

MythX is the premier security analysis service for Ethereum smart contracts. Their mission is to ensure development teams avoid costly errors and make Ethereum a more secure and trustworthy platform. In this article I will talk about how I can use Mythx during the life cycle of development, then how a client can use this plugin to automate smart contract security analysis of the Truffle Framework, after that I am going to talk about the vulnerability meaningful and test our Smart contract with Mythx.

## Secure your Remix project using MythX

---

In this section I will show to you how you can use MythX in remix Project, So MythX is a tool for finding smart contract weaknesses.

### Contract and scenario

In this section I will test the smart contract for Voting System:

```
pragma solidity ^0.4.20;

contract VotingSystem{
    struct Voter{
        bool alreadyVoted;
        uint vote;
    }

    struct Candidate{
        bytes32 name;
        uint votes;
    }

    mapping (address => Voter) public voters;
    Candidate[] public candidates;

    address public owner;

    bool public isOpen;

    function VotingSystem (bytes32[] candidatesNames) public
    {
        owner = msg.sender;
        isOpen = true;
        for (uint i=0; i < candidatesNames.length; ++i){
            // The candidate is included in the array
            candidates.push(Candidate({
                name: candidatesNames[i],
```

```

        votes: 0
    }));
}
}

function Vote (uint candidateVote) public {
    require(isOpen);
    Voter storage currentVoter = voters[msg.sender];
    require(!currentVoter.alreadyVoted);

    currentVoter.vote = candidateVote;
    currentVoter.alreadyVoted = true;

    candidates[candidateVote].votes++;
}

function CloseVoting () public {
    require((msg.sender == owner) && isOpen);
    isOpen = false;
}

// A function able to count votes and announce which
candidate is the winner
function GetWinner () constant returns (bytes32){
    require(!isOpen);
    uint maxAmountOfVotes = 0;
    uint winnerCandidate = 0;
    for (uint i=0; i < candidates.length; ++i){
        if (candidates[i].votes > maxAmountOfVotes){
            maxAmountOfVotes = candidates[i].votes;
            winnerCandidate = i;
        }
    }
    return candidates[winnerCandidate].name;
}
}

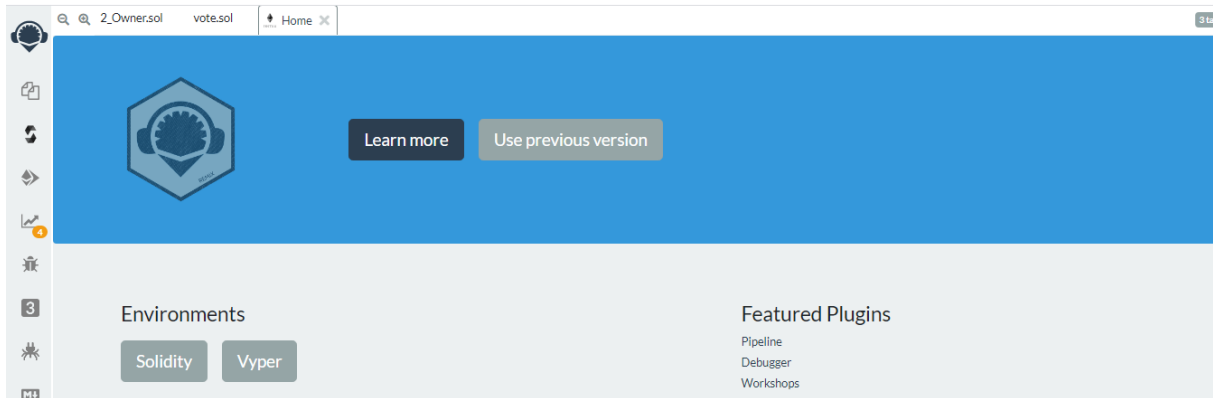
```

As a work in progress, I'm not quite ready for a manual audit yet, but wants to check the contract for weaknesses now with **MythX**.

