

Matildapp: Multi Analysis Toolkit (by IdeasLocas) on DAPPs

Ideas Locas – Discovery Unit
CDO - Telefónica
ideaslocas@telefonica.com

Resumen ejecutivo

La web3 es un paradigma que ha irrumpido en el mundo digital con fuerza. Millones de transacciones se realizan en diferentes tipos de Blockchain. La importancia de los Smart Contracts en Blockchain como Ethereum cobran mayor relevancia debido a las finanzas que se manejan. Un error o un fallo de seguridad puede costar millones de dólares, por lo que la ciberseguridad es un factor vital. En este paper se introduce la herramienta Madildapp, la cual permite llevar a cabo pruebas de tipo DAST y SAST para evaluar la seguridad de los SmartContracts y otros elementos dentro de la cadena de valor de la Web3 (como pueden ser las propias DAPPs). Es una herramienta innovadora que aglutina diferentes tipos de pruebas orientados a diferentes tipos de elementos como son el bytecode, el código fuente puro y el contrato en su propia ejecución a través de pruebas dinámicas. Matildapp es un todo en uno con implementación modular que ayudará a la comunidad a mejorar la herramienta.

1. Contexto de la herramienta

En nuestro mundo moderno y conectado, el concepto de Web3, también conocido como la web descentralizada, representa el próximo cambio significativo en la tecnología de Internet. La Web3, respaldada por la tecnología de blockchain y contratos inteligentes, ofrece posibilidades sin precedentes de descentralización, transparencia y soberanía del usuario. Sin embargo, con estas nuevas posibilidades surgen nuevos desafíos - uno de los más cruciales es la seguridad. La naturaleza descentralizada de la Web3 elimina los puntos de fallo centrales típicos en las aplicaciones de Web2, lo que lleva a muchos a considerarla inherentemente más segura. Sin embargo, las dinámicas de seguridad en la Web3 son diferentes, y un conjunto único de vulnerabilidades ha surgido.

Este artículo presenta Madildapp, una herramienta novedosa diseñada para evaluar la seguridad de contratos inteligentes y otros componentes dentro de la cadena de valor de Web3, incluyendo aplicaciones descentralizadas (DAPPs). Madildapp permite la ejecución de pruebas de seguridad dinámicas de aplicaciones (DAST) y pruebas de seguridad estáticas de aplicaciones (SAST) para evaluar la seguridad de los contratos inteligentes. Esta herramienta innovadora agrega varias modalidades de prueba, adaptándose a diferentes elementos, incluyendo bytecode, código fuente puro y ejecución de contratos a través de pruebas dinámicas.

2. Web3 Vs Web2

La Web3, también conocida como la web descentralizada, es un paradigma que busca revolucionar la forma en que interactuamos con Internet y los servicios en línea. La Web2, que es la web que conocemos hoy en día, tiene varias limitaciones y problemas que la Web3 intenta resolver. Algunas de las razones por las que se necesita la Web3 son:

1. Centralización: La Web2 es controlada por un pequeño grupo de empresas y gobiernos que tienen acceso y control sobre nuestros datos y actividades en línea. La Web3 busca descentralizar la web, otorgando más control a los usuarios y reducir la dependencia de intermediarios.
2. Privacidad: La Web2 ha demostrado ser vulnerable a la vigilancia y el robo de datos personales. La Web3 utiliza criptografía avanzada y tecnologías de blockchain para proteger la privacidad de los usuarios.
3. Interoperabilidad: La Web2 está fragmentada en silos de datos y aplicaciones que no se comunican entre sí. La Web3 busca crear un ecosistema más interconectado y abierto, permitiendo que los usuarios se beneficien de la interoperabilidad entre aplicaciones y servicios.
4. Monetización: La Web2 se basa en modelos de negocio que explotan los datos de los usuarios y los convierten en productos. La Web3 busca crear modelos de negocio más justos y transparentes, otorgando a los usuarios más control sobre sus datos y recompensándolos por su participación.

3. Matildapp

'Matildapp' (Multi Analysis Toolkit -by IdeasLocas- on DAPPs) es un proyecto Open Source en el que se proporciona un framework para entornos de Web3 en el mundo de la ciberseguridad y pentesting. La herramienta proporciona módulos para interactuar con diferentes tipos de

Blockchain y poder evaluar en SAST y DAST diferentes potenciales vulnerabilidades que puedan tener los SmartContracts.



```
python sc.py

[~*~] matildapp: Multi Analysis Toolkit (by IdeasLocas) on DAPPs [~*~]
[~*~] Created by: IdeasLocas(With Love!) [~*~]

[+] Starting the console...
[*] Console ready!

sc $ > hello world!!
```

Figura 1: 'Matildapp'

La arquitectura de 'matildapp' es modular. Cuenta con más de 15 módulos que permiten detectar diferentes vulnerabilidades en contratos inteligentes, tanto aportando el código en lenguaje Solidity y, en determinadas ocasiones, aportando únicamente el *bytecode* del contrato. Puedes ver la arquitectura de la herramienta en el siguiente esquema:

