# Fuzzing

## Of the DSS-Gate

## Smart Contract

Produced for Maker

By Deco

# What is Fuzzing ?

Fuzzing is a type of property based testing technique to discover software vulnerabilities. A process of generating a massive number of random scenarios, to find our values that produce unexpected results. In other words, Fuzzing helps to find call sequences to falsify the predicates(aka assertions) made against an Ethereum smart contract. In addition, It also assists in reporting max gas usage based on plotted fuzz campaigns.

# Echidna as Fuzzer

'Echidna' is a property-based fuzzer, which is based on abstract state machine modeling technique and automatic test case generation for verifying the properties of a program by executing a large set of inputs generated stochastically. Echidna is a library for generating random sequences of calls against a given contract's ABI and thus preserves user-defined predicates/invariants.

Echidna can be used to detect, assess and analyze the following :

1. Incorrect Access Control
2. Incorrect State Machine
3. Incorrect Arithmetic
4. User-defined custom Assertions
5. Consumption of Gas
6. Coverage and Corpus collection
7. Performance evaluation

`echidna-test` is an executable that takes a contract and list of user defined invariants as input. For each invariant, it generates random sequences of calls to the contracts and checks if the user-defined predicate/invariant holds. If it can find some way to falsify the invariant, it prints the call sequence that does so. If it can't, you have some assurance the contract is safe.

### Versioning

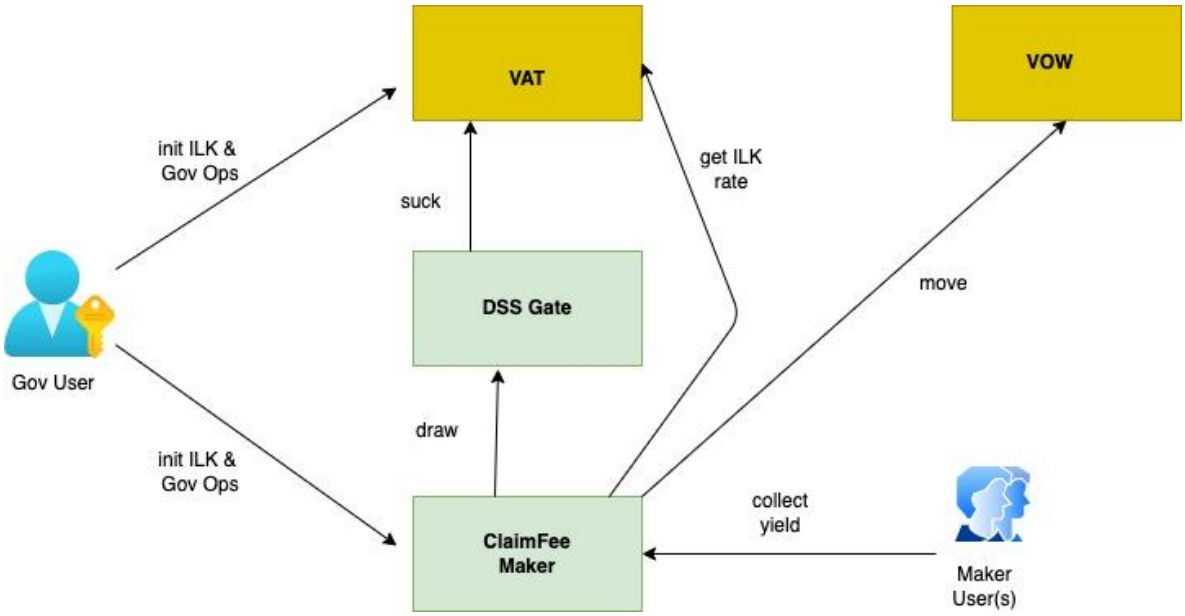
Echidna is compatible with various solidity compiler versions. Echidna has two versions namely Echidna 1.0 and 2.0 (being the latest stable release). Echidna 2.0 is compatible with solc 0.8+ to match with dss-gate.