## Running MythX

To check the weaknesses in my smart contract I will use Remix & Mythx

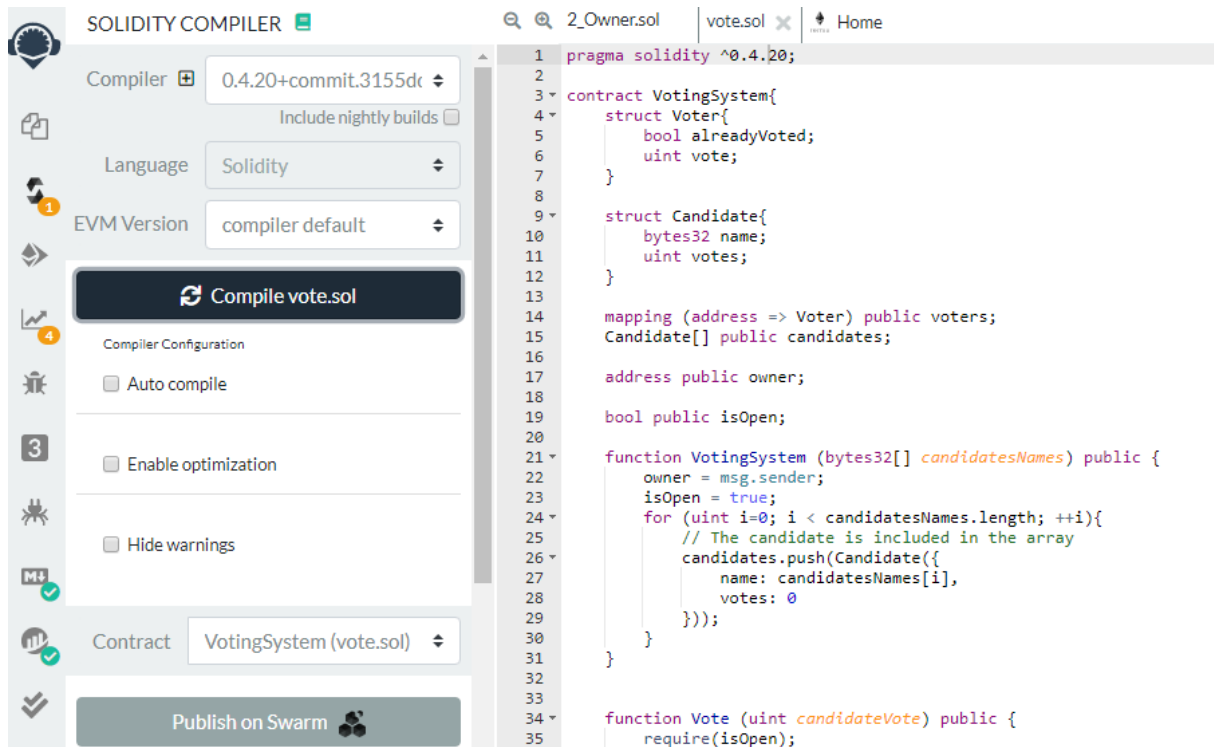
### 1. Open Remix.



