TRUFFLECON 2020

VERIFYING SMART CONTRACT SOURCE CODE ON ETHERSCAN

By Rosco Kalis

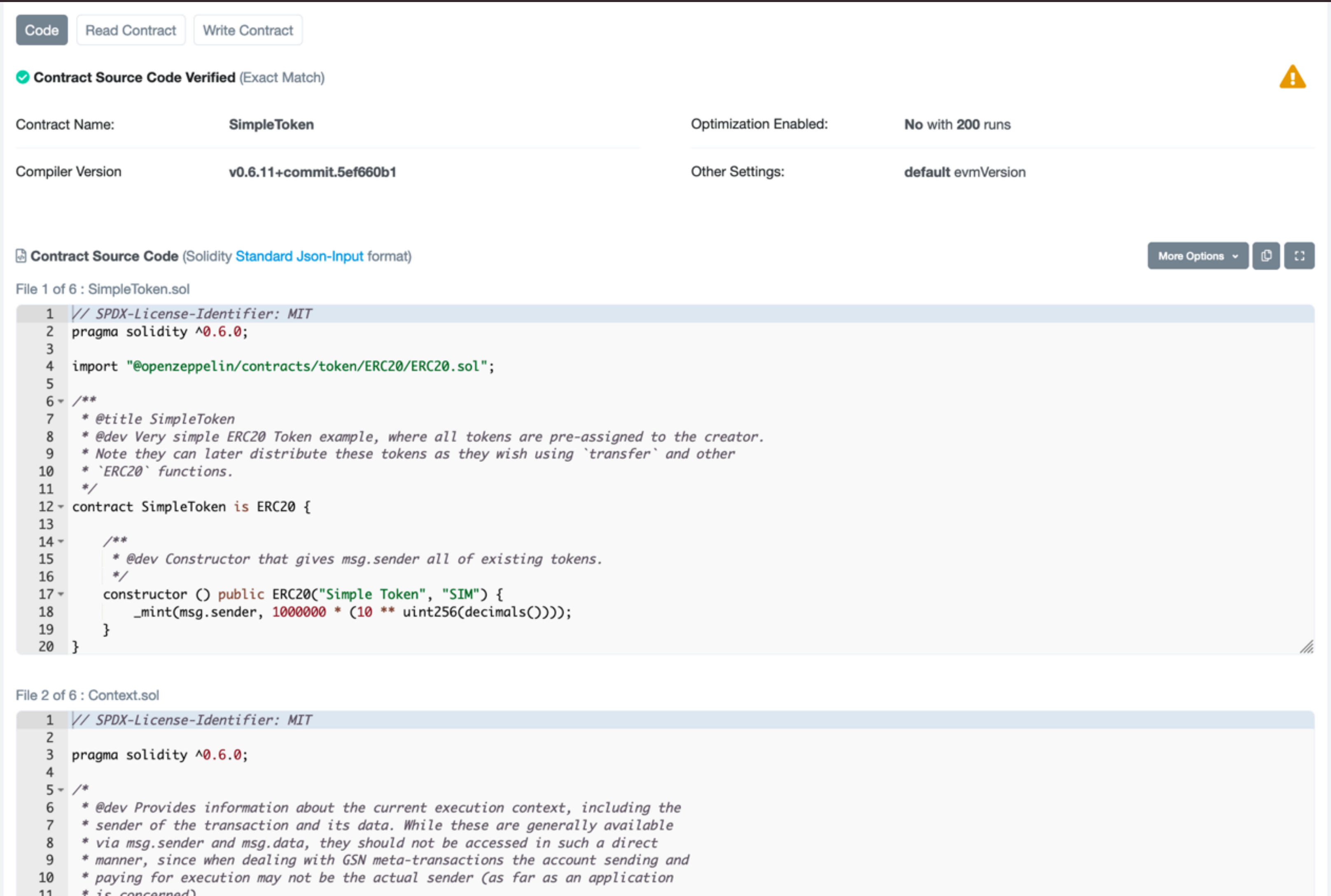
ABOUT ME

- Software Engineer @ General Protocols
- revoke.cash, CashScript
- truffle-assertions
- truffle-plugin-verify

CONTENTS

WHY VERIFYING IS IMPORTANT

WHY VERIFYING IS IMPORTANT



Code | Read Contract | Write Contract

Contract Source Code Verified (Exact Match)

Contract Name:	SimpleToken	Optimization Enabled:	No with 200 runs
Compiler Version	v0.6.11+commit.5ef660b1	Other Settings:	default evmVersion

Contract Source Code (Solidity Standard Json-Input format)

