IT-Security in smart grids: Defence strategies for Remote Terminal Units in SCADA networks with limited communication

Christof Duhme (421483), Marius Neumann (397664), Thomas Teodorowicz (421489)
Bachelor of Science Informatik
Westfälische Wilhelms-Universität Münster

***Abstract*—Power grids are getting rapidly more complex due to increasing quantity of Smart Home Automation, power delivery and even power production units like solar panels. To provide a secure and reliable power grid, intrusion detection systems have to be adapted to these new challenging circumstances. Due to the mentioned complexity the monitoring cannot be done on the top level but has to be moved to a local, decentralized level thus supervising just the local buses is a way to deal with the challenge. This approach will be presented as an implementation in a simulation written for the smart grid co-simulation framework mosaik.**

I. Introduction

Power grids are controlled through a central based Supervisory Control and Data Acquisition (SCADA) network [1]. It provides the functionality to monitor the system and send commands to remote units to control power flow. Normally, intrusion detection happens on a global level. A possibility is to analyse network traffic behaviour and create a flow whitelist based on the observations [2]. Approaches like this do not consider process-based information and due to rising complexity and non-linear power flow, control and intrusion detection are way to extensive to be done on a global level.

To tackle these problems, decentralising intrusion detection and control mechanisms that consider process-based information, can lead to a more stable, safer and securer network. To achieve this, Remote Terminal Units (RTU) are being located at key nodes where several branches run together. Therefore, the whole grid is controlled and stable if every RTU ensures stability at their own location. The downside of having several RTUs is that new attack possibilities arise, e.g. cyber-attacks used against Ukranian power grids [3]. To make RTUs more stable, a local intrusion detections system (IDS) is used at every RTU to enable the RTU to control the power lines, supervised by the RTU, based on the physical values read by the RTU's sensors. To prevent manipulated sensor information from leading to an insecure state, the IDS also validates sensor values with a behaviour specification based approach inspired by rules based on physical information as proposed in [1] and [4].

With the help of the smart grid co-simulation framework mosaik the performance of our IDS is tested by running several types of attack scenarios on several type of smart grids. The process of testing is made easier by our script interpreter, topology loader and improvements to the web visualisation provided by mosaik. The script interpreter enables us to create complex attack scenarios while the topology loader provides a convenient way to switch between different smart grid topologies.

This report contains the following sections. Section 2 introduces smart grid topologies for the simulation. Section 3 details which features we added to the simulation. Section 4 illustrates possible attack types on our smart grid simulation. Section 5 contains test results, their conclusion and ideas for future work.

II. The scenario

To simulate a smart grid with mosaik, a topology is needed. Several topologies are introduced that can be used to test the IDS efficiency in various situations.

A. *Structure of typologies*

A topology is composed of a grid and any number of RTUs attached to the grid. A grid is composed of a transformer, nodes and branches. The branches connect the transformer to a node from the grid, the nodes among themselves and houses as well as photovoltaic panels to nodes. Houses and photovoltaics (PV) are connected to most but not all nodes. A RTU can be placed at every node. The RTUs monitor the sensors of the nodes they are placed at and also the sensors of the branches connected to the nodes. Furthermore, RTUs are capable of closing and opening the switch of a branch which disables and enables the flow of electricity for the branch respectively.

B. *Different topologies*

We have 7 topologies that all have something unique to them. They are used in combination with attack scenarios to generate data which provides information of the performance of our IDS. It can also be used to fine tune the IDS's sensitivity. For our default topology we based our grid on a real life grid found in [1] and added RTUs to the 4 most central nodes. This way our IDS can monitor the most crucial points of our grid. The default topology has one parallel branch which is closed by default and can be opened and closed by the local RTU based on certain cut-off values. Topology 1a (fig. 1) is a clone of our default topology but it includes several RTUs that all supervise a node that posses parallel branches. This topology is used to test which parallel branches are needed and why