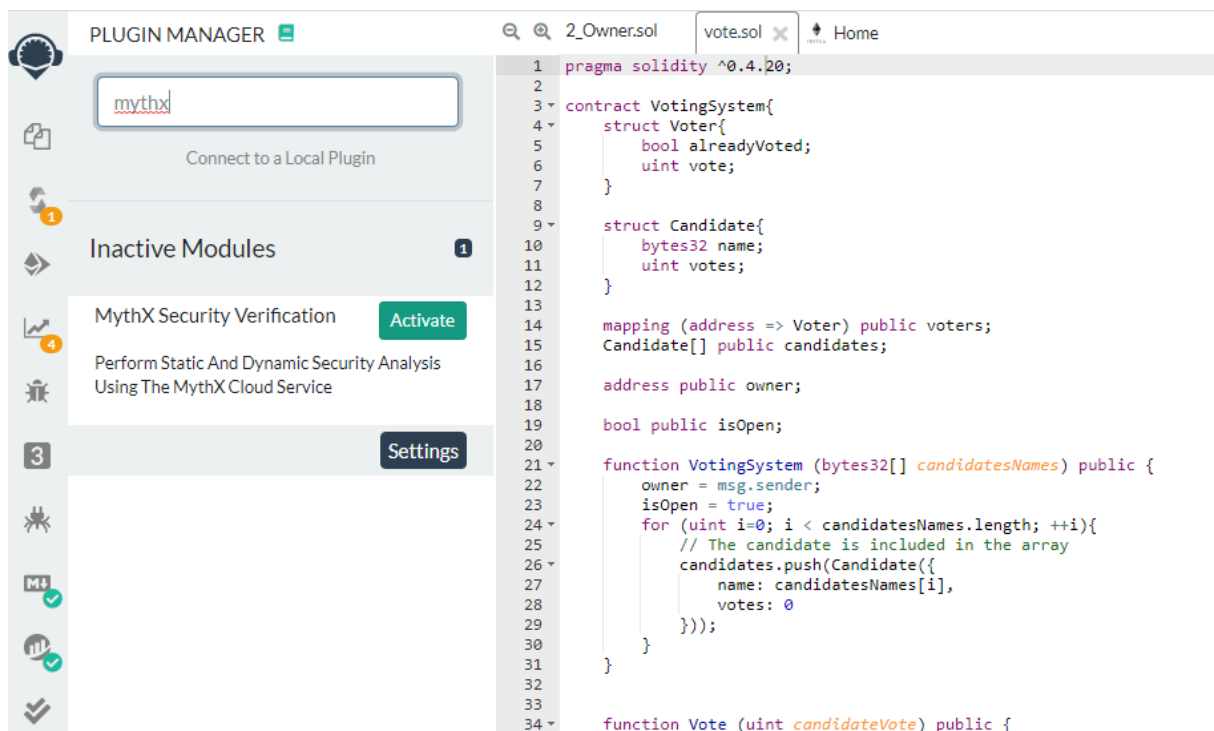
### 2. Open Remix.



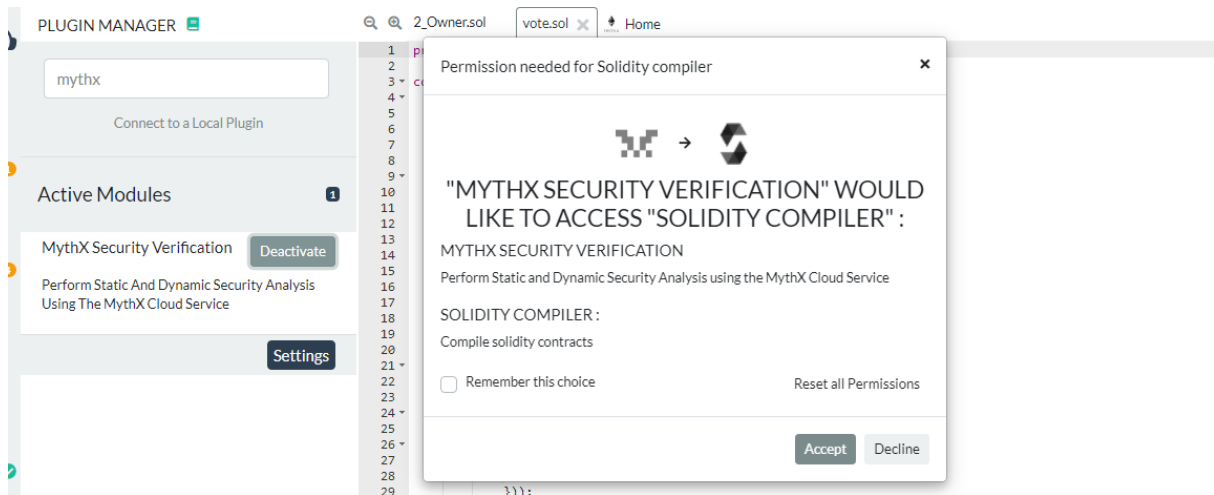
### 3. Use the Solidity compiler to compile your contract.



