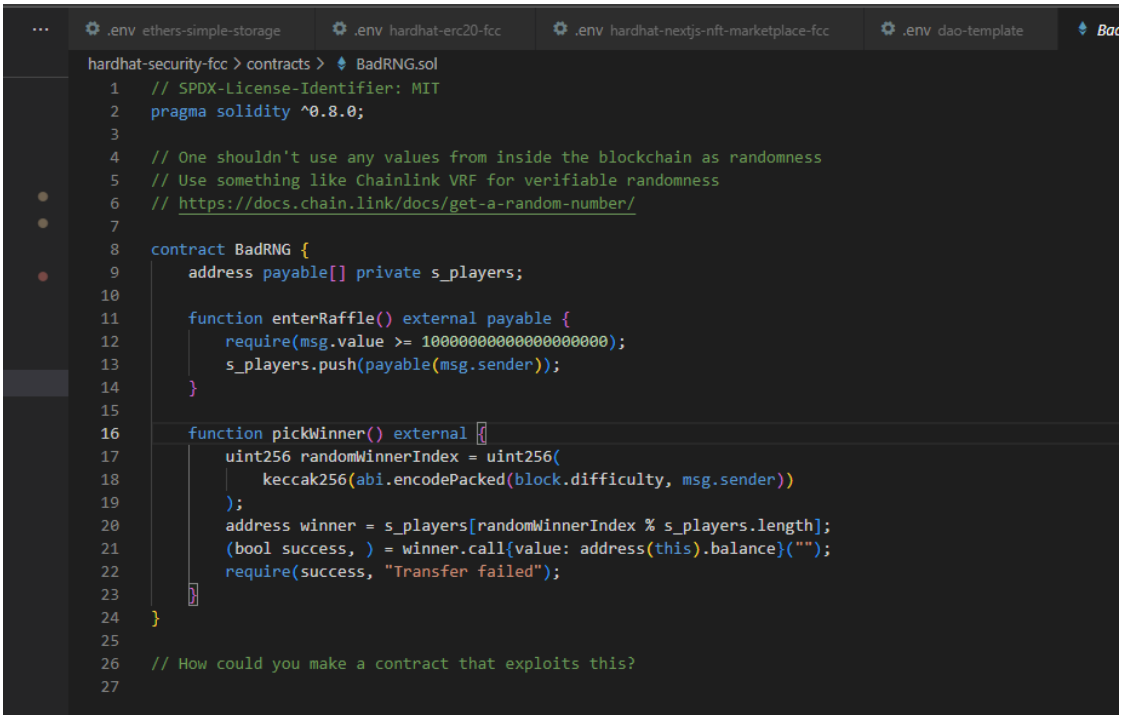
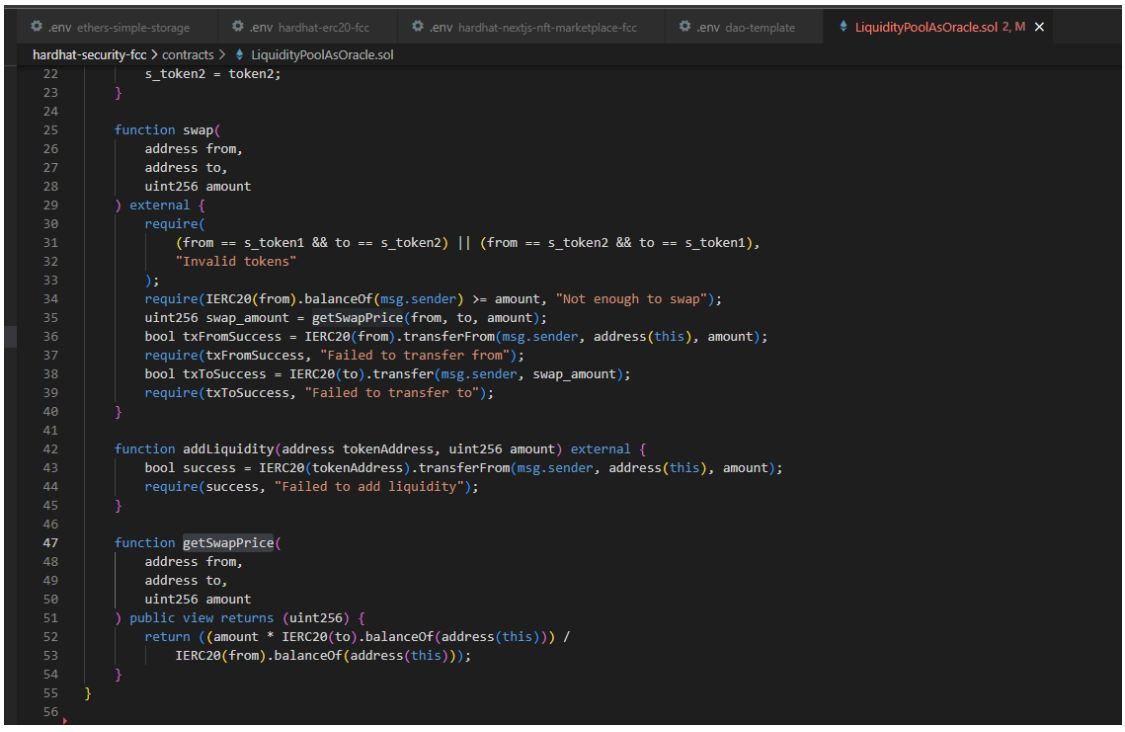
Security & Auditing :