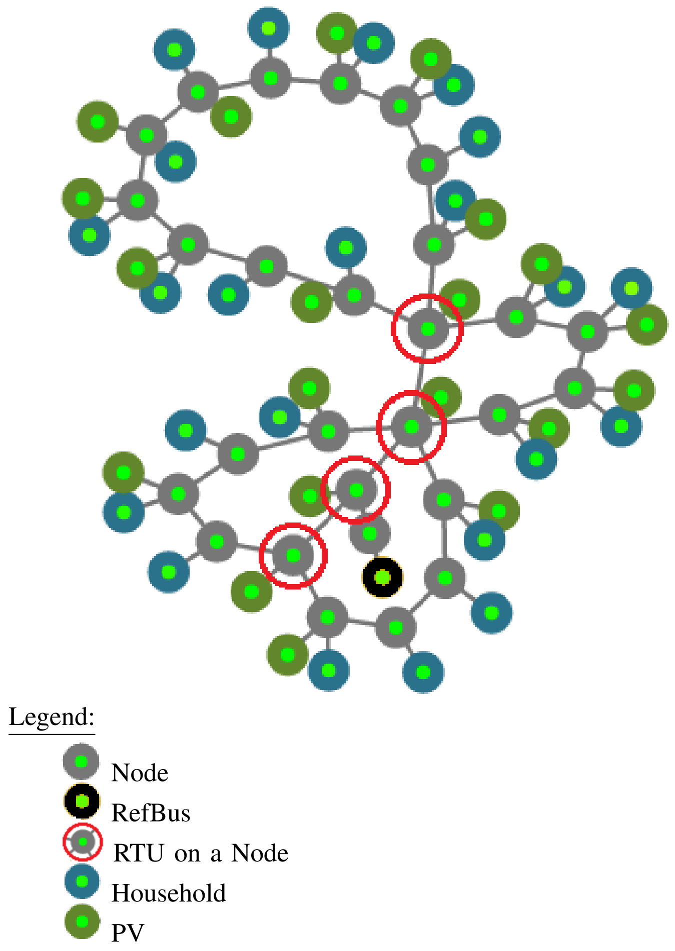


Figure 1: Topology 1 and 1a

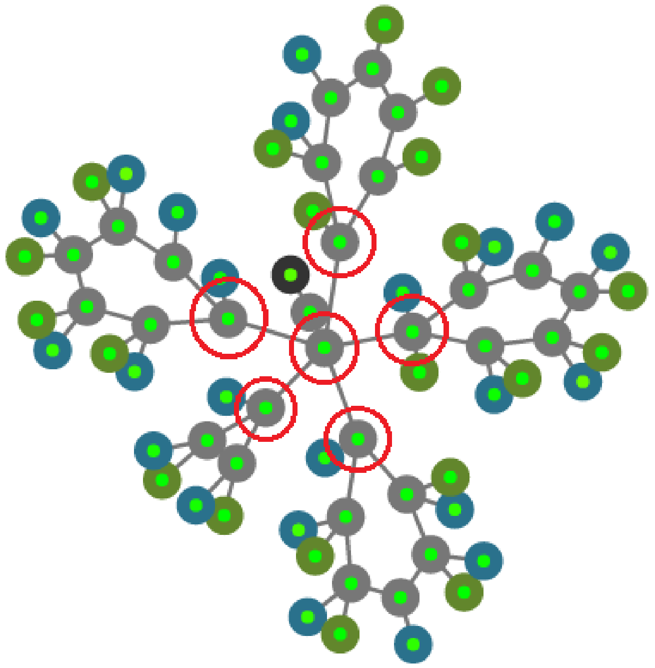


Figure 2: Topology 2

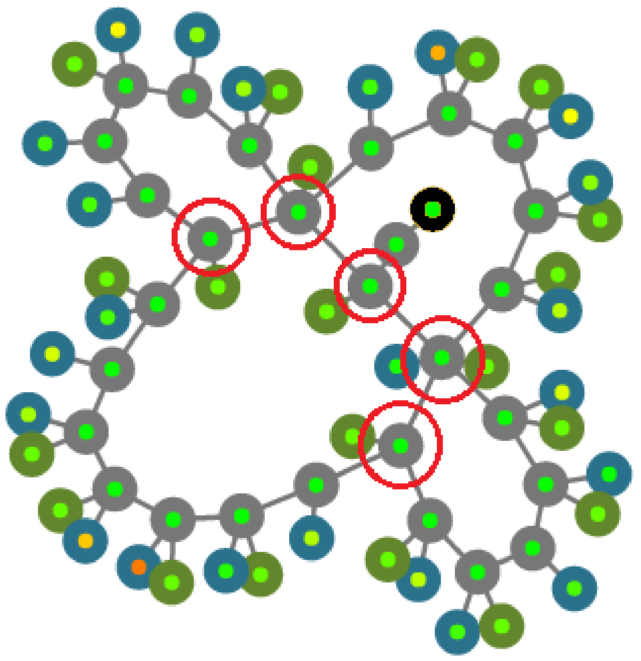


Figure 3: Topology 3 and 3a

they are needed or if parallel branches are needed at all.
Topology 2 (fig. 2) has a central node and 5 rings of nodes which are connected to the central node by branches. The central node is connected to the transformer. The RTUs are located at the central node and the nodes which are connected with the central node. The purpose of this topology is to test whether we need all 6 RTUs or if we can achieve comparable security by only using one RTU at the central node.
Topology 3 (fig. 3) is composed of a ring of nodes from which one is connected to the transformer and 4 semicircles. The RTUs are located at the nodes which connect the semicircles with the ring and one at the node which is connected to the transformer. We want to check if our attacks lead to drastically different result when we only change a small part of the topology. Topology 3a is used for similar testing purposes as topology 1a.
Topology 4 (fig. 4) is shaped like a string with one RTU at the node which is connected to the transformer, one RTU at the very end of the string and one RTU at the part of the string where there is no other RTU at the end. The purpose of this topology is to test the communication capabilities of our RTUs and if our RTU would activate the switch when there