# Repositories

This document entails to establish invariants for fuzzing on the below Smart Contract.

| Smart Contract | Repository | Commit Hash |
|---|---|---|
| Dss-Gate | https://github.com/deco-protocol/dss-gate | 43260bf |

# Overview



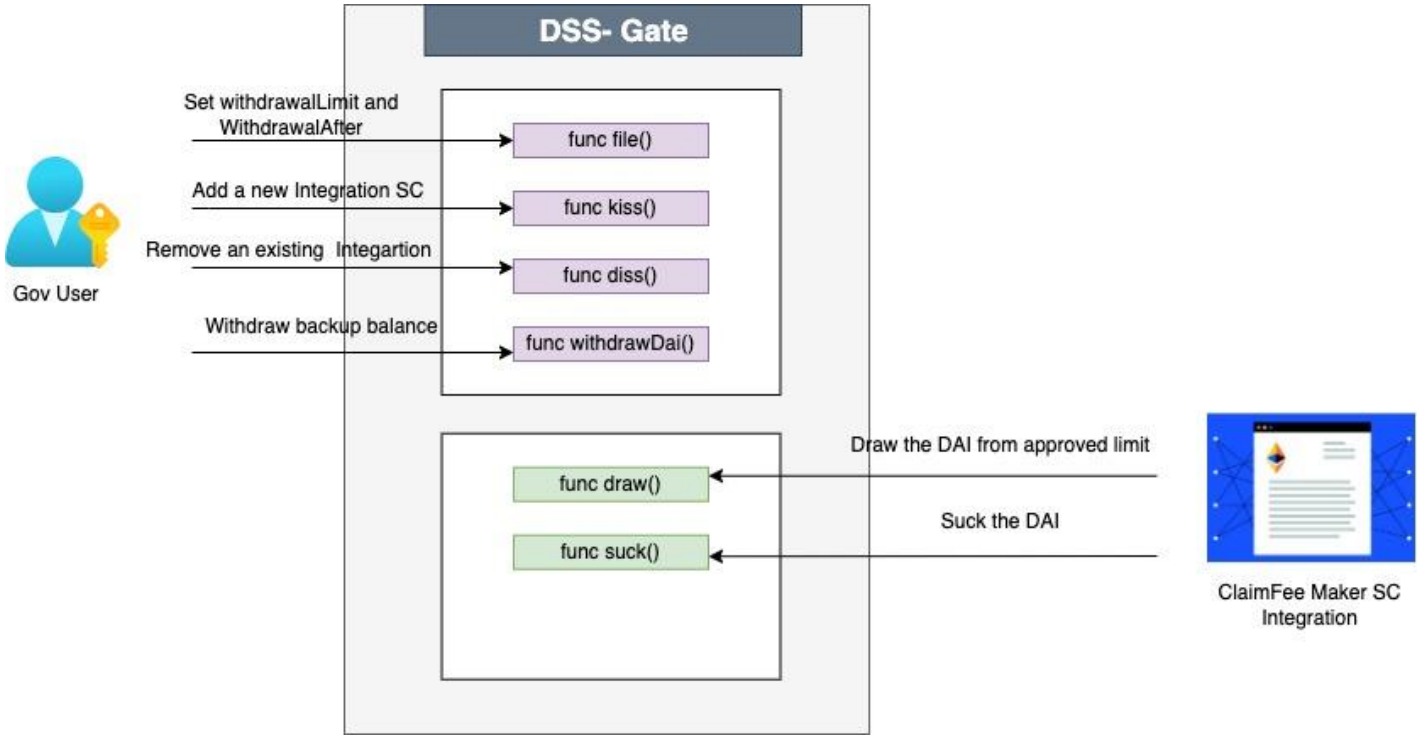

In the above diagram, the highlighted smart contracts (ClaimFeeMaker and DSS Gate) integrate with Core Maker contracts such as VAT and VOW.

**`DSS-Gate`** is mainly used as a safety lever to protect against the core VAT withdrawal capability. The VAT authorizes a withdrawal limit that it can `suck` DAI from. Upon vat.suck failures, This also holds backup DAI balance to fulfill requests. This approved DAI is further shared upon one or more gate integrations (termed as `buds`). As shown in the diagram, The ClaimFee Maker contract is one among those approved buds which is capable of drawing DAI, and thus transferring to its users.

