Framework for SCADA Cyber-attack Dataset Creation

Nicholas R. Rodofile[1], Kenneth Radke[2], Ernest Foo[3]
Information Security Discipline
Queensland University of Technology
Brisbane, Australia
{n.rodofile[1], k.radke[2], e.foo[3]}@qut.edu.au

ABSTRACT

Cyber-security research and development for SCADA is being inhibited by the lack of available SCADA attack datasets. This paper presents a modular dataset generation framework for SCADA cyber-attacks, to aid the development of attack datasets. The presented framework is based on requirements derived from related prior research, and is applicable to any standardised or proprietary SCADA protocol. We instantiate our framework and validate the requirements using a Python implementation. This paper provides experiments of the framework's usage on a state-of-the-art DNP3 critical infrastructure test-bed, thus proving framework's ability to generate SCADA cyber-attack datasets.

Keywords: Framework, Cyber-attacks, Injection, Man-in-the-middle, Critical Infrastructure, SCADA, DNP3, Datasets.

1. INTRODUCTION

Supervisory Control and Data Acquisition (SCADA) systems are utilised in transport, water treatment, and energy generation and distribution. SCADA systems are also utilised in manufacturing industries, providing critical automated services to provide consumer products [9]. In recent years, attacks such as Stuxnet, Black Energy, and the Ukraine substation cyber-attacks have shown the increasing threats to vulnerabilities in SCADA systems [15].

Researchers have identified the development of intrusion detection systems (IDSs) as an effective method for cyber-attack detection on SCADA [9]. This is due to the IDS's nature of analysing system behaviours and providing minimal impact on critical messages used for system operations. The research and development of IDSs, require the use of training and validation datasets containing anomalies and attacks. This enables developers to evaluate the performance of data mining and machine learning algorithms for new and intelligent IDSs. However, due to privacy issues, there is a lack of available SCADA datasets for SCADA IDS research [9]. This would require researchers to create their own datasets by stimulating their test-bed with attacks. However, there are limited methods and guidelines for generating anomalous behaviour on SCADA test-beds, thus generating data to aid the development of next-generation IDSs proves itself difficult [13]. Data generation methods include the stimulation of SCADA systems with attacks, resulting with the generation of attack datasets [9, 13]. To succeed with the SCADA attack generation process, a modular SCADA attack tool allowing for the ability to generate infinite amounts of anomalous behaviour. Without the availability of various SCADA attack data generation tools, the potential for new and intelligent methods of intrusion detection is limited. To enable the design of an appropriate SCADA attack generation framework, a set of attack generation requirement extracted from related work is necessary, thus allowing for the development of acceptable datasets for SCADA IDS. The presented framework and the requirements are not restricted to only Ethernet-based networks, but is modular to serial and wireless networks. The target *SCADA protocol* in the framework is interchangeable, thus allowing for the attack generation of standardised or propitiatory vendor protocols. The paper is organised as follows: In Section 2, we provide a review of the related work. In Section 3, we present the requirements for an attack generation framework based on the related work. In Section 4 we present the attack data generation framework and the required attack modules. In Section 5, we introduce the presented framework used to generate attack data. In addition, we provide a case study where we present the software implementation of our framework in DNP3, along with our findings as results. We provide a discussion of framework and it's implementation in Section 6 and conclude.

Contributions: From the analysis of previous work, this paper presents 10 requirements for the generation of anomalous traffic on SCADA networks. In addition, this paper presents a SCADA attack generation framework that satisfies the requirements, and provides a collection of algorithms aids the implementation of the presented framework. We verify the presented framework via case study involving a DNP3 implementation, and verify the framework implementation through experiments on our state-of-the-art SCADA DNP3 test-bed.

2. RELATED WORK

In order to identify the appropriate requirements for generating attack datasets, a review of work related to attacks, generating SCADA traffic, and the development of datasets is required. Therefore, we explore in this section the land-

ACSW '17, January 31-February 03, 2017, Geelong, Australia

DOI: http://dx.doi.org/10.1145/3014812.3014883

scape of SCADA attacks, review methods of generating traffic on SCADA networks, and finally assess previous methodologies for developing attack datasets.

2.1 SCADA Attack Taxonomies

East et al.[3], provides a DNP3 attack taxonomy, based on the DNP3 protocol stack: the data-link, transport, and application layer of DNP3, along with applicable attacks on each DNP3 layer. The taxonomy was based on the following threat categories: interruption, interception, modification and fabrication. East et al. does not provide a framework for generating attacks or anomalous behaviour, however, the attack concepts highlighted by East et al. are appropriate for deriving requirements for a SCADA attack dataset creation framework. Huitsing et al.[5] provides an attack taxonomy of the MODBUS SCADA protocol. Huitsing et al. identifies a combination of MODBUS over TCP/IP based attacks. With readily available off-the-shelf technology, MODBUS can be transported over TCP/IP or UDP/IP protocols, thus allowing SCADA based systems to be affected by IP-based attacks. Such attacks are further elaborated by Zhu et al.[17], providing extensive criticism of the lack of security features such as confidentiality and integrity of widely used SCADA protocols. This is emphasised through the identification of exploits against the Internet Protocol (IP) suite, thus expanding the attack landscape of widely used SCADA protocols. Drias et al.[2] asserts their SCADA attack landscape using a classification of threats, attacks, vulnerabilities and impacts (TAVI). The TAVI attack model provides several possible SCADA attack scenarios using the combination of MODBUS and DNP3 attacks proposed by East et al., and Huitsing et al.. Drias et al.'s attack scenarios focus on: probing, scanning, flooding, authentication, bypassing, spoofing, eavesdropping, misdirection, reading/copying, termination, execution, modification, and deletion. From the attack scenarios considered, there was emphasis on three targeted control network parties, *server*, *client* and *broadcasts*.

From the ideas provided in the SCADA attack taxonomies discussed, we are able to derive requirements in order to generate attack data to create SCADA attack datasets. To generate SCADA attacks over SCADA networks, we require a review of methods and tools used by other researchers, to further develop our SCADA attack generation requirements.

