Security Analysis of the Non-Aggressive Challenge Response of the DNP3 Protocol using a CPN Model

Raphael Amoah, Suriadi Suriadi, Seyit Camtepe, Ernest Foo
Information Security Discipline
Science and Engineering Faculty
Queensland University of Technology, Australia
{r.amoah, s.suriadi, seyit.camtepe, e.foo}@qut.edu.au

***Abstract*—Distributed Network Protocol Version 3 (DNP3) is the de-facto communication protocol for power grids. Standard-based interoperability among devices has made the protocol useful to other infrastructures such as water, sewage, oil and gas. DNP3 is designed to facilitate interaction between master stations and outstations. In this paper, we apply a formal modelling methodology called Coloured Petri Nets (CPN) to create an executable model representation of DNP3 protocol. The model facilitates the analysis of the protocol to ensure that the protocol will behave as expected. Also, we illustrate how to verify and validate the behaviour of the protocol, using the CPN model and the corresponding state space tool to determine if there are insecure states. With this approach, we were able to identify a Denial of Service (DoS) attack against the DNP3 protocol.**

Keywords*—*Supervisory Control and Data Acquisition Systems (SCADA), Distributed Network Protocol Version 3 (DNP3), Non-Aggressive Challenge Response (NACR), Coloured Petri Nets (CPN).

I. Introduction

Supervisory Control and Data Acquisition systems (SCADA) are systems that monitor and control industrial processes within utility industries and critical infrastructures. They are commonly used in industries such as water, electricity, transportation, oil and gas to monitor and to ensure their continued efficient operations [8].

SCADA systems comprise of master stations, outstations and protocols. Master stations are centralised computerised systems that interact with other devices by sending messages in the form of requests. They are usually associated with users (engineers). Outstations are the devices that take requests from master stations and get them processed. They comprise of Intelligent Electronic Devices (IEDs), Programmable Logic Controllers (PLCs) and Remote Telemetry Units (RTUs) [8]. SCADA protocols define the various rules for which master stations and outstations communicate via serial-line or TCP/IP protocols.

In recent years, reported attacks have shown that there is a growing number of malicious activities towards these critical systems [10]. As a result, security concerns of industries have been raised to ensure that these critical infrastructures are not compromised [3]. Although concerns have been raised, SCADA systems still face security issues as they (SCADA systems) continually connect to corporate networks [5], [7].

There are many SCADA protocols deployed in various industries. However, in this paper we focus mainly on the Distributed Network Protocol Version 3 (DNP3) [1] protocol. The purpose for our focus is because the protocol has a security mechanism in its application layer [4], which has not yet been formally analysed.

In this paper, we use Coloured Petri Nets (CPN) to create a model specifically for DNP3 (based on the protocol specification) and also uses its corresponding state space tool to perform the analyses.

CPN [6] is a formalism used to construct formal models of systems. Its supported state space tool facilitates the verification and validation procedures for analysing models. The applications of CPN and state space tool have been successful in modelling and performing analyses for various systems such as communication protocols, data networks and cryptographic protocols [2], [9]. However, to the best of our knowledge, our work so far is the first attempt at the formal verification of security properties of a SCADA protocol using CPN.

This paper proposes a CPN-based approach to construct a formal representation of one of the features of the DNP3 protocol, namely the non-aggressive challenge response. We also propose to verify security properties such as data integrity and availability of services of the protocol via the CPN model by using the state space analysis tool. Verifying the security properties of the protocol by using the model, we found a previously unidentified vulnerability that can be exploited by an attacker. Also, an additional benefit of the model is that it can serve as a framework. Other features of the DNP3 protocol can be added and other security properties can also be verified.

The rest of the paper is organised as follows: Section II introduces the DNP3 protocol. Section III presents the CPN-based approach for transforming the DNP3 protocol (Non-aggressive challenge response). The verification of the DNP3-NACR is described in Section IV. Section V and VI present the discussion, conclusions and the future research.

II. Distributed Network Protocol Version 3

DNP3 [1] is a layered and non-proprietary protocol, which is designed to facilitate communication between master stations and outstations via serial-line or TCP/IP protocols.

The application layer of the DNP3 protocol contains DNP3's security mechanism, which ensures that certain requests or responses are challenged and authenticated before they are processed. The mechanism involves a Keyed-Hash

Message Authentication Code (HMAC) and a challenge-response exchange [4]. Challenge response exchange, in accordance with the protocol's specification [1, p. 175] indicates that there are literally two different mode of its kind. Thus, challenge response mode and an aggressive challenge response mode. To avoid confusion, we refer to the challenge response mode as the non-aggressive challenge response (NACR). So, in this paper, our focus will be on NACR mode.

