Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

Jamie Oliver Sian
01/05/2018

The development of a layered Network Intrusion Detection and Prevention System for Industrial Control Systems

Project supervisor - Dr Leonardo Aniello
Second examiner - Professor Kirk Martinez

A project report submitted for the award of
MEng Computer Science with Mobile and Secure Systems

Abstract

As technology becomes ingrained in our lives more and more, our reliance on Industrial Control Systems (ICSs) is ever increasing. However as shown by Stuxnet increased reliance does not always mean increased security. The source of ICSs insecurity is rooted in their origins; a time before the Internet where system security was solely physical, if at all. As devices became connected to the Internet so did ICSs, and with the adoption of common protocols they adopted their flaws, a step that systems designed solely to work in real time were not prepared for. To combat this problem many measures were suggested to secure ICSs, two of which were the use of firewalls and the deployment of a defence in depth approach to security. In this project a layered Network Intrusion Detection and Prevention (NIDPS) is created to ascertain the viability of using Machine Learning (ML) to combat this security issue. The results of this investigation suggest that, whilst effective in the majority of cases, a 99% classification accuracy is unacceptable in an ICS environment due to the destructive nature of Type 2 errors in this setting.

Contents

1 Introduction

A modern Industrial Control System (ICS) can be defined as a Cyber-Physical System (CPS) that uses networked computers and sensors to automatically control and monitor industrial processes. Such processes take a wide array of forms, varying from the operation of general manufacturing lines in the private sector to the generation of power used to support Critical National Infrastructure (CNI).

Both the concept of an ICS and the invention of the first Programmable Logic Controller (PLC) precede the first transmission of a message over the Internet (then ARPANET); therefore in this brief period ICSs were intrinsically air-gapped networks. By being in a "safe" environment, often using proprietary hardware and network infrastructure, it would take a remarkable effort, including physical access, to exploit most ICSs. This remarkable effort came in the form of Stuxnet, a malicious worm, designed to interrupt Iran's nuclear program. Stuxnet was a pivotal moment for the cyber security community as it highlighted the incentive for offensive cyber, and the need for defensive cyber, in ICS environments.

As increasing numbers of devices have become interconnected via the Internet so have ICSs, with many now operating using TCP based protocols to provide additional functionality such as e–mail alerts. Shodan.io [1] identifies ~100,000 Internet connected ICSs at the time of writing, presenting many potential targets for remote exploitation. In the hands of rogue actors this information could be used to cause catastrophic economic, political and social damage.

In order to reduce informational failure among security professionals, NIST published a guide [2] detailing a variety of methods to secure ICSs. These methods range from staff training to network segregation, with key points being the need for Intrusion Detection/Prevention Systems (IDSs/IPSs) and a defence in depth approach. IDSs and IPSs are often classified into one, or in the case of a hybrid system both, of the following categories with regards to classification method:

- Signature based – requiring predefined rules to determine what constitutes an intrusion; meaning that exploitation via new methods or valid commands could still occur.
- Anomaly based – defines a notion of 'normal' traffic; abnormal traffic is detected/blocked.

In this paper a modular Network Intrusion Detection and Prevention System (NIDPS), similar to the hybrid system of Depren et al.[3] is developed. However this system extends the aforementioned notion of two detection categories and implements: a signature detection module, an anomaly detection module and a static ruleset combined with a custom voting system to evaluate threats in a loosely coupled manner synonymous with defence in depth. Specifically this NIDPS explores the costs and benefits of applying machine learning to the security issue posed in ICSs.

2 Literature Review

2.1 ICS History

One of the earliest examples of a physical control system was Bonnemain's temperature regulator, developed in 1777 it was initially intended for incubating chicken eggs, and later used in industry to regulate the heating of water [4]. Following the invention of control systems like Bonnemain's, more advanced electrical control systems were developed to replace older mechanical systems. In the case of the Avro Vulcan, a strategic bomber developed in the 1950s, circuits and relays were used to replace heavier hydro mechanical flight control systems [5].

One of the greatest drawbacks of relays was that they were difficult to maintain and update in environments subject to quick innovation like the automotive industry. In 1968, to replace relays, the Hydra-Matic Division of General Motors wrote a specification for a replacement device. The resulting device, the Programmable Logic Controller (PLC), could be programmed to make logical decisions that would affect output devices based on input device control signals. The PLC was produced commercially in the later part of 1969 [6].

Throughout the 1960s, before the specification for the PLC was written, ARPANET was under development by DARPA. The concept of ARPANET inspired the creation of the World Wide Web, developed and adopted by the general public in the 1990s [7]. As devices became increasingly connected, so did ICSs, with one of the earliest records of ICSs becoming connected to the Internet being a patent for accessing a PLC over a corporate communications network [8].

2.2 The Need for ICS Security

As explained in 2.1, ICSs were never designed to be secure against remote threats; their goal is to efficiently control real-time industrial processes. As a result of this, communications are stripped down and in many systems defence measures are non-existent.

As shown by Bodenheim et al. [9], ICSs can be quickly identified by tools such as Shodan when exposed to the Internet. However Stuxnet, a highly sophisticated worm targeting high-end Siemens S7 controllers used to enrich Uranium proved that Internet exposure is not always necessary for system exploitation. Stuxnet was initially delivered onto target machines "shielded" by air-gapped systems via USB. After installation it used multiple zero day exploits, evaded antivirus and utilised privilege escalation in order to covertly alter the behaviour of PLCs to ensure Uranium enrichment failed [10].

An example of an attack that traversed a network into an ICS is the 2014 German steel mill attack. The attacker used social engineering to gain access to the corporate network, from which they were able to interface with ICS components, including a furnace [11].

2.3 Attacks Specific to ICSs

An adversary can utilise a variety of attacks in order to obtain intelligence or alter the behaviour of an ICS. Here we specify the use of different ICS attack vectors.

Preceding an attack an adversary will need to gather intelligence about the PLC in question. In performing a reconnaissance operation using a tool such as PLC Scan over Modbus or S7Comm, they will be able to map out PLC specifics in a similar way to that of an Nmap scan on a traditional computer network. In doing so the attacker will be able to identify the location of PLCs, and determine how they should be exploited for the greatest gain [12].

To attack an ICS an adversary could employ a response injection attack (RIA). RIAs involve an adversary forging a response from a slave device in the ICS. The incorrect response is sent to and acted on by the PLC. In this type of attack, the incorrect value will also be reflected in the Human Machine Interface (HMI) that site supervisors will use to monitor ICS operations [13].

Unlike RIAs, a command injection attack (CIA) originates from the other side of the PLC. This variant of attack can occur if either: an attacker spoofs input of a plant supervisor, or in a more sophisticated manner, remotely modifies the logic used in the PLC to make it perform some other function [14].

A final variant of attack is a Denial of Service (DoS) attack. In ICSs DoS attacks involve flooding the network with crafted erroneous packets. These erroneous packets fill the network stack of the remote with errors to the point that it is unable to respond to further request; the link goes down and the state is left as it is, potentially having disastrous effects [13].

Whilst the aforementioned list of attack vectors differ in their execution signature, their mission statement is similar – to gain intelligence or cause disruptive behaviour in ICSs. Whether this is caused by the delivery of malicious traffic, or the incorrect sequencing of valid traffic, the focus of defensive cyber in critical ICS environments should "typically follow the priority of availability, integrity and confidentiality, in that order" [2]; defensive measures preventing this could be considered as harmful as an attack.

2.4 ICS Protocols

ICS protocols tend to be unlike other protocols, with a focus on quickly crossing a network to ensure real-time deadlines are met, their packets are stripped down in comparison to most other protocols. For example, a Modbus RTU message has a maximum size of 256 bytes, whereas the maximum size of a UDP datagram 65,535 bytes.

Modbus is one of the most widely used ICS protocols. Modbus uses a master/slave architecture, with the master, typically a PLC, polling slave devices in order to determine their state and act depending on its own logic. Communications over Modbus are unencrypted by default, as in certain applications the time taken to encrypt packets would cause messages to miss deadlines, increasing system operation risk to an unacceptable level. Modbus comes in many forms, however the most common are Modbus ASCII, Modbus RTU and Modbus TCP/IP, with the former two protocols running over serial connections, and the latter running over Ethernet using IP addresses [15].

