My main goal as an attacker here would be to get access and control over the pool and transfer all funds in my account, in 1 transaction and without depositing any funds myself in the pool first or having any money in my account.

Inside the main function of the contract the `flashloan` function. There is a balance check : ``` if (token.balanceOf(address(this)) < balanceBefore) {
            revert RepayFailed();
        }

        return true;
    }
```
to make sure that the balance of the address if the borrower, is equal to or bigger than it was before. And this is how flashloans work. If the borrower can't pay back their loan in the same transaction, then it reverts. So we obviously can't manipulate this part. However, within the same function we see this statement here : ```
            target.functionCall(data);
    ```

This is here is an external call with no validation or any kind of obvious restriction or control over who can call this function and what data they can pass into it. So this is a great opportunity for any attacker to exploit the contract through. Every time a lender or an attacker calls the flashloan function, the lender contract makes an external call to the target contract.

We can see the ` functioncall in the "@openzeppelin/contracts/utils/Address.sol" : ``` function functionCall(address target, bytes memory data) internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0);
    }
```

So the call is done from the pool contract, however, I can call any target contract on behalf of the pool and pass any data to it. But to transfer the funds to myself I need to be granted access and control over all the funds in the first place, and I need to do that in a different way than taking a loan because in that case, my transaction will revert.
So the tokens in the contract are ERC-20 tokens, and we know that 2 of the core fundamental functions in erc-20 contracts is the `approve(address spender, uint256 amount) ` and the transferFrom(address sender, address recipient, uint256 amount) function.
“The approve function Sets amount as the allowance of spender over the caller’s tokens.” and “ transferFrom Moves amount tokens from sender to recipient using the allowance mechanism. amount is then deducted from the caller’s allowance.”

So in other words, since I can make any external calls on behalf of the pool. I can make the pool call this function, pass the spender as my address, and pass the amount as the maximum amount available in the pool. And once it is approved and I gain access to all the funds, I can then use the transferfrom function to transfer all the funds into my account, passing the sender as the pool, the recipient as my account address, and the amount as the maximum amount of funds in the pool. All of that would be done in a single function in my attack contract, utilizing all of the things we discussed.

- The main vulnerability here was the fact that in the Trsuster contract that has an external call that anyone can take advantage of, there is no validation or restriction over who can call it, what data they can call, or any maximum limit for the ‘amount’ to pass per se. So an attacker can take advantage of this and utilize the pool as an intermediary to access and transfer all the funds to their address.
```

```
    }

    function attack(address attacker) external {
        uint256 poolBalance = token.balanceOf(address(pool));
        bytes memory data = abi.encodeWithSignature("approve(address,uint256)", attacker, poolBalance);
        pool.flashLoan(
            0,
            address(this),
            address(token), data );
        token.transferFrom(address(pool), attacker, poolBalance);
    }
}

```