NACR is described as a method where a master station sends a request to an outstation. On receipt of the request, the outstation inspects the content of the request to determine if the request contains a 'critical' code. A 'critical' code, according to the protocol's specification [1, p. 175] is considered to be any code that has the potential to control a station, perform set-point adjustments and set parameters. These codes are identified as *Mandatory* and for this reason, any station that uses any of these codes shall be challenged by the receiver's HMAC mechanism to prove its identity. Otherwise, the request will not be processed. However, in case the code in the request is not marked *Mandatory*, then it is treated as *Optional*. *Optional* codes are not challenged by the HMAC mechanism.

It is to be noted that in this paper, we will refer to every request that contain *Mandatory* code as critical request and every request that contain *Optional* code as non-critical request.

A. DNP3 Messages

A DNP3 request or response fragment consists of an application control field, a function code and an object header field. Except that with response, it also introduces an additional field called "internal indicator" [1, p. 20-21]. These fields are described below.

Application control: This contains various sub-fields that provide necessary information to build and reassemble multiple DNP3 fragment messages.

Function code: This defines the purpose of a message sent. That is, if an outstation receives a request from a master, it is the function code that tells the outstation what to do. In Table I, we list only the function codes that are considered for this paper with their corresponding meanings and their criticality.

Object header: This is an additional supplementary information that may be required to create a complete DNP3 message. They are associated with DNP3 objects (ie. various places that hold binary input/output data, analog data, and counters). It consists of various sub-fields such as group, variation, qualifier and range fields. Its purpose is to ensure that when a master station for instance sends a read command ("0x01") to an outstation, the accompanied object header in the request fragment specifies to the outstation what format, type or group of data the outstation must read and return as response. In Table I, we have combined all the sub-fields of the object header as a single element.

Internal indicator: This is a field that only appears in response fragments from outstations. It contains two sub-fields to indicate certain states and error conditions within outstations. In Table I, we combine these two fields as one and represent it as *IIN*.

TABLE I. DNP3 MESSAGES COMPONENTS

Fields	Message and Meanings	Criticality
Function Code (fc)	- 0x01 Read Function	Non-critical
	- 0x02 Write Function	Critical
	- 0x81 Standard Response	Not Applicable (Non-critical)
Object headers (oh)	g20v1 / g20v7 - If paired with Non-critical fc then return feedback in the format of xx|xx|xx, otherwise return in format xx-xx-xx	
	g20v1 / g20v7 - If paired with Critical fc then return feedback in the format of "gxxx"	
Internal Indicator (IIN)	00_IIN_1 - This is produced if an Non-critical fc is received	
	00_IIN_2 - This is produced if an Critical fc is received	

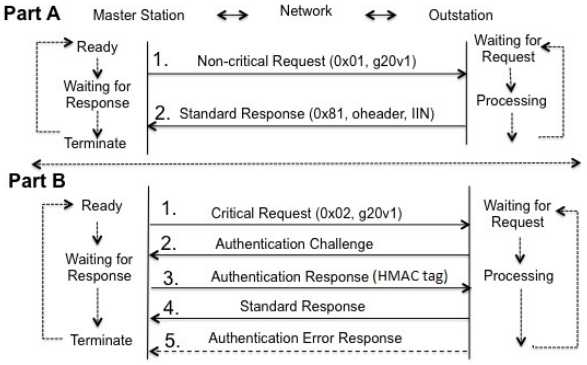

Fig. 1. The Non-aggressive Challenge Response Protocol

B. DNP3 NACR Protocol

Fig. 1 presents a message sequence chart that reflects the exact behaviour of NACR described in the DNP3 specification.

In Fig. 1, the master station sends a request to the outstation (see line 1, Part A & B of Fig. 1). As the outstation receives the request, a fragment inspection mechanism is triggered to inspect the content of the request (details not show in Fig. 1). Inspecting the request helps the outstation to determine if, the received request contains a critical function code or not (i.e. a critical or non-critical request). If the request received is a non-critical message, the request is processed. And if, feedback is required, the outstation replies with a "standard response" and then the protocol terminates (see line 2, Part A of Fig. 1). A standard response as we can see from line 2 of Fig. 1, contains three fields, namely: a function code (0x81, see Table I), object header and an internal indicator. It is to be noted that the object header and internal indicator in the standard response can be anything depending on request (see Table I row 2 & 3).

Alternatively, in case the request is a critical message, the outstation's HMAC mechanism is triggered to respond with an authentication challenge to the master station (see lines 1 and 2, Part B of Fig. 1). The use of the authentication challenge is to allow the user(s) associated with the master station, to prove their identity by providing a valid HMAC tag. After computation is completed and the HMAC tag is obtained, the master station sends the tag to the outstation as an authentication response (see line 3, Part B of Fig. 1). On receipt of the authentication response, the outstation creates its own

HMAC tag using the same computation elements previously used by the master station.