2.2 Traffic and Data Generation

Voyiatzis et al.[14] provides a study of MODBUS/TCP protocol implementations by developing and evaluating a MODBUS TCP fuzzing software tool (MTF). The requirements provided by Voyiatzis et al. are limited flooding and spoofing attacks for MODBUS only. Garitano et al.[4] presented a methodology to simulate Manufacturing Message Specification (MMS) traffic using a traffic generator algorithm implemented using the Python packet manipulation tool Scapy[1]. The work is only to provide a simulated test-bed for MMS without a working protocol stack. Scapy was also utilised by Lopes et al.[7], to stimulate Generic Object Oriented Substation Events (GOOSE) network architectures for substation, and analyse the performance and network flows. Unlike Garitano et al. and Lopes et al. who were using Scapy for generating traffic, Kobayashi et al.[6], provides an implementation of MODBUS to perform integrity tests using malformed packets to a MODBUS slave. Lopes et al.[7] show the use of packet manipulation to aid in the generation of traffic to stimulate network test-beds, but Kobayashi et al. shows the use of such tools to stimulate attacks against SCADA systems. The attack requirements presented by Voyiatzis et al. is relevant to data generation and is adapted for the presented requirements for flooding and spoofing attacks for generic SCADA protocols.

2.3 Datasets for Intrusion detection

Shiravi et al.[13], provides a guideline into benchmarking and generating what is considered as "normal behaviour" in corporate IT networks. The purpose of the process was to provide an environment that did not require anonymity during the data collection process. For the attack process, Shiravi et al. used the Metasploit software framework[2] which consists of various modules for network penetration testing. The authors used attack scenarios based on the Metasploit modules available based on vulnerable configurations in their test-bed. Shiravi et al. provided a set of guidelines which are useful for developing normal user traffic, Creech and Hu[1], provided an attack dataset that is to replace the outdated KDDcup99 attack dataset. Creech and Hu adapted methods similar to Shiravi et al., as they both utilised the Metasploit framework to conduct various contemporary wireless network-based attacks within a network test-bed. In both the works presented by Creech and Hu[1] and Shiravi et al.[13], there was no generic requirements for generating anomalous or malicious traffic. The authors had only resorted to the use of contemporary attacks that are available to specific applications, software services or operating systems. Morris and Gao[9] describes and presents four attack dataset for the SCADA protocol MODBUS. Morris and Gao generated 28 attacks on a MODBUS serial network test-bed. The dataset presented were of logs and did not provide network traffic. Morris and Gao produced four categories of attacks: reconnaissance, response injection, command injection, and denial-of-service. Morris and Gao provides a thorough description of each attack and it's implementation on C and C++ programs. However, there is no distinct framework provided for generating attacks or anomalies for SCADA protocols. Our presented requirements adapt the injection requirements presented by Morris and Gao, thus enabling us to provide an accurate framework for generating SCADA attacks for dataset creation.

By reviewing the ideas presented by related work, we were able to derive a set of requirements enabling us to design and implement a SCADA cyber-attack framework for attack dataset creation. We show our requirements in the following section.

3. ATTACK GENERATION REQUIREMENTS

After having reviewed the areas of attack taxonomies, data generation, and attack datasets in Section 2, we were able to derive 10 requirements to aid in the creation of SCADA cyber-attack datasets. We present the following ten requirements for the presented framework to satisfy:

R1. Able to *parse* SCADA protocol messages [6].

[1]Biondi, P. [2014], 'Scapy'. URL:http://www.secdev.org/projects/scapy/doc/build_dissect.html

[2]Rapid7. [2016], 'Metasploit'. URL:https://www.metasploit.com/

R2. Able to *replicate* the SCADA protocol stack [3, 14].

R3. Able to *sniff* local SCADA network traffic [2].

R4. *Inject* anomalous SCADA protocol messages into the network [9].

R5. *Modify* protocol message data in real-time [3].

R6. Provide a protocol *master service* for masquerading [14, 17, 6].

R7. Provide a protocol *slave service* for masquerading [17, 6].

R8. Provide SCADA *network discovery/reconnaissance* to target SCADA applications [9].

R9. Able to *replay* previous SCADA protocol messages [2, 14].

R10. Able to *flood* a SCADA service with anomalous messages [14, 17].

In the following section, we present the SCADA attack data creation framework. The framework incorporates several algorithms and procedures that demonstrates the ability to satisfy to the presented requirements.

4. SCADA ATTACK GENERATION FRAMEWORK

To satisfy the presented requirements in Section 3, we present a SCADA cyber-attack data generation framework. The framework, shown in Figure 1, four categories, *base network modules, SCADA modules, attack modules,* and *advance attack modules*. The *base network* modules are used to interface the framework with a networking medium i.e. Ethernet, wireless, or serial. The *base network* consist of: *socket, server,* and *client*. The *SCADA modules* are used to replicate the SCADA protocol in the target system. The SCADA modules allow for the framework's target SCADA protocol to be interchangeable, thus any major changes are to be made only among SCADA modules. The SCADA modules are: *Protocol Data Unit (PDU), spoofer, master* and *slave*. Each of these modules are extended, or are initialised as an instance, to create the following modules in the *attack* category: *slave masquerading, master masquerading, injection*, and *MITM*. The *attack* category allows for SCADA modules to extend to attacks based on the SCADA category. Only small adjustments are required to allow the attack modules to be adapted to the when a new protocol is added. The *advance attack* modules are used to provide specific attacks against the target SCADA protocols. The *advance attack* modules are: *Reconnaissance, slave flooding, MITM modification, MITM hijacking, injection replay, master replay*, and *master flooding*. In this section, we present each of the modules of the SCADA attack generation framework and illustrate it's use of the presented requirements.

4.1 PDU

The first requirement (R1) for developing the SCADA attack data generation framework, is to develop the structure of the target SCADA protocol's PDU, providing logic for network packet manipulation. The PDU module allows for an instance of a SCADA protocol's message based on the protocol's specification. The process of packet manipulation is essential for the operation of the entire framework, as it allows for the manipulation of the SCADA protocol, injecting a SCADA protocol message, and spoofing the SCADA protocol's stack. Thus allowing the data generation framework to be extend to a variety of SCADA attacks.

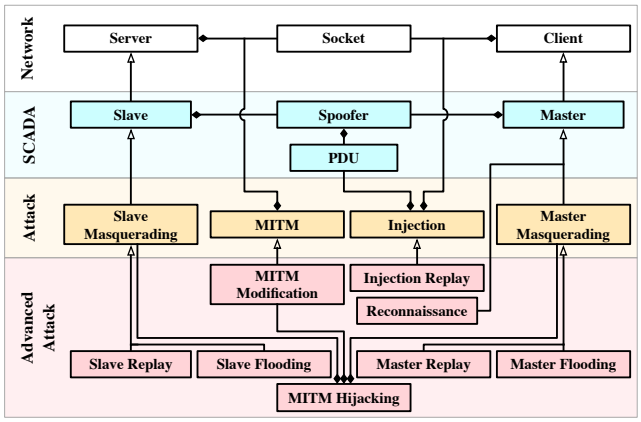

Figure 1: UML 2.5 Class Diagram of the presented SCADA cyber-attack data generation framework.