Other protocols for ICSs include Siemens S7, DNP3, BACnet and KNX, with the latter two being used in specialist building automation.

2.5 Existing Defences

Following Stuxnet and other ICS attacks, NIST published an extensive guide [2], detailing how to secure ICSs. Security suggestions highlighted in this report include the need for: IDSs/IPSs, defence-in-depth, encryption, and vulnerability assessment testing, views consistent with many other authors [16–18].

In accordance with the suggestions of NIST, many IDSs have been developed to protect ICSs specifically from adversaries. As with conventional IDSs, these can be categorised in different ways depending on how they function.

Host vs Network

Host IDSs (HIDSs) check for intrusions on a given device, alerting the user when the system has been exploited

Network IDSs (NIDSs) monitor traffic passing through the network interface that it is installed on, alerting the user when malicious traffic is detected.

Active vs Passive

Active IDSs block/allow traffic to cross a network autonomously; they do not require the oversight of a person in order to function. These can also be referred to as Intrusion Prevention Systems (IPSs).

Passive IDSs on the other hand, inform a user that an intrusion has been detected, and leave the decision of whether action should be taken to them. [19]

Both active and passive IDSs create some form of overhead in the system that they are protecting, some add network overhead only when abnormal behaviour is detected [20], whereas others operate by parsing all packets that travel through the IDS [21]. Other IDSs, for example SENAMI [22] use a hybrid approach for threat detection. SENAMI is both a passive NIDS and an active HIDS, proving that IDSs can be coupled in order to reap the benefits of different threat analysis techniques.

From this point onwards, unless stated explicitly we will consider IDSs/IPSs to be NIDSs/NIPSs.

Specifics of Threat Identification

IDSs/IPSs can employ either of the two following forms of threat identification methods to determine whether networked data constitutes a compromised system.

Signature-based detection (SBD), also known as misuse detection, is one of the most commonly employed techniques in IDSs/IPSs. Tools like Snort [23], can be used to create IDS/IPS rules to identify and/or block known malicious traffic. Exploitation via valid commands and unsignatured attacks will not be detected.

Anomaly-based detection (ABD) uses a baseline defining what is "normal" to ascertain whether a packet is normal or malicious. Useful for detecting unknown attacks such as zero day exploits. The likelihood of producing Type 1 and Type 2 errors is higher with this method.

2.6 Machine Learning in Network Security

Irrespective of use case, to test the accuracy of a classifier a labelled dataset is needed to provide a ground truth that classifier predictions can be compared against. In the sphere of IDSs, one of the most commonly used datasets is the KDD Cup 1999 (KDD–99) dataset [24] created by MIT's Lincoln Laboratory.

One of many examples of KDD–99 being used to create an IDS is the work of Depren et al. [3]. Depren et al. train a J48 decision tree using KDD–99 to autonomously create an SBD system to classify attacks in their misuse detection module. The novel aspect of this system is the combination of a decision tree with an ABD module utilising a Self-Organising Map (SOM) to create a hybrid NIDS (HNIDS); in doing so the correct classification of malicious packets increased. However in the context of ICSs, the HNIDS performs worse than misuse detection module alone as it increased the total false positive rate (FPR) and the false negative rate (FNR) of the system. Whilst the increased FPR + FNR (error rate) would transfer poorly to an ICS, the source of this appears to be the failure of the anomaly detection module and the decision support system; rectification of this in conjunction with the previously seen high correct classification could create a better tuned HNIDS suitable for use in ICSs.

Another variant of algorithms used for attack classification on KDD–99 is unsupervised clustering, this approach is taken by Portnoy et al. [25]. The level of accuracy obtained ranged from 40-55%; individually unacceptable for ICSs.

Whilst KDD-99 is used by many others as a training and testing set [26–28], most likely due to its availability and solidification as a benchmark dataset, it is heavily criticised. With respect to the generation of the dataset, McHugh [29] emphasises concerns surrounding the synthetic creation of background packets, the fuzzy nature of labelling, and how realistic the automated attack windows are. Furthermore in their study of unsupervised clustering Portnoy et al. [25] raise concerns about KDD–99 being inconveniently distributed for cross–validation. In an attempt to fix KDD–99 Tavallaee et al. [30] optimise the dataset by improving the record composition and distribution across KDD–99; creating a new dataset NSL–KDD. However this dataset still possesses the concerns of McHugh [29] as the underlying data remains unchanged.

From the perspective of network security KDD–99 and NSL–KDD are useful as often attackers will repurpose malware for usage on a completely different target. An example of this is Duqu, malware used for cyber espionage with striking similarities in implementation to Stuxnet [31]. Therefore in any bespoke dataset implementation, capturing the signatures of both generalised and specialised attacks is equally important, especially considering ICSs are now using common network protocols.

A bespoke dataset targeting Android based network intrusions is used by Kumar et al. [32] to create an Android specific NIDS. In a similar manner to [3], the effectiveness of the J48 algorithm was analysed, in this case individually, against others. Whilst J48 produced a superior true positive rate in certain circumstances, Random Forest always had a lower error rate; a promising quality for application in ICSs.

On balance, the best approach to create a NIDPS aimed at a specific ICS will be to couple modular system design feeding into a tuned decision support system with training and testing phases using a bespoke dataset containing both specialised and general traffic. This approach should combine the potential effectiveness of multiple systems, shown by Depren et al. [3], with the use case specificity of Kumar et al. [32], whilst maintaining resilience against conventional network–based attacks.

With respect to classifiers run in each module all options are to be considered, and the most appropriate will be implemented. This approach has been chosen as, provided a suitable voting matrix is implemented, the system should possess a level of resilience to internal models having a higher error rate provided the benefit offered outweighs this.

3 Design

3.1 NIDPS

This Section details the composition of the NIDPS and module interaction, for a visual representation of packet processing see Figure 1.

3.1.1 Packet Interception

When the NIDPS receives a packet to forward it will intercept it and parse it into a packet that can be interpreted. One copy of the parsed packet will then be sent to the Static Rule module for processing, and two copies will be sent to the Parsing module to be encoded before interpretation by the ABD module and the SBD module.

3.1.2 Classification and Voting

The logic behind whether a packet should be forwarded through the system is determined by the classification modules. The three intrusion detection modules have different strengths and have been implemented for the following reasons:

- Static Rules – Manually input ruleset, ascertains normal/malicious traffic, commonly used in industry, most companies will have a set of Static rules that reliably detect threats specific to them and amongst industry.
- ABD Module – Trained binary classifier, ascertains normal/malicious behaviour in the network, provides the greatest chance of sourcing a new attack/spyware.
- SBD Module – Trained n-ary classifier, asertains Traffic Type (map to Normal/Malicious), provides a generalised version of Static rules, learnt from attack data, useful for newly created ICSs or if certain attacks are omitted from Static rules.

To best utilise these defence mechanisms and ensure system safety, the NIDPS will act based on the voting matrix specified in Table 1. This matrix extends block/allow behaviour to ascertain whether a supervisor should be advised to take action.

Case 1 & 8 Trivial, unanimous agreement, allow/deny.
Case 2 & 5 Disagreement between Static and SBD, allow based on ABD to reduce chance of False Positive (FP) of False Negative (FN). Cause could be a misconfigured Static rule or SBD tree failure.
Case 3 Potential unknown attack, potential FP, allow.
Case 4 Potentially misconfigured Static, deny.
Case 6 Agreement between Static and SBD, deny
Case 7 Snort rules superior to SBD tree, SBD may need updating, deny
White Exceptionally high chance of True Positive (TP) or True Negative (TN)
Light Grey Potentially ambiguous case
Dark Grey Cause for concern

Alerts will be generated if the NIDPS is in doubt under either of the following conditions:

Case	1	2	3	4	5	6	7	8
Static Rules	0	0	0	0	1	1	1	1
ABD Module	0	0	1	1	0	0	1	1
SBD Module	0	1	0	1	0	1	0	1
Alert	0	1	1	0	1	0	1	0
Block	0	0	0	1	0	1	1	1

Table 1: Improved Voting Matrix

- Any situation where Static rules and the SBD module disagree, except case 4 (see Section 3.1.2) – Take the action suggested by the majority, as the agreement of intrusion detection methods is less likely to produce an FP/FN.
- Only the ABD detects a threat – Static rules and the SBD tree will not provide protection against attacks yet to be identified via a signature. Under this scenario alert a supervisor of the action the message was attempting to make.

