Power-Attack: A comprehensive tool-chain for modeling and simulating attacks in power systems

Ajay Chhokra, Carlos Barreto, Abhishek Dubey, Gabor Karsai and Xenofon Koutsoukos
{ajay.d.chhokra,carlos.a.barreto,abhishek.dubey,gabor.karsai,xenofon.koutsoukos}@vanderbilt.edu
Institute of Software Integrated Systems, Vanderbilt University
Nashville, TN, USA

ABSTRACT

Due to the increased deployment of novel communication, control and protection functions, the grid has become vulnerable to a variety of attacks. Designing robust machine learning based attack detection and mitigation algorithms require large amounts of data that rely heavily on a representative environment, where different attacks can be simulated. This paper presents a comprehensive tool-chain for modeling and simulating attacks in power systems. The paper makes the following contributions, first, we present a probabilistic domain specific language to define multiple attack scenarios and simulation configuration parameters. Secondly, we extend the PyPower-dynamics simulator with protection system components to simulate cyber attacks in control and protection layers of power system. In the end, we demonstrate multiple attack scenarios with a case study based on IEEE 39 bus system.

CCS CONCEPTS

• **Networks** → **Cyber-physical networks**; • **Software and its engineering** → **Domain specific languages**; • **Security and privacy** → *Hardware attacks and countermeasures.*

KEYWORDS

Cyber attacks, Power system simulation, Cyber security, Protection System

ACM Reference Format:
Ajay Chhokra, Carlos Barreto, Abhishek Dubey, Gabor Karsai and Xenofon Koutsoukos. 2021. Power-Attack: A comprehensive tool-chain for modeling and simulating attacks in power systems. In *9th Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MCPES'21), May 19–21, 2021, Nashville, TN, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3470481.3472705

1 INTRODUCTION

Securing the power system is one of the top national priorities. However, the adoption of new sensing and distributed control technologies such as Phasor Measurement Units (PMU) and advance protection relays have increased the complexity of the system as well as expanded the attack surface. As a result of the increased complexity and greater attack surface, identification and mitigation of potential threats have become very challenging.

To tackle system complexity, a number of machine learning based approaches have been proposed in the literature that are successful at detecting and preventing attacks [14]. It is well known that the accuracy of any machine learning algorithm depends heavily on the quality of operational data. Due to limited number of successful attacks, these approaches seldom rely upon synthetic data of different simulated attack scenarios. Several testbeds [10] and simulation frameworks [11], [20] exist in the scientific literature to simulate different scenarios. Despite these efforts, there are still gaps that need attention, such as, 1) Lack of an integrated framework to simulate sophisticated attacks involving protection relays, breakers, PMU and SCADA data. Majority of the frameworks allow cyber attacks associated with manipulation of sensor data (PMU and/or SCADA) to compromise state estimator and other controllers in a control center. These frameworks lack constructs to simulate attacks on protection relays and breakers that have been shown to cause significant damage to the system once compromised [24]. 2) Lack of a high level simulation configuration and attack specification language. Most of the simulation frameworks use academic or industry standard simulators such as PowerWorld [3], OPNET [6] to simulate different attack scenarios. Often these frameworks require setting up a large set of parameters to execute a simulation that requires in-depth knowledge of the complete parameter space, different components of the system and semantics of different simulators. A universal high level configuration and attack language can greatly help analysts to quickly specify various parameters and simulate different attack scenarios.

To this end, we present *Power-Attack*, a comprehensive tool-chain for modeling and simulating attacks in power systems. The contributions of this paper are the following: (1) We describe a PyPower [16] and PyPower-Dynamics [22] based simulation engine that can simulate both the continuous dynamics of generators, transmission lines, controllers (frequency and voltage) as well as discrete dynamics of protection relays. (2) We introduce a generic relay and breaker model to simulate nominal and compromised behavior within this simulation engine. (3) Finally, we present a domain specific language that can be used to create complex scenarios involving grounding faults in transmission lines and buses as well as cyber attacks on protection system, controllers, and sensing data. Our focus in this paper is limited to *device* attacks that aim to compromise a grid device in order to gain control and *data* attack attempts to insert, alter, delete data or control commands in the network traffic so as to mislead the smart grid to make wrong decisions/actions.