More Options

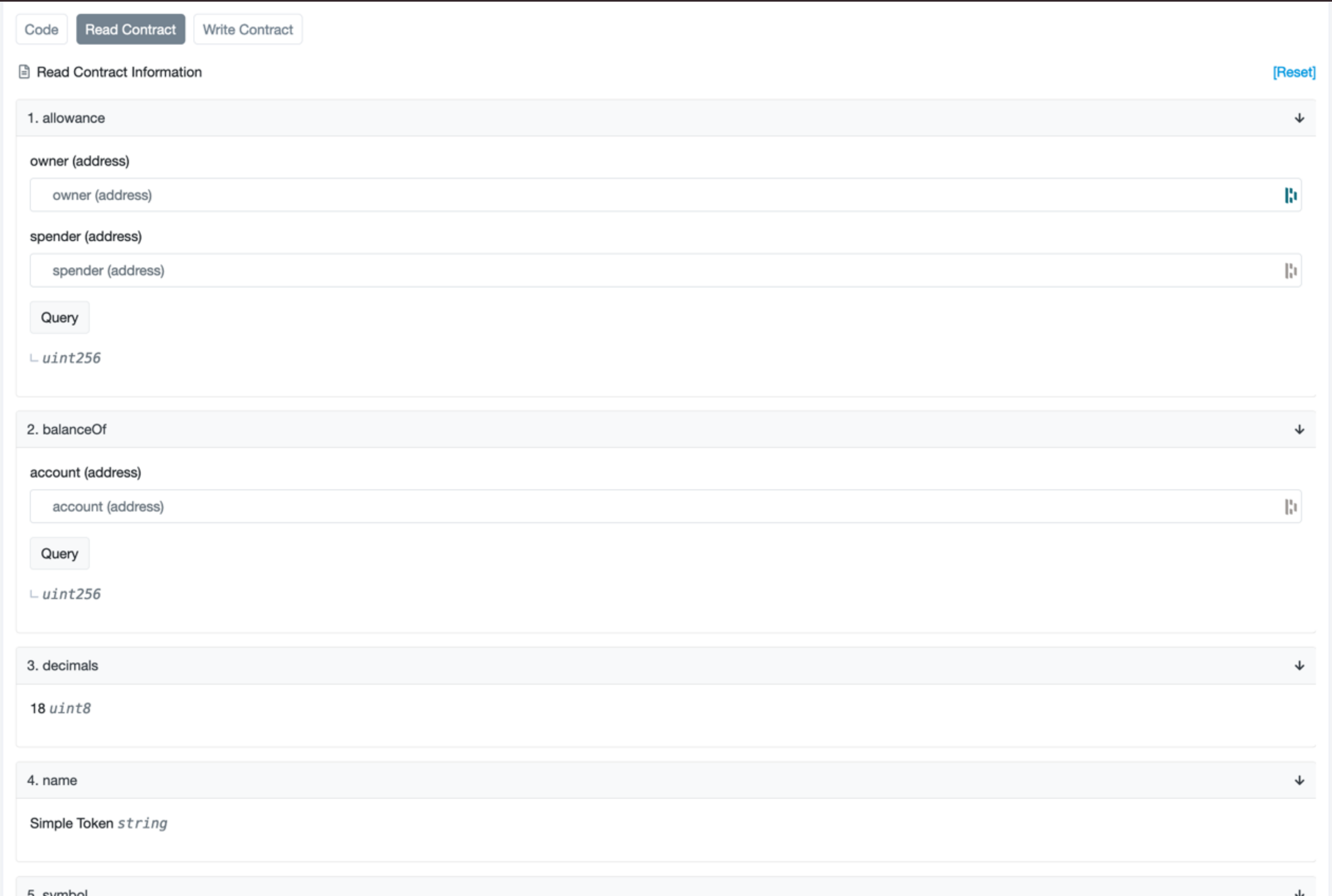
File 1 of 6 : SimpleToken.sol

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

/**
 * @title SimpleToken
 * @dev Very simple ERC20 Token example, where all tokens are pre-assigned to the creator.
 * Note they can later distribute these tokens as they wish using `transfer` and other
 * `ERC20` functions.
 */
contract SimpleToken is ERC20 {

    /**
     * @dev Constructor that gives msg.sender all of existing tokens.
     */
    constructor () public ERC20("Simple Token", "SIM") {
        _mint(msg.sender, 1000000 * (10 ** uint256(decimals())));
    }
}
```

File 2 of 6 : Context.sol

```solidity
// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
```

WHY VERIFYING IS IMPORTANT



Read Contract Information

1. allowance

owner (address)

spender (address)

Query

uint256

2. balanceOf

account (address)

Query

uint256

3. decimals

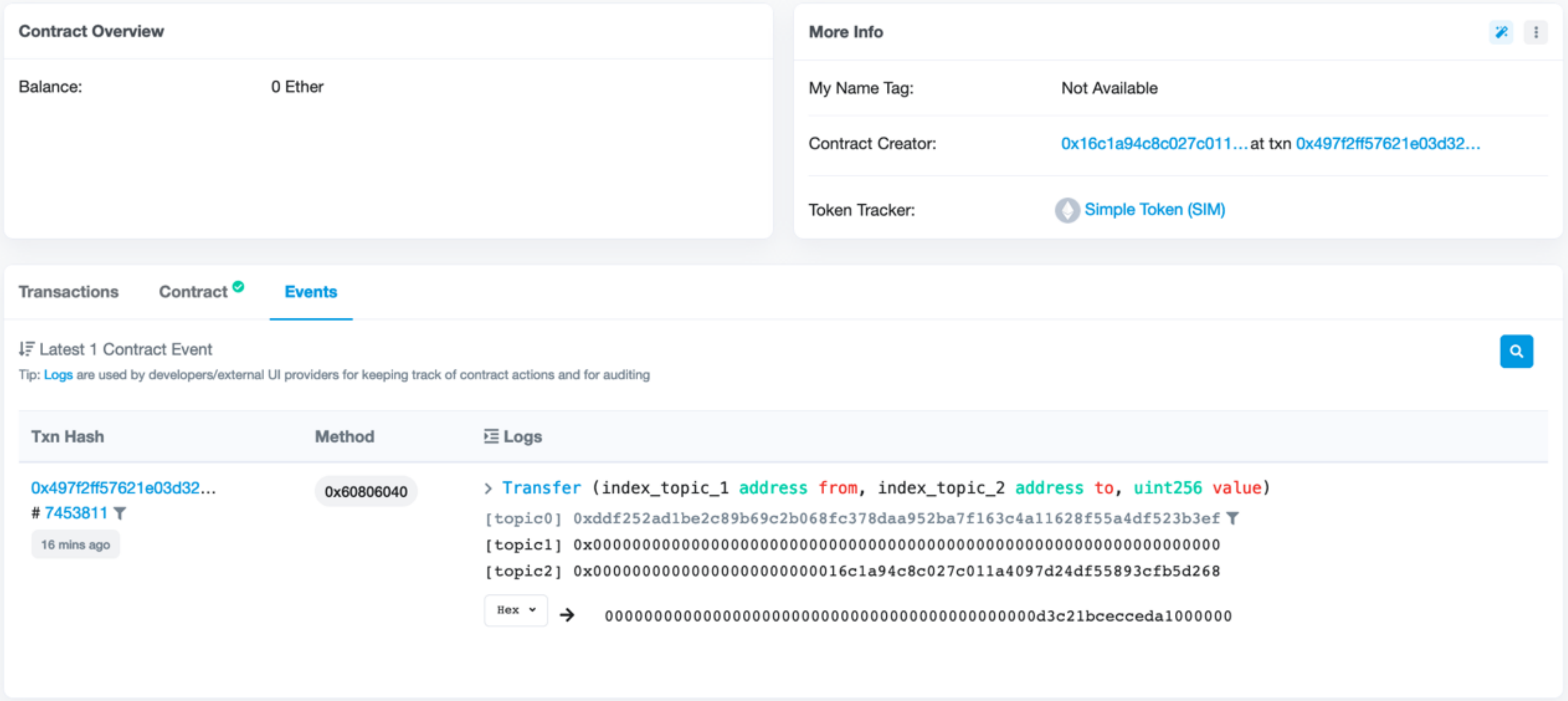
18 uint8

4. name

Simple Token string

5. symbol

WHY VERIFYING IS IMPORTANT



Contract Overview

Balance:	0 Ether

More Info

My Name Tag:	Not Available
Contract Creator:	0x16c1a94c8c027c011... at txn 0x497f2ff57621e03d32...
Token Tracker:	Simple Token (SIM)

Transactions · Contract · **Events**

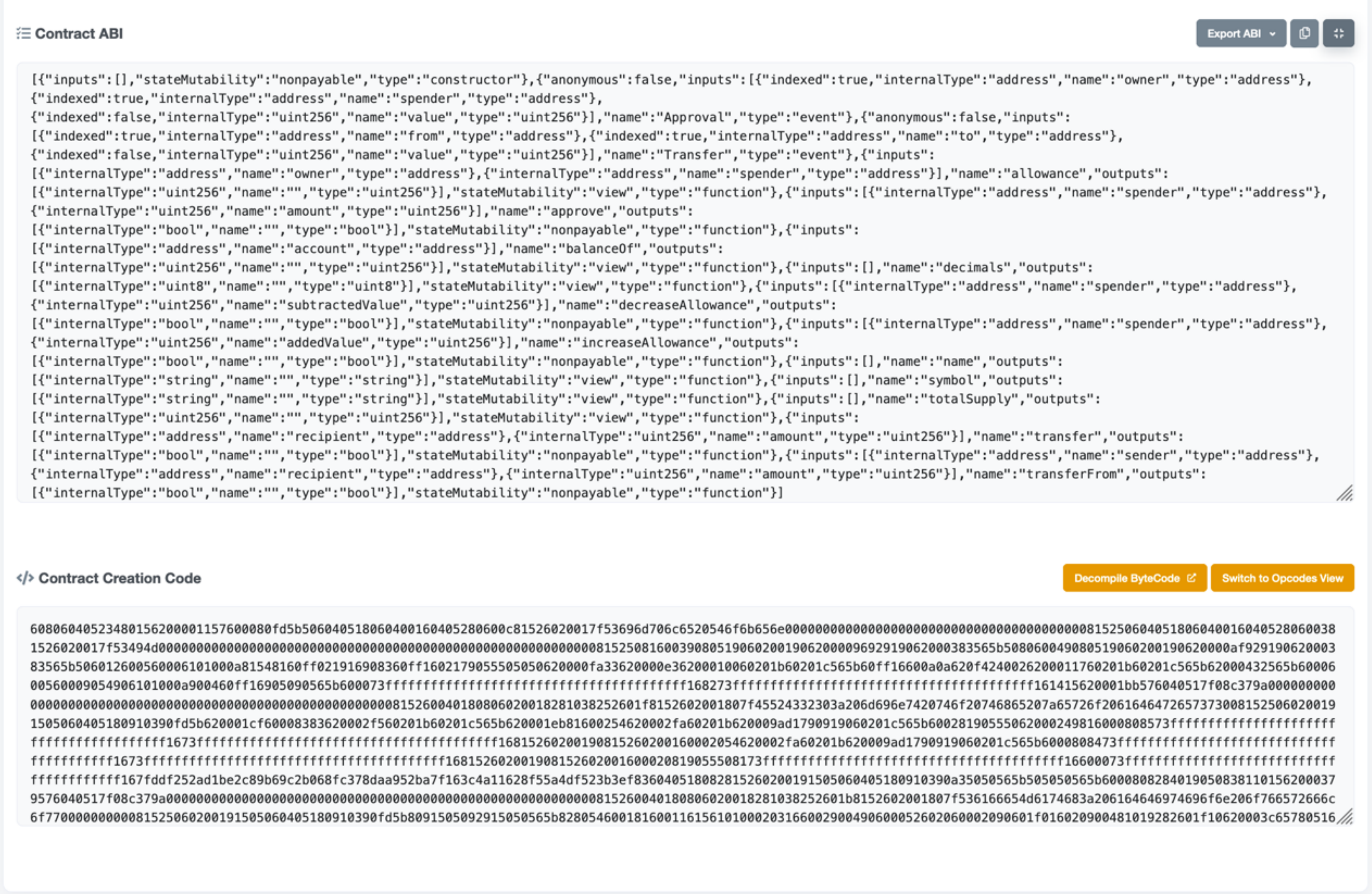
Latest 1 Contract Event

Tip: Logs are used by developers/external UI providers for keeping track of contract actions and for auditing

Txn Hash	Method	Logs
0x497f2ff57621e03d32... # 7453811 16 mins ago	0x60806040	see below