3.1.3 Alert/Blocking System

If no Alert Status or Block status is set the packet will not trigger an alert/be passed onto the next hop respectively.

In CNI it is of paramount importance that a NIDPS alerts a supervisor/logs events where a system could have been exploited, both for the safety of the workers in the vicinity, and to ensure the best collateral damage limitation policy is taken systems fail. In this NIDPS if an alert is invoked a console based message stating the case triggered and the predicted signature (if the SBD module is able to signature the packet) will be printed to the screen, potentially in conjunction with a packet blocking dependent upon Block Status.

To block traffic the Blocking System must first ascertain why the verdict of the voting matrix implies the packet needs to be blocked. Upon determining this reason the packet will either be classified as one that can/can not be dropped by the system to ensure the system does not go down. If the packet is not mission critical, if the packet is a member of a connectionless protocol it will be dropped on the fly, otherwise a reset packet will be sent to the source. If the packet is mission critical, the blocking system will sanitise it and forward it; acting as a kill switch.

On completion of these tasks the system will loop back to the initial state to poll the next packet.

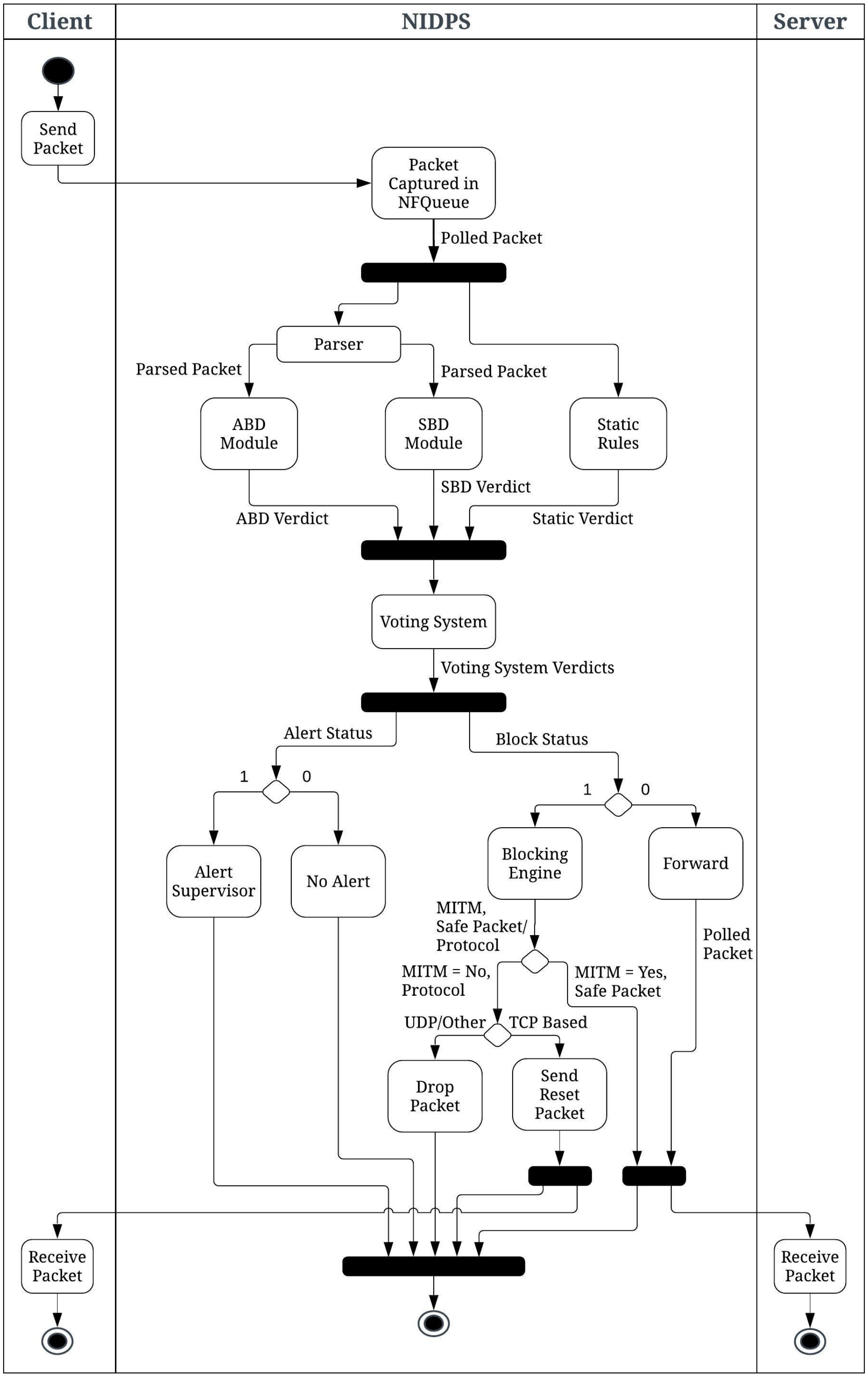


Figure 1: Activity Diagram detailing the handling process of an individual packet

3.2 Testbed Topology

In order to test the NIDPS a virtual testbed was used to simulate an Internet connected ICS. In the testbed all internal networking was implemented via static routes. This section details the nodes in the network and their purpose, with Figure 2 displaying where they exist in the topology relative to other nodes.

Remote Attacker
The left most laptop used in the testbed simulates a remote attacker. The skill level of this attacker could vary from a ‘Script Kiddie’ who could have located the network via Shodan to an advanced attacker for whom this is predefined target.

ICS Gateway
Represents the border between the internal ICS network and the wider Internet. On this box, to simulate cloud based communication from the remote attacker an IP and TTL spoofing script is run to simulate multi hop communication from the public IPv4 space and tag incoming packets with a random TTL between 32 and 60 between IPs. N.B as this spoofer operates between IPs, connections from the remote source port *x* will be mapped to a public IP address and TTL, which will stay consistent for connection–oriented protocols so long as connection between this port and the destination is maintained

Engineer
Represents an Engineer logged into the system remotely either via SSH or a VPN, normal technical commands will be executed from this machine.

NIDPS
Acts as a routing node in the middle of the ICS and is where intrusion detection will occur.

Modbus Master (MM)
Runs a script containing the underlying system control logic to provide control commands for the system.

Modbus Server (MServ)
Runs ModbusPal [33] representing typical Modbus storage between MM and MSlav

CI Attacker
Used to simulate perfect Command Injection (CI), could have occurred via logic upload. When running, captures all packets in NetfilterQueue [34], if it is a command from MM, uses Scapy [35] to maliciously craft the packet, then in all cases forward the packet to MServ.

RI Attacker
Simulates perfect Response Injection (RI), modifies and forwards Modbus responses from MSlav to MServ using the same technique as CI Attacker. Physical placement of this node would need to be undertaken by an insider.

Modbus Slave (MSlav)
Connects ICS Sensors to MServ and provides a GUI for analysis of the true ICS state prior to crafting from RI Attacker.

ICS Sensors
Of the simulation ICS detailed in Section 3.3, this node represents the mechanical systems and sensors used to regulate the ICS.

HMI
Outputs the state stored on the MServ; representing output to a plant supervisor.