Afterwards, the outstation compares the tags for equality in order to authenticate the master station. If the tag values match, then authentication is granted, otherwise authentication fails. If authentication is successfully granted, the outstation generates a standard response and sends the standard response to the master station (see line 4, Part B of Fig. 1). However, if authentication was not successful, an error message is also generated by the outstation and sent as a response to the master station (see line 5, Part B of Fig. 1).

III. The CPN-Based Approach

CPN [6] is a formalism used to graphically construct formal models of systems. It has the capabilities of Petri nets for creating graphical models and a high-level programming language that provides the primitives for defining data types, data manipulations and creating compact models. Moreover, it supported state space tool also facilitates the verification and validation procedures for analysing models.

A CPN model consists of *places*, *transitions* and directed *arcs*. Places capture various states of a model. They are assigned with *colour sets* (.i.e data types) and may contain data items called *tokens* or a *multiset* (collection data items). Transitions signifies events. Arcs connect places to transitions and transitions to places. In CPN, there are also variable that are used. A variable may represents a particular *colour set*. They are typed and can be assigned values. An assigned value to a variable is known as a *binding element*.

In CPN, the occurrences of transitions may remove various tokens (specified by the binding elements) from places and distribute or create new tokens to other places. When places in CPN have a given token(s), it is referred to as a *marking*, and whenever a transition also occurs, it represents *state changes*.

A. Transforming DNP3-NACR protocol into CPNs

In this section, we show how we translated the high-level description of DNP3-NACR into a formal model of CPN. To achieve this, we use the message sequence chart provided in Fig. 1 to informally describe how our transformation was accomplished. However, it is worth noting that translating SCADA processes or operations using CPN can be very complex and challenging as well, depending on what type of operations needs to be captured and how large the SCADA network system is. For example, in this paper, we only modelled a communication mode, which involved a security mechanism between two entities. However, statistics from Table III of initial our model show that, there are 922 nodes (places and transitions) and 1254 arcs involved in the model. This figures can get higher than these stated values (i.e. if, there should be any additional operations required).

To represent the coordinated activities of master station, outstation and the network (which all the details are not depicted in Fig. 1), we use CPNs' places, transitions and various arcs expressions and functions to model the protocol.

Additionally, to capture the various DNP3 messages, we model the representation of function code, object header, internal indicator and all the dynamic states of the protocol as CPN colour sets (See Table II), and then we captured its operations using CPN standard metalanguage functions (SML). Employing this approach made modelling easy and inclusive as virtually any type of DNP3 messages can be captured.

Moreover, by making use of the CPN colour set types such as *product* and *record*, we were effectively able to combine and encode all vital information to represent DNP3 messages from either the master station or the outstation, and use SML to simulate their operations. It is to be noted that the SML functions used in CPN are mostly symbolic rather than the real operation. For example, the outstation's way of generating appropriate messages based on a particular request from the master station, *does not* perform the actual generation of response expected from the protocol specification. Rather, we simply use SML to mimic the behaviour. In Table II, we demonstrate this approach by modelling the responses from the outstation.

B. The CPN Model of the DNP3-NACR protocol

In this section, we present the CPN model for the DNP3-NACR protocol. However, we focus more on the mechanism that inspects the contents of requests on the outstation. Before we present our model, we first present the assumptions made when creating the model and then define its data structure.

1) Modelling Assumptions:

- Communication is unicast.
- Both stations transmit requests and responses at time that fit in a single fragment.
- The underlying layers of the protocol are reliable. Therefore, we do not expect communication failures.
- Both stations are aware of which function code requires critical operation or non-critical operation.
- The security mechanism in the application layer is in place. And also, the user ID is pre-associated with both stations.

2) Declarations: Table II depicts the declaration for the CPN model of the DNP3-NACR protocol. The table has three columns; Remarks, Components and Corresponding CPN declarations, and four rows which also comprises of Requests, Responses, Dynamic states and Functions. Remarks in the table presents the various DNP3 objects. Components show the elements that form the DNP3 objects. Corresponding CPN declarations presents the translation of the DNP3 objects into the CPN language.

An example for instance is, under Remarks, the **Requests** row shows that a request fragment is made up of a function code and an object header components. These components have their corresponding CPN declaration to be colour set (*colset*) *fcode* & *oheader*, which both have strings as its data types. In addition, both colour sets have also been assigned with token values. Thus, *1‘(“0x01”)*, *1‘(“0x02”)*, and *1‘(“g20v1”)*, *1‘(“g20v1”)* respectively. Therefore, in summary, a complete CPN request fragment is modelled by assigning a data type called *product*, which binds both the function code and the object header together (see the last row of components and corresponding CPN declarations in Table II).