MCPES'21, May 19–21, 2021, Nashville, TN, USA

ACM ISBN 978-1-4503-8608-1/21/05…$15.00
https://doi.org/10.1145/3470481.3472705

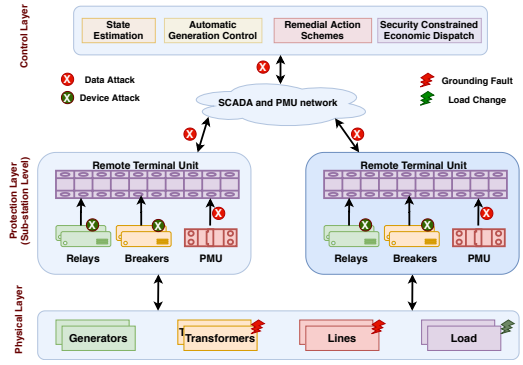


Figure 1: Power-Attack tool-chain capabilities

The outline of the paper is as follows. We start with related research in Section 2. Section 3 discusses the Power-Attack tool-chain. Section 4 presents the evaluation results with the help of three scenarios involving attacking multiple protection relays and a load frequency controller using an IEEE 39 Bus system [4] followed by concluding remarks in Section 5.

2 RELATED RESEARCH

Several groups have developed testbeds to study the cyber security of industrial control systems

[13]. For instance, one of the earliest initiative, National Supervisory Control And Data Acquisition TestBed (NSTB) implemented actual physical grid components including generation and transmission, while also incorporating industry standard software products [1]. Unfortunately, the substantial cost of deploying purely physical testbed limits its usability. A number of software based testbeds such as Virtual Control System Environment (VCSE) [18], Virtual Power System Testbed (VPST) [5], Testbed for Analyzing Security of SCADA Control Systems (TASSCS) [17] and SCADASim [21] have been designed to study performance of SCADA protocols and power system equipment under cyber attacks. These approaches use PowerWorld [3] to simulate physical layer and OPNET [6] or RINSE [15] to simulate network traffic. None of these approaches consider attacks on protection system components such as IEDs. The authors in [14] and [10] presented integrated testbeds that also allow users to simulate device attacks in protection system components. However, both of these testbeds rely on a licensed power simulator, PowerFactory [2] from DIgSILENT.

On the other hand, the Power-Attack aims at providing an open source simulation framework that gives the users ability to model and simulate different cyber attacks along with physical events in physical, control and protection layers to generate synthetic data. Our proposed approach is similar to the PSAT [19] based framework, GridSTAGE [20] to simulate cyber attacks (man in the middle) involving PMU and SCADA data. Similar to other SCADA based testbeds, this simulation framework does not allow device attacks on the protection system. Another key feature that distinguish Power-Attack from existing cyber attack simulation framework is the availability of a high level domain specific language (DSL) to define cyber attacks and simulation parameters. There exists some DSLs for power systems that target simulation of power systems for fault analysis [8], [12]. However, none of these language are suited for modeling and simulating cyber attacks in power systems.

3 APPROACH

In this tool-chain, we model a scenario as a sequence of events that includes both cyber attacks and physical events such as fault injection, disabling a branch, or changing the load demand on a bus etc., as shown in Figure 1. Power-Attack consists of two main components, 1) Domain Specific Language (DSL) to define simulation parameters and attack scenarios, and 2) A scalable simulation engine that executes the different attack scenarios. These components are described in more detail in the following sub-sections.

3.1 Domain Specific Language (DSL)

Power-Attack DSL is an imperative language to represent multiple configuration parameters and attack scenarios involving different protection system components, controllers and sensing devices. Table 1 lists the grammatical rules of the DSL. Each rule describes a concept of the Power-Attack language along with its syntactic structure in a textual format. These meta rules are expressed using Textx [9], a meta-language and parser tool for creating DSLs in Python.

Power-Attack DSL provides a collection of tuples, see rule *Pair* in Table 1 that allows users to specify arbitrary parameters as key value pairs. According to the rule, *Pair*, a key can be any `STRING` that signifies the parameter label and the associated value can be a one or two dimensional scalar, see rules *Value*, *Vector* and *Scalar*. The value associated with a key can either be a fixed or random value. The DSL support multiple distribution types to specify random variables such as `Uniform`, `Gaussian` and `Discrete-Uniform` as shown in rule *Distribution_type*. The language also defines types related to power system component, mentioned in rules *Bus_type*, *Branch_type*, *Controller_type* and set of types to specify attacks such as *Attack_type*, *Pmu_attack*, *Controller_attack*, *Relay_attack* and *Breaker_attack*. A key feature of the DSL is that it allows probabilistic variables, whose specific value is drawn from a defined distribution at simulation time.