**`ClaimFee Maker`** (aka CFM) is an integration of dss-gate module that offers 'Fixed-rate vaults as a Feature' on existing collateral types like ETH-A, WBTC-A etc over a fixed time window. A 'ClaimFee' balance is issued to a user with an existing Maker Vault. This 'Claim Fee' is in turn mapped to a specific ilk (Collateral Type), Issuance time and Maturity, which is coined as the term 'class'. This issued 'ClaimFee' is leveraged to offset the stability fee accrued on the Maker Vault. A user can redeem the 'ClaimFee' to offset the stability fees.

This document mainly focuses on providing a detailed overview of dss-gate smart contract and advanced testing of the same.

# DSS-Gate Contract Overview




[Figure 1]

The dss-gate (aka Gate1) possess an *approval limit* which indicates the maximum amount of dai that their integrations cumulatively, can withdraw from VAT. It also maintains a *backup balance* that allows integrations to withdraw upon vat.suck failures. Gate will ensure that only a single draw source is leveraged (either vat or backup balance but not both).

## Actors

- Ward - A privileged user or a smart contract with administrative authority. These users can set approval limits, withdrawal timestamp, add or remove integrations.

    - Ex : Governance.

- Bud - An integration smart contract which is authorized to draw and suck DAI from the gate. The gate inturn sucks/transfers from VAT.
    - Ex : claim-fee maker

## Public Functions

- file() - Used for setting approvedTotal and withdrawalAfter. The approvedTotal is the total DAI that gate integrations can withdraw from vat. The withdrawAfter is a timestamp set to which upon authorized governance addresses can withdraw backup balance.

- kiss() - Add a new integration to gate
- diss() - Remove an existing integration from gate
- rely() - Add a new admin
- deny() - Remove an existing admin
- withdrawDai() - Governance addresses to withdraw backup balance
- draw() - Suck the DAI to approved integration address
- suck() - Suck DAI (for backward compatibility with VAT)
- maxDrawAmount() - Maximum amount that is permissible to withdraw by gate integrations
- daiBalance() - Returns the dai balance held by gate contract.

## Private Functions

- accessSuck() - An internal helper function that invokes vat.suck()
- transferDai() - An internal helper function that transfers DAI to destination.

## Gate Invariants

A list of invariants (aka properties) in Gate contracts are depicted as below :

1. **Set Approval limit (aka approvedTotal)**
    a. Invariant : gate.approvedTotal will be set to input value.
    b. Access Invariant : A ward user is the only authorized entity to set approval limits (aka approvedTotal). This enables gate integration(s) to suck max Dai from VAT.

2. **Set WithdrawAfter**
    a. Invariant : gate.withdrawAfter will be set to input value.
    b. Access Invariant : A ward user is the only authorized entity to set withdrawafter, which enables governance addresses to withdraw dai from the backup balance.
    c. Conditional Invariant : The value for 'withdrawafter' key should always be greater than existing value.

3. **Add a new ward/administrator**
    a. Invariant : A new user is set as ward or administrator.
    b. Access Invariant : An existing ward can only add a new user as an admin.

4. **Remove an existing ward**
    a. Invariant : An existing ward is removed from administrators list.
    b. Access Invariant : An existing ward can only remove a user from admins.

5. **Add a new integration**
    a. Invariant : The address of integration must be added to buds.
    b. Access Invariant : Only an existing admin can add a new integration to gate.
    c. Conditional Invariant : A genesis address (0x0) cannot be added as integration address.
    d. Conditional Invariant : Cannot add any new integration after the withdrawAfter time is elapsed.

6. **Remove an integration**
    a. Invariant : An integration is removed from the buds. (set to 0)
    b. Access Invariant : Only an existing authorized ward/admin can remove integration.