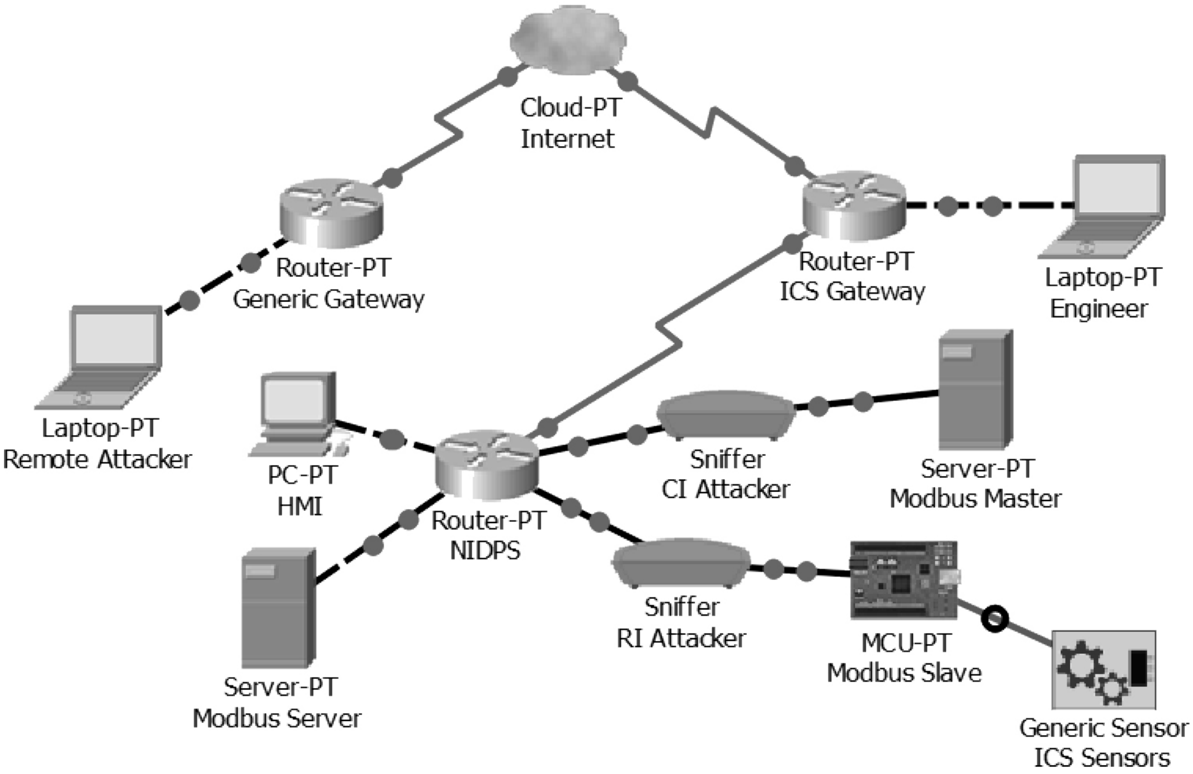


Figure 2: Testbed Topology

3.3 ICS Simulator

An industry that commonly uses ICSs is the power generation sector. In a thermal power plant a heat source source such as a coal fire is used to heat water and produce steam. The high pressure steam produced is then used to turn a turbine and produce electricity that can then be delivered to the power grid. In this project a thermal power plant is using the topology shown in Figure 2 is used to train and test the NIDPS.

Initially the tank starts with 500,000L (the normal level) of water. Once MSlav, HMI and MM are started, control logic from MM informs MSlav (via Mserv) to add coal; increasing the power in of the heat source. As power in increases up to of 2000MW (max), steam is produced to turn the turbine to produce electrical power out. As the water level decreases below the 500,000L mark, control logic is sent from MM to the MServ to water over the next time step to bring the water level back to ~500,000L.

Simulation notes:

- The simulator approximates basic thermodynamics and should be considered a good enough approximation of how water in the tank would truly behave.

- If the water level is below 500,000L, the injection of water is the difference between the current level and the normal level 500,000L. To this value a random ±20% noise factor is applied to simulate varied water flow rates in the pipe. If the level is below normal the volume of water added is 75% of the mass of steam produced in the last time step.

- Under this naive scenario we denote failure as the tank overflowing.

Simulation Access Invariants:

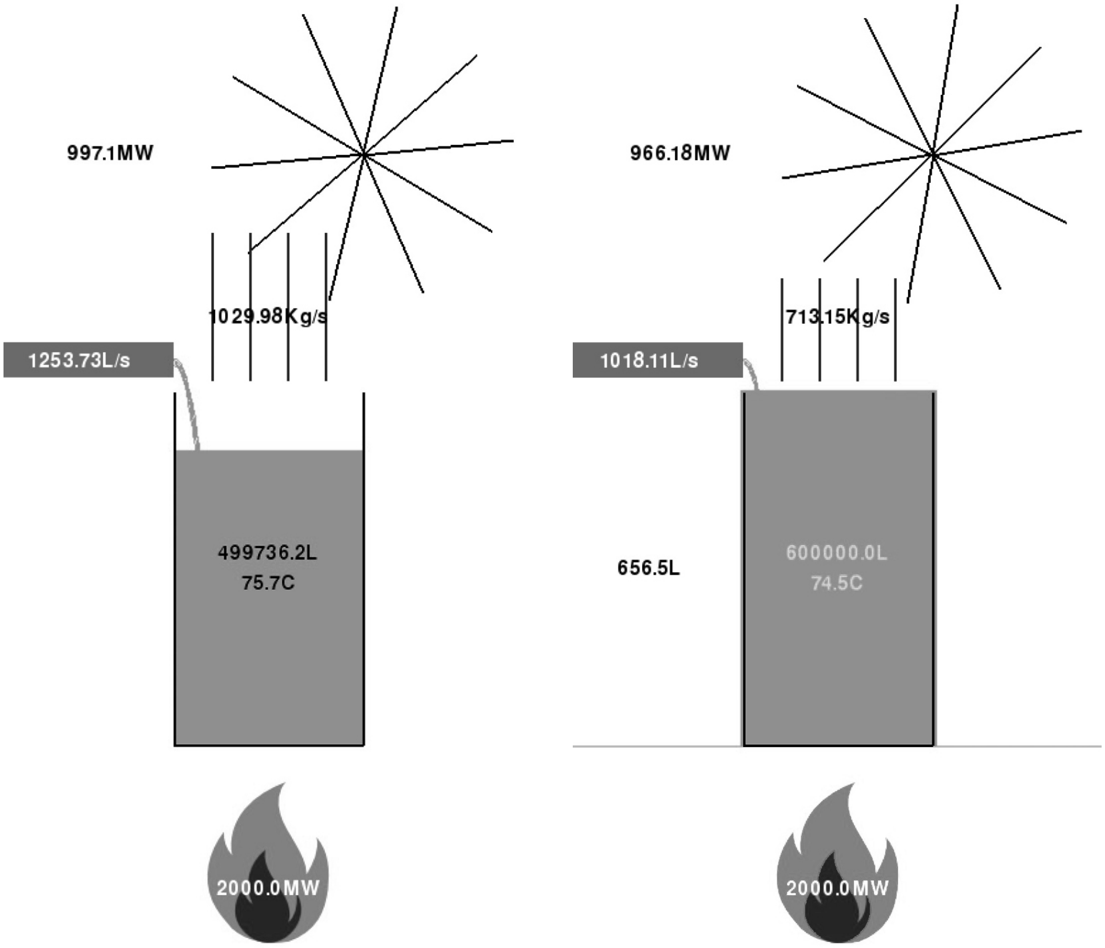
MM	Writes control inputs to MServ based on internal logic.
MSlav	Writes system state outputs to MServ based on physics engine.
MM	Reads the values supplied by MSlav via output holding registers of MServ.
MSlav	Reads the values supplied by MM via input holding registers of MServ.
HMI	Reads the values in MServ as current state.

In ModbusPal [33] on MServ, 16 bit holding registers [0,19] of unit 1 are used to store ICS related values, with register 20 used to synchronise HMI and MSlav for the purposes of simulation display. Note that in some instances decimal values are stored, as Modbus supports only unsigned integers; certain holding registers as, displayed in Table 2, are used to store bits $2^{-1}, 2^{-2}, ..., 2^{-n+1}, 2^{-n}$.

Direction	Usage	Register Range	Integer Registers	Decimal Registers
Input	Water Add Volume	0 – 1	1	1
	Power In (0/1)	2	1	0
Output	Water Level	3 – 6	2	2
	Water Temperature	7 – 8	1	1
	Power Out	9 – 12	3	1
	Steam Flow	13–15	1	2
	Power In	16–19	3	1
Synchronisation	Seconds Elapsed	20	1	0

Table 2: ICS Register Structure

Figure 3 presents two visual representations of ICS state, one normal, one failed. This visual representation will be displayed on the HMI (and for demonstration purposes MSlav) to provide an analysis of NIDPS performance. Flame graphic used in VSR from Vectors Market [36].