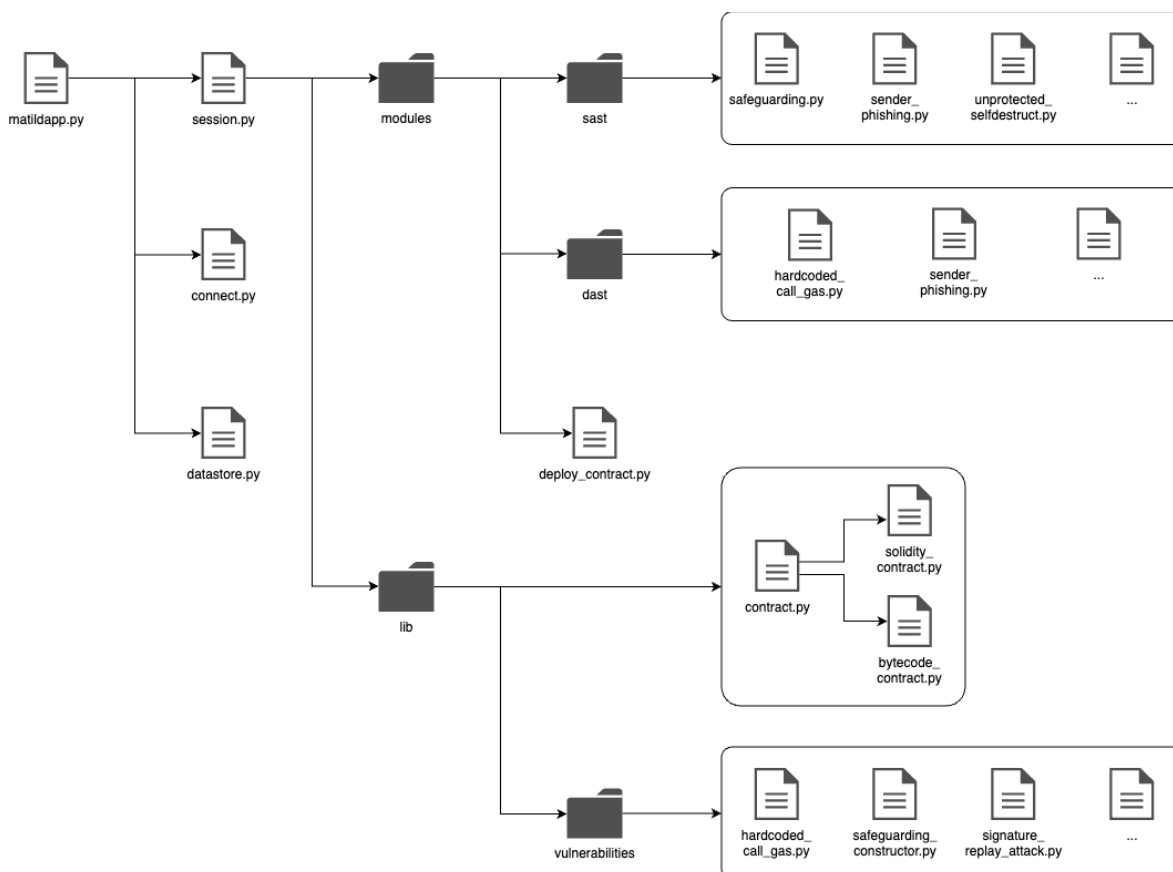


Figura 2: Esquema modular de 'matildapp'

En la ejecución de 'matildapp' se inicia una sesión interactiva desde la que podemos:

- Conectarnos a una blockchain para interactuar directamente con esta, por ejemplo, pudiendo compilar y desplegar contratos.

