Automated Malware and Exploit Data Collection Tool Documentation

version 0.3

Software Practicum Team 3

April 04, 2020

Contents

Welcome to Automated Malware and Exploit Data Collection Tool’s documentation!

src

MainServer module

`MainServer.`**`createScenario`**(`scenario_name`)
Creates a new scenario which includes the folders and the scenario JSON file :param scenario_name: String with the scenario name :return: True if the new scenario was successfully created

`MainServer.`**`deleteFile`**(`file_name`)

`MainServer.`**`deleteScenario`**(`scenario_name`)
Edits a current scenario with a JSON file :param scenario_name: String with the scenario name :return: True if the scenario has been successfully edited, otherwise False

`MainServer.`**`editScenario`**()
Edits a current scenario with a JSON file :param scenario_name: String with the scenario name :return: True if the scenario has been successfully edited, otherwise False

`MainServer.`**`getAvailableBoxes`**()
Gets the available boxes in the Vagrant context :return: A list of string with the available boxes

`MainServer.`**`getFileList`**()

`MainServer.`**`getScenario`**(`scenario_name`)
Gets the scenario as a JSON file :param scenario_name: String with the scenario name :return: JSON file with the scenario info

`MainServer.`**`getScenarios`**()
Gets the available scenarios :return: A list of strings with the available scenarios

`MainServer.`**`runVagrantUp`**(`scenario_name`)
Executes the vagrant up command for each machine in the scenario :param scenario_name: String with the scenario name :return: True if the vagrant up commands were successfully executed

`MainServer.`**`testPing`**(`scenario_name`, `source`, `destination`)
Tests network connectivity between two virtual machines :param scenario_name: String with the scenario name :param source: Source virtual machine :param destination: Destination virtual machine :return:

`MainServer.`**`uploadFile`**()

Entities package

Submodules

Entities.Entity module

class `Entities.Entity.`**`Entity`**
Bases: **`abc.ABC`**

abstract **`dictionary`****()**

abstract **`objectFromDictionary`****(**`dict`**)**

Entities.ExploitInfo module

class `Entities.ExploitInfo.`**`ExploitInfo`**(`name=''`, `type=''`, `download_link=''`)

Bases: **Entities.Entity.Entity**

dictionary ()
Generates a dictionary for the ExploitInfo object :return: A dictionary with ExploitInfo data

objectFromDictionary (dict**)**

Entities.NetworkSettings module

class Entities.NetworkSettings.**NetworkSettings** (network_name='', network_type='', ip_address='', auto_config=True)
Bases: **Entities.Entity.Entity**

dictionary ()
Generates a dictionary for the NetworkSettings object :return: A dictionary with NetworkSettings data

objectFromDictionary (dict**)**

Entities.Provision module

class Entities.Provision.**Provision** (name='', provision_type='shell')
Bases: **Entities.Entity.Entity**

dictionary ()
Generates a dictionary for the Provision object :return: A dictionary with Provision data

objectFromDictionary (dict**)**

setShellCommand (command**)**
Sets a new command to be executed as part of the provisioning :param command: Bash command intended for provisioning a virtual machine

Entities.Response module

class Entities.Response.**Response** (response='', code='', status='', task_id='', body='')
Bases: **Entities.Entity.Entity**

dictionary ()
Generates a dictionary for the Wrapper object :return: A dictionary with Wrapper data

objectFromDictionary (dict**)**

setBody (body**)**

setCode (code**)**

setResponse (response**)**

setStatus (status**)**

setTask_id (task_id**)**

Entities.Scenario module

class Entities.Scenario.**Scenario** (scenario_name)
Bases: **Entities.Entity.Entity**

addVM (vm)
Adds a new virtual machine to this scenario :param vm: Object which carries the virtual machine data

dictionary ()
Generates a dictionary for the Scenario object :return: A dictionary with Scenario data

objectFromDictionary (dict)

setExploitInfo (exploit_info)
Sets the exploit info for this scenario :param exploit_info: Object which carries the exploit info

setVulnerabilityInfo (vulnerability_info)
Sets the vulnerability info for this scenario :param vulnerability_info: Object which carries the vulnerability info

Entities.VagrantFile module

class Entities.VagrantFile.**VagrantFile**
Bases: **object**

vagrantFilePerMachine (machine, machine_path)
Creates a vagrant file for this machine :param machine: Object which carries the virtual machine data :param machine_path: Folder path to this machine :return: String used to write the vagrant file