Listing 1 shows a sample attack model that conforms to Power-Attack DSL. All attack files can be divided into three sections, 1) *set up*, 2) *preconditions* and 3) *scenarios*, where each section is enclosed within `Begin <> phase` ... `End <> phase` statements as highlighted in lines 1-6, 7-11, and 12-15. In *set up* section, a user defines structure of the power network and various parameters to execute the simulation. The current implementation requires the power network definition in the format accepted by PyPower, PandaPower or MATPOWER as a large number of standard network definitions already exists for these load flow solvers. Power-Attack DSL divides the configuration space into two categories, the first category consists of parameters related to simulation engine such as simulation time, tolerance, maximum iterations etc, while the second category of parameters is associated with equipment in physical, protection and control layers of power system such as relays, generators etc. A user can specify configurations of multiple components in *set up* section such that configuration statement of each component starts with keyword `Configure` followed by device identifiers and a parameter key-value map. For instance, the line 4 in Listing 1

Table 1: Power-Attack DSL Grammar

Deterministic_scalar: (NUMBER | FLOAT | STRING);
Primitive_pair: (key=Deterministic_scalar : val=Deterministic_scalar);
Probablistic_scalar: (distribution=Distribution_type , { params+=Primitive_pair[,] });
Distribution_type: (Uniform | Discrete-Uniform | Gaussian);
Scalar: (Deterministic_scalar | Probablistic_scalar);
Vector: (real=Scalar , imag=Scalar);
Value: (Scalar | Vector);
Pair: (key=STRING : val=Value);
Time: @ time Value;
Case_file: Case : case_file=STRING;
Setup_config: Configure (Breaker_config | Relay_config | Controller_config | Tracer_config | Simulator_config | Generator_config);
Breaker_config: breaker @ id=Branch_type with { params+=Pair[,] };
Relay_config: type=Relay_type relay @ id=Branch_type with { params+=Pair[,] };
Element_type: (instantaneous | time-delayed-1 | time-delayed-2) ;
Relay_type: (over-current | distance) ;
Bus_type: bus id=Deterministic_scalar;
Branch_type: branch between to_bus=Bus_type and from_bus=Bus_type;
Controller_type: (lfc | avr)
Controller_config: type=Controller_type in generator @ id=Bus_type with { params+=Pair[,] };
Tracer_config: trace of metrics+=Metric_type[,];
Metric_type: (bus voltages | power flows | islands | load-loss | generator frequency);
Simulator_config: simulation with { params+=Pair[,] };
Generator_config: generator @ id=Bus_type with { params+=Pair[,] };
Precondition_config:(Load_change | Trip_node | Fault_injection);
Load_change: Change load on id=Bus_type to val=Value time=Time;
Trip_node: Trip id=Node time=Time;
Node: (Bus_type | Branch_type);
Fault_injection: Inject fault in id=Node time=Time;
Attack_scenario: label=ID : < attack_sequence+=Attack[,] >;
Attack: Attack element=Attack_type time=Time;
Attack_type: (Pmu_attack | Controller_attack | Relay_attack | Breaker_attack);
Pmu_attack: pmu @ id=Bus_type with type= Data_attack_type attack of factor=Scalar;
Data_attack_type: (scaling | biasing);
Controller_attack: kind=Controller_type in generator @ id=Bus_type with type= Data_attack_type attack of factor=Scalar;
Relay_attack: kind=Relay_type relay element+=Element_type element @ id=Branch_type with type=Relay_attack_type attack;
Relay_attack_type: (missed | spurious);
Breaker_attack: breaker @ id=Branch_type with type=Breaker_attack_type attack;
Breaker_attack_type: (stuck open | stuck close);
PowerAttackModel:
Begin setup phase case_file=Case_file setup_configs*=Setup_config[,] End setup phase (Begin preconditions preconditions*=Precondition_config[,] End preconditions)? (Begin scenarios attack_scenarios+=Attack_scenario[,] End scenarios)?;

```
1  Begin setup phase
2  Case : '/Users/projects/gridmodels/arbitrary.py'
3  Configure simulation with { 't_sim' : 200} ,
4  Configure breaker @ branch between bus 10 and bus 11
       with { 'tto' : (Uniform, {'mue':0.050, 'sigma'
      :0.008}) },
5  Configure generator @ bus 30 with {'inertia' : 500},
6  End setup phase
7  Begin preconditions
8  Change load on bus 11 to (100, 43.78) @ time 9,
9  Trip branch between bus 11 and bus 9 @ time 90,
10 Inject fault in bus 21 @ time 76
11 End preconditions
12 Begin scenarios
13 Scenario1 : < Attack pmu @ bus 32 with scaling
       attack of 1.23 @ time 3.50, Attack breaker @
       branch between bus 4 and bus 7 with stuck close
        attack  @ time 9 >,
14 Scenario2 : < Attack lfc in generator @ bus 32 with
       biasing attack of (Gaussian, {'mue' : 0.2, '
       sigma' : 0.02}) @ time 4, Attack distance relay
        instantaneous element @ branch between bus 16
       and bus 21 with missed attack @ time 7.45 >
15 End scenarios
```

Listing 1: Sample Attack Model

changes the default time to open the breaker, `tto`, between buses 10 and 11 to a Gaussian random variable (0.050, 0.008). The device identifiers for different components are defined in rule *Setup_config*. The software repository [7] of the project maintains the list of all parameters along with their description and default values.