4.2 Spoofing

The ability to provide spoofing is necessary to satisfy requirement *R2*. The *Spoofer* module in the framework allows the attacker to imitate network stack logic of the target SCADA protocol. The *Spoofer* utilises the PDU module, in order to replicate the SCADA protocol's protocol stack. The *spoofer* module may need a database, or access to a file system, depending on the SCADA protocol's specification.

4.3 Injection

The *Injection* module is used to inject a series of message into the target system's connection, thus satisfying requirement R4. A usable algorithm for injection attacks is described in Algorithm 1. By using an instance of a *PDU* module, the injection module is able to craft one or many injectable PDUs, and insert the PDUs into the network, causing the manipulation of target SCADA devices. The *Injection* module will need to sniff an existing connection between the two targets, in order to track connection's sequencing information; a socket library can be used to sniff traffic, which can be passed to the injection module. As sniffing is required method for injection, the *Injection* module allows us to satisfy requirement R3. The *Injection* module is to craft the SCADA protocol's network transport layer, i.e. TCP/IP, to keep track of sequencing and correctness, this is shown in Sequence 8 - 19 Algorithm 1 using TCP/IP for illustrative purposes. Ideally the module requires access to the target connection via an Ethernet hub, port mirror on a switch, or the direct network interface on a target device. The injection module updates any sequencing fields that are required for the SCADA PDU. The *inject* procedure in Algorithm 1, is triggered by an additional function, condition, or user input, allowing for the injection process. The injectable packet's (*packet*) transport sequence has the length of the crafted PDU (*pdu*) and the malicious payload (*mal_pay*) append to it's sequence; allowing for the packet to be accepted by the target SCADA device when injected. Some proprietary SCADA protocols have their own transport layer, thus any injection module for that protocol must

have it's functionality catered for. The utility of the *Injection* module, is the possibility to extend to a variety of injection attacks against SCADA protocols.

Algorithm 1 Injection

```
1: packet ← {ethernet, ip, tcp}
2: pdu ← PDU()
3: procedure INJECT
4:     sequence ← packet_tcp_seq + len(pdu) + len(mal_pay)
5:     packet_tcp_seq ← sequence
6:     SOCKET.SEND({packet, pdu, mal_pay})
7: procedure UPDATE(p)
8:     packet_ethernet_src ← p_ethernet_src
9:     packet_ethernet_dst ← p_ethernet_dst
10:    packet_ip_src ← p_ip_src
11:    packet_ip_dst ← p_ip_dst
12:    packet_tcp_src ← p_tcp_src
13:    packet_tcp_dst ← p_tcp_dst
14:    packet_tcp_seq ← p_tcp_seq
15:    packet_tcp_ack ← p_tcp_ack
16:    if PDU ∈ p then
17:        pdu_src ← p_PDU_src
18:        pdu_dst ← p_PDU_dst
19:        pdu_seq ← (p_PDU_seq+1)
20: procedure SNIFF
21:    while running do
22:        p ← SOCKET.RECV
23:        if p ≠ ∅ then
24:            UPDATE(p)
```

4.4 Master

Clients provide the ability interact with a server and retrieve information [5]. In the presented framework, there is a need of a *client* module, enabling the attacker to interface with SCADA application services, to progress with requirement R6 in master masquerading. Some distinguished characteristics of a traditional SCADA master closely resembles the characteristics of a client, as through it's use of requesting information or interacting with source devices, i.e a SCADA slave. The *master* module inherits attributes from the *client* module, thus inheriting the capability of connecting to services including SCADA slaves. Unlike the *client* module, the master has the ability to process to responses made by various slaves that are managed by the slave. The *master* module allows the framework to satisfy requirement R6 as it will assist with SCADA master masquerading.

The *master* module is implemented to purely interface with it's target slave, and must not have any specified procedures based on the target slaves configuration. Thus allowing for other master masquerading modules to extend from the *master* module. The *master* module along with the use of the *Spoofer* module should only operate within the rules of the protocol and nothing more. In the following sections, we discuss extending the *master* module to cater for operations based on it's target slave and the target SCADA process.

4.5 Slave

A server is a network program that allow clients to retrieve, insert or update information [5]. In the presented framework, a *server* module is needed to enable the attacker to provide malicious services to unsuspecting client devices. The *server* module is purely used to accept connections from target clients. To extend it's functionality towards interacting with SCADA masters, the *server* module is extended to create a *slave* module. SCADA Slaves are used to provide information based on requests from a master devices and other equipment monitoring the control process. Slaves provide similar characteristics to servers, as it waits for a client like program to establish a connection, and allows the client to request, insert or update information. The *slave* module in the presented framework is an extension of the *server* module, inheriting characteristics of receiving connections from SCADA masters. The *server* module then allows for requirement R7 to be partially satisfied, as it will assist with SCADA slave masquerading modules. With the utility of a *Spoofer* class instance, the *slave* class is able to process requests from masters and generate responses based on the implemented SCADA protocol.

4.6 Masquerading

In the presented framework there exists two *masquerading* modules, the *Master Masquerading*, an extension of the *Master* class, and the *Slave Masquerading*, an extension of the *Slave* class. Each of the masquerading modules presented are implemented with logic to respond to target system's configuration. The masquerading process "pretends" to be an existing service and provides the same control process as the real master or slave service. The *Master Masquerading* and the *Slave Masquerading* modules allow the framework to completely satisfy both requirement R6 and R7.

4.7 Reconnaissance

Reconnaissance is the process of discovering information about the target system. This process could involve an ARP, TCP sync, or ping scans, but additional information about the SCADA systems protocol and functionality or configuration. The *Reconnaissance* module allows for the retrieval of information about the initialised data objects and data point values, discover memory registers, or other pieces of information stored on automation controllers [11]. By accessing and discovering such information about the SCADA process, the *Reconnaissance* module is able to initialise it's own data objects to further enhance the *masquerading* modules and and configure itself to perform more sophisticated attacks against the target SCADA system. Therefore, the *Reconnaissance* module allows requirement R8 to be satisfied in the presented framework. Rodofile et al.[11] provides several network discovery algorithms that are able to preform protocol specific reconnaissance against a target SCADA system.

4.8 Replay

By performing replay attacks, an attacker has the ability to disrupt or manipulate operations by exploiting previously used and legitimate traffic to cause a malicious affect on the system [8, 12]. Replay attacks are well known and have been used to attack systems without authentication. Each of the replay modules discussed below satisfies requirement R9.

