ð Encrypted Variables - Preventing Exposure

Ensuring that encrypted data and variables are not leaked is important when working with Fhenix. A common oversight when working with encrypted variables is revealing them to other contracts. Lets take a look at a scenario that leaks encrypted data:

contract

UserBalanceVulnerable

{ mapping (address

=> euint64)

public eUserBalances ;

function

addBalance (inEuint64 calldata _inBalance)

public

{ eUserBalances [msg . sender]

= eUserBalances [msg . sender] . add (FHE . asEuint64 (_inBalance)) ; } } This seems secure enough and no decrypted data is directly exposed, however thepublic access toeUserBalances leaks sensitive data. A malicious contract is able to fetch this data and then decrypt it:

contract

UserBalanceAttack

{ address

private vulnerableContract ;

function

revealUserBalance (address _user)

public

view

returns

(uint64)

{ return

UserBalanceVulnerable (vulnerableContract) . eUserBalances (_user) . decrypt ( ) ; } } All contracts on the Fhenix network share an encryption key, therefore an encrypted variable inContractA could be decrypted inContractB .

This is not inherently wrong, and many operations will require encrypted variables to be shared between contracts, but care must be taken to prevent open access to encrypted variables.

Hardhat Taskâ

Thefhenix-hardhat-plugin package contains a task that checks your contracts for any exposed encrypted variables. This task is run automatically when your contracts are compiled, but can also be run manually.

Task Exampleâ

The following contract exposes encrypted variables in 3 ways.

pragma

solidity

= 0.8.13

```
< 0.9.0 ;
import
"@fhenixprotocol/contracts/FHE.sol" ;
contract
ContractWithExposedVariables
{ // Example 1 mapping ( address
=> euint8 )
public eUserBalances ;
// Example 2 mapping ( address
=> euint8 )
private _eUserBalances ; function
getUserBalance ( address _user )
public
view
returns
( euint8 )
{ return _eUserBalances [ _user ] ; }
// Example 3 struct
Player
{ address player ; euint8 [ ] eCards ; uint256 chips ; uint256 bet ; } struct
Dealer
{ uint256 pot ; euint8 [ ] eFlopCards ; } struct
HoldEmGameState
{ Player [ ] players ; Dealer dealer ; }
HoldEmGameState private gameState ; // Encrypted card values is the Player and Dealer structs are leaked and can be exploited function
getGameState ( )
public
view
returns
( HoldEmGameState memory )
{ return gameState ; } }
```

Outputâ

Below is the output of the task when analyzing the aboveContractWithExposedVariables.sol

```
fhenix-hardhat-plugin:CheckExposedEncryptedVars checking for exposed encrypted variables....

contracts/ContractWithExposedVariables.sol:ContractWithExposedVariables eUserBalances(address) exposes 1 encrypted variables: pos-0 -euint8

getUserBalance(address) exposes 1 encrypted variables: pos-0 -euint8
```

getGameState() exposes 1 encrypted variables: pos-0 - struct ContractWithExposedVariables.HoldEmGameState players - struct ContractWithExposedVariables.Player[] eCards -euint8[] dealer - struct ContractWithExposedVariables.Dealer eFlopCards -euint8[]

Manual Task Executionâ

The task can be run manually with the command:

npx hardhat task:fhenix:checkExposedEncryptedVars Or as a part of a hardhat compilation:

npx hardhat compile Edit this page

Previous Examples & fheDapps Next ð Permits & Access Control