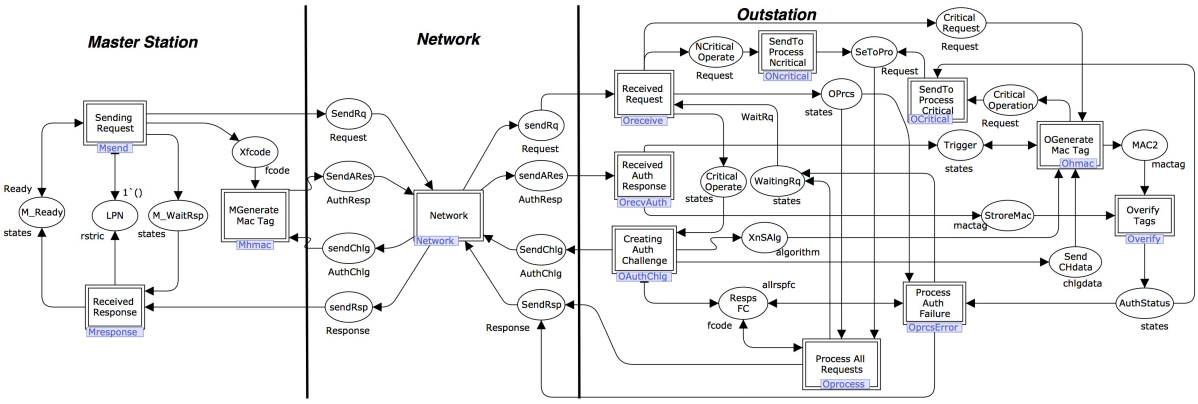

Fig. 2. CPN Diagram of the Non-aggressive Challenge Response Protocol - The Top-level Page

TABLE II. CPN DECLARATION FOR THE OUR MODEL

Remarks	Components	Corresponding CPN Declarations
Requests	Function Codes	Val allrqfc = 1‘(“0x01”), 1‘(“0x02”), Colset fcode = string; Var f:fcode;
	Object Headers	Val alloh = 1‘(“g20v1”), 1‘(“g20v7”), Colset oheader = string; Var oh:oheader;
	CPN Request Fragment	Colset Request = product fcode*oheader;
Responses	Function Codes	Val allrspfc = 1‘(“0x81”), Colset fcode = string; Var f:fcode;
	Object Headers	Colset oheader = string; Var oh:oheader;
	Internal Indicator	Colset IIN = string; Var i:IIN;
	CPN Response fragment	Colset Response = product fcode*oheader*IIN;
Dynamic States	Eg: Master station waiting for response	Colset states= with Ready |WaitRq|WaitRsp|Process |Critical|Terminate|Trigger|
Functions	Outstation Response - “0x01”	fun OutPro(f:fcode, oh:oheader) =let val(ff)=f val(ohh)=oh in if f=“0x01” andalso oh=“g20v1” then 1‘(“00|00|01”) else if f=“0x01” andalso oh=“g20v7” then 1‘(“111101”) else empty end;
	Outstation Response - “0x02”	fun Vresp(f:fcode, oh:oheader) =let val(ff)=f val(ohh)=oh in if f=“0x02” andalso oh=“g20v1” then 1‘(“10|01|10”) else if f=“0x02” andalso oh=“g20v7” 1‘(“11-00-11”) else empty end;
	Outstation Response with relation to the Internal Indicators (“0x01” or “0x02”)	fun OutIIN(f:fcode, oh:oheader) =let val(ff)=f val(ohh)=oh in if f=“0x01” then 1‘(“00_IIN_1”) else if f=“0x02” then 1‘(“00_IIN_2”) else empty end;

3) Model Structure and CPN Diagrams: The DNP3-NACR model is a hierarchical CPN consisting of two levels: a top-level page (shown in Fig. 2) and a second-level page (shown in Fig. 3). The top-level (Fig. 2) page shows the NACR model in a single page to depicts all the internal structures of the master station, outstation and the network (see the demarcations on Fig. 2). It consists of 26 places and 13 substitution transitions.

On the top-level page, the master station is shown to be on the far left, the network in the middle and the outstation on the far right (see Fig. 2). As we can see from the figure, all the coordinated activities of the master station have been modelled with three substitution transitions: *Sending Request*, *Received Response*, *MGenerate Mac Tag*, the network part with; *Network* and the outstation with; *Receive Request*, *Received Auth Response*, *Creating Auth Challenge*, *SendTo Process Ncritical*, *SendTo Process Critical*, *OGenerate Mac Tag*, *Overify Tags*, *Process Auth Failure* and *Process All Requests*.

The second-level page (Fig. 3) presents the details page of various substitutions transitions from the top-level page that we are interested. The purpose for this page is also to make it easy to visualise the flow messages while making reference to the protocol’s behaviour. The various substitutions transitions we consider include: *Sending Request*, *Received Response*, *ConnectA*, *ConnectB*, *Receive Request*, *SendTo Process Ncritical* and *Process All Requests* (see the top-level page).