The second section, *preconditions* allows users to define physical events to set up specific operating conditions for attacks in a scenario. The supported preconditions (events) include 1) changing a load attached to a bus, defined in rule *Load_change*. For instance, line 8 in Listing 1 changes the load demand on bus 11 to 100 +j43.78 at time 9 secs. 2) Injecting planned outages by removing buses or branches from the system, see rule *Trip_node* and 3) Injecting grounding faults in equipment as described in *Fault_injection* rule.

In the last section, *scenarios*, a user can define multiple attack scenarios according to the rule *Attack_scenarios*. An attack scenario is a sequence of different types of attacks. Power-Attack DSL supports four types of attacks described in rules *Pmu_attack*, *Controller_attack*, *Relay_attack* and *Breaker_attack* where the first two belong to data attack category and the last two are device attacks. According to rule *Data_attack_type*, a data attack can be of two types, *scaling* or *biasing*. A *scaling* attack scales the target value by a *factor*. For instance, in Listing 1 line 13, the attack scenario with label, `Scenario1` creates data falsification attack by scaling the output of PMU at bus 32 by a factor of 1.23. A *biasing* attack adds a bias to the target value as illustrated in scenario, `Scenario2` in Listing 1 (line 14) where a random bias with Gaussian distribution is added. The DSL also allows users to specify device attacks that compromise an IED such as numerical relays and breakers. In the current implementation, a relay can be compromised in two ways such that 1) It fails to detect fault conditions in its zone of protection and 2) It spuriously detects a fault conditions and instruct a breaker to remove an equipment. We label these relays attacks as *missed* and *spurious* respectively, as mentioned in rule *Relay_attack_type*. Similar to relay attacks, breakers can be subjected to two kinds of

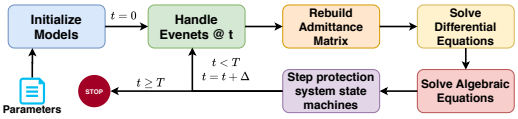


Figure 2: Simulation Loop

attacks, illustrated in rule *Breaker_attack_type*, where attacks *stuck open* and *stuck close* force the breaker to ignore the signals from relays and remain in open and close position respectively.

3.2 Simulation Engine

Power-Attack tool-chain uses a sequence of steps to simulate attack scenarios specified in an attack model file. In the first step, configuration parameters, $\mathcal{P}$, precondition events, $\mathcal{E}$ and attack scenarios, $\mathcal{A}$ are extracted from the model file which can be fixed or random in nature. In the second step, fixed and random parameters, precondition events and attacks in j^{th} scenario are segregated in disjoint sets as $(\mathcal{P}_f, \mathcal{P}_r)$, $(\mathcal{E}_f, \mathcal{E}_r)$, $(\mathcal{A}_{f,j}, \mathcal{A}_{r,j})$. In the third step, simulation space, $\mathcal{S}$ is created by sampling the distributions of random variables as shown in Equations (1), (2) and (3)

$$\mathcal{S}' = \left(\mathcal{E}_f \times \{\mathcal{E}_r^1, ., \mathcal{E}_r^K\}\right) \cup \left(\mathcal{P}_f \times \{\mathcal{P}_r^1, ., \mathcal{P}_r^K\}\right) \quad (1)$$

$$\mathcal{S}^j = \left(\mathcal{A}_{f,j} \times \{\mathcal{A}_{r,j}^1, ., \mathcal{A}_{r,j}^K\}\right) \times \mathcal{S}' \quad (2)$$

$$\mathcal{S} = \bigcup_j \mathcal{S}^j \quad (3)$$

where $\mathcal{A}_{r,j}^i$, $\mathcal{E}_r^i$, $\mathcal{P}_r^i$ are i^{th} samples of set of random variables in $\mathcal{A}_{r,j}$, $\mathcal{E}_r$, $\mathcal{P}_r$ respectively and K is tool-chain parameter that governs the maximum number of samples. In the last step, for each element, $s \in S$, a simulation process is spawned and assigned to a worker. Power-Attack utilizes one or more workers to execute multiple attack scenarios in parallel. Each worker process runs the simulation engine that accepts simulation parameters and attack scenario over REST end-point.