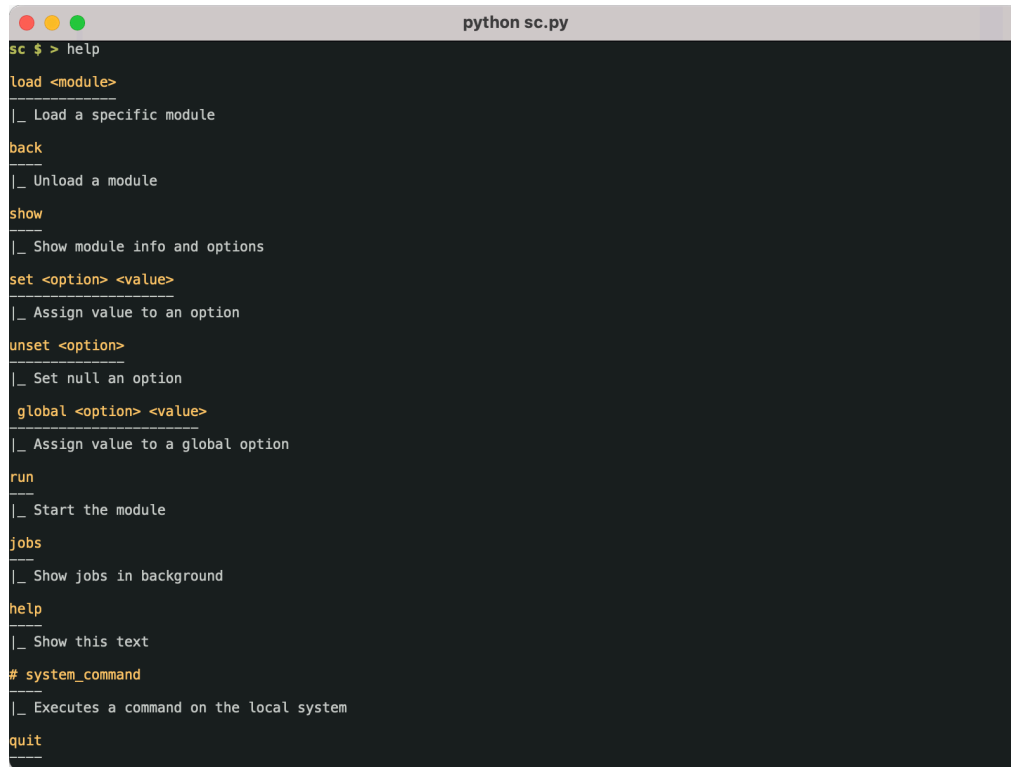
- Cargar diferentes módulos para el análisis SAST y DAST.
 - Los módulos de análisis SAST necesitan el código del smart contract aunque algunos módulos también pueden trabajar aportando únicamente el bytecode del contrato.
 - Los módulos de análisis DAST se conectan directamente a una dirección de contrato indicada y proceden al análisis.
 - También se dispone de un directorio con módulos de ejemplo para que cualquiera pueda crear sus propios módulos.
- Se encuentra un catálogo con diferentes vulnerabilidades basado en *Smart Contract Weakness Classification* (SWC) para facilitar su clasificación.
- Se dispone de un objeto para el almacenamiento de los datos, el *datastore*. Este va a permitir almacenar y gestionar distinta información, como por ejemplo la gestión de carteras, o la capacidad de ir añadiendo los resultados de los módulos de cada contrato con el que se trabaja.

```
python sc.py
sc $ >[bytecode]> load modules/
modules/dast/          modules/examples/          modules/utls
modules/deploy_contract modules/sast/
sc $ >[bytecode]> load modules/dast/
modules/dast/hardcoded_gas_call modules/dast/timestamp_dependence
modules/dast/sender_phishing modules/dast/unprotected_selfdestruct
sc $ >[bytecode]> load modules/sast/
modules/sast/array_length_manipulation/ modules/sast/signature_replay_attacks/
modules/sast/hardcoded_gas_call/ modules/sast/timestamp_dependence/
modules/sast/right_to_left_override_character/ modules/sast/unprotected_selfdestruct/
modules/sast/safeguarding_constructor/ modules/sast/untrusted_delegatecall/
modules/sast/sender_phishing/
sc $ >[bytecode]> load modules/sast/sender_phishing/
modules/sast/sender_phishing/bytecode modules/sast/sender_phishing/solidity
sc $ >[bytecode]> load modules/utls
modules/utls
sc $ >[bytecode]> load modules/utls
```

Los comandos que dispone la herramienta son los siguientes:

- El comando *help* guía sobre lo que hacen los comandos, como se muestra a continuación.
- El comando *load* permite cargar un módulo para su configuración y ejecución.
- El comando *back* hace la operación opuesta; devuelve el control al indicador principal.
- El comando *jobs* muestra los módulos que generan 'hilos' y están en ejecución.
- El comando *run* permite ejecutar un módulo. El comando *set* permite asignar valores a un atributo de un módulo.
- El comando *show* permite ver las opciones de un módulo antes de ejecutarlo.

- El comando *connect* permite conectarse con una cadena de bloques.
- El comando *disconnect* hace la operación opuesta; se desconecta de la cadena de bloques a la que estaba conectada.
- El comando *datastore* permite trabajar con este almacenamiento, pudiendo gestionar datos de carteras y contratos.



```

python sc.py
sc $ > help
load <module>
|_ Load a specific module
back
|_ Unload a module
show
|_ Show module info and options
set <option> <value>
|_ Assign value to an option
unset <option>
|_ Set null an option
global <option> <value>
|_ Assign value to a global option
run
|_ Start the module
jobs
|_ Show jobs in background
help
|_ Show this text
# system_command
|_ Executes a command on the local system
quit

```

Figura 5: Descripción de los comandos de 'matildapp'

4. Requisitos

'Matildapp' está escrito en Python y hace uso de librerías para trabajar con web3 como *web3.py* y *py-solc-x*. La gestión de versiones de estas librerías es importante, así como la versión de Python a utilizar. Se ha probado su funcionamiento en un entorno de Python con la versión 3.10.14 instalada y las versiones de *web3.py* 5.31.4 y *py-solc-x* 2.0.2.

Otras versiones de Python y de las librerías podrían causar problemas que impidieran la correcta ejecución de la herramienta.