4) Operation of the Model: Using our second-level page in conjunction with Fig. 1, we describe the operation of our CPN model. The initial markings of the master station constitute of four places: *M_Ready*, *Object Headers*, *Function Codes* and *LPN*. These four places have the token colours of *1‘Ready*, *1‘(“g20v1”) 1‘(“g20v7”)*, *1‘(“0x01”) 1‘(“0x02”)* and *1‘()* respectively (see **Sending Request** on Fig. 3).

In the model, *M_Ready* models the state where the master station is ready to transmit requests (see Fig. 1). *LPN* is introduced in the model to limit the number of requests sent at a time. *Object Headers* and *Function Codes* model the request contents from the master station to the outstation (see lines 1 and 2, Part A & B of Fig. 1).

The master station initiates request by firing transition *IniPack* if its enabled. *IniPack* gets its inputs (tokens) from place *Function Codes*, *Object Headers* and the support of *M_Ready* and *LPN* to form requests. After *IniPack* has occurred or fired, it creates a new token (multiset) to place *Request*. The availability of a token in *Request* enables transition *SendPack* to also fire or occur. As *SendPack* occurs, new tokens are concurrently created or distributed to place *M_WaitRsp*, *Xfcode* and *SendRq*. *M_WaitRsp* indicates that the master station is waiting for response (see *Waiting for response*, Part A & B of Fig. 1). *Xfcode* captures only critical request. *SendRq* indicates that the master station is sending a request to the outstation via the network (see lines 1 and 2, Part A & B of Fig. 1).

At this point, we assume that the request has been successfully transmitted via the network and transition *RecvRq* on the outstation is enabled (see **Received Request** on Fig. 3).

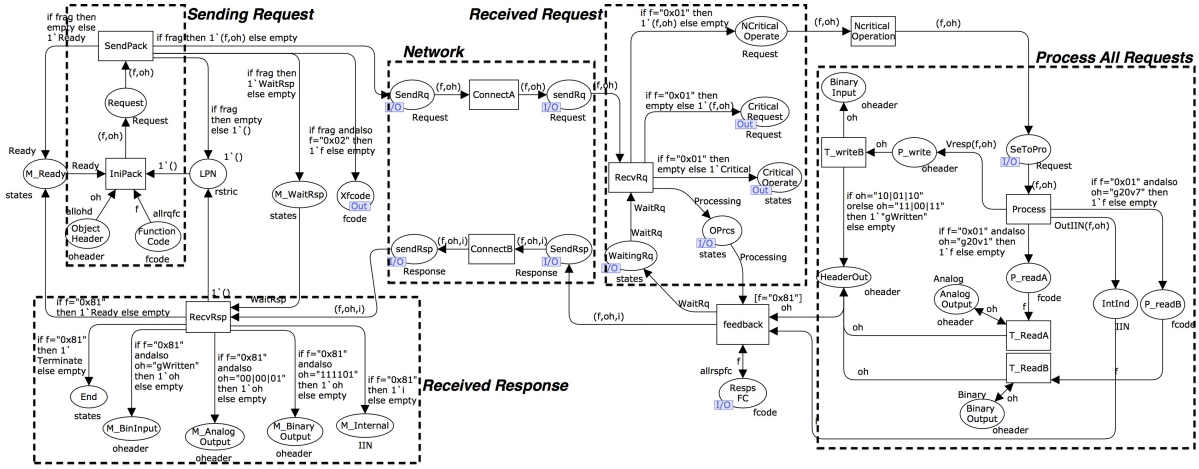

Fig. 3. The derived CPN Diagram from the Non-aggressive Challenge Response Protocol - The Second-level Page

RecvRq is enabled because it is already in the state *WaitingRq* and the request its been waiting for has arrived. This concurrently causes the outstation to 1) change its state from place *WaitingRq* to place *OPrcs* (see Processing, Part A & B of Fig. 1 and **Dynamic States** in Table II), and 2) also causing it to inspect the content of the request received (details are not shown in Fig. 1). Inspecting the content of the request is to determine if, the request requires a critical operation or not. We model that by the various arc expressions surrounding *RecvRq*.

As the received request requires no critical operation, then transition *Nritical Operation* (on the outstation) is enabled to forward the request to transition *Process* (see **Process All Requests** on Fig. 3). As *Process* is enabled, it concurrently accessed the rules provided on the arcs of place *P_readA*, *P_write*, *P_readB* and *IntInd* to generate standard response for the master station (see Functions on Table II and also lines 2 and 4, Part A & B of Fig. 1).

