“StorageVictim” Smart Contract: Initial Assessment

1. Scope

The smart contract called “StorageVictim” was reviewed for security flaws. Click on the following link to view the original code: https://github.com/jsanders108/StorageVictim-Audit/blob/main/Initial/StorageVictimOriginal.sol

Before commencing with the security review, it was requested that the contract be rewritten in a newer version of Solidity (0.8x) without changing the logic.

The following process was followed to complete this security audit:

a. The contract code was reviewed manually to 1) understand its mechanics and what it is trying to accomplish, and 2) identify any initial obvious security issues.
b. The contract was tested manually to determine if any intended functionality was not working correctly.
c. The static analysis tool called “Slither” was run on the contract to identify any red, yellow, or green issues.
d. Fuzz testing was run on the contract using a tool called “Echidna”.

2. Contract Upgrade

In order to compile successfully StorageVictim.sol in Remix with a newer version of Solidity (0.8x), the following changes to the code were implemented:

Issue #1: “Functions are not allowed to have the same name as the contract”:

```
function StorageVictim() public {
    owner = msg.sender;
}
```

Fix #1: The function name was changed to the following:

```
function storageVictim() public {
    owner = msg.sender;
}
```

Issue #2: Line #2 (“Storage str”): “Data location must be ‘storage’, ‘memory’, or ‘calldata’ for variable, but none was given”:

```
function store(uint _amount) public {
    Storage str;
    str.user = msg.sender;
    str.amount = _amount;
    storages[msg.sender] = str;
}
```

Fix #2: Line #2 (“Storage str”) was changed to the following:

```
function store(uint _amount) public {
    Storage memory str;
    str.user = msg.sender;
    str.amount = _amount;
    storages[msg.sender] = str;
}
```

Issue #3: Line #2 (“Storage str = storages[msg.sender]”): “Data location must be ‘storage’, ‘memory’, or ‘calldata’ for variable, but none was given”:

```
function getStore() public view returns (address, uint) {
    Storage str = storages[msg.sender];
    return (str.user, str.amount);
}
```

Fix #3: Line #2: “Storage str = storages[msg.sender]”) was fixed to the following:

```
function getStore() public view returns (address, uint) {
    Storage memory str = storages[msg.sender];
    return (str.user, str.amount);
}
```

---

*Note: The contract was also missing a license identifier at the top. Therefore, the following code snippet was added to the first line of the contract:

//SPDX-License-Identifier: MIT

3. Findings

3.1 Manual Review & Testing

Issue #1)

In the code below, anyone can set the “owner” of the contract to be themselves (msg.sender) simply by calling the public function “storageVictim”:

```
function storageVictim() public {
    owner = msg.sender;
}
```

Recommendation: It is best practice to use a constructor upon contract deployment to set the owner of the contract to be the deployer (msg.sender):

```
constructor() {
    owner = msg.sender;
}
```

Issue #2)

In the code below, anyone can adjust (at any time) the storage amount allocated to themselves (msg.sender) simply by calling the public function “store”:

```
function store(uint _amount) public {
    Storage memory str;
    str.user = msg.sender;
    str.amount = _amount;
    storages[msg.sender] = str;
}
```

Recommendation: If it is considered undesirable for anyone to be able to adjust their storage amount at any time (this depends on the contract’s objectives), then the ability to call this function should be restricted to the owner (which, per Issue #1, should be set upon contract deployment).

3.2 Static Analysis Testing: Slither

Red Issues
NONE

Yellow Issues
Issue #1)

In the code below, the variable “str” (line #2) is a local variable that is never initialized:

```
function store(uint _amount) public {
    Storage memory str;
    str.user = msg.sender;
    str.amount = _amount;
    storages[msg.sender] = str;
}
```

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variable

Recommendation: Rewrite/streamline the above code to the following:

```
function store(uint _amount) public {
    storages[msg.sender] = Storage(msg.sender, _amount);
}
```

Green Issues
Issue #1)

In the code below, the argument “_amount” is not in mixedCase:

```
function store(uint _amount) public {
    Storage memory str;
    str.user = msg.sender;
    str.amount = _amount;
    storages[msg.sender] = str;
}
```

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Recommendation: According to Solidity naming conventions, function arguments that are not private or unused should avoid using an underscore before the name. Therefore, consider rewriting the above code to the following:

```
    function store(uint amount) public {
        Storage memory str;
        str.user = msg.sender;
        str.amount = amount;
        storages[msg.sender] = str;
    }
```

Issue #2)

The contract was compiled using Solidity version 0.8.1. According to Slither, version 0.8.1 is not recommended for deployment.

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Recommendation: According to the reference document cited above, it is recommended that the contract should be deployed using Solidity version 0.8.16.

3.3 Fuzz Testing: Echidna

A fuzz test using Echidna was run to confirm that the following function—flagged during the manual review—is indeed problematic in that it is possible to change the owner's address after the contract has been deployed:

```
    function storageVictim() public {
        owner = msg.sender;
    }
```

Procedure:

1) The owner's address was hard coded into the StorageVictim contract:

```
//SPDX-License-Identifier: MIT
pragma solidity 0.8.16;

contract StorageVictim {

    //Hard coded an address in order to run an Echidna test
     address owner = 0xc5Fb0bCf2C7aB7Cb0B9E7d395337fA898D4E7E30;
```

2) The following Echidna test function was created to determine if it is possible to change the hard coded owner address to something else. The test fails if the function returns a boolean with the value of “false”:

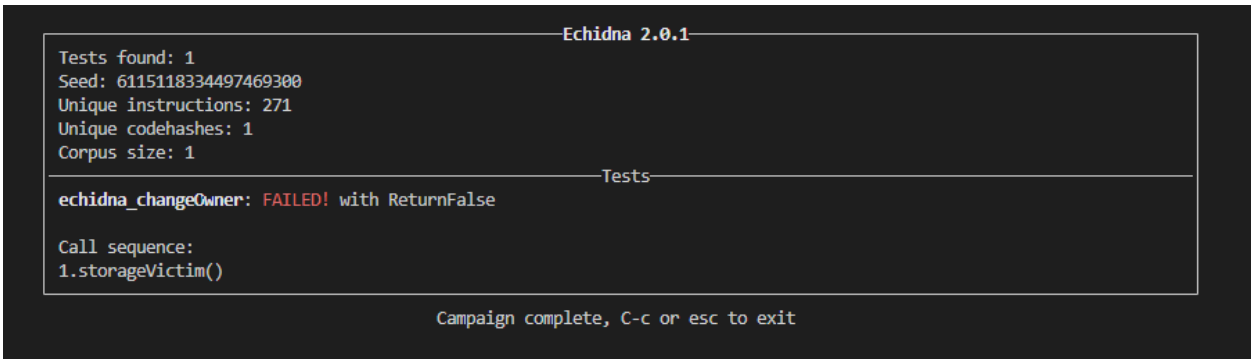
```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "./StorageVictim.sol";

contract FuzzStorageVictim is StorageVictim{

    function echidna_changeOwner() public view returns (bool){
        return owner == 0xc5Fb0bCf2C7aB7Cb0B9E7d395337fA898D4E7E30;
    }
}
```

3) The Echidna fuzzing test was run with the following result:



The fuzzing test failed. Further, the “call sequence” in the output above identifies that it is the “storageVictim” function that made this test fail (and therefore needs to be fixed).

Recommendation: The Echidna fuzz test confirms that using the “storageVictim” function to set the owner of the contract to msg.sender is problematic. Therefore, it is advised to use a constructor for this purpose.