Hay un archivo *requirements.txt* que debe ejecutarse la primera vez que se inicie la herramienta utilizando '*pip install -r requirements.txt*'. De nuevo, la versión de *pip* debe estar orientada a una versión de Python 3 testada como la 3.10.14.

Si se quiere trabajar en un entorno de pruebas de la cadena de bloques, será necesario que se cuente también con una utilidad que de este servicio. Para ello se puede hacer uso de herramientas como Ganache o Hardhat.

5. Cómo crear un módulo de 'matildapp'

La creación de un módulo es bastante sencilla y en la carpeta *examples* (se encuentra dentro del directorio de los módulos) se puede encontrar un ejemplo de ello. Este archivo implementa la clase 'CustomModule' que hereda de la clase 'Module' del archivo *module.py*.

Esta clase cuenta con un constructor que facilita la visualización de los datos informativos del módulo a través de la variable 'information'. La variable 'options' define los atributos necesarios para la clase que se implementará. Después, se utiliza 'super' para transferir los valores de 'information' y 'options' a la clase principal.

Al ejecutar un 'show' en la aplicación con este módulo cargado, se mostrarán las opciones especificadas en dicha variable, y podrán ser asignadas con el comando 'set'.

```
class CustomModule(Module):
    def __init__(self):
        information = {"Name": "My hello module",
                       "Description": "Test module",
                       "Author": "author"}

        # -----name-----default_value--description--required?
        options = {"message": ["hello world!", "Message for you", True],
                   "option2": [None, "Text description", False],
                   "option3": [None, "Text description", False]}

        # Constructor of the parent class
        super(CustomModule, self).__init__(information, options)

        # Class attributes, initialization in the run_module method
        # after the user has set the values
        self._option_name = None
```

Figura 6: Clase 'CustomModule' y su constructor

La clase 'CustomModule' también implementa un método llamado 'run_module'. Este método debe ser implementado en cada módulo para dar la funcionalidad. Podemos crear funciones o incluso métodos en nuestras clases, pero al ejecutar el comando 'run' se ejecutará este método 'run_module'.

En el siguiente ejemplo puede verse una acción tan sencilla como imprimir el contenido de la variable 'message' que se ha especificado anteriormente en las opciones del módulo.

```
# This module must be always implemented, it is called by the run option
def run_module(self):
    print(self.args["message"])
```

Figura 7: Método 'run_module'

Para finalizar, una vez creado nuestro módulo, se copiará a su categoría correspondiente. Por ejemplo, si se trata de un módulo de análisis estático, y es necesario pasar el código en lenguaje Solidity del contrato, se situará bajo 'sast/nombre_del_modulo/solidity.py'. Si por el contrario, se ha desarrollado con la capacidad de encontrar la vulnerabilidad dentro del bytecode, la ruta sería 'sast/nombre_del_modulo/bytecode.py'. De la misma forma, si se trata de un módulo de análisis dinámico se situaría bajo los módulos 'dast'.

Una vez que se encuentra dentro de su directorio correspondiente será cargado automáticamente por la herramienta en el momento de su ejecución.

6. Casos de uso

A continuación se muestran una serie de ejemplos realizados con la herramienta, mostrando así sus posibilidades. Todos los casos de uso se han probado bajo nuestro propio entorno de laboratorio.

a) Caso de uso: Detección de vulnerabilidad de posible phishing analizando el código fuente

En el siguiente ejemplo se va a mostrar el uso del módulo `'sender_phishing'`. Este módulo encuentra usos indebidos de la variable `'tx.origin'`.

La variable `'tx.origin'` es una variable global de Solidity que devuelve la dirección de la cartera que inició la transacción original. A diferencia de la variable `'msg.sender'`, que representa la dirección del remitente de la llamada actual (es decir, el contrato o cuenta que está invocando la función), `'tx.origin'` siempre muestra la dirección del usuario que comenzó la transacción, sin importar cuántos contratos intermedios se hayan llamado.

A continuación se dispone del código de un contrato inteligente que se quiere comprobar y se procede a cargar el módulo `'sast/sender_phishing/solidity.py'`.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.1;

import '@openzeppelin/contracts/access/Ownable.sol';
import '@openzeppelin/contracts/token/ERC20/ERC20.sol';

contract Token is Ownable, ERC20 {
    function Token() ERC20("HOLA","H") public {
        _mint(tx.origin, 10000 * (10**uint256(decimals())));
    }

    function mint(address _to, uint256 amount) public onlyOwner {
        _mint(_to, amount * (10**uint256(decimals())));
    }

    function burn(address _from, uint256 amount) public onlyOwner {
        _burn(_from, amount * (10**uint256(decimals())));
    }
}
```

Figura 8: Ejemplo de contrato inteligente haciendo uso incorrecto de `tx.origin`

En `'matildapp'` se encuentra una carpeta con varios ejemplos, y se puede encontrar el código usado para este ejemplo en `'examples/sol/phishing.sol'`. Aquí se puede ver dentro de la función `Token()` el uso de `'tx.origin'` a la hora de llamar a la función `_mint()`.

Una vez cargado el módulo, se procede a ver sus opciones de configuración haciendo uso del comando `'show'`. En las opciones puede verse que la única variable que hay que asignar es la ruta donde se encuentra el contrato inteligente ya que, al tratarse de un módulo SAST, se va a analizar el código fuente.