7. **Suck** - **Suck DAI from VAT and send to destination**
    a. Invariant : only one source is used (either backup balance or approved total)
    b. Invariant : If backup balance is modified, then dai balance of gate is reduced in VAT
    c. Invariant : If approvedTotal is modified, then the total approved limit is reduced for the gate.
    d. Access Invariant : Can be invoked by only authorized buds
    e. Conditional Invariant : Amount to be withdrawn must be available in either of sources(backup or limit).
    f. Conditional Invariant : The source address cannot be a genesis address (i.e 0x0)

8. **Draw - Transfer DAI from Gate to destination**
    a. Invariant : An amount of dai will be transferred to integration.
    b. Invariant : If backup balance is not used, approvedTotal is decreased by rad amount.
    c. Invariant : If approvedTotal is not used, backupBalance is decreased by rad amount.
    d. Access Invariant : Only authorized integrations are allowed to draw.
    e. Conditional Invariant : The amount to be drawn must be available in either of sources.

9. **WithdrawDai - Admin withdraws backup balance**
    a. Invariant : The amount will be transferred from gate to destination in VAT.
    b. Access Invariant : withdraw function can be invoked only by an authorized ward.
    c. Condition Invariant : VAT must be live
    d. Conditional Invariant : The amount to be drawn must be available in either of sources.

10. **MaxDrawAmount - Maximum permitted withdraw capacity**
    a. Invariant : The max permitted amount in RAD from all the sources

### Invariants Implementation

| Invariant | Echidna Test Function/Code Ref |
|---|---|
| 1 a, b | test_file() |
| 2 a, b | test_file() |
| 3 a, b | test_rely() |
| 4 a, b | test_deny() |
| 5 a, b , c, d | test_kiss() |
| 6 a, b | test_diss() |
| 7 a, b, c, d, e, f | test_suck() |
| 8 a, b, c, d, e | test_draw() |
| 9 a, b, c, d | test_withdrawdai() |
| 10 a | test_maxDrawAmount() |

## Advanced Testing

### Testing Infrastructure

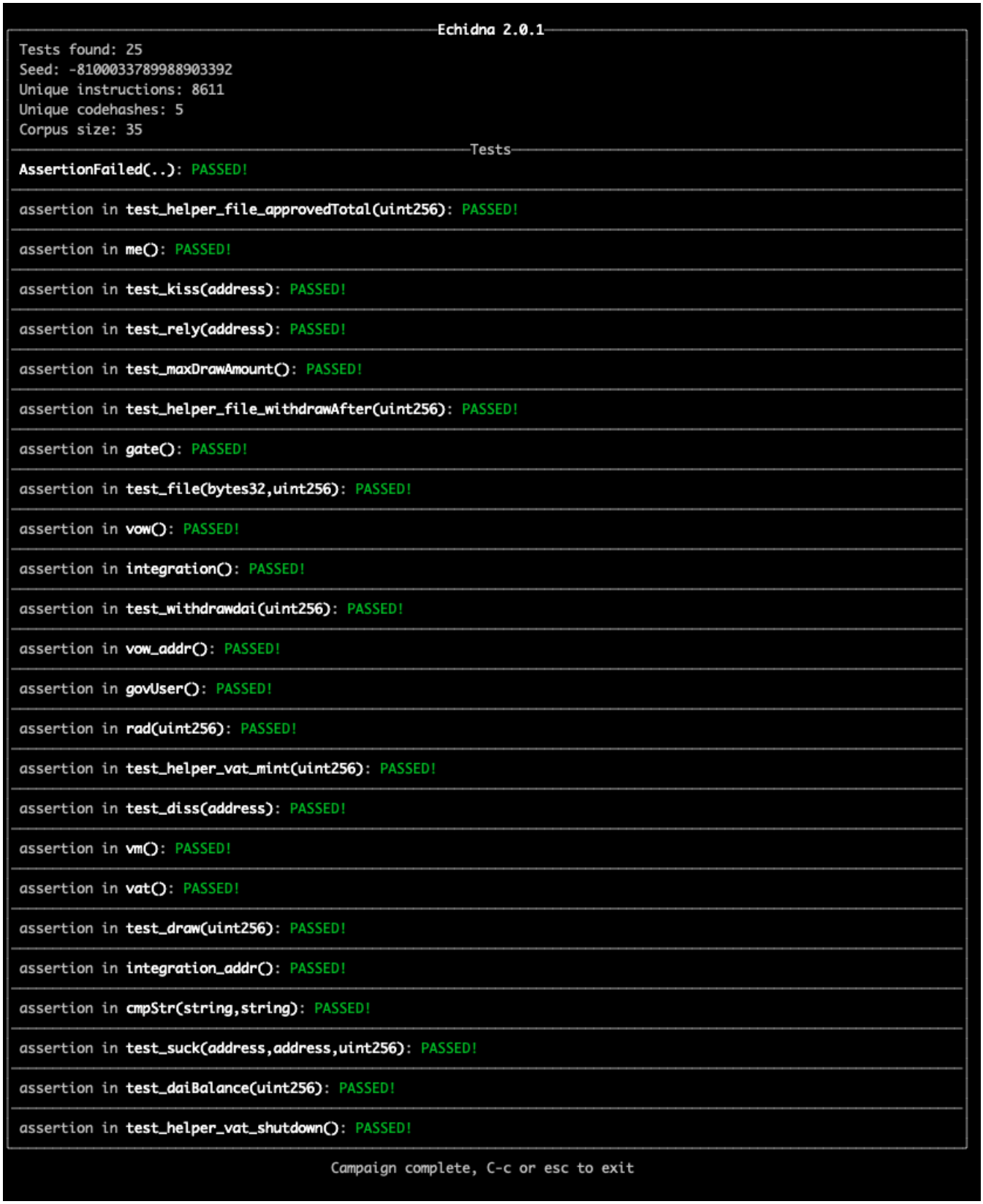
- ➢ Fuzzing Software : `Echidna 2.0`
- ➢ Cloud Compute : `AWS EC2 , Linux AMI (ami-0d9858aa3c6322f73)`
- ➢ Runtime : `Docker`
- ➢ Build : GNU Make
- ➢ Docker Image : `trailofbits/eth-security-toolbox`
- ➢ CPU : `32 Cores`