(a) Normal State

(b) Fail State

Figure 3: Example ICS state visualisations

4 Data Gathering

4.1 Attack Set

To create a dataset that could be trained against attacks, a set of exploits had to be compiled to be run against the ICS. Attacks running from a non–local device use the spoofer to spoof their credentials as explained in Section 3.2. This section details the attacks used and their behaviour.

4.1.1 Injection

In addition to the distinction of CIAs and RIAs specified in Section 2.3, in this project these attacks are divided further to analyse the difference in difficulty between classifying and preventing attacks from external hosts, considered ‘Simple’ and insider/multi–hop threats considered ‘Advanced’. In each of these four categories four further attack subclasses are included to provide more complex implementations of each technique.

Attacks are labelled using the following schematic (A)dvanced/(S)imple CIA/RIA (1-4) where the latter indicates the predicted difficulty of detection in relation to other attacks of the same class, e.g. SCIA1, is a Simple CIA of predicted detection difficulty 1, SCIA4 should be much harder to detect, and ARIA4 is not directly comparable.

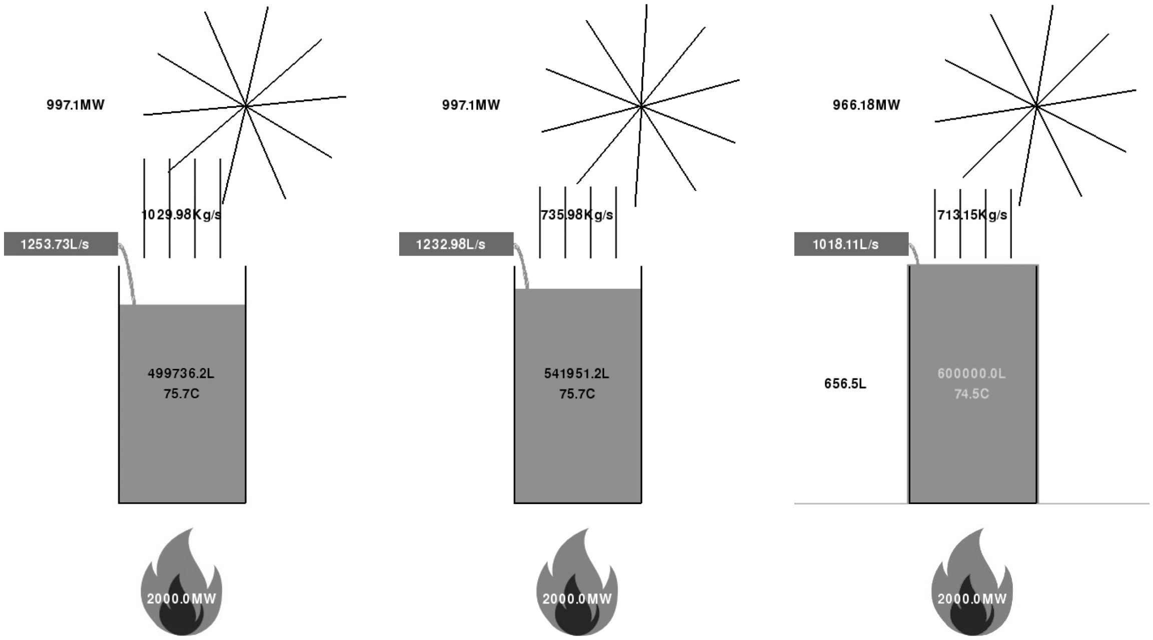
ACIA:
ACIA1 Water Add Volume (WAV) = 1500L, Power In = 0
ACIA2 WAV = 1500L
ACIA3 WAV = 1500L ±10%
ACIA4 If Power In == 1: WAV = Current WAV + random float [5,10]% + 500L
If Power In == 0: WAV = Current WAV + random float [0,5]% + 500L

Implementation Notes:
General – When ACIA is running 0% of MM control packets reach MServ.

ARIA:
ARIA1 Water Level (WL) = Normal WL (NWL) - 2000L
ARIA2 WL = NWL - 1000L, Steam Flow *= 1.5
ARIA3 WL = NWL - (1000L ±15%)
ARIA4 WL = NWL - (1000L ± 15%),
If Steam Flow < 1000L : Steam Flow += random int [275,300]

Implementation Notes:
All – Steam Flow represents received Steam Flow on wire.
General – An example of the effect of ARIA on HMI relative to True State can be seen in Figure 4.
General – When ARIA is running 0% of MSlav packets reach MServ.

SCIA: as ACIA attacks have no dependencies, SCIA attacks use the same implementation, albeit with the caveat that success is dependent on injections winning a significant number of



(a) HMI Output (b) True State (c) Overflow

Figure 4: Example of ARIA4 running on the ICS Simulator

races against the control traffic of MM. Furthermore injections must persist long enough to be viewed in the HMI and operated on by MSlav. In addition to this if races are won and lost in similar quantities the WAV on the HMI will flicker more than usual; potentially alerting a supervisor.

SRIA: These attacks operate on the assumption that the current state stored in MServ can be exfiltrated, interpreted and a malicious response can be injected before the HMI or MM poll MServ; a similar race condition to SCIA attacks.

SRIA1 WL = NWL - 2000L
SRIA2 WL = NWL - 1000L,
If Steam Flow *= 1.5 < 1300: Steam Flow *= 1.5
else: Steam Flow += random float [-20,20]Kg/s
SRIA3 WL = NWL - (1000L ±15%)
SRIA4 WL = NWL - (1000L ± 15%),
If Steam Flow < 1000L : Steam Flow += random int [275,300]

Implementation Notes:
All – Steam Flow derived from exfiltrated MServ state.
SRIA2 – Condition 1 required as the attacker could win multiple races and be detected if Steam Flow becomes obscenely large.
SRIA2 – Condition 2 applied to investigate a small amount of random noise.

4.1.2 Common Attacks

The following commands are some of the most commonly used attacks/reconnaissance tools to used to exploit systems:

hping3	– A penetration testing tool used for packet generation and analysis.
Amap	– A penetration testing tool used for service enumeration.
Nmap	– Benchmark scanner for network enumeration.
PLC Scan	– Custom built for recon missions against S7 and Modbus PLCs [12]
Patator	– A brute forcing tool compatible with a wide array of protocols.

The following commands are used against the specified IPs in the internal ICS network (Regex used for conciseness):

```
//DoS
hping3 −1 −−faster 10.10.(1|2|3|4|10).(100|101) //Malicous ICMP
hping3 −1 −−spoof [any target IP] 10.10.(1|2|4).255 //Smurf
hping3 −S −−flood −p (102|502) 10.10.(3|4).101 //SYN Flood
hping3 −2 −−faster 10.10.(1|2|3|4).101 //UDP Flood

//Recon
amap −bqv 10.10.(1|2|3|4).101 (22|102|502) //Amap
nmap −T(1|2|4|5) 10.10.(1|2|3|4).0/24 //Nmap
python plcscan.py (−−brute−uid)? //PLC Scan

//Brute Force
python patator.py ssh_login host=10.10.(1|2|3|4).101 user=FILE0 0=logins.txt password=FILE1 1=
    ↪ passwords.txt −x ignore:mesg=‘Authentication Failed.’ //Malicious SSH
python patator.py telnet_login host=10.10.(1|2|3|4).101 inputs=‘FILE0\nFILE1’ 0=logins.txt 1=passwords.
    ↪ txt persistent=0 prompt_re=‘Username:|Password:’ −x ignore:egrep=’Login incorrect.+Username:’
```

Listing 1: Common Attacks

4.2 Captured Data Set

$$
\begin{gathered}
\mathrm{X}_{100} = X_n \cup \mathrm{X}_m \quad s.t \\
|\mathrm{X}_n| = |\mathrm{X}_{100}| \cdot \frac{n}{100} \quad \wedge \quad |\mathrm{X}_m| = |\mathrm{X}_{100}| \cdot \frac{m}{100} \quad \wedge \quad \mathrm{X}_n \cap \mathrm{X}_m = \emptyset
\end{gathered} \tag{1}
$$