```
> Transfer (index_topic_1 address from, index_topic_2 address to, uint256 value)
[topic0] 0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef
[topic1] 0x0000000000000000000000000000000000000000000000000000000000000000
[topic2] 0x00000000000000000000000016c1a94c8c027c011a4097d24df55893cfb5d268
Hex → 00000000000000000000000000000000000000000000d3c21bcecceda1000000
```

WHY VERIFYING IS IMPORTANT



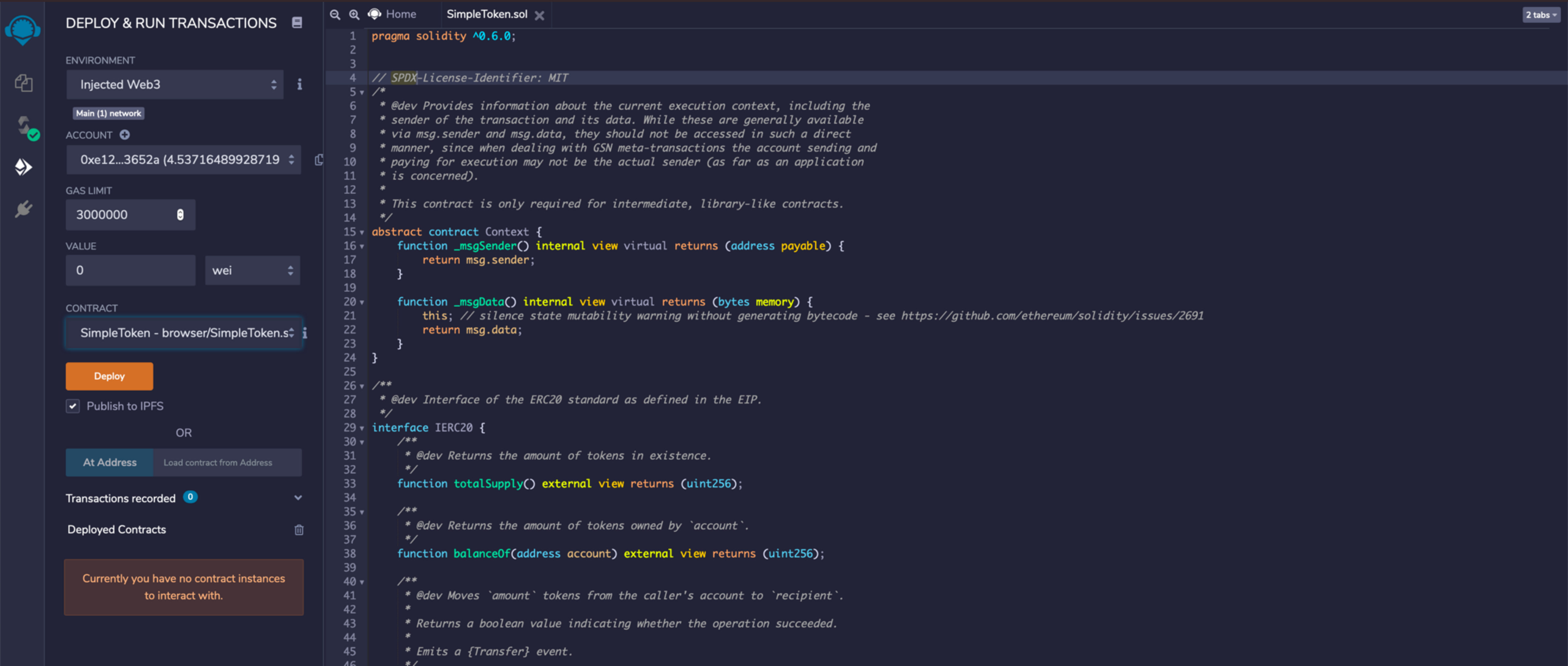
Contract ABI