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

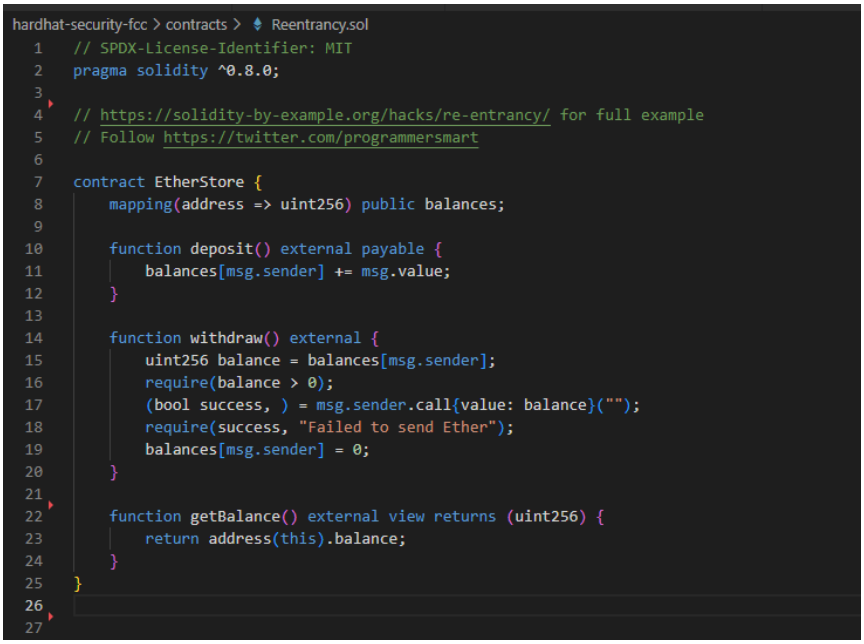


Selanjutnya adalah LiquidityPool sebagai Oracle, dimana banyak orang dapat membeli, menjual, atau menukar berbagai macam aset dengan menggunakan fungsi penukaran tunggal itu tidak baik karena seharusnya diterapkan sistem yang terdesentralisasi agar aman dari ancaman perubahan data.