The underlying simulation engine in Power-Attack is based on PyPower-Dynamics. Although PyPower-Dynamics can simulate the dynamic behavior of synchronous machines and controllers such as automatic voltage regulators (AVR), but it lacks the constructs to model devices with discrete dynamics such as numerical relays. Power-Attack simulation engine integrates the PyPower-Dynamics with state machine based models to capture behavior of components in protection layer. Figure 2 highlights the simulation loop of the extended PyPower-Dynamics simulator. The simulation process begins with solving the load flow using PyPower and building an admittance matrix to initialize various dynamic models in the system. At any time step in the loop, the simulator first executes the scheduled events at current time step. These events can be 1) changing the load attached to a bus, 2) removing a branch or bus from the system, 3) generating signals to induce data (scaling or biasing) or device (spurious, missed, stuck open or close) attacks. In case the structure of the network is changed as a result of tripping of branch or bus, the admittance matrix is rebuilt. After handling external events, differential equations and algebraic network equations (DAEs) are solved using partitioned solution approach. Based on the solution, the transition function of state machines is evaluated. All the events produced in current time step are added to the event queue and the simulator advances to the next time step ($t + \Delta$). The simulation ends when time step reaches user-defined threshold (T) or the system fails to converge. The following sub-section give more details on the implementation of device and data attacks.

3.2.1 Device Attacks. Modern numerical relays consist of different types of elements that implement various protection functions. These elements can be categorized into two types 1) Instantaneous and 2) Time-delayed elements. An instantaneous element instructs a breaker to open as soon as the tripping condition is satisfied, for instance, a zone 1 element of distance protection. Whereas the time delayed elements wait for a pre-defined duration of time before sending the trip command, for example zone 2 or 3 elements of distance protection.

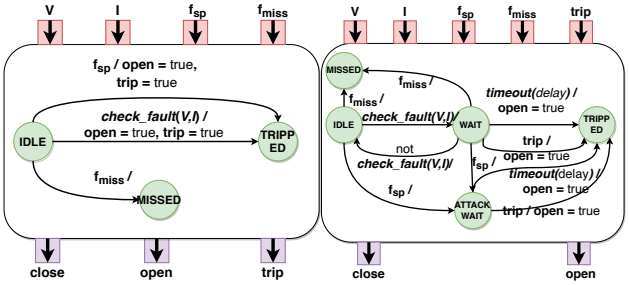
Figure 3a shows a generic state machine template that captures the behavior of an instantaneous element. The state machine contains 3 states (`IDLE`, `MISSED`, `TRIPPED`), 4 input ports (f_{sp}, f_{miss}, V, I) and 3 output ports (*trip*, *open*, *close*). The input ports are read only state variables that are used by the state machine to synchronize with the connected output ports of other state machines as well as changes in voltage and current values in the physical layer. According to Figure 3a, the state machine is in `IDLE` state initially and at every time step, it checks for the condition on all the outgoing transitions based on the latest value of the input ports. If the function $check\_fault(V, I)$ returns `true`, then the state machine jumps to `TRIPPED` and instructs the associated breaker to open by setting the output port *open* along with generating a trip transfer signal for the relay element connected on the opposite side of the branch. Similarly, if spurious attack is performed in the current time step by setting the input port f_{sp}, then the state machine jumps to `TRIPPED` while updating the output ports as before. However, in the case of missed attack, the value of port f_{miss} is `true` which forces the state machine to transition to `MISSED` state.

The behavior of the time delayed element as highlighted in Figure 3b is same except after detecting fault conditions or under the influence of spurious attack, the state machine first transitions to an intermediate state, `WAIT` or `ATTAK-WAIT` states respectively, for predefined amount of time, *delay* before jumping to `TRIPPED`. While in the `WAIT` state, the automaton checks for the fault conditions at every step. If the fault is cleared by other relay elements, the state machine returns back to `IDLE` state. In case missed or spurious attack is injected while in `WAIT` state, the state machine transitions to `MISSED` or `ATTACK-WAIT` respectively.

Figure 4 shows a state machine that captures behavior of a breaker. Under normal conditions, the breaker transitions from `CLOSE` to `OPENING` and then eventually to `OPEN` state in *tto* secs after receiving command from a relay on input port, *open*. However, when the breaker is attacked, by setting the input port, f_{stuck}, the breaker transition to `STUCK_CLOSE` or `STUCK_OPEN` from (`CLOSE`, `CLOSING`) or (`OPEN`, `OPENING`) states respectively.[1]

3.2.2 Data Attack. One of the primary focus of the smart grid paradigm is to move away from a centralized architecture and

[1]The execution of all state machines follows a pre-defined order in which state of each automaton is updated and the order of checking outgoing transitions from every state is also fixed.