```
[{"inputs":[],"stateMutability":"nonpayable","type":"constructor"},{"anonymous":false,"inputs":[{"indexed":true,"internalType":"address","name":"owner","type":"address"},
{"indexed":true,"internalType":"address","name":"spender","type":"address"},
{"indexed":false,"internalType":"uint256","name":"value","type":"uint256"}],"name":"Approval","type":"event"},{"anonymous":false,"inputs":
[{"indexed":true,"internalType":"address","name":"from","type":"address"},{"indexed":true,"internalType":"address","name":"to","type":"address"},
{"indexed":false,"internalType":"uint256","name":"value","type":"uint256"}],"name":"Transfer","type":"event"},{"inputs":
[{"internalType":"address","name":"owner","type":"address"},{"internalType":"address","name":"spender","type":"address"}],"name":"allowance","outputs":
[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"spender","type":"address"},
{"internalType":"uint256","name":"amount","type":"uint256"}],"name":"approve","outputs":
[{"internalType":"bool","name":"","type":"bool"}],"stateMutability":"nonpayable","type":"function"},{"inputs":
[{"internalType":"address","name":"account","type":"address"}],"name":"balanceOf","outputs":
[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[],"name":"decimals","outputs":
[{"internalType":"uint8","name":"","type":"uint8"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"spender","type":"address"},
{"internalType":"uint256","name":"subtractedValue","type":"uint256"}],"name":"decreaseAllowance","outputs":
[{"internalType":"bool","name":"","type":"bool"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"spender","type":"address"},
{"internalType":"uint256","name":"addedValue","type":"uint256"}],"name":"increaseAllowance","outputs":
[{"internalType":"bool","name":"","type":"bool"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[],"name":"name","outputs":
[{"internalType":"string","name":"","type":"string"}],"stateMutability":"view","type":"function"},{"inputs":[],"name":"symbol","outputs":
[{"internalType":"string","name":"","type":"string"}],"stateMutability":"view","type":"function"},{"inputs":[],"name":"totalSupply","outputs":
[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":
[{"internalType":"address","name":"recipient","type":"address"},{"internalType":"uint256","name":"amount","type":"uint256"}],"name":"transfer","outputs":
[{"internalType":"bool","name":"","type":"bool"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"sender","type":"address"},
{"internalType":"address","name":"recipient","type":"address"},{"internalType":"uint256","name":"amount","type":"uint256"}],"name":"transferFrom","outputs":
[{"internalType":"bool","name":"","type":"bool"}],"stateMutability":"nonpayable","type":"function"}]
```

Contract Creation Code

```
608060405234801562000011576000080fd5b506040518060400160405280600c81526020017f53696d706c6520546f6b656e00000000000000000000000000000000000000000000000000081525060405180604001604052806003 8
1526020017f53494d000000000000000000000000000000000000000000000000000000000081525081600390805190602001906200009692919062000383565b508060049080519060200190620000af929190620003
83565b5060126005600006101000a81548160ff021916908360ff1602179055505050620000fa33620000e36200010060201b60201c565b60ff16600a0a620f424002620001176 0201b60201c565b62000432565b60006
0056000905490610 1000a900460ff16905090565b600073ffffffffffffffffffffffffffffffffffffffff168273ffffffffffffffffffffffffffffffffffffffff161415620001bb576040517f08c379a000000000
000000000000000000000000000000000000000000000000000815260040180806020018281038252601f8152602001807f45524332303a206d696e7420746f20746865207a65726f20616464726573730081525060 20019
15050604051809103 90fd5b620001cf6000838362 0002f560201b60201c565b620001eb81600254620002fa60201b620009ad1790919060201c565b6002819055506200024981600080857 3ffffffffffffffffffff
ffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002054620002fa60201b620009ad1790919060201c565b6000808473ffffffffffffffffffffffffffffff
ffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002081905550817 3ffffffffffffffffffffffffffffffffffffffff16600073ffffffffffffffffffffffffffffff
ffffffffff167fddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef8360405180828152602001915050604051809103 90a35050565b505050565b600080828401905083811015620000 37
9576040517f08c379a0000000000000000000000000000000000000000000000000000000000815260040180806020018281038252601b8152602001807f536166654d6174683a206164646974696f6e206f766572666c
6f77000000000000815250602001915050604051809103 90fd5b8091505092915050565b828054600181600116156101000203166002900490600052602060002090601f016020900481019282601f10620003c65780516
```

THE TRADITIONAL METHOD FOR VERIFICATION



DEPLOY & RUN TRANSACTIONS

ENVIRONMENT
Injected Web3
Main (1) network

ACCOUNT
0xe12...3652a (4.53716489928719

GAS LIMIT
3000000

VALUE
0 wei

CONTRACT
SimpleToken - browser/SimpleToken.s

Deploy

✓ Publish to IPFS

OR

At Address — Load contract from Address

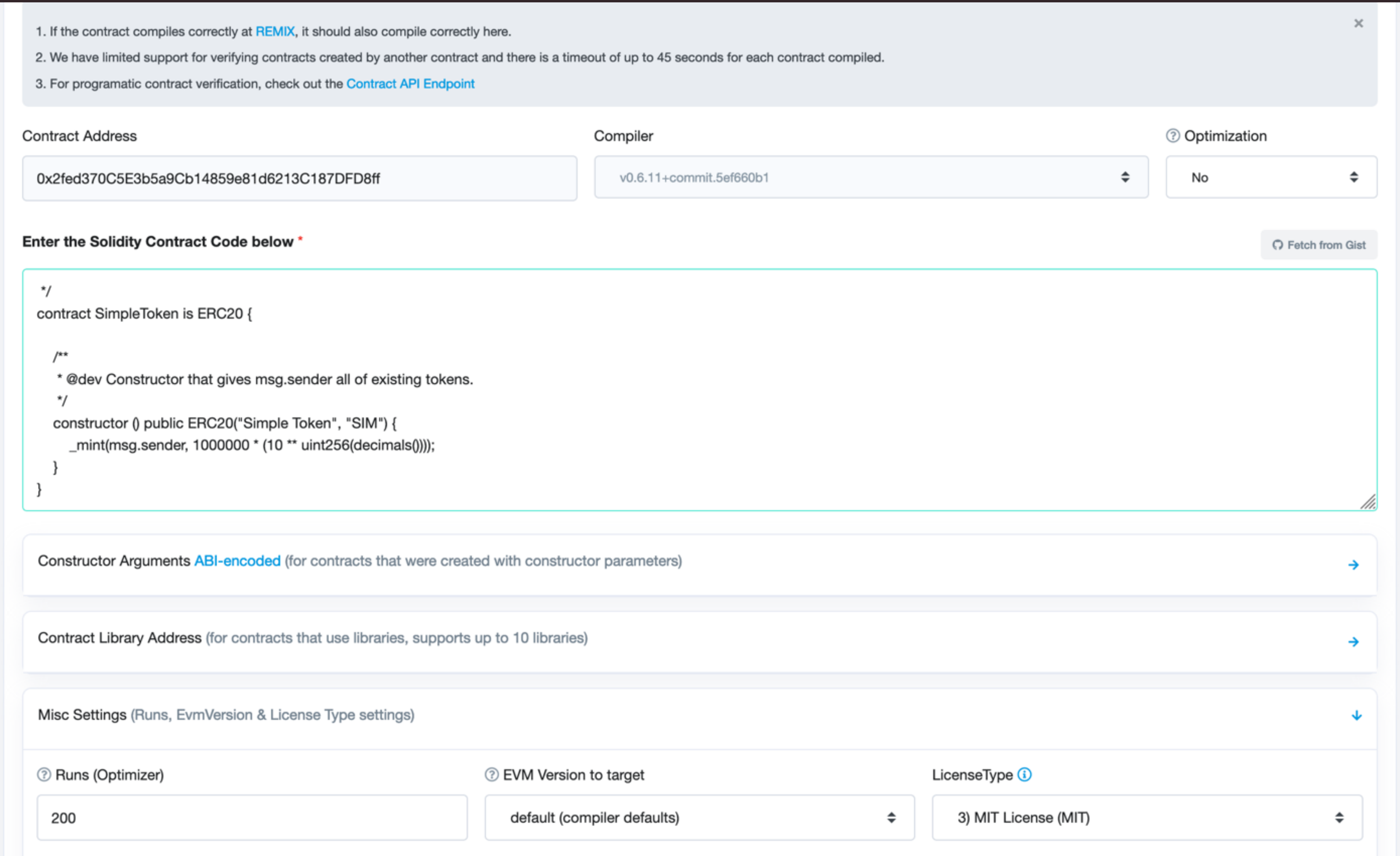
Transactions recorded 0

Deployed Contracts

Currently you have no contract instances to interact with.