- Memory : `60 GB`
- Architecture : `x86_64`

### Echidna Test Configuration

- # Echidna Test Mode
  - **`testMode: "assertion"`**
- # Total number of test sequences to run
  - **`testLimit: 1000000 ( 1 million)`**
- Defines how many transactions are in a test sequence
  - **`seqLen: 200`**
- # solcArgs allows special args to solc
  - **`solcArgs: "--optimize --optimize-runs 200"`**
- # Maximum time between generated txs; default is one week
  - **`maxTimeDelay: 15778800 # approximately 6 months`**
- # Deployer is address of the contract deployer (who often is privileged owner, etc.)
  - **`deployer: "0x41414141"`**
- # Sender is set of addresses transactions may originate from
  - **`sender: ["0x42424242", "0x43434343"]`**

## Echidna Test Results



```
ethsec@cc4b1ff081ea:/home/training$ make echidna-dss-gate
./echidna/echidna.sh
Running ECHIDNA tests for dss-gate
Loaded total of 26 transactions from corpus/coverage
Analyzing contract: /home/training/echidna/DssGateEchidnaTest.sol:DssGateEchidnaTest
test_helper_vat_shutdown():  passed! 🎉
test_daiBalance(uint256):  passed! 🎉
test_suck(address,address,uint256):  passed! 🎉
cmpStr(string,string):  passed! 🎉
integration_addr():  passed! 🎉
test_draw(uint256):  passed! 🎉
vat():  passed! 🎉
vm():  passed! 🎉
test_diss(address):  passed! 🎉
test_helper_vat_mint(uint256):  passed! 🎉
rad(uint256):  passed! 🎉
govUser():  passed! 🎉
vow_addr():  passed! 🎉
test_withdrawdai(uint256):  passed! 🎉
integration():  passed! 🎉
vow():  passed! 🎉
test_file(bytes32,uint256):  passed! 🎉
gate():  passed! 🎉
test_helper_file_withdrawAfter(uint256):  passed! 🎉
test_maxDrawAmount():  passed! 🎉
test_rely(address):  passed! 🎉
test_kiss(address):  passed! 🎉
me():  passed! 🎉
test_helper_file_approvedTotal(uint256):  passed! 🎉
AssertionFailed(..):  passed! 🎉

Unique instructions: 8611
Unique codehashes: 5
Corpus size: 35
Seed: -8100033789988903392
```