(a) Instantaneous Relay Element (b) Time-delayed Relay Element

Figure 3: Protection Relay Automata

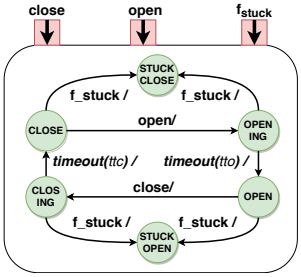


Figure 4: Breaker Automaton

adopt distributed or decentralized control methodologies in which different control centers or sub-stations communicate with each other and take consensus based decisions. Power-Attack simulator provides such distributed consensus based control algorithm for frequency control[23] defined by Equations (4), (5), (6) and (7)

$$P_{m,i} = P^{ref}_{m,i} + \mu_i \tag{4}$$

$$\mu_i = P^d_i + \Omega_i \tag{5}$$

$$P^d_i = (\omega^* - \omega_i)D_i \tag{6}$$

$$\dot{\Omega}_i = (\omega^* - \omega_i) - \sum\nolimits_j a_{ij}(C_i\Omega_i - C_j\Omega_j) \tag{7}$$

where $P_{m,i}$ is the mechanical power injected in the i^{th} generator, $P^{ref}_{m,i}$ is the power generation scheduled for the ith generator, μ_i is the control variable, P^d_i corresponds to droop control and D_i is the droop constant. The term Ω_i is a consensus variable and a_{ij} denotes the connectivity between the generators i and j. The consensus-based control requires that generators to share their consensus variables Ω_i. Now we analyze the effect of data attacks, scaling and biasing, in which an adversary manages to compromise the consensus variable.

Biasing Attack: The adversary implements a man in the middle attack on the ith generator and injects a bias to the consensus variable, $\hat{\Omega}_i = \Omega_i + b_i$ and transmits to the neighbors of i. With this attack, the adversary manages to change the setpoint of some generators as shown in Equations (8), (9).

$$\dot{\Omega}_i = \omega^* - \omega_i - \sum\nolimits_k a_{ik}(C_i\Omega_i - C_k\Omega_k) \tag{8}$$

$$\dot{\Omega}_j = \omega^* - \omega_j - \sum\nolimits_k a_{jk}(C_j\Omega_j - C_k\Omega_k) + a_{ji}C_ib_i \tag{9}$$

```
1  Begin setup phase
2  Case : '/Users/projects/power-attack/grid-models/case39
       .py'
3  End setup phase
4  Begin preconditions
5  Change load on bus 30 to (80, 30) @ time 1,
6  Change load on bus 30 to (97.60, 44.20) @ time 2
7  End preconditions
8  Begin scenarios
9  Sc1 : < Attack lfc in generator @ bus 30 with biasing
       attack of (Gaussian, {'mue': 0.01, 'sigma': 0.005}
        @ time 5 >,
10 Sc2 : < Attack lfc in generator @ bus 30 with scaling
       attack of 2.1 @ time 5 >,
11 Sc3 : < Attack distance relay time-delayed-2 element @
       branch between bus 15 and bus 16 with spurious
       attack @ time 16 >
12 End scenarios
```

Listing 2: Attack Model for Scenarios Sc1 Sc2 and Sc3

Scaling Attack: In this attack, the adversary scales the consensus variable, $\hat{\Omega}_i = \gamma_i\Omega_i$ and transmits to the neighbors of i. With this attack the droop control, P^d_i can still reduce the frequency error, that is, $\omega_i \rightarrow \omega^*$ but the attack changes the power produced and leads to higher expenses as shown in Equations (10) and (11).

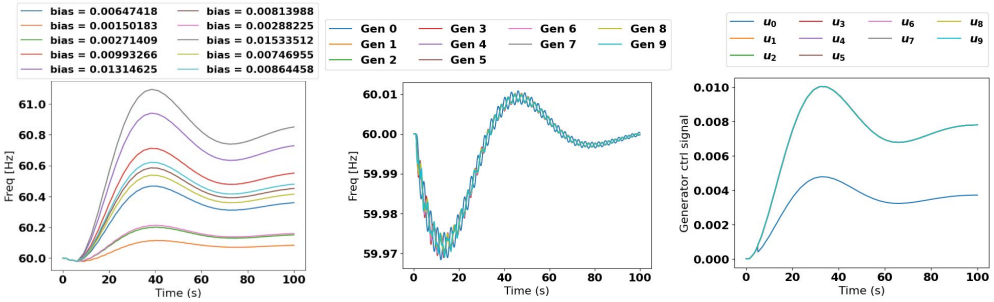
$$\dot{\Omega}_i = \omega^* - \omega_i - \sum\nolimits_k a_{ik}(C_i\gamma_i\Omega_i - C_k\Omega_k) \tag{10}$$

$$\dot{\Omega}_j = \omega^* - \omega_j - \sum_{k\neq i} a_{jk}(C_j\Omega_j - C_k\Omega_k) - a_{ji}(C_j\Omega_j - C_i\gamma_i) \tag{11}$$