```
pragma solidity ^0.6.0;


// SPDX-License-Identifier: MIT
/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
```

THE TRADITIONAL METHOD FOR VERIFICATION



1. If the contract compiles correctly at REMIX, it should also compile correctly here.
2. We have limited support for verifying contracts created by another contract and there is a timeout of up to 45 seconds for each contract compiled.
3. For programatic contract verification, check out the Contract API Endpoint

Contract Address: 0x2fed370C5E3b5a9Cb14859e81d6213C187DFD8ff

Compiler: v0.6.11+commit.5ef660b1

Optimization: No

Enter the Solidity Contract Code below *

Fetch from Gist

```
 */
contract SimpleToken is ERC20 {

    /**
     * @dev Constructor that gives msg.sender all of existing tokens.
     */
    constructor () public ERC20("Simple Token", "SIM") {
        _mint(msg.sender, 1000000 * (10 ** uint256(decimals())));
    }
}
```

Constructor Arguments ABI-encoded (for contracts that were created with constructor parameters)

Contract Library Address (for contracts that use libraries, supports up to 10 libraries)

Misc Settings (Runs, EvmVersion & License Type settings)

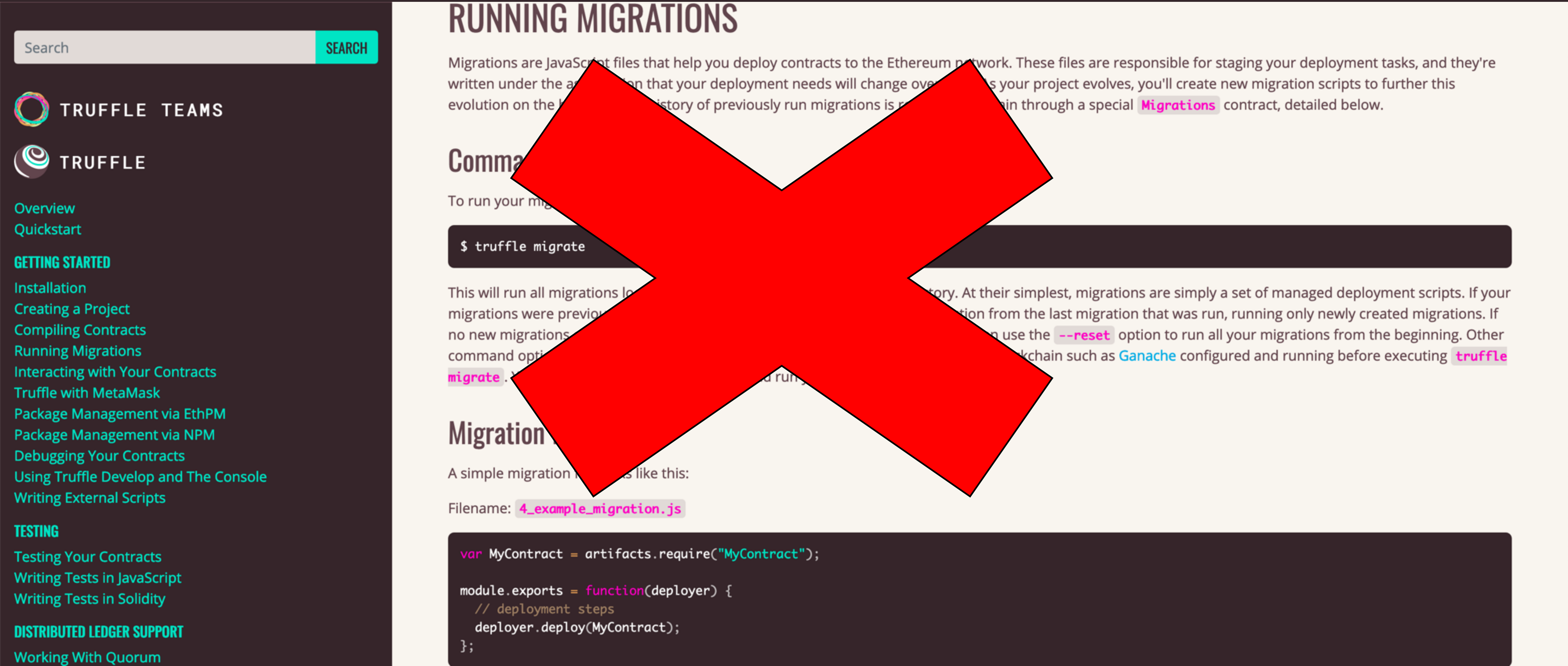
Runs (Optimizer): 200

EVM Version to target: default (compiler defaults)

LicenseType: 3) MIT License (MIT)

THE TRADITIONAL METHOD FOR VERIFICATION

THE TRADITIONAL METHOD FOR VERIFICATION



Search SEARCH

TRUFFLE TEAMS

TRUFFLE

Overview
Quickstart

GETTING STARTED

Installation
Creating a Project
Compiling Contracts
Running Migrations
Interacting with Your Contracts
Truffle with MetaMask
Package Management via EthPM
Package Management via NPM
Debugging Your Contracts
Using Truffle Develop and The Console
Writing External Scripts

TESTING

Testing Your Contracts
Writing Tests in JavaScript
Writing Tests in Solidity

DISTRIBUTED LEDGER SUPPORT

Working With Quorum

RUNNING MIGRATIONS

Migrations are JavaSc[illegible]t files that help you deploy contracts to the Ethereum n[illegible]twork. These files are responsible for staging your deployment tasks, and they're written under the a[illegible]on that your deployment needs will change ove[illegible]s your project evolves, you'll create new migration scripts to further this evolution on the [illegible]istory of previously run migrations is r[illegible]in through a special `Migrations` contract, detailed below.

Comma

To run your mi

