Contents

1 Overview

2 Installation

SaTC project depends on two external libraries:

- **commons-csv-1.4.jar**
  Commons CSV reads and writes files in variations of the Comma Separated Value (CSV) format. More details can be found on the project official website: Commons CSV Home
- **mason.19.jar**

3 Running

4 Input

4.1 netmap.csv

Short introduction to netmap.csv

column	meaning
id	TBD
type	TBD
sd_type	TBD
up_id	TBD
channel	TBD
cost	TBD
cap	TBD
security	TBD

Example:

column	value
id	1
type	3
sd_type	D
up_id	201
channel	type1
cost	1
cap	2000
security	100

4.2 virtualmap.csv

column	meaning
id	TBD
type	TBD
configuration	TBD
channel	TBD
agent	TBD
intel_level	TBD
intel	TBD
security	TBD

4.3 testdraw.csv

column	meaning
n	TBD
type	TBD
load	TBD
elast	TBD

5 Output

5.1 Run Configuration Files

column	meaning
netmap	TBD
virtualmap	TBD
draws	TBD
history	TBD
bids	TBD
transcost	TBD
transcap	TBD
seed	TBD
debug	TBD

5.2 Log Files

5.3 Net Demand Files / Debugging Files

column	meaning
pop	TBD
id	TBD
tag	TBD
dos	TBD
steps	TBD
p	TBD
q_min	TBD
q_max	TBD

5.4 Output Files

column	meaning
pop	TBD
dos	TBD
id	TBD
rblock	TBD
blocked	TBD
p	TBD
q	TBD
upcon	TBD

6 Modeling Framework

6.1 Simulation Environment

Env

6.2 Traders and Markets

Trader, market, demand, net demands

6.3 Price Determination

Transmission costs, congestion, aggregation, population

6.4 Communications

Channel, msg

6.5 DoS

6.6 Adversaries

History, intel

7 Objects Reference

7.1 Adversary

Adversary is an abstract class for virtual agents who try to disrupt the power grid. It has several subclasses that can perform different kinds of attacks.

Adv_Adam
An **Adv_Adam** object can send false bids to the other nodes inside the grid. It looks up the **Intel** hash map and picks all the target nodes that be connected to. Then constructs the false bid and sends the bid to the targets.

Adv_Beth
Adv_Beth class is the promoted version for **Adv_Adam** class. It has all the functionalities provided by **Adv_Adam** as well as to forge credentials and to decipt the recipient with that.

Adv_Darth
An **Adv_Darth** object is able to attack the market with constrained tranmission to their upstream nodes. It is supposed to have compromised one trader from the target market. It first intercepts all messages sent by the compromised trader. Then extracts the trader's historical demands. Shifts the demand curve by the attacker customized distance. Finally injects the fake demands back into the channel and these demands will be sent to the target market.

Adv_Elvira
Adv_Elvira performs similar behavior to **Adv_Darth**. We suppose that **Adv_Elvira** has compromised several traders instead of only one. Then it performs the attack from all of these compromised traders, and shifts all their demand curves by a smaller value, with a total shift distance that equals to the attacker customized distance. Compared with the **Darth**, **Elvira** performs the attack more softly but with more traders to achieve the same result, which means it has less possbility to be discovered by the anomalous data detection.

Adv_Faust

An **Adv_Faust** object listens the target market channel and intercepts messages sent by the target. Then injects the DEMAND message back to the channel without any modification, but tampers the price info in the PRICE message and injects it back to the channel before traders calculate loads.

7.2 Agent

Agent is an abstract class that provides basic features for entities that communicate. It has two subclasses: **Grid**, for agents actually connected to the power grid, and **Virtual**, for agents that only have communication links.

7.3 Channel

A **Channel** object is communications channel. An arbitrary number are allowed and they can have different properties. Each agent has a specific Channel that it uses to communicate with its upstream parent node. A Channel is used to by one agent to send Msg objects to another.

Each Channel has one main method, **send()**. It looks up the sender and recipient from the corresponding fields of the Msg object it is given and checks whether random denial of service filtering applies to the sender. If so, the message is dropped. Otherwise, as long as the message is not diverted (discussed below) it is passed to the recipient via the recipient's **deliver()** method.

Three hooks are available to support man in the middle attacks and other interventions. Message diversions can be set up via each channel's **divert_to()** and **divert_from()** methods. The first diverts all messages sent *to* a given node and the second diverts all messages sent *by* a given node. The *from* diversion is processed first and when both apply to a given message it takes precedence. Messages can be reinserted downstream from the diversions via the channel's **inject()** method. Future features to be implemented:

- Random DOS loss rates that can vary by channel
- VPN channels that prohibit diversions

- Authentication of senders
- Authentication of recipients – blocking **divert_to()**

7.4 Demand

A **Demand** object holds a net demand curve expressed as a list of steps. Positive quantities indicate demand and negative quantites indicate supply. **Trader** nodes send **Demand** objects to **Market** nodes. Lower tier **Market** nodes send aggregated **Demand** objects to higher-tier **Market** nodes. In all cases the curves are sent via **Msg** objects.

7.5 Env

The **Env** object represents the environment under which the simulation is running. It includes various global variables and is also responsible for loading data, configuring the network of **Agent** nodes, the **Channel** objects they use to communicate, and then starting the simulation.

7.6 Grid

Abstract class for grid-connected agents (that is, agents through which power can flow). Has two subclasses: **Trader** and **Market**.

7.7 History

A **History** object stores historic price and quantity data for a specific agent. Provides several methods used to store and retrieve demand, price, quantity, and constraint information. The simulator can load historic data from preset history file before it runs. Adversaries can extract relevant information from history object to build fake demand curves and then perform attacks.

7.8 Intel

Stores a virtual agent's information about another agent within the grid. Contains functionality to store an agent's grid-level (transmission cost, price, parent, children, tier), historic (price, quantity, bid), and some status (compromised, interceptTo, interceptFrom, forge, send ,learned) information. Can also retrieve max/min/avg information regarding p and q.

7.9 Market

Represents a market. Aggregates demands by child nodes, which can be **Trader** or other **Market** nodes. If the node has a parent, it adjusts the aggregate demand for transmission costs and capacity to the parent and passes the adjusted demand up. If the node does not have a parent, it determines the equilibrium price. Market nodes receive prices from upstream, adjust them for transmission parameters, and then pass them down to child nodes.

7.10 Msg

A single message from one agent to another. At the moment, two types of messages can be sent: one with a **Demand** object and one with a price. All **Msg** objects are passed via **Channel** objects. Future features to be implemented:

- Encryption that prevents reading by diverters
- Signing that prevents spoofing the sender

7.11 Trader

Abstract class for end agents. **Trader** nodes have upstream parents that are **Market** nodes. They submit **Demand** objects to their parent **Market** nodes and then receive prices back. Final demand or supply results from the prices received.

Provides method **getOneDemand()** to retrieve demand, static method **readDraws()** to load draws from history files, and abstract method **drawLoad()** to build the agent's demand curve.

Has subclass: **TraderMonte**.

TraderMonte

A **TraderMonte** object represents the trader under Monte Carlo mode. Has two types: end users and suppliers. Implements method **drawLoad()**, and provides method **readDraws()**.

7.12 Util

A utility class that includes a few general purpose methods for opening files with built-in exception handling.

7.13 Virtual

Abstract class for agents connected to the communications network but not connected directly to the power grid.