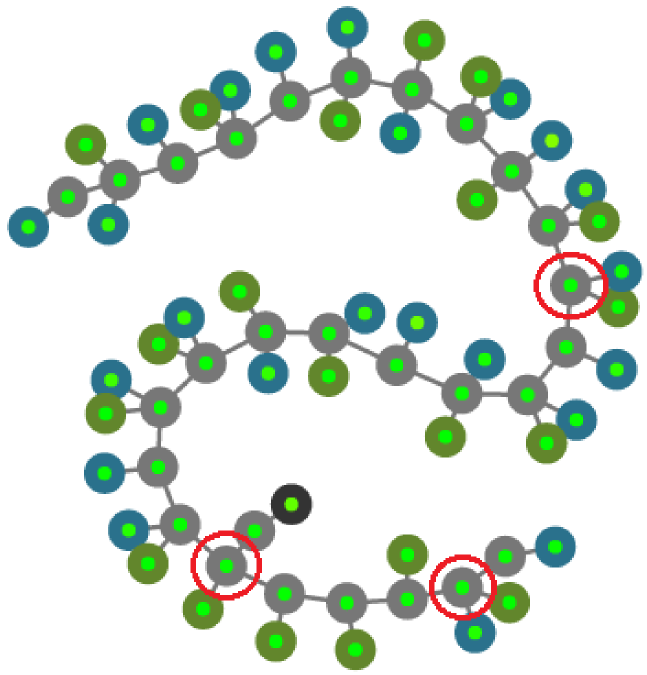


Figure 4: Topology 4

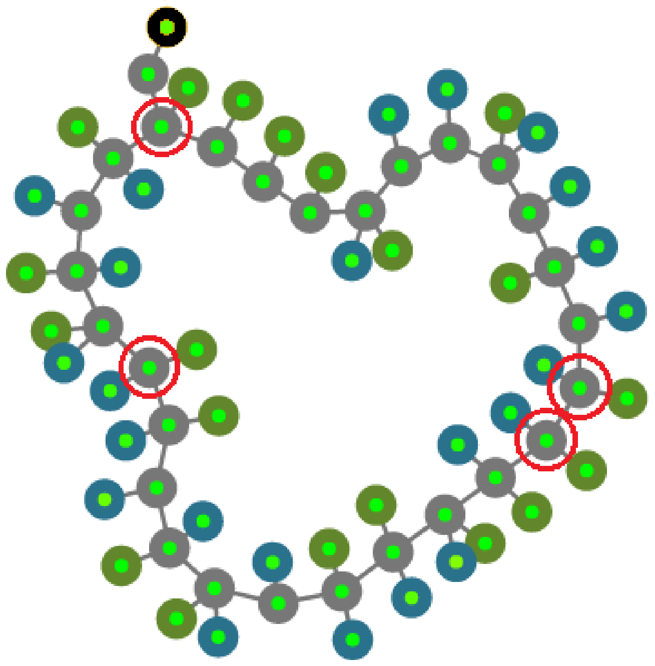


Figure 5: Topology 5

is no RTU after this one even if this would mean that a part of or grid with multiple households is left without power.
Topology 5 (fig. 5) is shaped like a ring and has a RTU at the node connecting the ring to the transformer and multiple RTUs on the ring.The purpose of this topology is to further test the communication capabilities of our RTUs. If one RTU decides to activate a switch it has to be ensured that no other RTU on the ring also activates a switch because then a part of our grid with multiple households would be left without power.

III. Implementation in mosaik

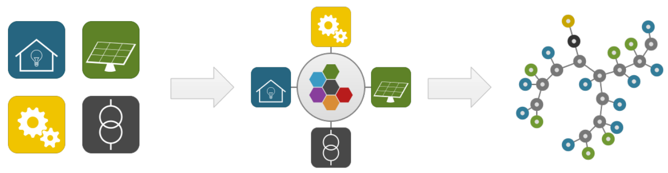


Figure 6: Schematic of mosaik's functionality connecting separate simulators to work together. [5]

Mosaik is a co-simulation framework written in Python and published open source by OFFIS under GNU LGPL [5].
It provides an API for simulators to implement and then handles the data transfer between them and simulation execution order.
Figure 6 is a simple example and was the base scenario for our work. There are simulators for households (blue), photovoltaics (green), a load flow analysis tool (grey) and a monitoring and analysis tool (yellow). When the simulation is run, the household and PV simulators calculate one step of data. mosaik collects the values and passes them to the flow analysis tool which performs one simulation step. After that mosaik sends the data to the motoring and analysis tool. From there on, these steps are repeated until the simulation is finished.
To manage these interactions, mosaik provides the following components:

1) The *mosaik Sim API* handles the communication between the simulators and mosaik.
2) The *Scenario API* is for defining the simulation scenario.
3) The *Simulator Manager* manages the simulator processes and their communication.
4) mosaik's *simulator* itself is based on the event-discrete simulation library SimPy to execute the scenario simulation.