The *Injection Replay* module, an extension of the *Injection* module, is configured to utilise one or many critical messages from a collection of previously captured SCADA traffic. The message is then parsed by an instance of the *PDU* module, thus allowing for the message to synchronise with the target connection. Once the injection is triggered, the replayable messages are injected into the connection (See

Algorithm 1). The *Master Replay* module, an extension of the *Master Masquerading* module, only replays master messages from the previously collected traffic. This process is achieved by filtering the messages prior the attacks then retransmit the filtered messages sequentially and in relative time after each previous message (See Algorithm 2).

Algorithm 2 SCADA PDU Replay

```
1: replay_pcap ← Queue(pcap)
2: procedure REPLAY
3:     last_out_time ← 0
4:     while replay_pcap ≠ ∅ ∧ running do
5:         msg ← replay_pcap_1
6:         wait_time ← (msg_t − last_out_time)
7:         sleep(wait_time)
8:         socket.send(msg_PDU)
9:         last_out_time ← msg_t
```

4.9 Flooding

Flooding is the process of sending a series of messages to a target SCADA service that causes it to stay in the attacker's desired state [2]. The targets either lose legitimate requests, or will be placed back into a state which is desired by the attacker. The frameworks make use of two modules, the *Master Flooding* and *Slave Flooding* to allow the framework to satisfy requirement R10.

The *Master Flooding* module, an extension of the *Master Masquerading* module, would have a collection of predefined/ configured messages. The messages will be sent to the target slave as fast as possible to place the slave in the desired attack state by writing over memory registers or data points. The same method is adapted by the *Slave Flooding* module, an extension of the *Slave Masquerading* module, in which a collection of critical messages are flooded back to the target master, placing the master in a critical state.

4.10 Man-in-the-middle

A Man-in-the-middle (MITM) attack, is the process in which an attacker insinuates itself in the middle of an existing connection and intercepts messages between the communicating parties [10]. The *MITM* class presented in the framework is utilised to satisfy requirement R9. This module allows traffic from the target parties to be intercepted, read by the *PDU* class, and forward on to the designated party. The *MITM* module has far more potential than just reading and forwarding information from a connection (See Algorithm 3). In the following sections, we discuss the extension of the module to satisfy requirements R5, R8 and R3, using a modification module and connection hijacking modules.

4.10.1 Modification

The process of *modification* entails the manipulation of protocol messages received by the attacker during a MITM attack. The *Modification* module uses the *PDU* module to modify messages that have been intercepted from a SCADA connection, via the *MITM* module. The modification logic is able to update any PDU control fields i.e. length, redundancy when updating, function codes, removing, modifying or inserting protocol data objects. This functionality allows the framework to satisfy requirement R5. Due to the ability to intercept and read messages, the module satisfies requirement R8 also.

Algorithm 3 MITM Forwarding

```
1: procedure FORWARDING(packet)
2:     if packet_ether_dst ≡ Attacker_mac then
3:         if packet_ip_src ≡ Master_ip then
4:             packet_ether_dst ← Master_mac
5:             SOCKET.SEND(packet)
6:         else if packet_ip_src ≡ Slave_ip then
7:             packet_ether_dst ← Slave_mac
8:             SOCKET.SEND(packet)
9:         else Drop packet
```

Algorithm 4 MITM Connection Hijacking

```
1: masterMasq ← MasterMasquerading()
2: slaveMasq ← SlaveMasquerading()
3: hijacked ← False
4: procedure HIJACK(packet)
5:     if packet_ip_src ≡ Master_ip then
6:         MASTERMASQ.PROCESS(packet)
7:     else if packet_ip_src ≡ Slave_ip then
8:         SLAVEMASQ.PROCESS(packet)
9:     else Drop packet
10: procedure FORWARDING(packet)
11:     if packet_ether_dst ≡ Attacker_mac then
12:         if hijacked then
13:             hijack(packet)
14:             return
15:         else
16:             MITM.forwarding(packet)
```

4.10.2 Connection Hijacking

Connection hijacking is the process of hijacking a existing connection between two communicating parties. The *Connection Hijacking* module enables the framework to break an existing connection and take-over the communication between the two parties. Some of the required procedures are shown in Algorithm 4. The hijacking procedure in the *connection hijacking* module intercepts each message and synchronises itself with the connection. Once the module has synchronised to the connection, it is then able to trigger a connection hijack, thus severing the connection between the targets. The *Connection Hijacking* module is an extension of the *modification* module and utilises an instance of the *Slave masquerading* and the *Master masquerading* modules. The *modification* class is utilised to allow the *Connection Hijacking* module to include further modification characteristics for future extensions. The utility of the *Slave masquerading* and the *Master masquerading* modules allow for the *Connection Hijacking* module to masquerade by providing the relevant responses to each targets as both a slave and a master after hijacking a connection. As the *Connection Hijacking* module allows for the injection of messages into the network and is able to masquerade as both a master and a slave, the *connection hijacking* module addresses requirements R4, R5, R6, R7, and R8.

5. CASE STUDY

To verify our framework presented in Section 4, and vali-

date it's adherence to the requirements for generating anomalous data on SCADA systems, we present the implementation of a software tool based on the presented framework. We evaluate our framework implementation on our state-of-the-art DNP3 SCADA test-bed network, by conducting SCADA-based attacks. We then provide an analysis of the network impact of the implementation on our SCADA test-bed. We finally present our findings in the form of results, and show that the presented framework and it's implementation reflects the requirements discussed in Section 3.

5.1 Framework Implementation for DNP3

Our software implementation was developed to target the DNP3 protocol, as DNP3 is a prominently used SCADA protocol in power transmission networks [2]. To further understand DNP3 and it's operation in SCADA networks, please review the DNP3 standard[3]. We used Python 2.7 as our development language in order to use the Scapy packet manipulation library. We required Scapy, as there is a DNP3 plug-in that enables the ability to build the *Spoofer* module described in 4.2 and the PDU module described in Section 4.1. Scapy also provides sniffer function that allows for the implementation of MITM and injection functions. In addition to Scapy, the Python socket library was used to interface the implementation to the network, and enabled us to build the master and slave masquerading modules described in Section 4.4 and Section 4.5. Shown in Table 1 is the collection of attacks implemented using the software tool. As the framework is extendible, we were able to create a variety of attacks based on the presented framework modules, totalling to 32 attacks. To aid us in the analysis process for our case study, the 32 attacks were placed into 6 categories: Reconnaissance, Injection attacks, Masquerading, Replay, Flooding, and MITM.