4 RESULTS

In this section, we show three scenarios involving data and device attacks using an IEEE 39 Bus system. These attack scenarios are outlined in Listing 2. All scenarios depend upon a common operating point that stems from creating a load disturbance at bus 30, specified in *preconditions* section (Lines 4-7). The scenarios labeled as `Sc1` (Line 9) and `Sc2` (Line 10) inject biasing and scaling attacks in the load frequency controller of a generator at bus 30 (labled as `Gen0`) at time t = 5 secs. In `Sc1`, the bias parameter, b_{Gen0}, is a random variable such that yields in 10 simulations (K = 10) where b_{Gen0} for each simulation is sampled from the Gaussian distribution, $\mathcal{N}(0.01, 0.005)$ defined in `Sc1` attack specification (Line 9). The biasing attack with $b_{Gen0} \geq 0.01$ raises the frequency of the system to an abnormal range as shown in Figure 5a. The scaling attack (with parameter γ_{Gen0} = 2.1) doesn't change the system's frequency, but can increase the cost of generation output, indicated by the reduction germination control signal $\mu 0$ as shown in Figures 5b and 5c.

The scenario, `Sc3` in Listing 2 is related to a device attack in the protection system. The scenario, `Sc3` compromises a time-delayed-2 or zone 3 element of the distance relay attached to a branch between buses 15 and 16 with spurious attack at time t = 16.00 secs. As a result, the input port, f_{sp} of the corresponding state machine is set to `true` and the state machine jumps from `IDLE` to `ATTACK-WAIT` at t=16.00. The state machine stays in the `ATTACk-WAIT` state for *delay* = 1.44 secs (zone 3 wait time). At t=17.44 secs, the condition on the outgoing transition, $timeout(delay)$ evaluates to `true` and the state machine moves to `TRIPPED` while setting the output port *open*



(a) Biasing attack: System Frequency **(b) Scaling attack: System Frequency** **(c) Scaling attack: Generator control**

Figure 5: Effect of biasing and scaling attack

to `true`. At t=17.44 secs, the connected breaker acknowledges the change in the input port *open* and moves to `OPENING` . The breaker automaton stays in `OPENING` for tto = 0.048 secs and at t=17.488 secs transitions to `OPEN`.

5 CONCLUSIONS

In this paper, we presented a comprehensive tool-chain for modeling and simulating attacks in the modern power system. We described the core components of the tool-chain, 1) a domain specific modeling language to define configuration parameters and attack scenarios. 2) Python based simulator that is capable of simulating continuous dynamics of components in physical and control layer as well as discrete behavior of protection system. In the end, we demonstrated the simulation of multiple attack scenarios, involving data and device attacks on a controller and a numerical relay in IEEE 39 Bus System. The current implementation only supports two types of protection functions, over-current and distance. However, in future we would like to extend the simulation engine to include other relay elements such as differential protection etc. We also plan to integrate a discrete event network simulator to allow user to analyze the effects of network availability attacks.

ACKNOWLEDGMENTS

This work is funded in part by the National Science Foundation under the award numbers 1329803 and 1840052.

REFERENCES

[1] [n.d.]. National SCADA Test Bed | Department of Energy. https://www.energy.gov/oe/technology-development/energy-delivery-systems-cybersecurity/national-scada-test-bed

[2] [n.d.]. PowerFactory - DIgSILENT. https://www.digsilent.de/en/powerfactory.html

[3] [n.d.]. PowerWorld: The visual approach to electric power systems. https://www.powerworld.com/

[4] T. Athay, R. Podmore, and S. Virmani. 1979. A Practical Method for the Direct Analysis of Transient Stability. *IEEE Transactions on Power Apparatus and Systems* PAS-98, 2 (1979), 573–584. https://doi.org/10.1109/TPAS.1979.319407

[5] David C. Bergman, Dong Jin, David M. Nicol, and Tim Yardley. 2009. The Virtual Power System Testbed and Inter-Testbed Integration. In *Proceedings of the 2nd Conference on Cyber Security Experimentation and Test* (Montreal, Canada) *(CSET'09)*. USENIX Association, USA, 5.