On the outstation, we have been able to model the processing of standard response fields with 6 places: *Resps FC*, *HeaderOut*, *Binary Input*, *Analog Output*, *Binary Output*, and *IntInd* respectively (see **Process All Requests** on Fig. 3). Place *Binary Input*, *Analog Output*, *Binary Output* and *HeaderOut* have the colour sets *oheader* (see Table II). They model the various DNP3 objects on the outstation (details are not shown in Fig. 1). *Analog Output*, and *Binary Output* have markings of one token each (i.e. 1‘(“00|00|01”) and 1‘(“111101”) respectively). *HeaderOut* represents the object header for the *Binary Input*, *Analog Output*, and *Binary Output* places. Place *IntInd* also models the internal indicator on the outstation. *IntInd* is assigned with colour set *IIN*, but it has no token colour (see Table II). Place *Resps FC*, which is also assigned with colour set *fcode*, models the repository for response function codes (see Table II). *Resps FC* has a token colour of 1‘(“0x81”) (see line 2, Part A of Fig. 1).

As the non-critical request requires to read a value, the object header in the request specifies which format and what type of data value to read and return (see line 1, Part A & B of Fig. 1). After the request has been executed from *Process*, a token is taken from either *Analog Output* or *Binary Output* and placed on *HeadertOut*. The presence of tokens on *HeaderOut*, *Resps FC*, and *IntInd* enable transition *feedback* to transmit standard responses to the master station via the network (see lines 2, 4 and 5, Part A & B of Fig. 1).

However, if the received request requires a critical operation (i.e. request contains a critical function code), then tokens are distributed to place *Critical Request* and *Critical Operate* to trigger the outstation’s HMAC mechanism.

On the master station, transition *RecvRsp* gets enabled as the outstation transmits the response via the network (see **Received Response** on Fig. 3). Firing *RecvRsp* concurrently distributes appropriate tokens to the various places associated with the transition. The associated places model the DNP3 objects on the master station (details are not shown in Fig. 1).

IV. Verification of the DNP3-NACR Protocol

In this section, we present the behavioural analysis of the CPN model of the DNP3-NACR protocol. The purpose for this analysis is to 1) find out how the protocol operates or behaves under normal circumstance and 2) how the protocol behaves under an attack scenario. This, therefore divides our analysis into two parts. The first part being the analysis of our initial model while the second part deals with an integrated adversary in the network portion of the second-level page (see Fig. 4). In our analysis, we will refer to the model with the adversary as the “modified model”.

A. Baseline Behavioural Analysis of our Initial Model

TABLE III. State Space Report - Initial Model

State Space Report	
State Space Nodes	922
State Space Arcs	1254
SCC Graph Nodes	922
SCC Graph Arcs	1254
Time	0
Home Markings	None
Dead Markings	19
Dead Transitions	None
Live Transitions	None

TABLE IV. State Space Report - Modified Model

State Space Report	
State Space Nodes	121
State Space Arcs	130
SCC Graph Nodes	121
SCC Graph Arcs	130
Time	0
Home Markings	None
Dead Markings	6
Dead Transitions	1
Live Transitions	None

Table III presents the full state space report of our initial model. It shows that the State Space and the Strongly Connected Components (SCC) Graph have the same number of

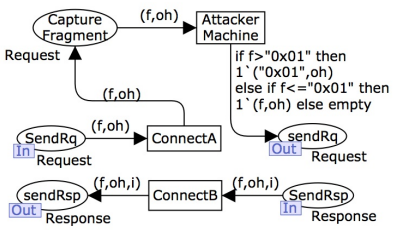

Fig. 4. The Attacker Model - The Attack Page

nodes and arcs. SCC Graphs are usually employed to test for reachability and for identifying loops in a given model [6]. Since our SCC graph values match our state space values, every node in our model *can be* reached from at least one other node. Furthermore, there are *no loops* in our model. Also, the table shows the Home Markings row as none. A home marking is a state or marking which can be reached from every state. Since the home marking shows none, it also implies that our model has a termination point.

Moreover, Table III also states 19 Dead Markings but no Dead & Live transitions. A dead marking is a state in our model, which has *no binding elements* to enabled it. These identified dead markings contribute to the absence of home markings, as well as the absence of dead & live transitions. A transition is considered *dead* if, it has *no path* from a reachable marking to enable it. And also, a transition is considered *live* if, from *any path* of a reachable marking we can always find an occurrence sequence containing the transition [6]. So, with the absence of these transitions, it means that all transitions in our model get activated at certain instances.

To inspect all the occurrences of the dead markings, we used CPN's SML query functions and also the state space to individually simulate all the dead markings identified. A close inspection on our results confirmed that all the dead markings obtained and the absence of dead & live transitions from our model are expected. This implies that the DNP3-NACR protocol we modelled is terminating as expected.