The six categories encompass 2 reconnaissance mechanisms, 6 injection attacks, 7 masquerading attacks, 2 replay attacks, 6 flooding attacks, and 9 MITM attacks. We note the ability to implement a combination of attacks based on the flexibility of the presented framework. An example is the TCP hijacking functionality, which is based on the implementation of the *Injection* module described in Section 4.3 and a *Masquerading Master* module described in Section 4.6. The process of hijacking an existing TCP connection using a TCP sync packet. This enables the use sniffing the target connection and injecting a TCP sequenced sync message to break the TCP connection, and hijacking the server application instance from the master by connecting a masquerading master first. A second example of the flexibility of the framework, is the use of *flooding replay* (See attack 4.2 in Table 1), showing the ability to combine attacks for target SCADA protocols.

5.2 SCADA test-bed

To evaluate our presented framework, and validate the ability to generate anomalous traffic on SCADA networks, we required the use of a physical and realistic SCADA test-bed network. Shown in Figure 2, is the test-bed used for the attack data generation experiment. The test-bed was able to simulate real substation events through the use of real-world GOOSE traffic collected from industry partners. The test-bed consisted of a GOOSE Publisher, a SCADA gateway running a GOOSE subscriber and DNP3 slave (*Slave*), a DNP3 master, a GPS clock (*Clock*) for time synchronisation, a human machine interface (HMI), and an attacker disguised as a backup HMI (*Attacker*). The GOOSE publisher intelligent electronic device (IED) was a Raspberry Pi B+ that was transmitting real-world GOOSE traffic (*IED*), which is then subscribed by the SCADA gateway in a GOOSE subscriber application (*Slave*). In the SCADA gateway was also a DNP3 slave application which converted the GOOSE physical input data into DNP3 binary objects. The DNP3 master would then collect the data objects from the slave via polling, or receiving unsolicited responses from the DNP3 slave. A Hub is connecting each of the SCADA gateways to the switch allowing us to analyse the network traffic for the experiment using the attacker device. The *mirror* connection was used to monitor the master device to allow for the sniffing process of the injection-related attacks.

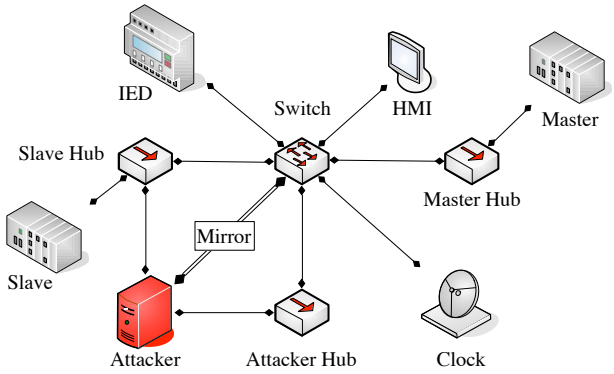

Figure 2: Critical Infrastructure Test-bed.

5.3 Experiment and Results

The attack data generation and capture experiment takes place in a space of 7200s. The attacks finished at the 4942s mark with each of the attack taking place approximately 180s apart. All the attacks executed were configured to last approximately 60s. The MITM modification attacks were configured to sniff 60 packets during each attack to provide minimal impact to the DNP3 process. In total, 32 attacks were configured to be executed in which two where reconnaissance mechanisms. Each of the attacks are described in Table 1 and is described in Section 5.1.

Shown in Table 2 in *Attacks* column are the list of attacks that had been executed during the experiment. These attacks can be referred to in Table 1. The *Attack Start* column shows the start time of the attack in the correlating row, shown relative in seconds from the start time of the experiment. *Attack End* provides the end time of the correlating attack, also in seconds, relative to the start of the experiment.

Shown in Figure 3 are four graphs that depict the flow of protocol traffic per second. Each Graph shows the flow of TCP, DNP3 (TCP messages containing at least one DNP3 message in it's application layer), ARP and GOOSE network frames. Each of the depicted graphs requires a logarithmic y axis as there is a significant spike in network traffic dur-

[3]Power and Energy Society [2012], IEEE Standard for Electric Power Systems Communications DNP3, Technical report, The Institute of Electrical and Electronics Engineers, Inc.

Attack 0 Reconnaissance	
0.1 Discover all devices on the network	0.2 Discover DNP3 slaves on the network
Attack 1 Injection Attacks	
1.1 Replay a previously collected message into the master via injection	1.4 Inject a malicious command into the slave (Cold Restart)
1.2 Replay a previously collected message into the master via injection and acknowledge any response from target	1.5 Inject a malicious command into the slave (Warm Restart)
1.3 Inject a malicious command into the slave (Freeze Objects)	1.6 Inject a malicious command into the master (Unsolicited Response with Object data)
Attack 2 Masquerading	
2.1 Connect to the slave and masquerade as a master device	2.4 Masquerade as the target slave and spoof object binary data
2.2 Use TCP hijacking to steal the existing service instance connection from the target master	2.5 Masquerade as the target slave and spoof object count data
2.3 Masquerade as the target slave and accept the connection from the target master	2.6 Masquerade as the target slave and fuzz binary object data
	2.7 Masquerade as the target slave and fuzz count object data
Attack 3 Replay	
3.1 Replay previously captured DNP3 messages to the slave	3.2 Use TCP hijacking to steal the existing service instance connection from the target master and replay previously captured DNP3 messages to the slave
Attack 4 Flooding	
4.1 Flood malformed messages to the slave	4.4 Flood malicious messages to the slave (Freeze Objects)
4.2 Flood previously captured messages to the slave	4.5 Flood malicious messages to the slave (Update Time)
4.3 Use TCP hijacking to steal the existing service instance connection from the target master and flood previously captured messages to the slave	4.6 Masquerade as the target slave and flood critical messages back to the target master
Attack 5 MITM	
5.1 Intercept and read all DNP3 communication between the targets	5.5 Intercept DNP3 messages and update binary object data point
5.2 Intercept DNP3 messages and inject DNP3 communication between the targets	5.6 Intercept DNP3 messages and update counter object data point
5.3 Intercept DNP3 messages and update function code	5.7 Intercept DNP3 messages and delete binary object
5.4 Intercept DNP3 messages and update binary status object	5.8 Intercept DNP3 messages and delete binary object data point
	5.9 Intercept DNP3 messages and insert binary object data point

Table 1: The attacks implemented in DNP3 using the presented framework.

ing the attacks thus making the data difficult to interpret. Figure 3A shows the flow of the test-bed's SCADA traffic without any attacks taking place thus making this chart our control. Figure 3B shows the flow of traffic from the attackers perspective, containing DNP3, TCP, and ARP. In this chart we exclude GOOSE frames as we wish to only analyse the affects of out attack generation on the DNP3 protocol for our experiment. Figure 3C shows the SCADA attack traffic from the perspective of the master and Figure 3D shows the same attack traffic but from the perspective of the slave.