```
$ truffle migrate
```

This will run all migrations lo[illegible]tory. At their simplest, migrations are simply a set of managed deployment scripts. If your migrations were previo[illegible]tion from the last migration that was run, running only newly created migrations. If no new migrations[illegible]n use the `--reset` option to run all your migrations from the beginning. Other command opti[illegible]kchain such as Ganache configured and running before executing `truffle migrate`. [illegible]d run[illegible]

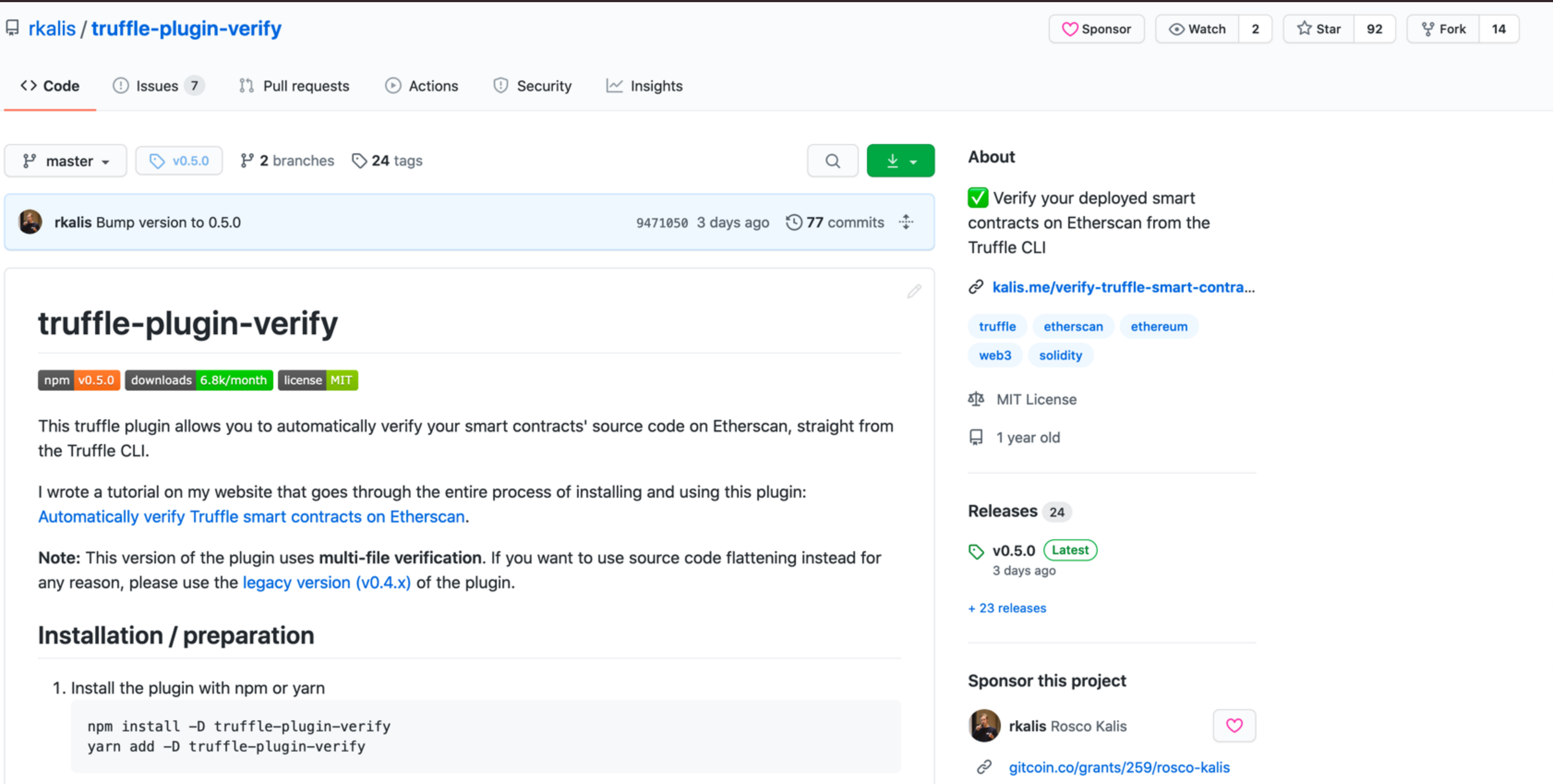
Migration

A simple migration [illegible]s like this:

Filename: `4_example_migration.js`