```

python sc.py
sc $ > load modules/sast/sender_phishing/
modules/sast/sender_phishing/bytecode modules/sast/sender_phishing/solidity
sc $ > load modules/sast/sender_phishing/solidity
[+] Loading module...
[+] Module loaded!
sc $ > [solidity]> show

Name
----
|_Sender Phishing Solidity check

Description
-----
|_
SWC: 115
CWE RELATED: CWE-477
Title: Authorization through tx.origin
Description: tx.origin is used for finding the wallet that triggered the original transaction.
             Can be used for Phishing if a wallet calls a malicious smart contract.
             Can be catastrophic if the wallet is a multisig wallet, as it will only store the
             reference to the person who triggered the last vote.

Detection cases: Can occur when tx.origin is used.
1. tx.origin appears in the code
2. tx.origin is in a comment
Final conclusion: 1 && !2

Author
-----
|_@chgara

Options (Field = Value)
-----
|_[REQUIRED] contract = None (Contract path, should be a solidity file)

sc $ > [solidity]> set contract examples/bytecode/sender_phishing.txt
contract >> examples/bytecode/sender_phishing.txt
sc $ > [solidity]>

```

Figura 9: Configuración del módulo 'sast/sender_phishing/solidity.py'

Una vez definida esta variable, se puede lanzar el módulo haciendo uso del comando 'run' y esperar a ver el resultado. En este ejemplo, el contrato es vulnerable.

```

python sc.py
sc $ > [solidity]> run
[+] Running module...
Vulnerability found (surely)
- SWC: 115
- CWE-related: CWE-477
- Title: Authorization through tx.origin
- Effect: Critical
- Description:
  > X Phishing alert
  > The contract uses tx.origin, which can be used for phishing attacks
  > More info: https://swcregistry.io/docs/SWC-115

Vulnerability found (surely)
- SWC: 115
- CWE-related: CWE-477
- Title: Authorization through tx.origin
- Effect: Critical
- Description:
  > X Phishing alert
  > The contract uses tx.origin, which can be used for phishing attacks
  > found in line 9 of contract Token
  > mint(tx.origin, 10000 * (10**uint256(decimals())));
  > More info: https://swcregistry.io/docs/SWC-115
sc $ > [solidity]>

```

Figura 10: Resultado del módulo 'sast/sender_phishing/solidity.py'

b) Caso de uso: Detección de vulnerabilidad selfdestruct analizando el bytecode

En el siguiente ejemplo se muestra el uso del módulo "unprotected_selfdestruct". Este módulo encuentra el uso de la función 'selfdestruct' que esté sin proteger.

La función 'selfdestruct' (anteriormente conocida como 'suicide') es una instrucción que permite eliminar un contrato inteligente de la blockchain y transferir el saldo restante a una dirección especificada. Al ejecutar 'selfdestruct', el contrato se elimina completamente del estado de la blockchain, y todas las interacciones futuras con el contrato serán imposibles.

En esta ocasión se dispone del bytecode del contrato inteligente, y el módulo a cargar es 'sast/unprotected_selfdestruct/bytecode.py'.

Figura 11: Fragmento de bytecode de un contrato inteligente

Similar al caso de uso anterior, en esta ocasión hay que asignar la variable 'bytecode' que permite introducir el bytecode directamente o bien la ruta a un archivo .txt que contenga el bytecode.


Figura 12: Configuración del módulo `sast/unprotected_selfdestruct/bytecode.py`

Figura 13: Resultado del módulo `‘ast/unprotected_selfdestruct/bytecode.py’`

c) Caso de uso: Despliegue de un contrato y uso del datastore

El ejemplo que se detalla a continuación mostrará el uso de un despliegue de un contrato y su almacenamiento en el objeto 'datastore' de 'matildapp'.

Para comenzar, es posible conectarse a una cadena de bloques conocida, como Ethereum, Polygon... o bien levantar nuestra propia cadena de bloques de desarrollo haciendo uso de herramientas como Ganache o Hardhat. Se hace uso del comando 'connect' para establecer la conexión.

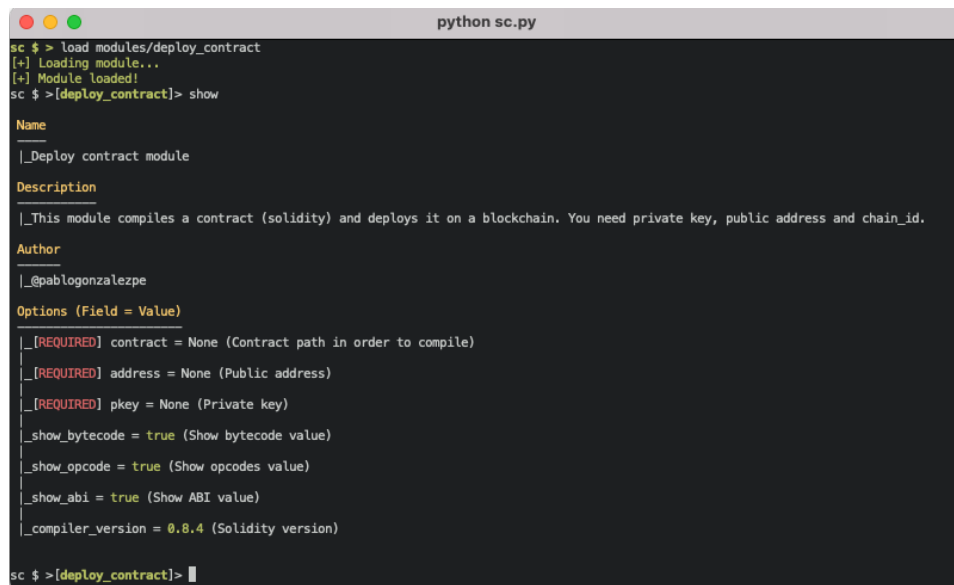