In the following sections, we discuss the findings of the attack data generation experiment, we provide an analysis of the attack data generated and its impact on the network. Shown in Table 2 column *Total*, outlines the number of DNP3 messages produced during each attack. These messages were the total number DNP3 fragments generated, counting each DNP3 fragment encapsulated in a single TCP packet. The *Generated* column in Table 2 provides the total number of DNP3 fragments produced by the attacker. The *Modified* column in Table 2 provides the total number of DNP3 fragments that were modified during the correlating attack in the *Attack* column. Finally columns *R1 - R10* were the attack data generation requirements discussed in Section 3. As each attack is executed and stopped there is a time-stamp taken allowing for the logging of each attack and its time duration.

5.3.1 Reconnaissance

The reconnaissance processes *0.1* and *0.2* (See Reconnaissance in Table 1) occurred at the 120.21s and 132.22s mark during the experiment, shown in Table 2. During the process, *0.1* only produced ARP scans as the first stage of an attack requires the discovery of potential DNP3 devices. We can view the increase of ARP traffic in all the attack graphs in Figure 3 reaching approximately 400 ARP frames/s. As the process was able to discover the DNP3 service, the Requirement *R8* has been met. Attack *0.2* created a total of 30 DNP3 fragments in which the attack tool generated 29. One of the total DNP3 fragments was a DNP3 response containing the slave's data objects, DNP3 address and it's corresponding master. The execution of attack *0.2* demonstrates the use of R1, R2 and R8.

5.3.2 Injection

Injection attacks were only one single DNP3/TCP packet that was injected into a connection, however due to the injection messages causing de-resynchronisation at the TCP layer, a TCP conflict occurs as both the master and slave were de-synchronised. This caused between 6000-7000 frames per second on 3 occasions, 400s, 650s and 1100s mark shown in both Figure 3C and 3D. We correlate the TCP acknowledgement conflict with Injection attacks *1.2*, *1.3* and *1.6* as it was the injection of these packets that would have caused each of the target devices to de-synchronise. De-synchronisation also occurs at the 4000s mark after the MITM injection attack (5.2), this is due to the process of the MITM injecting four DNP3 messages between the master and slave. Attacks *1.1*, *1.4* and *1.5* had been accepted by the target and caused it to reset it's connection. Each of the injection attacks were able to demonstrate requirements *R1*, synchronising to the target DNP3/TCP connection; *R3* process of sniffing to aid the synchronising process; and *R4*, the ability

Attacks	Start	End	Total DNP3	Generated	Modified	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
0.1	120.21	123.13	0	0	0								✓		
0.2	133.22	137.46	30	29	0	✓	✓				✓		✓		
1.1	257.76	262.05	2	1	0	✓		✓	✓					✓	
1.2	382.17	390.08	2	1	0	✓		✓	✓					✓	
1.3	510.21	537.59	9	6	0	✓		✓	✓						
1.4	657.64	736.01	5	2	0	✓		✓	✓						
1.5	856.11	931.72	4	1	0	✓		✓	✓						
1.6	1051.93	1073.24	6	3	0	✓		✓	✓						
2.1	1193.38	1268.94	27	27	0	✓	✓				✓				
2.2	1389.09	1547.95	158	80	0	✓	✓	✓	✓		✓				
2.3	1668.07	1722.58	89	36	0	✓	✓					✓			
2.4	1842.66	1870.03	42	17	0	✓	✓					✓			
2.5	1990.17	2050.74	102	42	0	✓	✓					✓			
2.6	2170.88	2185.71	46	16	0	✓	✓					✓			
2.7	2305.81	2344.94	139	51	0	✓	✓					✓			
3.1	2465.05	2515.36	63	31	0	✓	✓		✓		✓			✓	
3.2	2635.39	2713.65	58	28	0	✓	✓	✓			✓			✓	
4.1	2833.75	2853.30	17	14	0	✓	✓				✓				✓
4.2	2973.40	3034.50	27	18	0	✓	✓				✓				✓
4.3	3154.57	3225.74	6166	3096	0	✓	✓	✓	✓		✓			✓	✓
4.4	3345.85	3416.96	5544	3449	0	✓	✓				✓				✓
4.5	3537.09	3610.20	2976	2976	0	✓	✓					✓			✓
4.6	3730.21	3763.90	1872	774	0	✓	✓					✓			✓
5.1	3884.00	3901.91	57	0	0	✓		✓					✓		
5.2	4022.04	4051.77	77	18	0	✓	✓	✓	✓		✓	✓	✓		
5.3	4171.92	4192.01	36	0	12	✓		✓		✓					
5.4	4312.04	4327.47	37	0	8	✓		✓		✓					
5.5	4447.58	4461.44	36	0	0	✓		✓							
5.6	4581.52	4604.50	33	0	14	✓		✓		✓					
5.7	4724.61	4735.18	36	0	6	✓		✓		✓					
5.8	4855.29	4875.33	36	0	14	✓		✓		✓					
5.9	4995.39	5022.30	43	0	14	✓		✓		✓					

Table 2: Experimental Results showing total Attack number (See Table 1), start time (relative) and end time of attack, total DNP3 fragments, generated DNP3 fragments and modified DNP3 fragments.

to inject anomalous messages into the SCADA network. Attacks *1.1* and *1.2* were replaying unsolicited responses, thus allowing them to provide *R9*.

5.3.3 Masquerading

The Masquerading attacks were identified as *2.1*, *2.2*, *2.3*, *2.4*, *2.5*, *2.5* and *2.7* (see the attack occurrences in Table 2). During the masquerading master attacks, no visual impacts were shown in Figures 3C, and 3D. However, the masquerading masters were able to interact with the DNP3 slave using but did not cause any harm to the service. During each of the attacks, a number of DNP3 fragments were generated and were transmitted to the slave and master targets, seen in 3B. We can see around the times of the slave masquerading attacks in Figure 3D, a lack of traffic directed to the slave, thus concluding the master traffic being directed to the attacker instead. Shown in Table 2 were the total number of DNP3 fragments generated during each of the masquerading attacks. See the "Generated" column in Table 2 to view the results. Each of the attacks were able to meet requirement R1 and R2, as they all required the protocol stack of DNP3 and able to parse and react to responses and requests from the targets. Attacks *2.1*, and *2.2* meet requirement R6 as they were able to provide a masquerading master. *2.3*, *2.4*, *2.5*, *2.5* and *2.7* meet requirement R7 as they were able to provide a masquerading slave. Attack *2.2* was able to combine TCP hijacking into the attack, showing the extendibility of the framework. Requirements R3 and R4 were also met by Attack *2.2* as the attack used sniffing and injection.