B. Introducing an Attack Model in Our Initial Model

This section introduces an attack model in our initial CPN model. It is assumed that the attacker is an insider. As a result, our attack is modelled to be part of the network on the second-level page (see Fig. 3). Due to space constraints, we only present the network part of the second-level page where the attacker is modelled (see Fig. 4).

In Fig. 4, the attacker is modelled with one transition (*Attacker Machine*) and one place (*Capture Fragment*). *Attacker Machine* and *Capture Fragment* model the attacker's ability to capture request packet, modify its contents and re-route the packet to its destination (see the arcs inscriptions on Fig. 4). Also, we have assumed that the attacker is only interested in modifying critical request to non-critical request since data in the fragments are in 'clear text'.

Now, we simulate our modified model with the state space tool to obtain the second state space report.

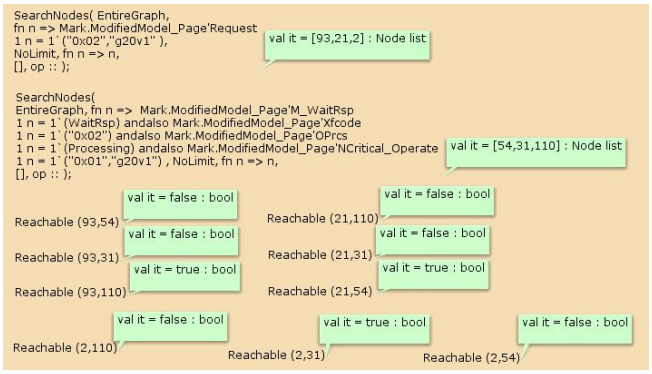

Fig. 5. The State Space Analysis of the Modified Model

C. Behavioural Analysis of our Modified Model

Table IV presents the state space analysis of our modified model. Comparing Table IV to Table III, we have identified some major differences. First of all, in Table IV, we see a reduction in the number of nodes and arcs present in the state space and SCC Graph. This reduction indicates that there is an unusual activity within the model. Secondly, we also see a reduction in the the dead markings (thus, from 19 to 6). Lastly, it is also stated that we have obtained a dead transition.

The decrease in the dead markings stated in Table IV indicates that there are certain instances in our modified model which prevent the model from terminating correctly (i.e. comparing to the analysis of the initial model). Also, obtaining a dead transition signifies that there is a transition in our modified model that does not get activated or fired. These behaviours (dead markings and transition) are not expected in our model. Therefore, we must perform further investigations.

First-level Investigations: We firstly used the SML query (*ListDeadMarkings ()*) to search throughout the model and list all the dead markings. The dead markings results we obtained are: 99, 97, 96, 24, 121, and 100. Afterwards, we also simulated individually, all the dead markings obtained from the query by using the state space tool. This, was to determine if all the listed dead markings were expected in our modified model (comparing to our initial dead markings in Table III). The results we obtained from the simulations indicated that dead marking 99, 97 and 121 were not expected.

Second-level Investigation: As we obtained unexpected dead markings, we needed to create SML queries that will have the inputs (markings) of the unexpected dead markings and determine if there *exists* insecure states in our modified model. If there *exists* insecure states in our model, then it indicates that the protocol we modelled has a vulnerability.

As it is shown in Fig. 5, two *SearchNodes* queries were defined. The first *SearchNodes* query was defined to have its input from place *Request* while the second *SearchNodes* query had its inputs from *M_WaitRsp*, *Xfcode*, *OPrcs* and *Ncritical Operate* from our modified model (see the first and second *SearchNodes* queries of Fig. 5). The purpose for these queries having their markings (inputs) from the mentioned places is because, we have assumed that the master station has sent a critical request to the outstation and therefore, we expect tokens on place *Critical Request* and *Critical Operate* (see

Received Request on Fig. 3). However, if there *exists* a token in any other place apart from *Critical Request* and *Critical Operate*, then it indicates that there are insecure states in our model (a vulnerability in the protocol).

From Fig. 5, the two queries have returned the node lists of (93, 21, 2) and (54, 31, 110) respectively (see the comments for both queries). By obtaining these node lists, we also needed to verify and validate if they were reachable from each other. The purpose for verifying and validating the reachability properties of these node lists, is to determine or get the assurance that it is not possible to get tokens in our expected places (i.e. place *Critical Request* and *Critical Operate*).

The last section of Fig. 5 presents the use of SML query, *Reachable ()*, for pairing the node lists obtained. Results from the pairings indicated that it is possible to reach node 110 from node 93, node 54 from node 21, and node 31 from node 2 (see the comments, where *it* is indicated *true* on Fig. 5). Reaching node 110 from node 93, node 54 from node 21, and node 31 from node 2 indicate that there are instances in our modified model, where *Critical Request* and *Critical Operate* do not possess tokens but rather *Ncritical* gets a token. The presence of a token on *Ncritical* indicate that there are insecure states in our modified model, which eventually results in unexpected dead markings. This, therefore indicates that the DNP3-NACR protocol we modelled, will not always terminate as expected due to the undesirable dead markings identified in our model.