```
python sc.py
sc $ > connect
[!] Arguments incorrect: connect provider <rpc_server>
sc $ > connect provider http://127.0.0.1:7545
Connection created with Blockchain
sc $ >
```

Figura 14: Conexión a una cadena de bloques

Para que la conexión tenga éxito hay que indicar el proveedor. En este ejemplo el proveedor se encuentra en la misma máquina. Puede apreciarse el mensaje de la conexión establecida correctamente.

El siguiente paso es cargar el módulo para hacer el despliegue del contrato. Este se localiza bajo el módulo 'modules/deploy_contract'. Se pueden listar las opciones que hay disponibles para su configuración y se encuentran varias que son requeridas: la ruta donde se encuentra el contrato inteligente, la dirección pública y la clave privada de la cuenta que va a subir el contrato. Esta cuenta deberá tener saldo suficiente para poder pagar el gas que cuesta hacer el despliegue.



```
python sc.py
sc $ > load modules/deploy_contract
[+] Loading module...
[+] Module loaded!
sc $ > [deploy_contract]> show

Name
|_Deploy contract module

Description
|_This module compiles a contract (solidity) and deploys it on a blockchain. You need private key, public address and chain_id.

Author
|_@pablogonzalezpe

Options (Field = Value)
|_ [REQUIRED] contract = None (Contract path in order to compile)
|_ [REQUIRED] address = None (Public address)
|_ [REQUIRED] pkey = None (Private key)
|_ _show_bytecode = true (Show bytecode value)
|_ _show_opcode = true (Show opcodes value)
|_ _show_abi = true (Show ABI value)
|_ _compiler_version = 0.8.4 (Solidity version)