A. Overview

The diagram 7 shows the overall structure of the simulation and how the individual packages and classes work together.
The following features were implemented by us. A topology loader for easier switching between scenarios, simulations for every RTU that can communicate with each other, an IDS that is used by the RTUs, improvements to the web visualisation for easier scenario testing, tools for trusted operators and a hacker

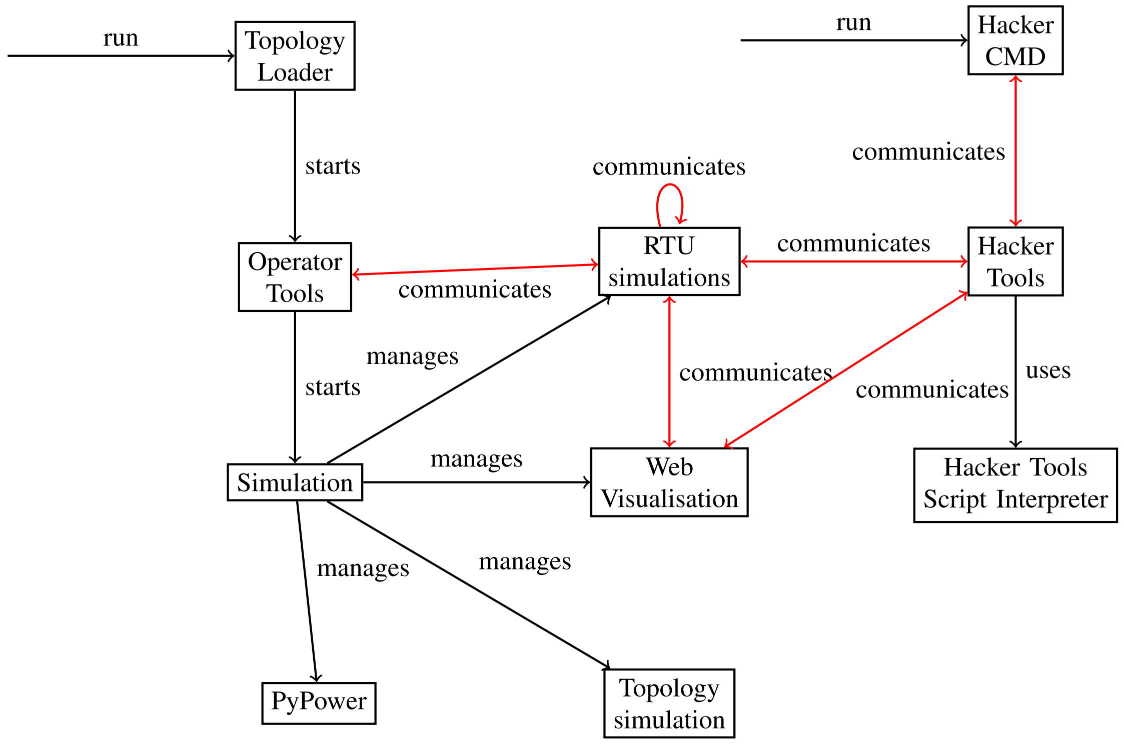


Figure 7: The diagram shows an overview over the structure of the simulation and the communication and management of the subsimulations and external packages.

client to either manually carry out attacks or to execute attack scripts. These features are used for either the simulation itself or for tests concerning the IDS safety.
When running the simulation, the Topology Loader is started first. It provides a GUI for choosing a topology and configuring some simulation parameter. After that the Operator Tools start a GUI for the user to monitor RTU messages and resetting them. It also starts the simulation itself. The simulation persists of the RTU and topology simulation, PyPower for the physics calculations and the Web Visualisation.
When the Hacker CMD is run, it starts the Hacker Tools which handles the communication with the RTU simulation and the Web Visualisation. For executing attack scripts, the Hacker Tools use the Hacker Tools Script Interpreter for translation of the commands to function calls.

B. *Topology loader*

Working with different topologies to test the defence algorithms can be exhausting and annoying because every time either multiple files or several paths in the code had to be changed. As a quality of life improvement the topology loader was introduced. It needs every topology to have a configuration file that declares all needed files and directories. There is also the possibility to choose the default settings so large standard files like the PV production table do not need to be copied several times. This prevents the overall size of the data from blowing up.
The loader also provides a graphic user interface (GUI) to select a topology. It also shows an image of the topology layout to make the selection process easier.

C. *RTU simulation*

There are multiple RTUs in the grid so the simulation has to simulate all of them. Therefore, more than one RTU simulation has to be started. mosaik gives a parameter option for this but due to a hard coded 1 in the source code, this possibility cannot be used. So there can only be one RTU simulation. To still simulate a various number of RTUs in a power grid, the RTU simulation class is used as a wrapper for the real simulation. It creates a new thread for each RTU and lets them run independently. It also forwards the simulation data to them in every simulation step.

D. *Intrusion Detection System*

The IDS is based on the requirements and physical laws suggested in [1].
RTUs use R2 and R4 as defined in [1] to regulate the smart grid state of the branches monitored by itself. R2 is the rule that power lines have a max current that is not to be exceeded and R4 requires that if the current of a branch exceeds a cut-off current a second branch, if existing, is opened. As we do not know if commands are trusted, commands are prevented from being executed if R2 or R4 are not violated. To validate sensor values we use a behaviour specification based IDS as suggested in [6] as it can deal with unknown attacks and yields a low false negative rate. Our basis are internal trust values (ITV) assigned to each sensor. If an

ITV surpasses a cut-off value we mark a sensor as untrusted. The RTU broadcasts to other RTUs if a sensor was marked as untrustworthy. Untrusted sensors will be ignored until an operator resets the trust. Inspired by [4] we check for voltage angle and voltage magnitude at nodes, current at lines that the margin of values of two successive steps do not exceed a cut-off value. If the cut-off value is exceeded we increase the ITV to it's cut-off value. As advised in [4] the cut-off principle is implemented for the voltage angle margin between adjacent RTUs. As suggested in [4] warnings and great-warnings are given if a margin reaches a percentage of the cut-off value. Warnings and great-warnings increase the ITV in amounts smaller than the cut-off value. This prevents single outliers to lead to marking sensors as untrustworthy. Warnings allow several, smaller outliers while great-warnings allow for less, bigger outliers. To increase adaptability to different smart grids cut-off values and the standard increases to the ITV are easily modifiable. In combination with the ITV, RTUs check R1 and P1 as defined in [1] to validate sensor data. If R1 or P1 are breached by a sensor it's ITV are increased. R1 demands that the voltage on all lines stays within the boundaries 230V +-10% in low voltage areas. In our simulation we used 10000V as we simulate a middle voltage area.