Equation 1: Data Set Notation

$$
\begin{gathered}
\mathrm{A}_{100} \subset \mathrm{B}_{100} \subset \mathrm{C}_{80} \\
\mathrm{C}_{100} \cap \mathrm{D}_{100} = \emptyset
\end{gathered} \tag{2}
$$

Equation 2: Data Set Relationships

$$U_{100} = C_{100} \cup D_{100} \tag{3}$$

Equation 3: Universal composition

Nature	Class	Signature	A_{100}	B_{100}	C_{100}	D_{100}
Normal	Operational	ARP	100	365	460	1132
		DNS	120	385	950	2408
		IPv6 TCP	10	10	20	20
		NTP	40	80	100	100
		Modbus	190	620	30000	75215
		TCP	160	520	5000	12502
	Technical	Diagnostic ICMP	90	280	350	914
		SSH	100	280	350	904
Malicious	Injection	SCIA1	100	100	410	972
		SCIA2	120	120	700	1669
		SCIA3	120	120	1000	2451
		SCIA4	120	120	1300	3195
		SRIA1	100	100	410	1021
		SRIA2	120	120	700	1779
		SRIA3	120	120	1000	2483
		SRIA4	120	120	1300	3296
		ACIA1	80	80	175	436
		ACIA2	100	100	275	676
		ACIA3	100	100	400	1027
		ACIA4	100	100	500	1234
		ARIA1	80	80	175	424
		ARIA2	100	100	275	684
		ARIA3	100	100	300	752
		ARIA4	100	100	350	885
	DoS	Malicious ICMP	100	100	400	993
		Smurf	100	100	300	772
		SYN Flood	100	100	300	731
		UDP Flood	100	100	300	773
	Recon	Amap	100	100	600	1549
		Nmap	100	100	600	1499
		PLC Scan	100	100	600	1518
	Brute Force	Malicious SSH	80	80	200	492
		Malicious Telnet	80	80	200	494

Table 3: Composition of Bespoke Dataset

In order to produce a bespoke dataset Wireshark was run in promiscuous mode on the NIDPS machine to capture and log both normal and malicious traffic. Following each round of data collection the PCAP was filtered to match the appropriate traffic criteria, parsed into CSV format using Scapy [35] and padded with the value 0.0 such that all parsed packets had the same length to enable interpretation by the models detailed in Section 5.

The data procedure of data collection was undertaken over multiple sessions consisting of 7200 ICS clock ticks, in this scenario a tick of 1s was considered a reasonable time given the scale of the scenario. Certain attacks were automated to ensure a reasonable number of packets were generated within the timeframe, approaches to achieve this varied from custom scripts to run attacks to more hosts faster, to using a backoff timers to start attacks on certain ICS state conditions. In advanced injection attacks, the latter approach was taken to ensure the ICS tank overflowed and had time to recover and return to normal operation. In this instance the difference between recovery and attack packets was recorded by logging the sequence numbers where the packet had been maliciously modified, as this was unique amongst control traffic on a per session basis. After testing this data was found by searching the logged PCAP for packets containing a sequence number in the list that also matched the criteria of Modbus traffic, and labelled as appropriate. For other classes this approach was also used but the search for sequence number was omitted.

Following multiple sessions of data collection to ensure an accurate, well distributed capture of system operation, the parsed CSV rows were shuffled and used to randomly produce datasets C_{100} and D_{100} whose composition is detailed in Table 3. C_{100} is to be used as a training and a testing dataset, whereas D_{100} is used only as an evaluative dataset. From C_{80}, as shown in Equation 2 A_{100} and B_{100} were formed to provide stochastic datasets such that the composition of attacks in A_{100} and $B_{100} \propto ln\, C_{80}$, with an additional scaling factor applied to B_{100} to ensure the probability of Normal and Malicious packets is 0.5.

5 Implementation

To produce an optimal model using optimal modules, a degree of testing and experimentation is required. In this section implementation specifics for both the model and modules produced are discussed.

5.1 Packet Interception and Parsing

To intercept packets, the NIDPS uses rules on the forward and input chains of iptables to pass recieved packets into a NetfilterQueue [34] queue. Upon packet retrieval, the packet is cast to a Scapy [35] packet for easier manipulation and processing. Prior to parsing, TCP timestamps, sequence numbers and acknowledgements are zeroed out as they were in dataset creation to ensure consistency between dataset generated in Section 4.2 and live traffic. As ML algorithms cannot process raw strings and large values, the parsing module preprocesses the packets by mapping the hexidecimal value of each byte onto the interval [0.0,1.0] by dividing by 255; after this process each packet is correctly normalised for input into ML models.

5.2 Reduction

In the case of algorithms whose time and space complexities scale poorly as the dimensionality and the number of points in the dataset increase, reduction is required to ensure timely completion of classification using a reasonable amount of memory. The form of reduction used in this project is Principal Component Analysis (PCA), implemented using [37] . Where n is the number of directions to project the dataset onto, PCA projects the dataset in the n orthogonal (perfectly uncorrelated) directions (Principal Components (PCs)), PC1, PC2, ... , PCn that maximise the variance of the dataset. Under this notation PC1 is the most discriminating feature, with following features being increasingly less discriminating. A final useful feature of PCA is that it reduces the likelihood of statistical bias in a dataset via the removal of feature correlation and the reduction of feature collinearity [38].

5.3 Anomaly Based Detection

5.3.1 Metrics

$$
\begin{gathered}
Precision = \frac{TP}{TP+FP} \qquad Recall = \frac{TP}{TP+FN} \\
F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \equiv \frac{2TP}{2TP+FP+FN}
\end{gathered} \tag{4}
$$

Equation 4: The derivation of F_1 score

In assessment of ABD modules F_1 score (see Equation 4) provides a measure of the correct classification of normal traffic flow relative to incorrectly classified traffic flow; a useful metric as a low error rate is necessary in ICSs.

5.3.2 Discriminant Analysis

Algorithms such as Linear/Quadratic Discriminant Analysis (L/QDA) have been both applied practically to problems such as HTTP traffic analysis [39] and compared against other algorithms [40]; providing a fair baseline to build further ABD module assumptions off.

- LDA & QDA assume classes form a multivariate gaussian.
- LDA & QDA classify based on a decision boundary/hyperplane.
- LDA assumes equal covariance matrices.

Experimental Setup:
– LDA & QDA: Independent Variable (IV) – Vary number of PCs (#PCs)=[1,50]

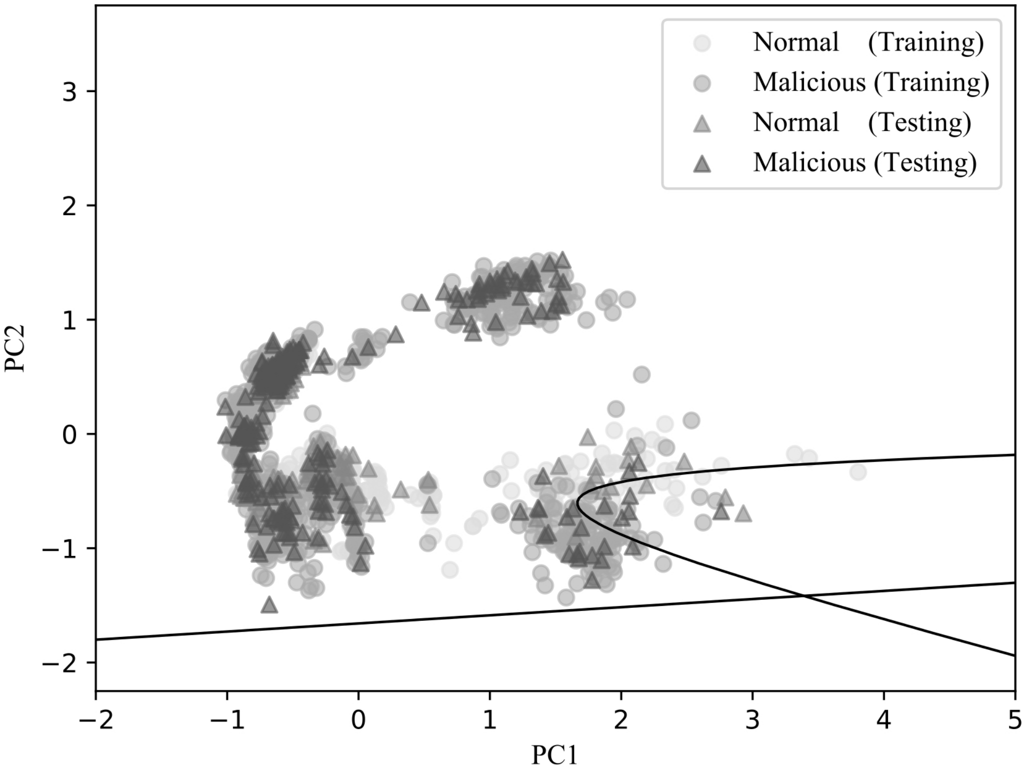


Figure 5: Projection of C_{80} (train) and C_{20} (test) via PCA into 2 dimensions. N.B. For clarity this diagram contains 5% of C_{80} and 10% of C_{20}. Linear decision boundary - LDA; quadratic decision boundary - QDA.