### Test Setup Instructions

## AWS

**Use a Amazon Linux AMI**

Start an AWS ec2 instance from AWS console

**SSH in to aws console, Spin a ec2 instance/Set up a EC keypair**

Download the private key

```
$> chmod 400 ~/<privatekey>.pem
```

```
$> ssh -i "<privatekey>.pem"
ec2-user@ec2-54-219-121-253.us-west-1.compute.amazonaws.com
```

**Install Docker**

```
$> sudo yum install docker
```

**Install git**

```
$> sudo yum install git
```

**Start Docker**

```
$> sudo service docker start
```

**Docker pull Image - Trail of bits**

```
$> docker pull trailofbits/eth-security-toolbox
```

**Clone dss-gate repo**

```
$> git clone https://github.com/monseabrb/dss-gate.git
```

**Run trail of bits from dss-gate folder**

```
$> docker run -it -v "$PWD":/home/training
trailofbits/eth-security-toolbox
```

**Run the echidna-tests**

```
$> cd /home/training
$> make echidna-dss-gate
```

## Linux Machine

**Pre-requisites**

- Clone the dss-gate repository
  `$> git clone git@github.com:deco-protocol/dss-gate.git`
- Install Docker - Follow instructions based on your Operating System : https://docs.docker.com/get-docker/

**Setup and Run ECHIDNA**

Step 1 : Pull the Trail-Of-Bits ETH Sec ToolBox container image

- Pull the docker image
  `$> docker pull trailofbits/eth-security-toolbox`
- Change to dss-gate directory
  `$> cd .../dss-gate`

Step 2 : Run the Trail-Of-Bits container

```
$> docker run -it -v "$PWD":/home/training
trailofbits/eth-security-toolbox
```

Note : Make sure you run this command when your pwd is set to cloned repo folder(i.e dss-gate). This will enable you to access all the files and folders in the current working directory inside the container under `/home/training` folder.

Step 3 : Run ECHIDNA tests

The echidna tests of dss-gate can be run by below command :

```
$> make echidna-dss-gate
```

The corpus will be collected in 'dss-gate/corpus folder'. The corpus is collection of seed input, random inputs, coverage and execution flow of the target contract being tested.

# Glossary

1. *ClaimFee* - A special type of token balance that is issued for a Maker user with vault to offset stability fee.
2. VAT - Vault Engine is a database that tracks all user balances, accounting, fungibility etc
3. *VOW* - A settlement module which tracks the debt in system
4. *ilk* - A collateral type such as ETH, WBTC etc which is onboarded to Maker platform
5. *Class* - A keccak256 encoded hash of ilk, issuance and maturity.
6. *Governance User* - A user with MKR and has power of voting.
7. ClaimFee User/ Maker User - A user who has a vault in Maker.
8. *Ward* - A user with administrator privileges and has ability to invoke privileged operations.
9. *DAI* - A stablecoin cryptocurrency on ETH pegged against USD.

# References


[1] dss-gate https://github.com/deco-protocol/dss-gate/blob/master/docs/gate1.md

[2] claim-fee-maker https://github.com/deco-protocol/claim-fee-maker

[3] Diagrams https://app.diagrams.net/#G1jT1NokQ3RSkdxB4xfSe3pd0eA5MaPLT1

[4] Echidna : https://github.com/crytic/echidna

[5] blogs https://blog.trailofbits.com/2018/03/09/echidna-a-smart-fuzzer-for-ethereum/
https://blog.trailofbits.com/2018/05/03/state-machine-testing-with-echidna/
https://blog.trailofbits.com/2020/03/30/an-echidna-for-all-seasons/

[6] Echidna
https://github.com/crytic/building-secure-contracts/blob/master/program-analysis/echidna/fuzzing-introduction.md

[7] Maker units : https://github.com/makerdao/dss/blob/master/DEVELOPING.md#units

[8] Maker Docs. https://docs.makerdao.com/