[6] Xinjie Chang. 1999. Network simulations with OPNET. In *WSC'99. 1999 Winter Simulation Conference Proceedings.'Simulation-A Bridge to the Future'(Cat. No. 99CH37038)*, Vol. 1. IEEE, 307–314.

[7] Ajay Chhokra. [n.d.]. Power Attack. https://github.com/chhokrad/power-attack

[8] A. Chhokra, A. Dubey, N. Mahadevan, and G. Karsai. 2015. A component-based approach for modeling failure propagations in power systems. In *2015 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems*. https://doi.org/10.1109/MSCPES.2015.7115412

[9] I. Dejanović, R. Vaderna, G. Milosavljević, and Ž. Vuković. 2017. TextX: A Python tool for Domain-Specific Languages implementation. *Knowledge-Based Systems* 115 (2017), 1–4. https://doi.org/10.1016/j.knosys.2016.10.023

[10] A. Hahn, A. Ashok, S. Sridhar, and M. Govindarasu. 2013. Cyber-Physical Security Testbeds: Architecture, Application, and Evaluation for Smart Grid. *IEEE Transactions on Smart Grid* (2013), 847–855. https://doi.org/10.1109/TSG.2012.2226919

[11] Saqib Hasan, Ajay Chhokra, Abhishek Dubey, Nagabhushan Mahadevan, Gabor Karsai, Rishabh Jain, and Srdjan Lukic. 2017. A simulation testbed for cascade analysis. In *2017 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*. IEEE, 1–5.

[12] S. Hasan, A. Dubey, A. Chhokra, N. Mahadevan, G. Karsai, and X. Koutsoukos. 2017. A modeling framework to integrate exogenous tools for identifying critical components in power systems. In *2017 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*. 1–6. https://doi.org/10.1109/MSCPES.2017.8064540

[13] Hannes Holm, Martin Karresand, Arne Vidström, and Erik Westring. 2015. A survey of industrial control system testbeds. In *Nordic Conference on Secure IT Systems*. Springer, 11–26.

[14] J. Hong, S. Wu, A. Stefanov, A. Fshosha, C. Liu, P. Gladyshev, and M. Govindarasu. 2011. An intrusion and defense testbed in a cyber-power system environment. In *2011 IEEE Power and Energy Society General Meeting*. 1–5. https://doi.org/10.1109/PES.2011.6039375

[15] M. Liljenstam, J. Liu, D. Nicol, Y. Yuan, G. Yan, and C. Grier. 2005. RINSE: the real-time immersive network simulation environment for network security exercises. In *Workshop on Principles of Advanced and Distributed Simulation (PADS'05)*. 119–128. https://doi.org/10.1109/PADS.2005.23

[16] Richard Lincoln. [n.d.]. PYPOWER. https://github.com/rwl/PYPOWER

[17] M. Mallouhi, Y. Al-Nashif, D. Cox, T. Chadaga, and S. Hariri. 2011. A testbed for analyzing security of SCADA control systems (TASSCS). In *ISGT 2011*. 1–7. https://doi.org/10.1109/ISGT.2011.5759169

[18] Michael McDonald, John Mulder, Bryan Richardson, Regis Cassidy, Adrian Chavez, Nicholas Pattengale, Guylaine Pollock, Jorge Urrea, Moses Schwartz, William Atkins, et al. 2010. Modeling and simulation for cyber-physical system security research, development and applications. *Sandia National Laboratories, Tech. Rep. Sandia Report SAND2010-0568* (2010).

[19] Federico Milano. 2005. An open source power system analysis toolbox. *IEEE Transactions on Power systems* 20, 3 (2005), 1199–1206.

[20] Sai Pushpak Nandanoori, Soumya Kundu, Seemita Pal, Khushbu Agarwal, and Sutanay Choudhury. 2020. Model-Agnostic Algorithm for Real-Time Attack Identification in Power Grid using Koopman Modes. *IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids*. (2020).

[21] C. Queiroz, A. Mahmood, and Z. Tari. 2011. SCADASim—A Framework for Building SCADA Simulations. *IEEE Transactions on Smart Grid* 2, 4 (2011), 589–597. https://doi.org/10.1109/TSG.2011.2162432

[22] Julius Susanto. [n.d.]. PYPOWER Dynamics. https://github.com/susantoj/PYPOWER-Dynamics

[23] Hao Tu, Yuhua Du, Hui Yu, Abhishek Dubey, Srdjan Lukic, and Gabor Karsai. 2019. Resilient information architecture platform for the smart grid (riaps): a novel open-source platform for microgrid control. *IEEE Transactions on Industrial Electronics* (2019).

[24] Joe Weiss. 2016. Aurora generator test. *Handbook of SCADA/Control Systems Security* (2016), 107.