```
var MyContract = artifacts.require("MyContract");

module.exports = function(deployer) {
  // deployment steps
  deployer.deploy(MyContract);
};
```

TRUFFLE-PLUGIN-VERIFY



rkalis / truffle-plugin-verify

Sponsor | Watch 2 | Star 92 | Fork 14

Code | Issues 7 | Pull requests | Actions | Security | Insights

master | v0.5.0 | 2 branches | 24 tags

rkalis Bump version to 0.5.0 — 9471050 3 days ago — 77 commits

truffle-plugin-verify

npm v0.5.0 | downloads 6.8k/month | license MIT

This truffle plugin allows you to automatically verify your smart contracts' source code on Etherscan, straight from the Truffle CLI.

I wrote a tutorial on my website that goes through the entire process of installing and using this plugin: Automatically verify Truffle smart contracts on Etherscan.

Note: This version of the plugin uses **multi-file verification**. If you want to use source code flattening instead for any reason, please use the legacy version (v0.4.x) of the plugin.

Installation / preparation

1. Install the plugin with npm or yarn

```
npm install -D truffle-plugin-verify
yarn add -D truffle-plugin-verify
```

About

✅ Verify your deployed smart contracts on Etherscan from the Truffle CLI

kalis.me/verify-truffle-smart-contra...

truffle | etherscan | ethereum | web3 | solidity

MIT License

1 year old

Releases 24

v0.5.0 Latest
3 days ago

+ 23 releases

Sponsor this project

rkalis Rosco Kalis

gitcoin.co/grants/259/rosco-kalis

INSTALLATION & SETUP

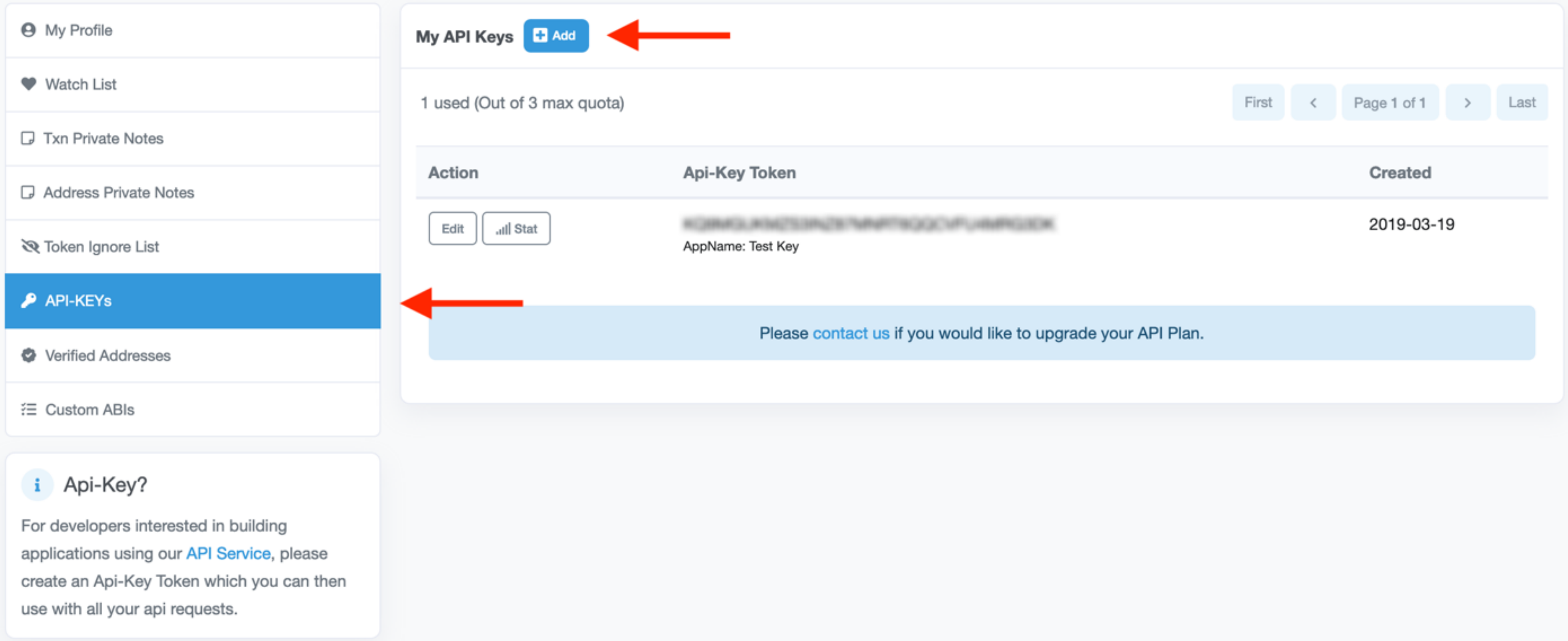
1. Install the plugin with npm or yarn

```
npm install -D truffle-plugin-verify
yarn add -D truffle-plugin-verify
```

2. Add the plugin to your `truffle-config.js` file

```
module.exports = {
  /* ... rest of truffle-config */

  plugins: [
    'truffle-plugin-verify'
  ]
}
```

INSTALLATION & SETUP



- My Profile
- Watch List
- Txn Private Notes
- Address Private Notes
- Token Ignore List
- API-KEYs
- Verified Addresses
- Custom ABIs

Api-Key?

For developers interested in building applications using our API Service, please create an Api-Key Token which you can then use with all your api requests.

My API Keys Add

1 used (Out of 3 max quota)

First ‹ Page 1 of 1 › Last

Action	Api-Key Token	Created
Edit Stat	[redacted] AppName: Test Key	2019-03-19

Please contact us if you would like to upgrade your API Plan.

INSTALLATION & SETUP

3. Add your Etherscan API key to your truffle config (make sure to use something like dotenv so you don't commit the api key)

```
module.exports = {
  /* ... rest of truffle-config */

  api_keys: {
    etherscan: 'MY_API_KEY'
  }
}
```

RUNNING VERIFICATION

1. Compile & deploy contracts

```
truffle compile
truffle migrate --network rinkeby
```

2. Verify deployed contract