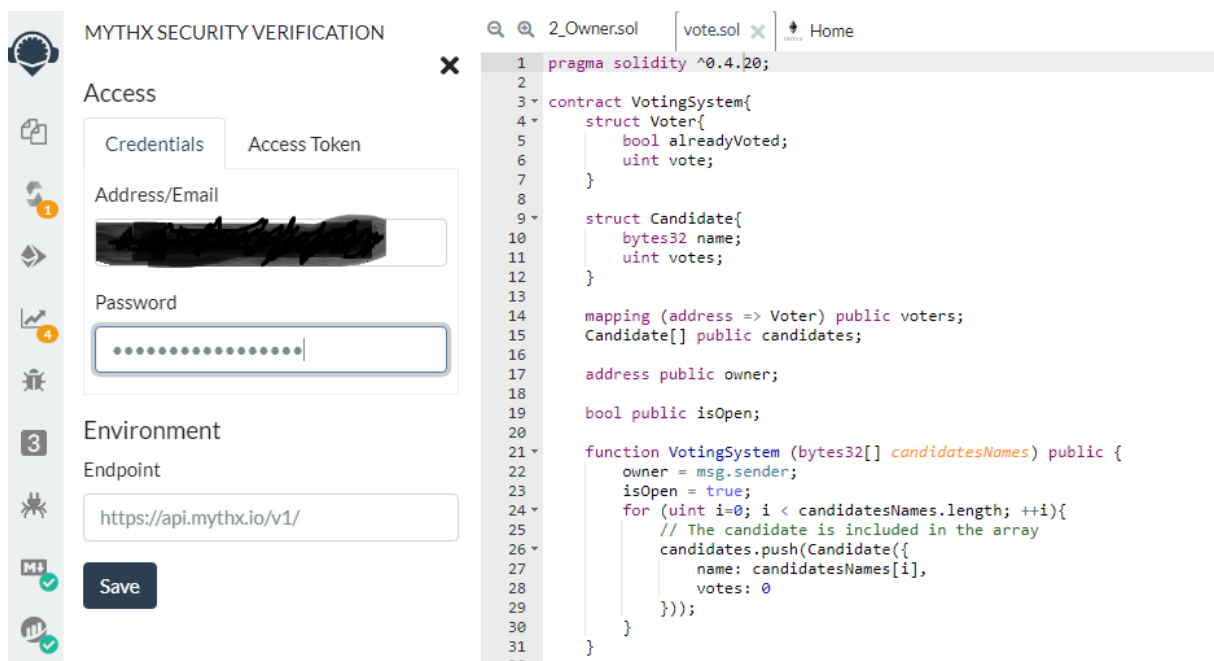
4. Enable the MythX plugin by going to the Plugin Manager, finding MythX Security Verification, and click Activate.




(If you see a “permission needed” window, click Accept.)



5. Enter your MythX credentials and click "Save". (You can sign up for a free account at [mythx.io](https://mythx.io).)



## MYTHX SECURITY VERIFICATION

Credentials 

hide 


Address

Password

Save


browser/Payable.sol::Demo 




Analyze 


Run full mode

6. Click "Analyze".



### MYTHX SECURITY VERIFICATION

browser/vote.sol::VotingSyste 


**Analyze** 


We are analyzing your contract. This should take up to 2 minutes



[Log](#) [Report](#)


[04/08/2020 12:50:29 AM] Your quick analysis is completed. See your results at [b5a3d266-cf2f-48a3-9cfa-7eb5580edc62](#)