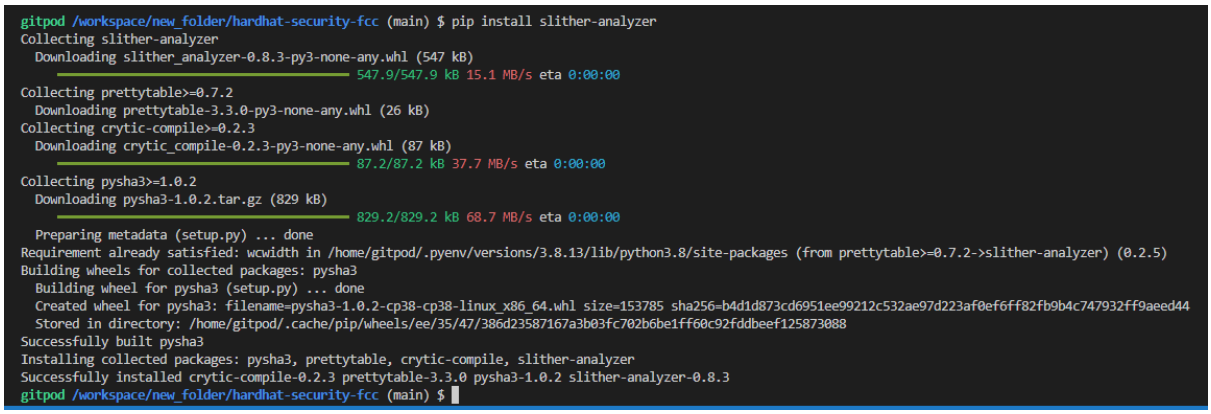


Yang lainnya adalah Metamorphic Contract dimana contract dapat diinisialisasi, namun tidak dapat menjamin contract tersebut berhasil di inisialisasi

Dan juga masalah keamanan seperti Reentrancy



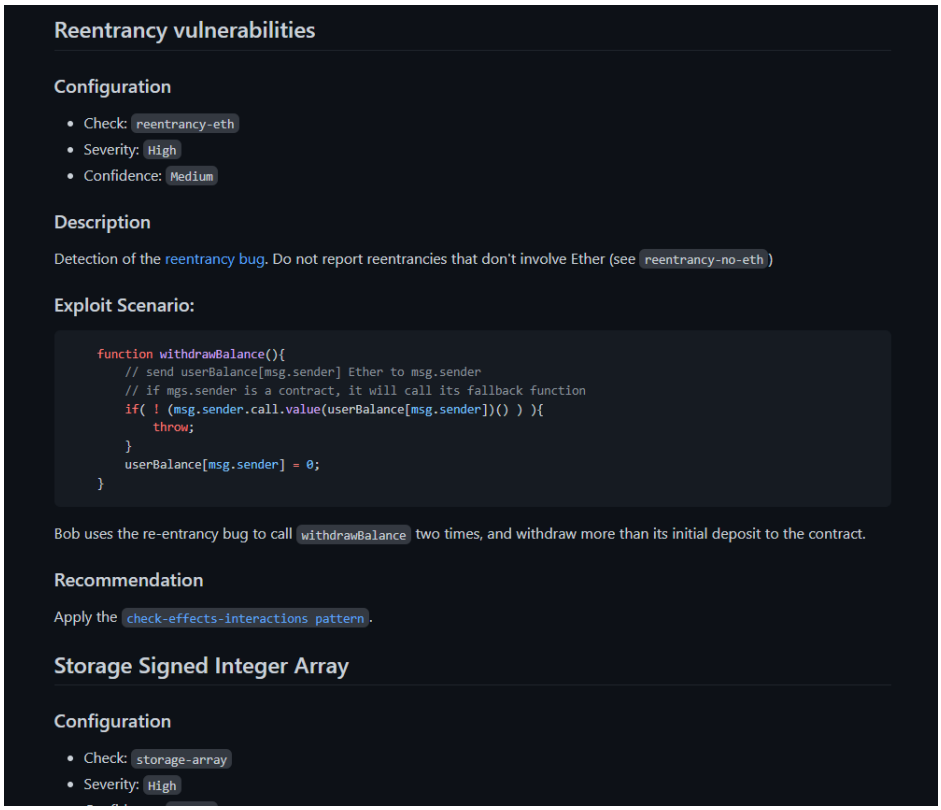
Untuk melihat celah keamanan dari smart contract ini, dapat digunakan tools seperti slither yang dapat diinstall dengan perintal pip install slither-analyzer



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 mgs.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`