

Security & Auditing :

Berikut adalah beberapa contract yang memiliki celah keamanannya masing masing. Yang pertama adalah BadRNG, dimana contract ini mengambil pemenang dari raffle berdasarkan kesusahan block dan pesan pengirim.

```
... .env ethers-simple-storage .env hardhat-erc20-fcc .env hardhat-nextjs-nft-marketplace-fcc .env dao-template Back
hardhat-security-fcc > contracts > BadRNG.sol
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 // One shouldn't use any values from inside the blockchain as randomness
5 // Use something like Chainlink VRF for verifiable randomness
6 // https://docs.chain.link/docs/get-a-random-number/
7
8 contract BadRNG {
9     address payable[] private s_players;
10
11     function enterRaffle() external payable {
12         require(msg.value >= 1000000000000000000);
13         s_players.push(payable(msg.sender));
14     }
15
16     function pickWinner() external {
17         uint256 randomWinnerIndex = uint256(
18             keccak256(abi.encodePacked(block.difficulty, msg.sender))
19         );
20         address winner = s_players[randomWinnerIndex % s_players.length];
21         (bool success, ) = winner.call{value: address(this).balance}("");
22         require(success, "Transfer failed");
23     }
24 }
25
26 // How could you make a contract that exploits this?
27
```

Selanjutnya adalah LiquidityPool sebagai Oracle, dimana banyak orang dapat membeli, menjual, atau menukar berbagai macam aset dengan menggunakan fungsi penukaran tunggal itu tidak baik karena seharusnya diterapkan sistem yang terdesentralisasi agar aman dari ancaman perubahan data.

```
.env ethers-simple-storage .env hardhat-erc20-fcc .env hardhat-nextjs-nft-marketplace-fcc .env dao-template LiquidityPoolAsOracle.sol 2, M X
hardhat-security-fcc > contracts > LiquidityPoolAsOracle.sol
22 s_token2 = token2;
23 }
24
25 function swap(
26     address from,
27     address to,
28     uint256 amount
29 ) external {
30     require(
31         (from == s_token1 && to == s_token2) || (from == s_token2 && to == s_token1),
32         "Invalid tokens"
33     );
34     require(IERC20(from).balanceOf(msg.sender) >= amount, "Not enough to swap");
35     uint256 swap_amount = getSwapPrice(from, to, amount);
36     bool txFromSuccess = IERC20(from).transferFrom(msg.sender, address(this), amount);
37     require(txFromSuccess, "Failed to transfer from");
38     bool txToSuccess = IERC20(to).transfer(msg.sender, swap_amount);
39     require(txToSuccess, "Failed to transfer to");
40 }
41
42 function addLiquidity(address tokenAddress, uint256 amount) external {
43     bool success = IERC20(tokenAddress).transferFrom(msg.sender, address(this), amount);
44     require(success, "Failed to add liquidity");
45 }
46
47 function getSwapPrice(
48     address from,
49     address to,
50     uint256 amount
51 ) public view returns (uint256) {
52     return ((amount * IERC20(to).balanceOf(address(this))) /
53         IERC20(from).balanceOf(address(this)));
54 }
55 }
56
```

Yang lainnya adalah Metamorphic Contract dimana contract dapat diinisialisasi, namun tidak dapat menjamin contract tersebut berhasil di inisialisasi

```
.env ethers-simple-storage .env hardhat-erc20-fcc .env hardhat-nextjs-nft-marketplace
hardhat-security-fcc > contracts > MetamorphicContract.sol
1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.8.7;
3 import "@openzeppelin/contracts/proxy/utils/Initializable.sol";
4
5 contract MetamorphicContract is Initializable {
6     address payable owner;
7
8     function kill() external {
9         require(msg.sender == owner);
10        selfdestruct(owner);
11    }
12 }
13
```

Dan juga masalah keamanan seperti Reentrancy

```
hardhat-security-fcc > contracts > Reentrancy.sol
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 // https://solidity-by-example.org/hacks/re-entrancy/ for full example
5 // Follow https://twitter.com/programmersmart
6
7 contract EtherStore {
8     mapping(address => uint256) public balances;
9
10    function deposit() external payable {
11        balances[msg.sender] += msg.value;
12    }
13
14    function withdraw() external {
15        uint256 balance = balances[msg.sender];
16        require(balance > 0);
17        (bool success, ) = msg.sender.call{value: balance}("");
18        require(success, "Failed to send Ether");
19        balances[msg.sender] = 0;
20    }
21
22    function getBalance() external view returns (uint256) {
23        return address(this).balance;
24    }
25 }
26
27
```