A notable point about P1, known as Kirchhoff's Law, is the way how it is calculated. For example, if there is a node with four branches a, b, c and d connected, the formula for Kirchhoff's Law with I_i as the current on branch i looks like this presuming the combination of branches is the valid one:

$$(I_a + I_b) - (I_c + I_d) = 0 \tag{1}$$

To change the values without violating Kirchhoff's Law, this constraint with δ indicating the value change between two discrete points in time has to be met:

$$\delta I_a + \delta I_b = \delta I_c + \delta I_d \tag{2}$$

It assures that the equation 1 stays true. Because we do not know the flow direction of the current, the only way to get an equation like 1 is by trial and error. This is done by checking which combination of signs yields the sum closest to zero. If there is no value close to zero, P1 is violated. The reason why we check for the sum to be only close to zero and not equal is that we are missing the data of PVs and households to consider in the calculations.

For every branch exists a sensor which means that several sensors are on the same node and belong to the same RTU. RTUs check if every sensor at a node reads the same value. If they do not share the same value we mark every sensor at the node as untrusted. As suggested by Prof. Remke, we also added the possibility to use majority rule instead. Our system yields a higher false positive rate if a minority of sensors is attacked while the majority rule yields a higher false negative rate if the majority of sensors is attacked.

Considering this and the adaptability for different smart grids, our IDS design allows to easily switch off or swap out features.

E. WebVis

The web visualisation was previously handled by an executable but was changed to python script for easier adjustments.

When being started the web server provides the HTML main page for the browser and through that the JS script which handles the network visualisation by the 3d.js framework and updating the data given by the simulation over the web server. The server also manages the visual representation of hacker attacks and RTU interventions. The data for this feature is sent to the JS script along with the simulation data. These visualisations are helpful to understand the extent of an attack and the defence behaviour of the RTUs.

F. Hacker Tools

To simulate attacks on the grid, it is assumed that the hacker has already gained access to the network e.g. by bypassing a firewall or infecting a PC with a defected USB key.

1) Hacker Tools CMD: For I/O a simple command line shell is used with a limited set of commands. The hacker can connect to RTUs, read the current state of specific branches and send malicious data to override real sensor data with the provided commands. Furthermore, switches on the branches can be turned on and off resulting in a real change of the grid in contrast to the sensor data manipulation which just misleads the IDS.

The data transfer works over the TCP protocol. The server on the RTUs provide a TCP server with a distinct IP address onto which the Hacker Tools can connect with a TCP client.

The link to the web visualisation works in the same way over a TCP based server-client connection.

These clients each run in a separate thread to guarantee responsiveness and let the main thread run even when waiting for a response.

2) Script Interpreter: To automate complex attacks, a script interpreter was added for easy and even intelligent attacks. The used script language is self-developed and simple to learn for ease of use.

The interpreter can handle all normal Hacker Tools commands and following constructs and functions:

- set and get for variables
- if - then - else
- for-loop
  - over a range of values
  - over an array
- random-function
  - number in range
  - element from array
- array length function
- wait function (waits a given amount of seconds)

It can calculate $+$, $-$, $*$, / and mod. The comparators $==$, $>$, $<$, $>=$, $<=$, $!=$ can be used with logic functions and and or.

To make reviewing the script easier, it allows to insert line comments with $\#$ as a token.

```
for i in 0 to 1000
  # choose random RTU
  connect random get listservers False
  # iterate through all branches
  for branch in get listbranches
    # per cent to modify sensordata
    set a random 25 300
    # get sensor value of current branch
    set v get sensordata of getstate branch
    False
    if random 0 1 > 0
      setsensor branch, v*(1+a/100)
    else
      setsensor branch, v*(a/100)
    ifEnd
    wait 0.5
  forEnd
forEnd
```

Figure 8: Example attack script for a random style attack.

Figure 8 is an example for an attack script. It executes a random attack on the grid by connecting to a random RTU, iterating through all its branches and changing the sensor values by a randomly chosen percentage. Additionally, the decision whether the change is positive or negative is also randomized.

```
for i in 0 to 1000
  for server in get listservers
    connect server
    for branch in get listbranches
      set v get sensordata of getstate
    branch False
      setsensor branch, v*1.01
      wait 0.5
    forEnd
  forEnd
forEnd
```

Figure 9: Example attack script for increasing all sensor values continuously.

The attack script in figure 9 iterates through all RTU servers and their branches 1000 times. It then increases the branches sensor value by one per cent.