5.3.4 Replay

The Replay attacks were identified as *3.1* and, *3.2* (See the attack occurrences in Table 2). Attack *3.1* was able to generate 31 DNP3 fragments resulting in a total of 63, thus demonstrating the interaction with a target utilising previous traffic, with no modification of DNP3 sequence or CRCs. This was also the case with attack *3.2*, which generated 28 DNP3 fragments and a total of 58 fragments. Attack *3.2* also used TCP hijacking thus meeting requirements *R3*, and *R4*. Both of the attacks were able to achieve requirement *R6*.

5.3.5 Flooding

The Flooding attacks were identified as *4.1, 4.2, 4.3, 4.4, 4.5,* and *4.6* (See the attack occurrences in Table 2). Attack *4.1* generated 17 DNP3 fragments and totalled 17 DNP3. Attack *4.1* resulted in the slave disconnecting the master due to the malformed DNP3 fragments. Attack *4.2* generated 18 DNP3 fragments and totalled 27 DNP3. Attack *4.2* was able to flood repayable DNP3 fragments to the slave, allowing the attack to satisfy *R10*. Attacks *4.3, 4.4*, and *4.5* were able to provide a visual impact to network showing a spike from the slaves view in Figure 3D. Attack *4.3* generated 3096 DNP3 fragments and totalled 6166 DNP3 fragments. Attack *4.4* generated 3449 DNP3 fragments and totalled 5544. Attack *4.5* generated 2976 and totalled 2976 DNP3 fragments. Attack *4.5* had no affect on the target slave. The flooding during attack *4.5*, caused the slave service to stop responding as the slave due to the DNP3 service loosing sequence of messages. Figure 3C, shows a spike in TCP and DNP3 traffic around the 3700s mark, outlining the

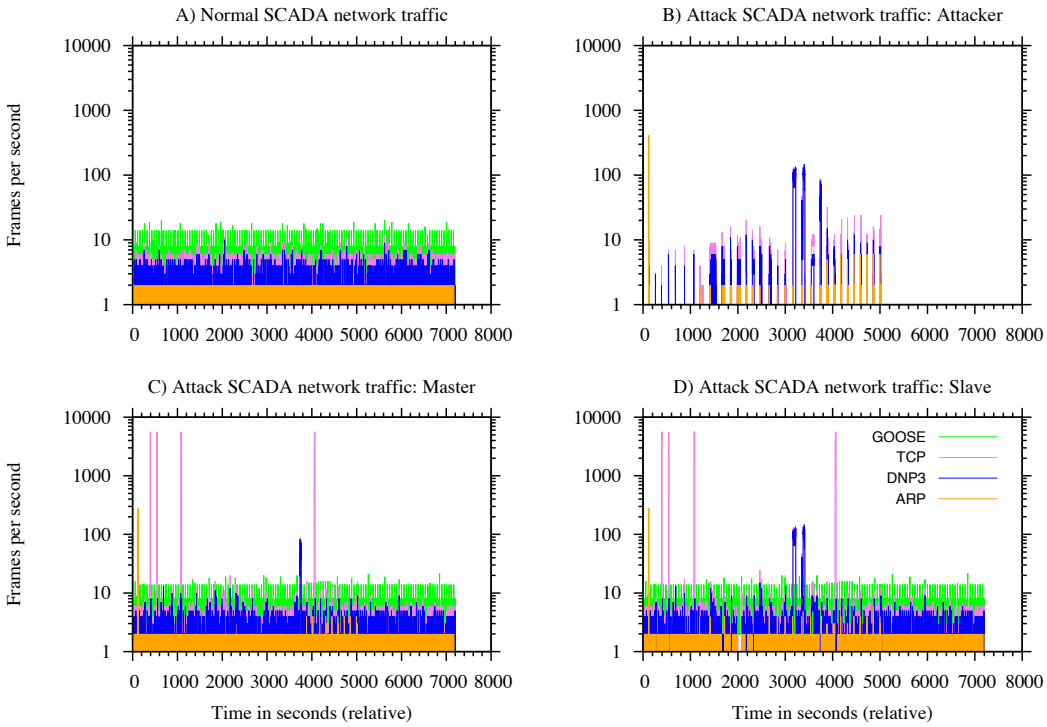

Figure 3: SCADA Attack traffic Analysis.

significant DNP3 generation from the *4.6* attack. The attack shows the spike of encapsulated DNP3/TCP and TCP traffic at a rate of approximately 500 frames/sec, thus demonstrating a significant impact on network traffic. As the attacks *4.3, 4.4, and 4.5* and *4.6* were able to flood a large number of DNP3 fragments to each of the targets, they were able to satisfy requirement *R10*.

5.3.6 MITM

The MITM attacks were identified as *5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8*, and *5.9* (See the attack occurrences in Table 2). Attack *5.1* is purely to provide reconnaissance as it was able to intercept and read a total of 57 DNP3 fragments, thus meeting requirements *R3*, and *R8*. Attack 5.2 was able to inject 18 DNP3 fragments during the experiment, 10 response fragments to masquerade as the slave, and 8 request fragments to masquerade as the master. The attack was able to meet the *R1, R2, R3, R4, R6, R7*, and *R8*. Attacks *5.3, 5.4, 5.5, 5.6, 5.7*, and *5.8* were significant, as they were the only attacks that satisfy the requirement of *R5*. See column "Modified" in Table 2 to view the results. Attack *5.5* did not have any modified fragments as it's target object (See Table 1) was not generated in the duration of the attack. During the MITM attacks we can view additional TCP and DNP3 traffic in Figure 3B. This was the case as there was DNP3 being intercepted and forwarded by the attacker, thus explaining the spikes. In addition, there is an increase of ARP messages which demonstrates the use of ARP poisoning.

In closing, a total of *10716* DNP3 fragments were generated by the attacker, which provided a total of *17775* DNP3 fragments added to the network in the space of 7200s. In which 68 DNP3 fragments were modified. From the results provided in this section, we verify the presented framework, and demonstrate it's usefulness, as the implementation of the framework allows for the generation of anomalous SCADA traffic. We also validate the utility the requirements defined in Section 3, as it allows for the creation of SCADA attack datasets.