Untuk melihat celah keamanan dari smart contract ini, dapat digunakan tools seperti slither yang dapat diinstall dengan perintah pip install slither-analyzer

```
gitpod /workspace/new_folder/hardhat-security-fcc (main) $ pip install slither-analyzer
Collecting slither-analyzer
  Downloading slither_analyzer-0.8.3-py3-none-any.whl (547 kB)
    547.9/547.9 kB 15.1 MB/s eta 0:00:00
Collecting prettytable>=0.7.2
  Downloading prettytable-3.3.0-py3-none-any.whl (26 kB)
Collecting crytic-compile>=0.2.3
  Downloading crytic_compile-0.2.3-py3-none-any.whl (87 kB)
    87.2/87.2 kB 37.7 MB/s eta 0:00:00
Collecting pysha3>=1.0.2
  Downloading pysha3-1.0.2.tar.gz (829 kB)
    829.2/829.2 kB 68.7 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: wcwidth in /home/gitpod/.pyenv/versions/3.8.13/lib/python3.8/site-packages (from prettytable>=0.7.2->slither-analyzer) (0.2.5)
Building wheels for collected packages: pysha3
  Building wheel for pysha3 (setup.py) ... done
  Created wheel for pysha3: filename=pysha3-1.0.2-cp38-cp38-linux_x86_64.whl size=153785 sha256=b4d1d873cd6951ee99212c532ae97d223af0ef6ff82fb9b4c747932ff9aeed44
  Stored in directory: /home/gitpod/.cache/pip/wheels/ee/35/47/386d23587167a3b03fc702b6be1ff60c92fddbeef125873888
Successfully built pysha3
Installing collected packages: pysha3, prettytable, crytic-compile, slither-analyzer
Successfully installed crytic-compile-0.2.3 prettytable-3.3.0 pysha3-1.0.2 slither-analyzer-0.8.3
gitpod /workspace/new_folder/hardhat-security-fcc (main) $
```

Setelah slither terinstall kita dapat menjalankannya dengan perintah yarn slither dan akan menampilkan beberapa baris kode yang memiliki celah keamanan. Untuk

```
--generate-patches Generate patches (json output only)
gitpod /workspace/new_folder/hardhat-security-fcc (main) $ yarn slither
yarn run v1.22.19
$ slither . --solc-remaps '@openzeppelin=node_modules/@openzeppelin @chainlink=node_modules/@chainlink' --exclude naming-convention,external-function,low-level-calls
'npx hardhat compile --force' running
Compiled 12 Solidity files successfully

Reentrancy in EtherStore.withdraw() (contracts/Reentrancy.sol#15-21):
  External calls:
    - (success) = msg.sender.call{value: balance}() (contracts/Reentrancy.sol#18)
  State variables written after the call(s):
    - balances[msg.sender] = 0 (contracts/Reentrancy.sol#20)
  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities

MetamorphicContract.owner (contracts/MetamorphicContract.sol#6) is never initialized. It is used in:
  - MetamorphicContract.kill() (contracts/MetamorphicContract.sol#8-11)
  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables

Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#201-221) uses assembly
  - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#213-216)
  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

VaultFuzzTest.echidna_test_find_password() (contracts/test/fuzzing/VaultFuzzTest.sol#9-11) compares to a boolean constant:
  - s_locked == true (contracts/test/fuzzing/VaultFuzzTest.sol#10)
  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
```

Bila terdapat kata berwarna merah, berarti terdapat celah keamanan pada contract yang perlu kita tindak lanjuti. Selain itu juga terdapat referensi dari permasalahan celah keamanan yang terkait ke website yang akan membantu cara mengatasi masalah tersebut.

Reentrancy vulnerabilities

Configuration

- Check: `reentrancy-eth`
- Severity: `High`
- Confidence: `Medium`

Description

Detection of the `reentrancy` bug. Do not report reentrancies that don't involve Ether (see `reentrancy-no-eth`)

Exploit Scenario:

```
function withdrawBalance(){
    // send userBalance[msg.sender] Ether to msg.sender
    // if msg.sender is a contract, it will call its fallback function
    if( ! (msg.sender.call.value(userBalance[msg.sender]))() ) ){
        throw;
    }
    userBalance[msg.sender] = 0;
}
```

Bob uses the re-entrancy bug to call `withdrawBalance` two times, and withdraw more than its initial deposit to the contract.

Recommendation

Apply the `check-effects-interactions pattern`.

Storage Signed Integer Array

Configuration

- Check: `storage-array`
- Severity: `High`