**3**





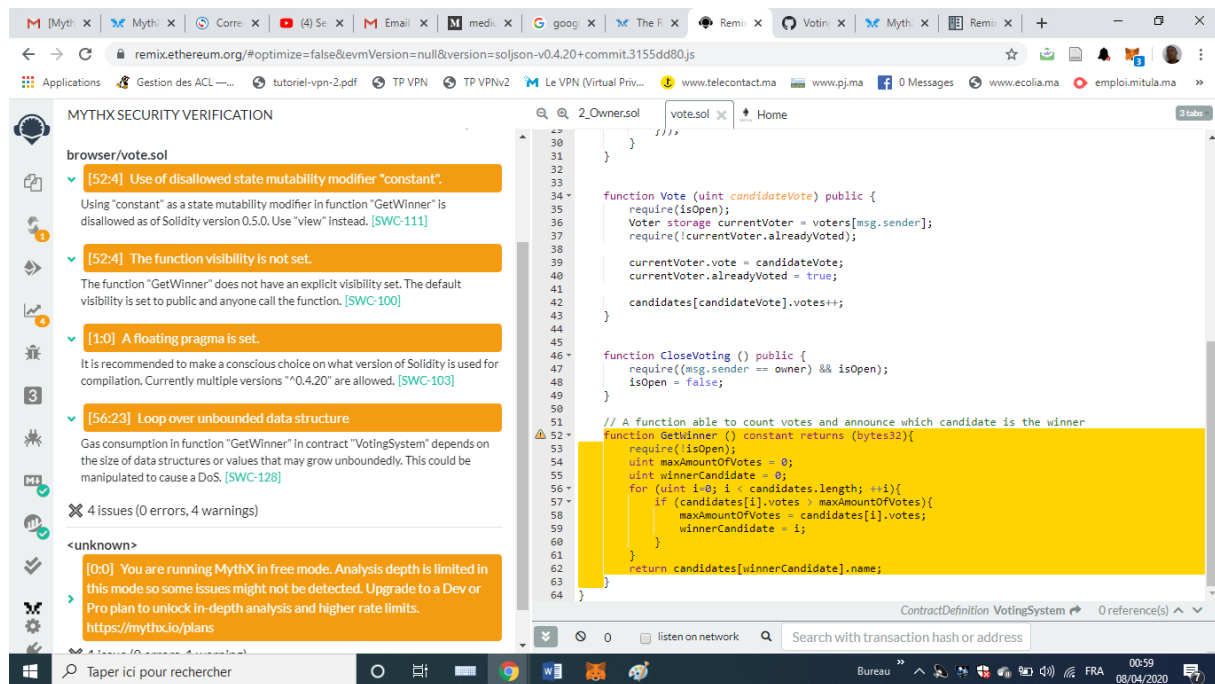
2\_Owner.sol vote.sol  Home 



## Environments

[Solidity](#) [Vyper](#)

By default, MythX will analyze your contract using the standard "quick" mode of a 2 minute scan. When finished, the results will be displayed in the main window.



✖ 4 issues (0 errors, 4 warnings)

<unknown>

✓ [0:0] You are running MythX in free mode. Analysis depth is limited in this mode so some issues might not be detected. Upgrade to a Dev or Pro plan to unlock in-depth analysis and higher rate limits. <https://mythx.io/plans>

Warning: Free mode only detects certain types of smart contract vulnerabilities. Your contract may still be unsafe. Upgrade to a Dev or Pro plan to unlock the ability to test for even more vulnerabilities, perform deeper security analysis, and more. <https://mythx.io/plans>

✖ 1 issue (0 errors, 1 warning)

## Viewing results

Let's take a look at each of the results in greater detail.

[52:4] Use of disallowed state mutability modifier "constant". [SWC-111]

Several functions and operators in Solidity are deprecated. Using them leads to reduced code quality. With new major versions of the Solidity compiler, deprecated functions and operators may result in side effects and compile errors.

```
[52:4]The function visibility is not set. [SWC-100]
```

Functions that do not have a function visibility type specified are public by default. This can lead to a vulnerability if a developer forgot to set the visibility and a malicious user is able to make unauthorized or unintended state changes.

```
[1:0]A floating pragma is set. [SWC-103]
```

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

```
[56:23]A floating pragma is set. [SWC-128]
```

SWC-128 refers to the potential for arrays to grow larger over time. If the array grows large enough, the gas required could exceed the block gas limit.

## Remediation

```
[52:4] Use of disallowed state mutability modifier "constant". [SWC-111]
```

Solidity provides alternatives to the deprecated constructions. Most of them are aliases, thus replacing old constructions will not break current behavior. For example, sha3 can be replaced with keccak256

```
[56:23]A floating pragma is set. [SWC-128]
```

Caution is advised when you expect to have large arrays that grow over time. Actions that require looping across the entire data structure should be avoided.