sc $ > [deploy_contract]>
```

Figura 14: Conexión a una cadena de bloques

Otra opción que hay que tener en cuenta es la versión del compilador, ya que podría dar problemas si se trabaja con una versión no compatible.

Por último, las opciones 'show_bytecode', 'show_opcode' y 'show_abi' permiten definir si se quiere mostrar esta información por pantalla a la hora de lanzar el módulo.

Una vez configurado todo lo necesario, se procede a la ejecución del módulo y vemos la salida.

[illegible]

Figura 15: Resultado de la salida del despliegue del contrato inteligente

Si todo ha ido bien el contrato se ha desplegado en la cadena de bloques y, además, puede notarse que cierta información se ha añadido a la estructura del *datastore*. Si se dispone de un explorador de bloques se puede comprobar el detalle de la transacción realizada.

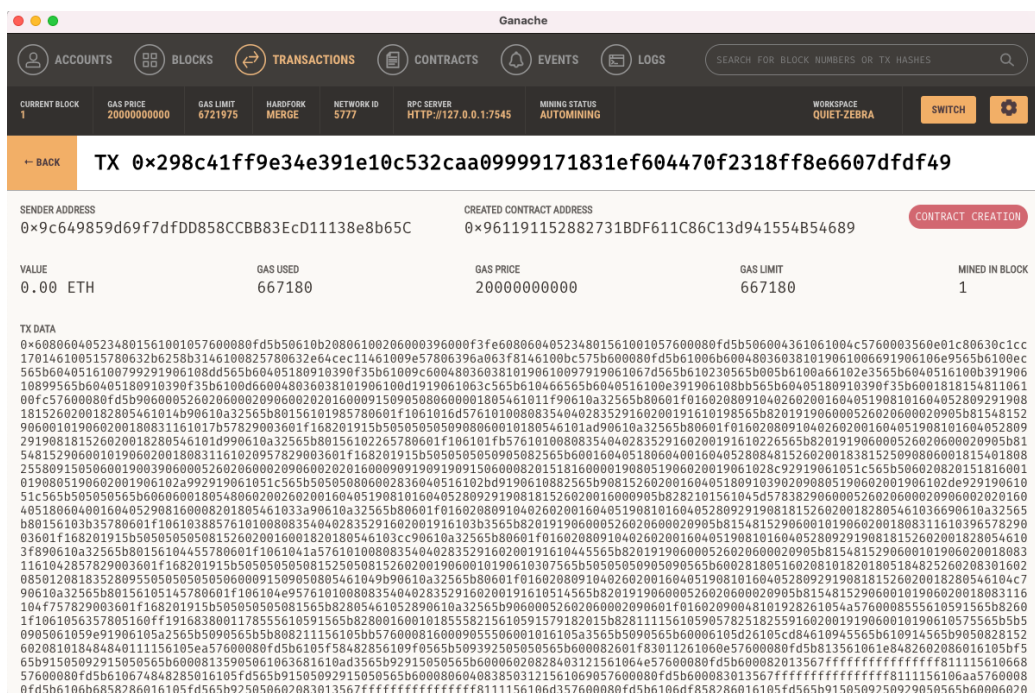
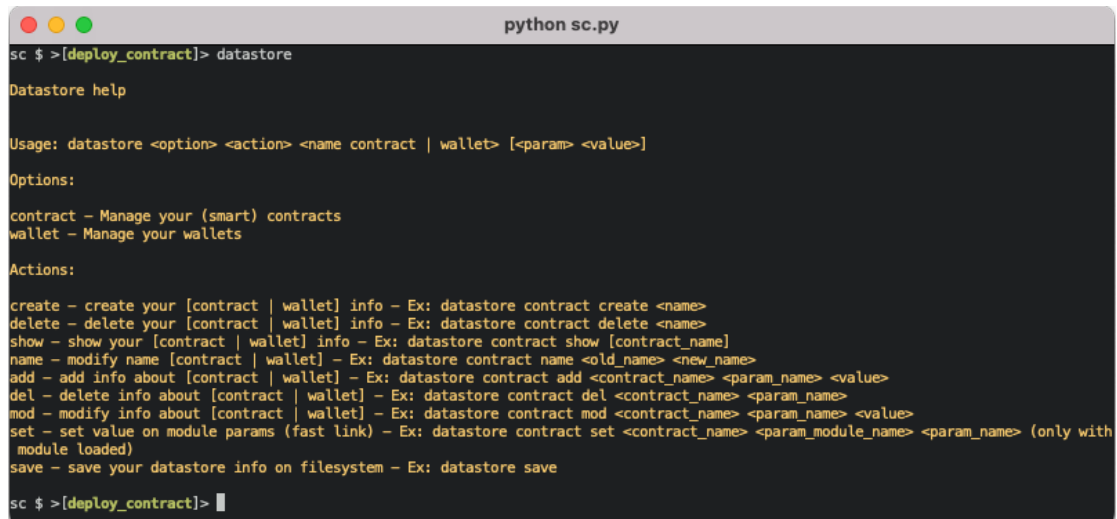


Figura 16: Detalle del despliegue en Ganache GUI

Por último, se puede trabajar con el objeto *datastore* para comprobar los datos que han sido almacenados y utilizar estos datos como parámetros de entrada de los propios módulos de *'matildapp'*.

Para comenzar a trabajar con *datastore*, utilizando el comando sin ningún parámetros se mostrará la ayuda con las diferentes opciones y acciones disponibles para su uso. Entre las opciones se distingue si se va a trabajar con una cartera o con un contrato. Las acciones son las siguientes:

- *create*: Crea un nuevo registro de contrato o cartera
- *delete*: Elimina un registro de un contrato o cartera
- *show*: Muestra la información de un contrato o cartera
- *name*: Asigna un nuevo nombre a un contrato o cartera
- *add*: Añade una nueva información a un contrato o cartera ya existente
- *del*: Elimina información de un contrato o cartera ya existente
- *mod*: Modifica la información de un contrato o cartera ya existente
- *set*: Asigna el valor al parámetro de un módulo
- *save*: Guarda una copia del *datastore* en el disco



```
python sc.py
sc $ >[deploy_contract]> datastore

Datastore help

Usage: datastore <option> <action> <name contract | wallet> [<param> <value>]

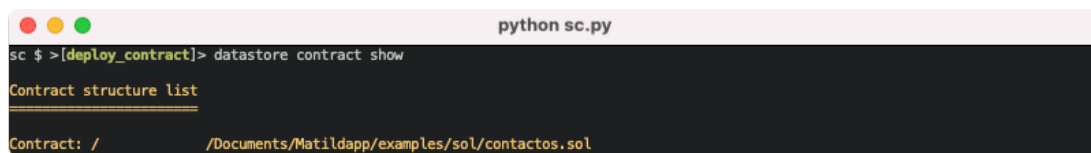
Options:
contract - Manage your (smart) contracts
wallet - Manage your wallets

Actions:
create - create your [contract | wallet] info - Ex: datastore contract create <name>
delete - delete your [contract | wallet] info - Ex: datastore contract delete <name>
show - show your [contract | wallet] info - Ex: datastore contract show [contract_name]
name - modify name [contract | wallet] - Ex: datastore contract name <old_name> <new_name>
add - add info about [contract | wallet] - Ex: datastore contract add <contract_name> <param_name> <value>
del - delete info about [contract | wallet] - Ex: datastore contract del <contract_name> <param_name>
mod - modify info about [contract | wallet] - Ex: datastore contract mod <contract_name> <param_name> <value>
set - set value on module params (fast link) - Ex: datastore contract set <contract_name> <param_module_name> <param_name> (only with module loaded)
save - save your datastore info on filesystem - Ex: datastore save

sc $ >[deploy_contract]> 
```

Figura 17: Uso del comando *datastore*

Si se quiere mostrar una lista con la información que se tiene almacenada sobre los contratos inteligentes se hará uso del comando: '*datastore contract show*'.