G. Operator Tools

The Operator Tools provide a simple GUI for hard resetting the local IDSs by making every sensor trusted. It also gives an overview over detected attacks whether true or false positives. It shows all notifications when an RTU's defence mechanism gets triggered.

IV. Attack scenarios

When attacking the grid, the hacker has several options on how to structure their attack. They range from simple straight forward to complex and IDS targeted ones.

A. Deterministic attacks

Deterministic attacks are the easiest to implement because we can list all RTUs in the simulation with the help of Hacker Tools, connect to them and get the data read by their sensors. This enables us to start specific attacks on a single or more RTUs, which can alter specific physical values read by specific sensors of these RTUs or open/close specific switches. For example one could connect to a RTU via the TCP connection and increase a physical value, read by a sensor of this RTU, by a specific amount.

B. Random attacks

Random attacks try to abuse that they have no pattern, which is not a problem for our IDS as it is not based on pattern recognition. For example we could connect to a random RTU, add a random number to a physical value read by a random amount of sensors of this RTU and then repeat the whole process.

C. Defence mechanism specialized attacks

Attacks that are engineered to bypass certain defence mechanisms. These attacks require the attacker to know which defensive mechanism are implemented to secure the system.

1) For example if the attacker wants to bypass the Kirchhoff's law he could use an attack on the lines of connecting to a certain RTU, increase the voltage angle value of half of the sensors and decrease the other sensor's voltage angle values by the same amount. In theory these attacks easily circumvent our IDS but they not only require knowledge about how our IDS works but also knowledge of the specific cut-off values used.
2) Another approach is to increase the sensor data just by a fraction of its value. This prevents the IDS from triggering its control mechanism because the change of the sensor's value seems to be a normal gradient. When the value reaches the maximal current, the branch will be turned off with no real world reason to do so. This way, drastic changes can be made to the grid without the IDS interfering.

All IDS defence mechanisms have to be considered to make a successful attack, as different mechanisms cover each other. E.g. you cannot trick the gradient check without also tricking Kirchhoff's Law.

D. Attack to kill the IDS

Instead of trying to trick the IDS we can also prevent the IDS from checking for an attack. To make this possible, a first attack is run, that will definitely be detected so the IDS will mark those nodes as unsafe. Due to this, these nodes will be ignored by the IDS. This impairs the IDS's control functions, meaning that power lines are not turned off if they exceed their maximally allowed current nor will parallel branches be utilized to reduce their main branches load. At this point we cannot guarantee the grids stability.

V. Discussion

To discuss the work, there is an evaluation at first. After that, we draw a conclusion to our findings and take a look at future work to be done afterwards.

A. Evaluation

To evaluate our IDS we compare the effect of specific and randomized attacks. They are tested with topology 1 (fig. 1).

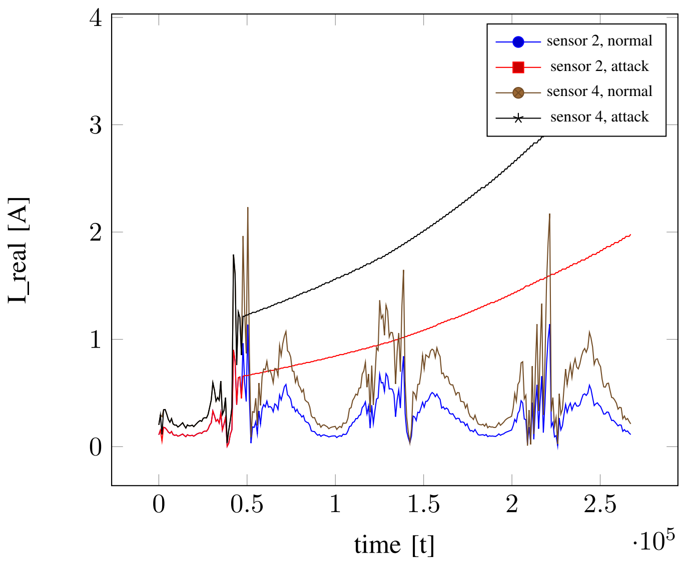


Figure 10: The plot shows I_real value against simulation time. For sensor 2 and sensor 4 the values are displayed for both, a normal simulation execution and a simulation where a specific attack was made.

1) Specific attack: This attack increases the current of $branch_2$ and $branch_4$ by 1 per cent each simulation step which is similar to the attack in figure 9. In figure 10 this continuous increase can be seen as a steady rising of the black and red graph. The attack started at simulation time 47700 and the IDS triggered great warnings for Kirchhoff's Law immediately. After 3 simulation steps, the attacked sensors warning values reach the unsafe status, meaning the IDS detected the attack. As the attack was detected by the violation of Kirchhoff's Law all sensors on the attacked node were marked as unsafe. Therefore we have two false positives in this scenario.
Figure 11 shows an example IDS output when an attack is detected.