If you absolutely must loop over an array of unknown size, then you should plan for it to potentially take multiple blocks, and therefore require multiple transactions



```
[1:0]A floating pragma is set. [SWC-103]
```

Lock the pragma version and also consider known bugs (<https://github.com/ethereum/solidity/releases>) for the compiler version that is chosen. Pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package. Otherwise, the developer would need to manually update the pragma in order to compile locally.

```
[56:23]A floating pragma is set. [SWC-128]
```

Caution is advised when you expect to have large arrays that grow over time. Actions that require looping across the entire data structure should be avoided.

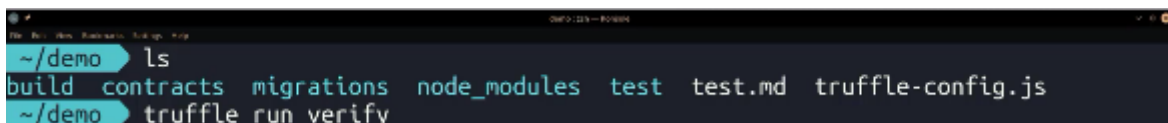
If you absolutely must loop over an array of unknown size, then you should plan for it to potentially take multiple blocks, and therefore require multiple transactions.

## MythX Security Analysis Plugin for Truffle Framework

This plugin is compatible with Truffle 5.0 or higher in this section I am going to demonstrate how to use the Mythx Security Analysis plugin for Truffle

to analyze these contracts to see what wrong run the following command `truffle run verify` inside your truffle project.

In our case we're going to analyze all of our contracts though there is a contract built into all truffle projects called migrations and that one will be skipped.

A terminal window screenshot showing the command `truffle run verify` being executed. The prompt is `~/demo`. The output shows a progress bar and the command `truffle run verify` being executed. The output also shows the command `truffle run verify` being executed. The output also shows the command `truffle run verify` being executed.

the results take a few seconds to come in the length of time is determined by a number of factors, including the type of analysis desired the progress bar shows the status of the analysis once completed here we see the issues detected by mythX for each contract we see violation and violation severity as well as the ID of the violation the output also shows the specific line number and location of where the violation occurred in the code the swc Id refers to the smart contract weakness registry a standard list of vulnerabilities.

```
~/demo ls
build contracts migrations node_modules test test.md truffle-config.js
~/demo truffle run verify
Map |*****| 100% || Elapsed: 3.0s ✓ completed
SimpleDAO |*****| 100% || Elapsed: 4.6s ✓ completed

/mapping_write.sol
  5:0  error  The binary addition can overflow  SWC-101
 11:25 error  The binary addition can overflow  SWC-101
 14:8  error  write to arbitrary storage key    SWC-124

/simple_dao.sol
 12:4  error  The binary addition can overflow  SWC-101
 18:6  error  persistent state read after call  SWC-107
 18:6  error  persistent state write after call SWC-107

> 6 problems (6 errors, 0 warnings)

~/demo
```

We originally ran the command from the terminal, but like most truffle commands you can also run the mythx plug-in from within the truffle develop console so let's do that while we're doing this let's add a few new options first the style flag, let's use configure how the output result should be formatted and now we are only running the analysis on the simple smart contract (Many option to add).

There are a number of option for running the mythx plugin we can bring up the full list of options by using the help flag, let's talk about some of the most commonly used options mode determines how in-depth you want your analysis to be the default is quick full which uses a more in-depth range of analyses can take on the order of minutes to return style as we saw previously allows the user to format the output results timeout will let you decide how long you want the command run on the terminal before giving up and returning control back to you note that processing may still continue on the Mythix server so you'll get a unique identifier that you can later use to retrieve results the debug flag provides additional debugging output there is also debug too for even more verbose output there is lots more to talk about with mythX and its integration with truffle but we suggest you giving it a try yourself mythX for Truffle is available as standard NPM module and you can use the mythx service on a trial basis for free right now if you'd like to learn more about the mythX platform go to [MythX .io](https://mythx.io)

## Conclusion

Here we've given you look how you can use MythX to check a smart contract weakness and how you can remediate it.