```
python sc.py
sc $ >[deploy_contract]> datastore contract show

Contract structure list

Contract: / /Documents/Matildapp/examples/sol/contactos.sol
```

Figura 18: Listado de los contratos almacenados en el *datastore*

Una vez que se conoce como se llaman los contratos disponibles se puede acceder a la información concreta de cada uno de ellos, por ejemplo: '*datastore contract show mi_contrato.sol*'.

```
python sc.py
sc $ >[deploy_contract]> datastore contract show /Documents/Matildapp/examples/sol/contactos.sol
{
  'address': '0x961191152882731B0F611C86C13d941554B54689',
  'abi': [
    {
      'inputs': [
        {
          'internalType': 'string',
          'name': 'name',
          'type': 'string'
        }
      ],
      'name': 'addContact',
      'outputs': [],
      'stateMutability': 'nonpayable',
      'type': 'function'
    },
    {
      'inputs': [
        {
          'internalType': 'uint256',
          'name': '',
          'type': 'uint256'
        }
      ],
      'name': 'contact',
      'outputs': [
        {
          'internalType': 'string',
          'name': 'name',
          'type': 'string'
        },
        {
          'internalType': 'string',
          'name': 'phoneNumber',
          'type': 'string'
        }
      ],
      'stateMutability': 'view',
      'type': 'function'
    },
    {
      'inputs': [
        {
          'internalType': 'string',
          'name': '',
          'type': 'string'
        }
      ],
      'name': 'nameToPhoneNumber',
      'outputs': [
        {
          'internalType': 'string',
          'name': '',
          'type': 'string'
        }
      ],
      'stateMutability': 'view',
      'type': 'function'
    },
    {
      'inputs': [],
      'name': 'retrieve',
      'outputs': [
        {
          'components': [
            {
              'internalType': 'string',
              'name': 'name',
              'type': 'string'
            },
            {
              'internalType': 'string',
              'name': 'phoneNumber',
              'type': 'string'
            }
          ],
          'internalType': 'struct ContactList.Contact[]',
          'name': '',
          'type': 'tuple[]'
        }
      ],
      'stateMutability': 'view',
      'type': 'function'
    }
  ],
  'opcode': 'PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH2 0x10 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH1 0x4 CALLDATASIZE LT PUSH2 0x4C JUMPI PUSH1 0x0 CALLDATALOAD PUSH1 0xE0 SHR DUP1 PUSH4 0xC1CC170 EQ PUSH2 0x51 JUMPI DUP1 PUSH4 0x2B6258B3 EQ PUSH2 0x82 JUMPI DUP1 PUSH4 0x2E64CEC1 EQ PUSH2 0x9E JUMPI DUP1 PUSH4 0x96A063F8 EQ PUSH2 0xBC JUMPI JUMPDEST PUSH1 0x0 DUP1 REVERT JUMPDEST PUSH2 0x6B PUSH1 0x4 DUP1 CALLDATASIZE SUB DUP2 ADD SWAP1 PUSH2 0x66 SWAP2 SWAP1 PUSH2 0x6E9 JUMP JUMPDEST PUSH2 0xEC JUMP JUMPDEST PUSH1 0x40 MLOAD PUSH2 0x79 SWAP3 SWAP2 SWAP1 PUSH2 0x8DD JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH2 0x9C PUSH1 0x4 DUP1 CALLDATASIZE SUB DUP2 ADD SWAP1 PUSH2 0x97 SWAP2 SWAP1 PUSH2 0x67D JUMP JUMPDEST PUSH2 0x230 JUMP JUMPDEST STOP JUMPDEST PUSH2 0xA6 PUSH2 0x2E3 JUMP JUMPDEST PUSH1 0x40 MLOAD PUSH2 0xB3 SWAP2 SWAP1 PUSH2 0x899 JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH2 0xD6 PUSH1 0x4 DUP1 CALLDATASIZE SUB DUP2 ADD SWAP1 PUSH2 0xD1 SWAP2 SWAP1 PUSH2 0x63C JUMP JUMPDEST PUSH2 0x466 JUMP JUMPDEST PUSH1 0x40 MLOAD PUSH2 0xE3 SWAP2 SWAP1 PUSH2 0x8BB JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH
```

Figura 19: Detalle de un contrato almacenado en el datastore

De la misma forma se procede a trabajar con la información de las carteras almacenadas.