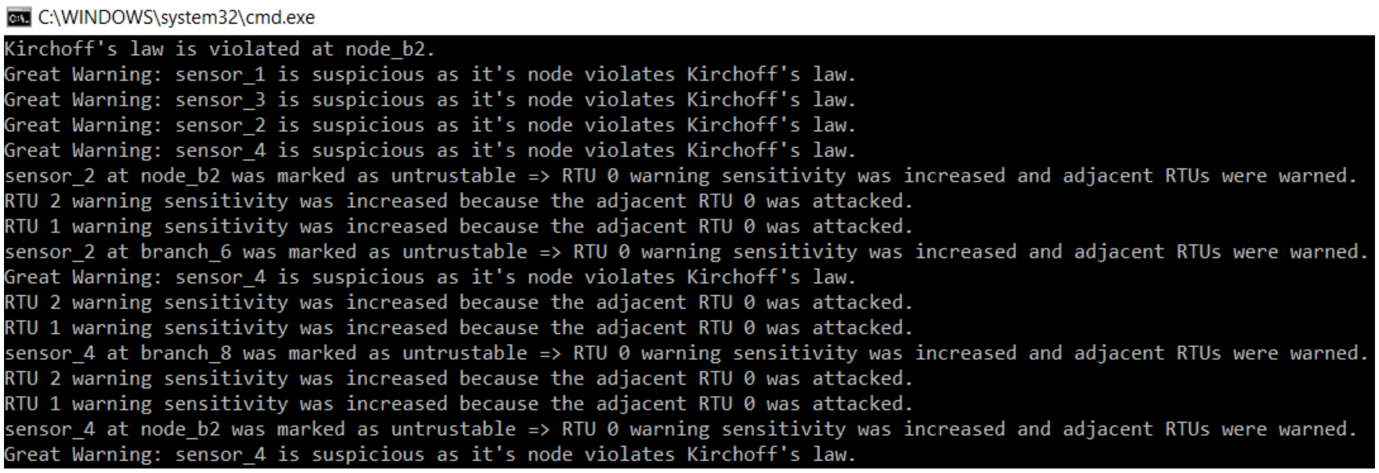
2) Random attack: This attack is the one presented in (fig. 8. It randomly picks a RTU, then randomly picks a sensor and then changes the value read by the sensor by a random percentage either positive or negative in the range between 25 and 75. Due to this the graphs in the plot (fig. 12) look like step functions. It is notable that one branch does not appear to have been chosen for an attack during this run of the simulation. Before even drastic jumps can trigger gradient checks, Kirchhoff's Law detects the intrusion.

B. Conclusion

Due to the severe consequences that attacks on smart grids can have, the security is the top priority with current approaches not being comprehensive enough to cover every potential attack possibility. This work tests the performance of a local system that acts based on locally attainable physical information.
We discovered that it is very complicated to bypass Kirchhoff's Law. Like our IDS an attacker cannot separate in- and output branches of a node. Due to this, the attacker has to do the same calculations as the IDS to determinate current flow directions. Only then he can successfully evade Kirchhoff's Law. This makes Kirchoff's Law a strong point of our IDS.
We also noted that the cut-off values used in our attack scenarios are too high to trigger an unsafe marking as Kirchhoff's Law is too accurate.
As a conclusion we consider neither deterministic nor random attacks to be sufficient enough to harm the grid without the IDS noticing.
Especially Kirchhoff's Law prevents these attack types by definition.
As Kirchhoff's Law is more sensitive than expected and by declaring all sensors at a node unsafe when a violation is detected, we get many false positives. Therefore, we conclude that using the majority rule feature instead we can reduce the amount of false positives while not increasing false negatives except for attacks targeted against this defence mechanism.
Test results show that using a local system that acts on process-based information is able to consistently detect several types of attacks while having a low amount of false positives. However, attacks that abuse knowledge of the behaviour specifications are expected to be successful.

C. Future work

Although a lot of work was done in this project, there are still some things to consider for improvement and more functionality.
At the moment, all nodes in a grid are treated the same way. But the first node after the reference bus is exposed to a much greater current than all the other ones. A special treatment for nodes like that would be great addition, so the sensitivity for normal nodes can be increased. On a similar note the current generally decreases the further away a node is from the reference bus which could be used to fine tune more distant RTUs with smaller cut-off values.
When handling commandos we use a very restricted system. The first problem is that suspicious commands are not prevented from being executed but we only execute a reverse command in the next step. Ideally we would prevent suspicious commands from being executed in the first place. Another problem is that commands are not checked at all if an untrusted sensor has to be checked to validate the command. Handling this problem in a more complex way, e.g. something like a pattern recognition to filter out greater chunks of commands that are not harmful alone but all together, would increase the system's safety.



```
C:\WINDOWS\system32\cmd.exe
Kirchoff's law is violated at node_b2.
Great Warning: sensor_1 is suspicious as it's node violates Kirchoff's law.
Great Warning: sensor_3 is suspicious as it's node violates Kirchoff's law.
Great Warning: sensor_2 is suspicious as it's node violates Kirchoff's law.
Great Warning: sensor_4 is suspicious as it's node violates Kirchoff's law.
sensor_2 at node_b2 was marked as untrustable => RTU 0 warning sensitivity was increased and adjacent RTUs were warned.
RTU 2 warning sensitivity was increased because the adjacent RTU 0 was attacked.
RTU 1 warning sensitivity was increased because the adjacent RTU 0 was attacked.
sensor_2 at branch_6 was marked as untrustable => RTU 0 warning sensitivity was increased and adjacent RTUs were warned.
Great Warning: sensor_4 is suspicious as it's node violates Kirchoff's law.
RTU 2 warning sensitivity was increased because the adjacent RTU 0 was attacked.
RTU 1 warning sensitivity was increased because the adjacent RTU 0 was attacked.
sensor_4 at branch_8 was marked as untrustable => RTU 0 warning sensitivity was increased and adjacent RTUs were warned.
RTU 2 warning sensitivity was increased because the adjacent RTU 0 was attacked.
RTU 1 warning sensitivity was increased because the adjacent RTU 0 was attacked.
sensor_4 at node_b2 was marked as untrustable => RTU 0 warning sensitivity was increased and adjacent RTUs were warned.
Great Warning: sensor_4 is suspicious as it's node violates Kirchoff's law.
```