6. DISCUSSION AND CONCLUSION

Due to the success of the results, we can verify the framework as an acceptable methodology for creating software tools for generating attack traffic to aid in the development of IDSs. We provided the attack experiments to validate the use of the framework by conducting a series of attacks. By using this methodology for creating attack datasets, the attacks may require a break between the attacks, as demonstrated with the presented experiment, we provided 180s. This is due to the affect of some of the attacks against SCADA applications. When viewing the attacks and their duration in Table 2, and comparing them with the graphs in Figure 3 we can observe the completion of the attack process and the ending timestamps, however, the effects of the attacks may take up to 60s to recover from TCP sequence issues, crashing/rebooting of the SCADA application. In this possible case, failure of executing the attacks may arise as an issue, as some systems may need time to recover.

Injecting messages into a connection may break synchronisation as the sequence numbers are out of sync. These numbers are incremented based on the number of bytes transferred in the TCP payload. Stealthy approaches could require the attacker injecting an even number of bytes between the two targets thus allowing the connecting target devices to stay in-sync after the injection process.

For the masquerading attacks presented in our experi-

ments we exploited the use of a second instance available on the target slave. To overcome issues of an occupied port, the use of TCP hijacking would allow the attacker to steal the existing instance from the authorised master.

The techniques we used to masquerade as the slave involved the allocation of the target slave's IP address and MAC address to the attacker's interface. Techniques used to masquerade as a slave may include ARP poisoning or DNS poisoning. Some vendors may implement a "heartbeat" service to check the status of SCADA equipment. When creating masquerading SCADA devices, spoofing "heartbeat" services maybe be required.

In some cases, a target SCADA protocol may already have security mechanisms such as authentication or cryptographic features, an example of such protocols is DNP3 SA. The presented framework is still applicable, as the implementation will allow for these mechanisms to exist in the Spoofer module (See Section 4.2)

Some of the more basic implementations of SCADA applications may only provide a small buffer for SCADA protocols over TCP/IP or UDP/IP, in which one SCADA application PDU or fragment may exist within a TCP or UDP payload. However, due to more recent implementations of SCADA applications, an internet protocol frame may contain multiple SCADA fragments. This suggests any implementation of the presented framework, must cater for the use of multiple fragments within a single internet protocol frame.

Using other programming languages, or software frameworks, may allow for some of the functionality presented, however the master and slave services need to be implemented. Metasploit has several modules that allow for the presented framework's *attack* layer, but none of which provide the *SCADA* layer, thus not satisfying requirements R4, R5, R6, and R7 (See Section 3). The requirements and the framework presented contributes a guide for the development of SCADA-based plug-ins for well know pen-testing frameworks such as Metasploit or Scapy, to aid in the generation of SCADA cyber-attack datasets.

In conclusion, we presented 10 requirements, inspired by the analysis of previous work, for generating attack traffic on SCADA networks. In addition, this paper presented a framework for generating cyber attack data on SCADA systems which satisfies the requirements. We verify the presented framework via case study involving a DNP3 implementation, and show the framework implementation is able to satisfy the presented requirements through experiments on our state-of-the-art SCADA DNP3 test-bed. Future work includes the generation of labelled attack datasets for multiple standardised and proprietary vendor SCADA protocols.

Acknowledgement

This work was supported in part by Australian Research Council Linkage Grant LP120200246, Practical Cyber Security for Next Generation Power Transmission Networks.

References

[1] Creech, G. and Hu, J. [2013], Generation of a new IDS test dataset: Time to retire the KDD collection, *in* 'Proceedings of WCNC, 2013 IEEE', pp. 4487–4492.

[2] Drias, Z., Serhrouchni, A. and Vogel, O. [2015], Taxonomy of attacks on industrial control protocols, *in* '2015 ICPE - NTDS'.

[3] East, S., Butts, J., Papa, M. and Shenoi, S. [2009], A Taxonomy of Attacks on the DNP3 Protocol, *in* 'Critical Infrastructure Protection III', Springer, pp. 67–81.

[4] Garitano, I., Siaterlis, C., Genge, B., Uribeetxeberria, R. and Zurutuza, U. [2012], A method to construct network traffic models for process control systems, *in* 'Proceedings of ETFA 2012'.

[5] Huitsing, P., Chandia, R., Papa, M. and Shenoi, S. [2008], 'Attack Taxonomies for the Modbus Protocols', *International Journal of Critical Infrastructure Protection* **1**(0), 37 – 44.

[6] Kobayashi, T., Batista, A., Brito, A. and Motta Pires, P. [2007], Using a Packet Manipulation Tool for Security Analysis of Industrial Network Protocols, *in* 'Proceedings of ETFA, 2007'.

[7] Lopes, Y., Muchaluat-Saade, D. C., Fernandes, N. C. and Fortes, M. Z. [2015], Geese: A traffic generator for performance and security evaluation of iec 61850 networks, *in* 'Proceedings of ISIE 2015'.

[8] Mo, Y., Chabukswar, R. and Sinopoli, B. [2013], 'Detecting Integrity Attacks on SCADA Systems', *IEEE Transactions on Control Systems Technology* .

[9] Morris, T. and Gao, W. [2014], Industrial control system traffic data sets for intrusion detection research, *in* 'Critical Infrastructure Protection VIII', Springer.

[10] Rodofile, N., Radke, K. and Foo, E. [2015], Real-Time and Interactive Attacks on DNP3 Critical Infrastructure Using Scapy, *in* 'Proceedings of (ACSW-AISC 2015)'.

[11] Rodofile, N., Radke, K. and Foo, E. [2016], DNP3 Network Scanning and Reconnaissance For Critical Infrastructure, *in* 'Proceedings of ACSW-AISC 2016'.

[12] Sayegh, N., Chehab, A., Elhajj, I. and Kayssi, A. [2013], Internal security attacks on SCADA systems, *in* 'Proceedings of 2013 ICCIT'.

[13] Shiravi, A., Shiravi, H., Tavallaee, M. and Ghorbani, A. A. [2012], 'Toward Developing a Systematic Approach to Generate Benchmark Datasets for Intrusion Detection', *Computers & Security* **31**(3).

[14] Voyiatzis, A. G., Katsigiannis, K. and Koubias, S. [2015], A modbus/tcp fuzzer for testing internetworked industrial systems, *in* 'ETFA'.

[15] Zetter, K. [2016], 'Inside the cunning, unprecedented hack of ukraine's power grid', *WIRED, March* **3**, 2016.

[16] Zhang, P., Li, F. and Bhatt, N. [2010], 'Next-generation monitoring, analysis, and control for the future smart control center', *Smart Grid, IEEE Transactions on* .

[17] Zhu, B., Joseph, A. and Sastry, S. [2011], A Taxonomy of Cyber Attacks on SCADA Systems, *in* 'Proceedings of 2011 iThings/CPSCom'.