Entities.VirtualMachine module

class Entities.VirtualMachine.**VirtualMachine** (name='', os='', is_attacker=False)
Bases: **Entities.Entity.Entity**

addSharedFolder (hostPath, guestPath)
Adds the shared folder between the host and the guest :param hostPath: String with the host path :param guestPath: String with the guest path

dictionary ()
Generates a dictionary for the Virtual Machine object :return: A dictionary with Virtual Machine data

enableGUI (isVisible)
Enables the GUI for this virtual machine :param isVisible: Boolean to enable or disable the GUI in a virtual machine

objectFromDictionary (dict)

setName (name)
Sets the name for this virtual machine :param name: String with the virtual machine name

setNetworkSettings (network_settings)
Sets the network settings for this virtual machine :param network_settings: Object which carries the network settings data

setOS (os)
Sets the OS for this virtual machine :param os: String with the virtual machine OS

setProvision (provision)
Sets the provision for this virtual machine :param provision: Object which carries the provision data

Entities.VulnerabilityInfo module

class Entities.VulnerabilityInfo.**VulnerabilityInfo** (name='', type='', cve_link='', download_link='')
Bases: **Entities.Entity.Entity**

dictionary ()
Generates a dictionary for the VulnerabilityInfo object :return: A dictionary with VulnerabilityInfo data

objectFromDictionary (dict)

Module contents

Managers package

Submodules

Managers.DatabaseManager module

Run mongoDB server:

- cd C:Program FilesMongoDBServer■.2■in
- mongod

Create database:

- use soft_prac

Create collections:

- db.createCollection('scenarios')

class Managers.DatabaseManager.**DatabaseManager** (url='')
Bases: **object**

deleteScenario (scenario_name)

editScenario (scenario_json)

getScenario (scenario_name)

getScenarioNames ()

getScenarios ()

insertScenario (scenario_json)

Managers.FileManager module

class Managers.FileManager.**FileManager**
Bases: **object**

createMachineFolders (scenario_json)
Creates a folder for each machine in the scenario :param scenario_json: String with the scenario name :return: True if machine folders are created successfully

createScenarioFolders (scenario_name)
Creates a scenario folder with the JSON, Exploit, Vulnerability and Machines subfolders :param scenario_name: String with the scenario name :return: True if the scenario is created successfully

getCurrentPath ()

Gets the project folder path :return: String with the project path

getJSONPath (scenario_name)

getScenariosPath ()
Gets the scenarios folder path :return: String with the scenarios project path

Managers.ScenarioManager module

class Managers.ScenarioManager.**ScenarioManager** (db_manager='')
Bases: **object**

deleteScenario (scenario_name)

editScenario (new_scenario)
Edits a current scenario with a JSON file :param scenario_name: String with the scenario name :param scenario_json: JSON file with the new scenario :return: True if the scenario has been successfully edited, otherwise False

getScenario (scenario_name)
Gets the scenario as a JSON file :param scenario_name: String with the scenario name :return: JSON file with the scenario info

getScenarios ()
Gets the available scenarios :return: A list of strings with the available scenarios

newEmptyScenario (scenario_name)
Creates a new scenario which includes the folders and the scenario JSON file :param scenario_name: String with the scenario name :return: True if the new scenario was successfully created

scenarioExists (scenario_name)
Check if a scenario exists :param scenario_name: String with the scenario name :return: False if the scenario JSON file does not exist and the path to the JSON file if it exist

testDB (scenario_name)

testDB2 ()

Managers.VagrantManager module

class Managers.VagrantManager.**VagrantManager**
Bases: **object**

createVagrantFiles (scenario_name)
Creates a vagrant file per machine in a scenario :param scenario_json: String with the scenario name :return: True if vagrant files were successfully created

getAvailableBoxes ()
Gets the available boxes in the Vagrant context :return: A list of string with the available boxes

haltVM (machine_name)

restartVM (machine_name)

runVagrantUp (scenario_name)
Executes the vagrant up command for each machine in the scenario :param scenario_name: String with the scenario name :return: True if the vagrant up commands were successfully executed

sendCommand (scenario_name, machine_name, command, default_timeout=5, show_output=True)

testNetworkPing (scenario_name, machine_name, destination_machine_name, count=1)

Module contents

Indices and tables

- **genindex**
- **modindex**
- **search**

Index

Python Module Index

e

m