Figure 11: CMD output for the specific attack

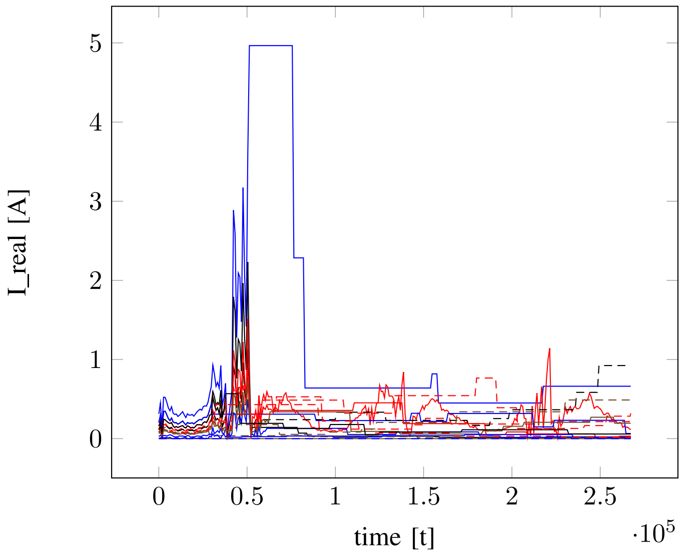


Figure 12: The plot shows I_real value against simulation time. A graph is shown for every sensors value during a simulation where a random attack was executed.

Currently a RTU cannot get the current readings of PVs and households. This results in rather inaccurate calculations such as the check for Kirchhoff's Law, for example. Getting these values would drastically improve the simulation because there would be no need to round and estimate results.

At the moment our IDS is rather easily tricked by gradually increasing and decreasing current values read by branch sensors on a single node in a way that Kirchhoff's Law is not violated at the node. This leads to closing of power lines as soon as they exceed the maximally allowed current. Therefore, it would be interesting to see if pattern based detection methods would be a good extension for our current IDS.

Additionally, when a real error occurs on a node that was marked as unsafe, this is neither detected nor handled. For sensors on a node this may be able to be fixed with a system, that automatically restores trust for a sensor when it's read data values are matching the other node's sensor's readings for a certain amount of time.

In [1] other requirements and physical laws were defined which we either could not implement, did not need to implement or did not have the time to implement. P2 and P4 are restricted by the simulation. Lines that are turned of to not have zero current within the simulation so P2 cannot be checked. To realize P4 we need to know the voltage from power lines. As we could not attain this value we did not implement P4. R3 cannot be checked because we are missing the source power production values. This is due to the fact that the simulation does not contain these but just equals power loads out at the reference bus. P5 and P6 concerned transformers which are not the focus of our group. P3 could be implemented if we get the specific branch data, e.g. it's length, and check if the two power values given at a branch are the same after considering the resistances.

As for the Hacker Tools Script Interpreter, a great improvement would be an error detection system that handles syntactical mistakes so the whole Hacker Tools shell does not crash every time.

References

[1] J. J. Chromik, A. Remke, and B. R. Haverkort, "Improving SCADA security of a local process with a power grid model," in *4th International Symposium for ICS & SCADA Cyber Security Research, ICS-CSR 2016*, ser. Electronic Workshops in Computing. BCS Learning & Development Ltd., August 2016. [Online]. Available: http://doc.utwente.nl/101363/

[2] R. R. R. Barbosa, R. Sadrea, and A. Prasb, "Flow whitelisting in scada networks," in *Vol. 6, Issues 3-4*, ser. International Journal of Critical Infrastructure Protection. ScienceDirect, August 2013. [Online]. Available: http://dx.doi.org/10.1016/j.ijcip.2013.08.003

[3] I. C. S. C. E. R. Team. (2017, february) Cyber-attack against ukrainian critical infrastructure. [Online]. Available: https://ics-cert.us-cert.gov/alerts/IR-ALERT-H-16-056-01

[4] H. Bao, R. Lu, and R. Deng, "BLITHE: Behavior rule-based insider threat detection for smart grid," in *Vol. 3, No. 2*, ser. IEEE internet of things journal. IEEE, April 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7163500/

[5] OFFIS. (2017, february) mosaik documentation. [Online]. Available: http://mosaik.readthedocs.io/en/latest/overview.html

[6] J. J. Chromik, “Cyber threats to smart grid,” in *DACS. Design and Analysis of Communication systems*. University of Twente, November 2016.