As shown in Figure 5, in lower dimensionality space it is very difficult to form a meaningful decision boundary; LDA defaults to placing the boundary on one side of the set and classifies everything as normal due to class imbalance. In attempting to exclude a small set of Malicious traffic QDA produces a boundary that generalises worse than this to the test data. Overall from this one can deduce that in low dimensional space this dataset is too random to provide a meaningful classifier.

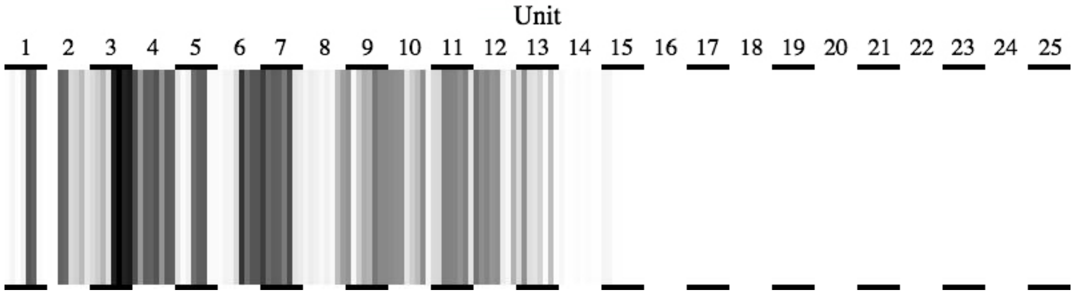


Figure 6: Per byte variance analysis of packets in U_{100}. Each unit specified consists of 8 bytes; the 25 units displayed are comprised of 200 bytes.

One of the major disadvantages of using PCA is the loss of interpretability. The goal of PCA is to maximise the variance of the dataset, as displayed in Figure 6, the variance of bytes 105+ across U_{100} is consistently large, most likely as a result of encrypted traffic such as SSH. Therefore poor projections in the initial steps of PCA, masked by the cardinality of Normal traffic, seem logical, with slight variances being more likely to approximate a multivariate gaussian that conclusions can be deduced from.

5.3.3 Unsupervised Learning

Using the idea of Portnoy et al. [25] that normal/attack data could be classified by unsupervised algorithms, in this case with raw packet data, we look to take advantage of unsupervised learning for anomaly detection, as it offers unparalleled economic benefits for business compared to supervised learning that requires the costly labelling of data [41].

Experimental Setup:
Contamination = Estimated proportion of outliers in the dataset.

One-Class Support Vector Machine (SVM)	IV – #PCs= [1,50], kernel= rbf, degree= 3, contamination= C_{100}–0.5, D_{100}–0.3
Local Outlier Factor (LOF)	IV – #PCs= [1,50], neighbours= 20, contamination= C_{100}–0.5, D_{100}–0.3

5.3.4 Other Learning Technniques

Experimental Setup:

K-Nearest Neighbours (KNN)	IV #PCs=[1,50], neighbours=20, Ball Tree and KD tree (optimally chosen by Scikit-learn [37])
MLP (NN)	IV epoch=[1,50], mini batch size= 64, hidden layers= 2, hidden layer size= 15, hidden layer activation= relu, optimiser= rmsprop, loss function= binary crossentropy, final activation= sigmoid, learning rate reduction factor= 0.2, patience= 5

5.3.5 Results

ABD Module		Test 1			Test 2		
Algorithm	Train Set	Data Set	Max Accuracy	Max F_1	Data Set	Max Accuracy	Max F_1
NN	C_{80}	C_{20}	0.980	0.987	D_{100}	0.981	0.987
LDA	C_{80}	C_{20}	0.949	0.967	D_{100}	0.949	0.966
QDA	C_{80}	C_{20}	0.946	0.963	D_{100}	0.944	0.962
KNN	B_{100}	C_{20}	0.914	0.940	D_{100}	0.918	0.942
LOF	–	C_{100}	0.577	0.660	D_{100}	0.674	0.774
SVM	–	C_{100}	0.655	0.761	D_{100}	0.655	0.761

Table 4: Final ABD Module Results

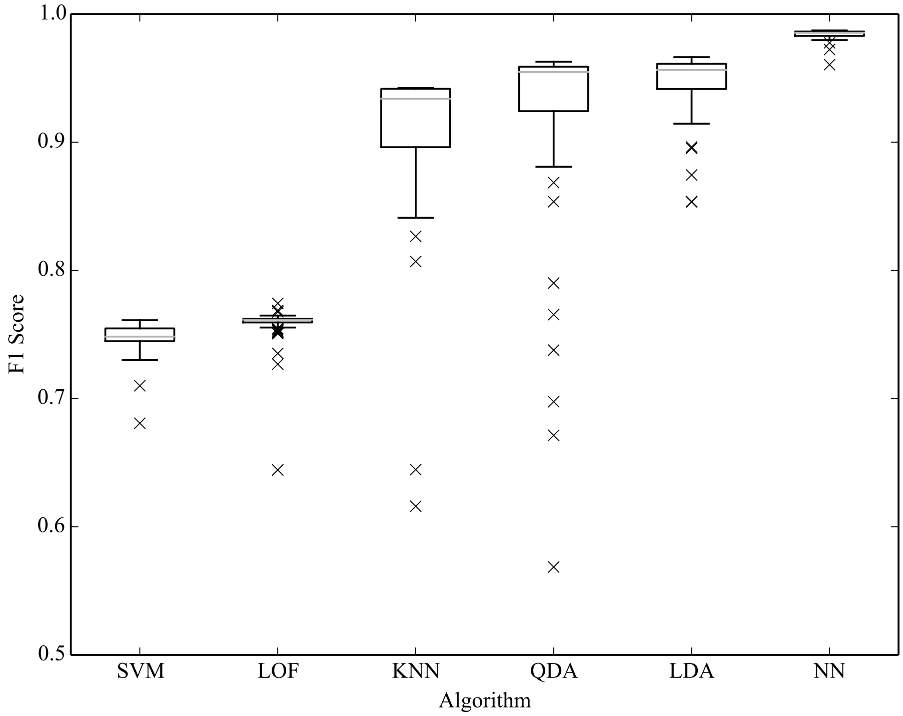


Figure 7: Box plot analysis displaying F_1 score for first $min(IVsize, 50)$ IVs when tested against D_{100}

As displayed in Figure 7 we would expect a distribution with a strong negative skew that learns quickly with respect to IV, and maintains a high level of accuracy with respect to error rate, as defined by F_1. As a result of this the optimal model for the NN was saved and is integrated into the ABD module.

5.4 Signature Based Detection

5.4.1 Metrics

Whilst F_1 score is a method used to evaluate the effectiveness of binary classifier, Cohen's kappa coefficient (κ) is better suited for multinomial classification and derived by the following formula:

$$\kappa = \frac{p_o - p_e}{1 - p_e} \tag{5}$$

In an abstract form κ = the likelihood this result occurred by chance. With: perfect accuracy $\kappa = 1$, chance $\kappa = 0$, and worse than chance $\kappa < 0$.

5.4.2 Supervised learning

Standard supervised learning algorithms to be tested and justification.

LDA: Baseline comparison.
KNN: Performed okay as SBD, does this transfer better to increasingly specialised classes.
Multinomial Logistic Regression (MNLR): Ascertain how well LR handles the problem.
NN: Will not be used twice, but use as comparison to other algorithms.