A close inspection of our investigations (especially with the second-level investigation) revealed that the presence of token on *Ncritical* contributes to the presence of a dead transition in the state space report (see Table IV). In terms of our model, it means that as *Ncritical* on the outstation gets a token, every critical request sent from the master station to outstation is being treated as a non-critical request. Due to this reason, transition *T_writeB* on the outstation never gets enabled because the data integrity has been compromised (see Fig. 3). Moreover, it also implies that as *T_writeB* cannot be executed to generate the desirable response for the master, it results in a Denial of Service (DoS) attack.

V. Discussion

Through our model and its formal analysis, we were able to identify insecure states that an attacker can exploit to cause a DoS attack. DoS is possible if the attacker continues to flip every critical request from the master station to a non-critical request sent to the outstation. This behaviour will cause the master station to be receiving unexpected messages (i.e. unexpected dead marking in our model) from the outstation, as the outstation will treat every request received from the master station as valid. Secondly, this behaviour also indicates that the availability of a particular service needed by the master station is denied.

In the protocol specification [1, p. 206], it is stated that there could be unexpected messages from a corresponding station. However, occurrences of such behaviours are logged after a threshold. We argue that even though there is a log for such behaviour, there are no mechanisms that verifies data integrity at the receipt of messages on any of the stations. Additionally, there are no mechanisms to ensure the protocol takes appropriate actions against this unusual activity. This gives the attacker the chance to continually ensure that the master station receives unexpected messages. Moreover, as the availability of service is also compromised, there could be serious inconveniences in the SCADA environment. For example, an outstation of a sewage industry is expecting a request from the master station to control the flow of its sewage. As the availability of service is compromised, controlling the flow of the sewage will not be possible.

A solution that may counter this behaviour is to introduce a digital signature scheme for all request or response messages exchanged between the stations. A signed message will allow any station to determine if a given message is authentic or the message has been tampered with.

VI. Conclusions and Future Work

We have shown that CPN can be used to model DNP3's non-aggressive challenge response protocol effectively. We have shown that the state space tool supports various properties that can verify availability and data integrity. We have illustrated how these properties can be used to identifying unsecured states in the model which translates to attack on DNP3's non-aggressive challenge response mode.

Future work involves the refinement and generation of the modelling and analysing of HMAC mechanism for both non-aggressive and aggressive challenge-response. We will also build a front-end framework to simplify the modelling and analysis of SCADA protocols. Such a framework will be generic and therefore applicable to widely used SCADA protocols.

References

[1] "IEEE Standard for Electric Power Systems Communications-Distributed Network Protocol (DNP3)," *IEEE Std 1815-2012*, no. 1815-2012 (Revision of 1815-2010), pp. 1–866, 2012.

[2] I. Al-Azzoni, D. G. Down, and R. Khedri, "Modeling and Verification of Cryptographic Protocols using Coloured Petri Nets and Design/CPN," *Nordic Journal of Computing*, vol. 12, no. 3, p. 201, 2005.

[3] E. Byres and J. Lowe, "The myths and facts behind cyber security risks for industrial control systems," in *Proceedings of the VDE Kongress*, vol. 116, 2004.

[4] G. Gilchrist, "Secure authentication for DNP3," in *Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE*. IEEE, 2008, pp. 1–3.

[5] V. M. Igure, S. A. Laughter, and R. D. Williams, "Security issues in SCADA networks," *Computers & Security*, vol. 25, no. 7, pp. 498–506, 2006.

[6] K. Jensen, L. Kristensen, and L. Wells, "Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 9, no. 3, pp. 213–254, 2007.

[7] A. Nicholson, S. Webber, S. Dyer, T. Patel, and H. Janicke, "SCADA security in the light of Cyber-Warfare," *Computers & Security*, vol. 31, no. 4, pp. 418–436, June 2012. [Online]. Available: http://dx.doi.org/10.1016/j.cose.2012.02.009; http://www.sciencedirect.com/science/article/pii/S0167404812000429

[8] K. Stouffer, J. Falco, and K. Scarfone, "Guide to Industrial Control Systems (ICS) Security," *NIST Special Publication*, vol. 800, p. 82, 2008.

[9] S. Suriadi, C. Ouyang, and E. Foo, "Privacy compliance verification in cryptographic protocols," *Transactions on Petri Nets and Other Models of Concurrency VI: Lecture Notes in Computer Science*, vol. 7400, pp. 251–276, 2012.

[10] K. Wilhoit, "The SCADA That Didn't Cry Wolf," Trend Micro Inc., White Paper, 2013.