```
truffle run verify SimpleToken --network rinkeby
```

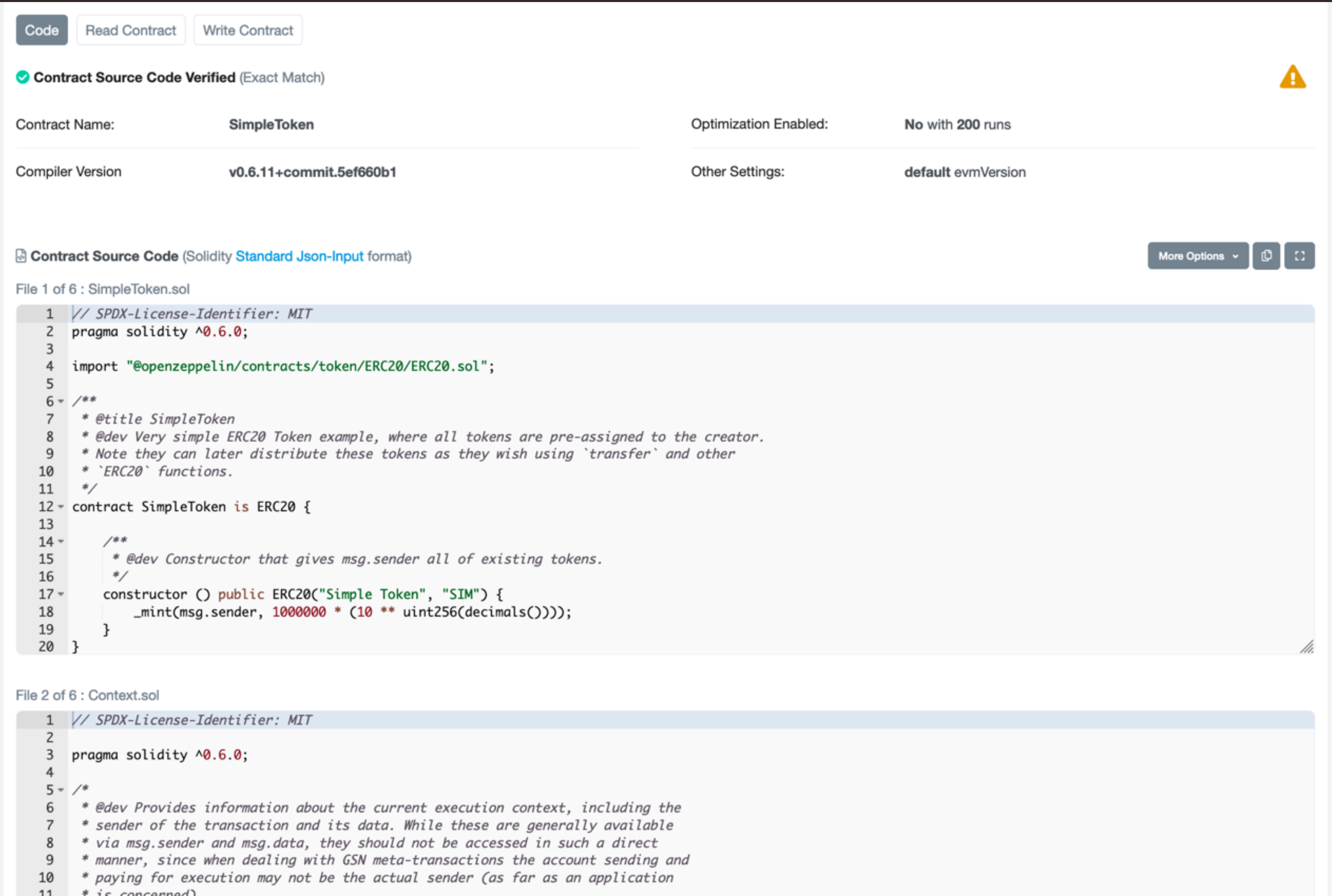
2. (Alternatively) Verify deployed contract with custom address

```
truffle run verify SimpleToken@0x2fed370C5E3b5a9Cb14859e81d6213C187DFD8ff --network rinkeby
```

3. Enjoy your verified contract

```
> Pass - Verified: https://rinkeby.etherscan.io/address/0x2fed370C5E3b5a9Cb14859e81d6213C187DFD8ff#contracts
```

RUNNING VERIFICATION



Contract Source Code Verified (Exact Match)

Contract Name:	SimpleToken	Optimization Enabled:	No with 200 runs
Compiler Version	v0.6.11+commit.5ef660b1	Other Settings:	default evmVersion

Contract Source Code (Solidity Standard Json-Input format)

File 1 of 6 : SimpleToken.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

/**
 * @title SimpleToken
 * @dev Very simple ERC20 Token example, where all tokens are pre-assigned to the creator.
 * Note they can later distribute these tokens as they wish using `transfer` and other
 * `ERC20` functions.
 */
contract SimpleToken is ERC20 {

    /**
     * @dev Constructor that gives msg.sender all of existing tokens.
     */
    constructor () public ERC20("Simple Token", "SIM") {
        _mint(msg.sender, 1000000 * (10 ** uint256(decimals())));
    }
}
```

File 2 of 6 : Context.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
```

TECHNICAL DETAILS

- Extract config information

```
// Truffle handles network stuff, just need to get network_id
const networkId = config.network_id
const apiUrl = API_URLS[networkId]
enforce(apiUrl, `Etherscan has no support for network ${config.network} with id ${networkId}`, logger)

const apiKey = config.api_keys && config.api_keys.etherscan
enforce(apiKey, 'No Etherscan API key specified', logger)

const workingDir = config.working_directory
const contractsBuildDir = config.contracts_build_directory

enforce(config._.length > 1, 'No contract name(s) specified', logger)
const contractNames = config._.slice(1)
```

TECHNICAL DETAILS

- Extract & format data from artifact
- Retrieve constructor data from Etherscan

```
const artifactPath = path.resolve(options.contractsBuildDir, `${contractName}.json`)
const artifact = require(artifactPath)
const inputJSON = await fetchInputJSON(artifact, options)
const constructorArgs = await fetchConstructorArgs(artifact, options)
```

TECHNICAL DETAILS

- Build & send Etherscan verification request

```
const postQueries = {
  apikey: options.apiKey,
  module: 'contract',
  action: 'verifysourcecode',
  contractaddress: artifact.networks[`${options.networkId}`].address,
  sourceCode: JSON.stringify(inputJSON),
  codeformat: 'solidity-standard-json-input',
  contractname: `${artifact.sourcePath}:${artifact.contractName}`,
  compilerversion: `v${artifact.compiler.version.replace('.Emscripten.clang', '')}`,
  constructorArguements: encodedConstructorArgs
}

const guid = await axios.post(options.apiUrl, querystring.stringify(postQueries))
```

TECHNICAL DETAILS

- Retrieve verification result

```
while (true) {
  await delay(1000)

  const qs = querystring.stringify({
    apiKey: options.apiKey,
    module: 'contract',
    action: 'checkverifystatus',
    guid
  })

  const verificationResult = await axios.get(`${options.apiUrl}?${qs}`)
  if (verificationResult.data.result !== VerificationStatus.PENDING) {
    return verificationResult.data.result
  }
}
```

HAPPY VERIFYING

```
$ truffle run verify SimpleToken --network rinkeby
Verifying SimpleToken
Pass - Verified: https://rinkeby.etherscan.io/address/0x7Eaf86d770FAd2d495E7923555a1553DEdC6B172#contracts
Successfully verified 1 contract(s).
```

FURTHER READING

- https://kalis.me/verify-truffle-smart-contracts-etherscan/
- https://github.com/rkalis/truffle-plugin-verify
- https://kalis.me/uploads/trufflecon2020.pdf