Experimental Setup:
– LDA, IV #PCs [1,50]
– KNN: IV #PCs [1,50], neighbours=20
– Multinomial Logistic Regression (MNLR): IV #PCs [1,50]
– NN : IV epoch=[1,50], mini batch size= 64, hidden layers= 2, hidden layer size= 50, hidden layer activation= relu, optimiser= rmsprop, loss function= categorical crossentropy, final activation= sotmax, learning rate reduction factor= 0.2, patience= 5

using sklearn [37] and keras [42] and Scapy [35]

5.4.3 Tree based supervised learning

The advantage of having a tree is simple packet analysis, the tree can be easily analysed manually.
Random Forest (RF): Investigate the impact of lots of shallow trees for classification. May be unacceptable for real time use as multiple threads may be needed for many trees.
Classification and Regression Tree (CART): Using Scikit's optimised version of CART, varying the tree depth, should lead to convergence on higher levels, but may induce overfitting.

Experimental Setup:
– RF: IV #Number of Trees [1,25], Tree Depth = 10
– CART: IV #Tree Depth [1,50]

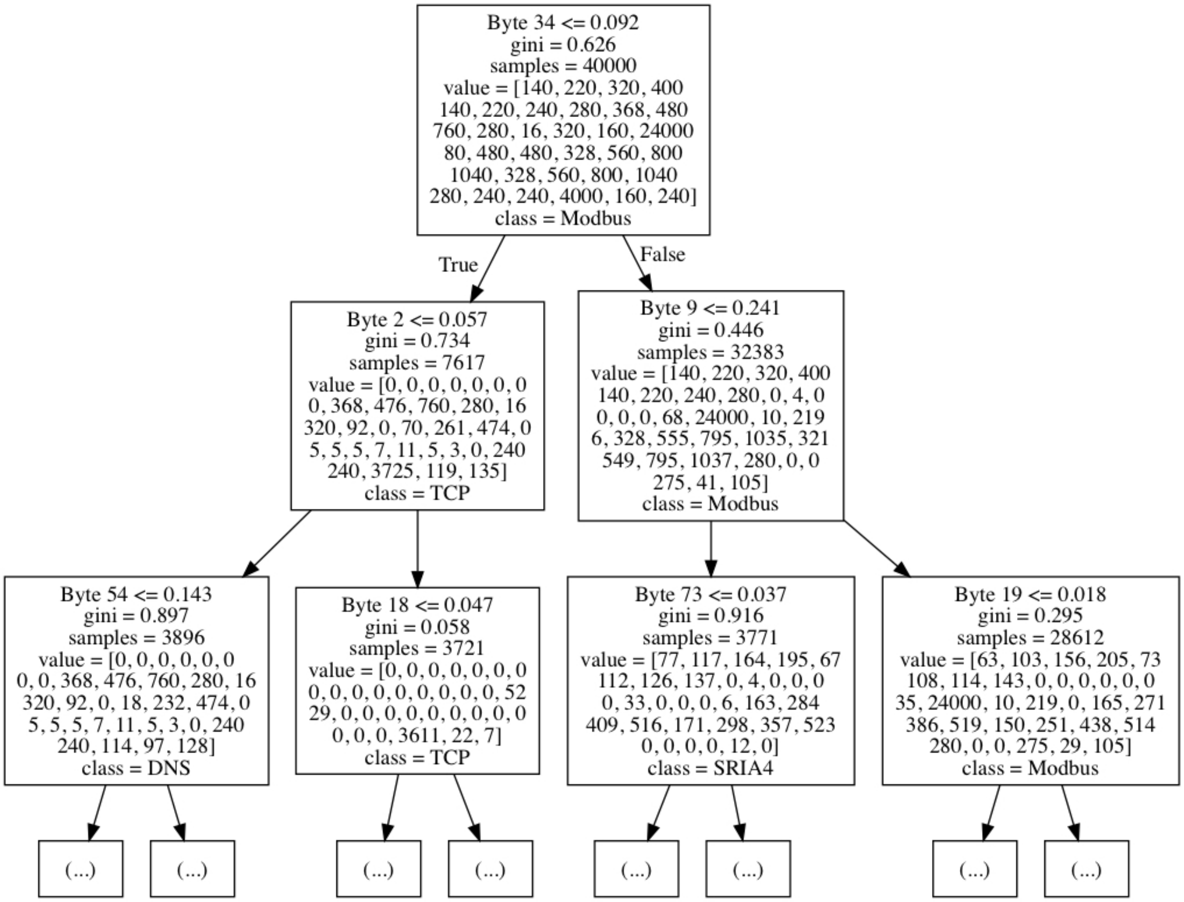


Figure 8: CART Visualisation

The following is an analysis CART decision process shown in Figure 8 with respect to TCP.
Byte 34 – Byte 2 of the TCP flags section of the TCP header
Byte 2 – Differentiated Services Field, including Differentiated Services Codepoint (DSCP) and ECN, this is used to differentiate between Recon/DoS protocols (True) and TCP and Nmap (False)
Byte 9 – TTL < 61.7: spoofer set external TTLs between 32 and 60; discriminate between local/external
Byte 54 – TCP TS echo reply < 14.5 (zeroed for TCP by parser); determines TCP
Byte 18 – Octet 2 of dstIP < 36.6, classes below this include TCP, NTP, Nmap, Mal. Telnet and UDP Flood
Byte 73 – For a Modbus write packet, upper byte of 4th transmitted holding register (see Table 2) (required when WL exceeds 65,535L), checks whether WL < ~621,000L, an invariant in write packets as the tank only has a capacity of 600,000L. The alternative leads to Amap and Malicious SSH packets.
Byte 19 – Octet 3 of dstIP, check under 4.6, useful as devices using Modbus on the 10.10.0.0/16 network in subnets of 10.10.1.0/24 – 10.10.4.0/24

	Prediction	Normal								Malicious							
	Actual	ARP	DNS	IPv6 TCP	NTP	Modbus	TCP	D. ICMP	SSH	SCIA1	SCIA2	SCIA3	SCIA4	SRIA1	SRIA2	SRIA3	SRIA4
Normal	ARP	1132															
	DNS		2408														
	IPv6 TCP			20													
	NTP		1		91		5										
	Modbus					75156	5		17								
	TCP					6	12448		2						4		
	Diagnostic ICMP							914									
	SSH					26	7		864	1					3		
Malicious	SCIA1									805	82	7	15	14	12	4	28
	SCIA2									69	1498	14	7	16	31	10	22
	SCIA3									14	35	2385	3	2	2		5
	SCIA4									15	18	20	3117	5	4	6	3
	SRIA1									11	18	27	11	924	10	6	9
	SRIA2									14	10	18	8	22	1663	13	18
	SRIA3									7	11	14	5	8	17	2381	24
	SRIA4									19	3	12	11	15	13	17	3200
	ACIA1																
	ACIA2																
	ACIA3																
	ACIA4					6											
	ARIA1																
	ARIA2					4											
	ARIA3					11											
	ARIA4					5											
	Malicious ICMP																
	Smurf																
	SYN Flood																
	UDP Flood					1				1		7					
	Amap						8			3			3	2	3	3	
	Nmap											3					
	PLC Scan						18			1	4			6	16		
	Malicious SSH								5	3	5	5	9	4	12	11	10
	Malicious Telnet						3			1	1	4	3			1	8

Table 5: CART confusion matrix produced by testset D_{100} part 1. Note blank cell denotes 0

	Prediction	Malicious																
	Actual	ACIA1	ACIA2	ACIA3	ACIA4	ARIA1	ARIA2	ARIA3	ARIA4	Mal. ICMP	Smurf	SYN Flood	UDP Flood	Amap	Nmap	PLC Scan	Mal. SSH	Mal. Telnet
Normal	ARP																	
	DNS																	
	IPv6 TCP																	
	NTP												3					
	Modbus		1	1	16	2	2	7	7									1
	TCP													13		18	10	1
	Diagnostic ICMP																	
	SSH				1										1			1
Malicious	SCIA1															1	4	
	SCIA2													1			1	
	SCIA3													1		4		
	SCIA4													4		3		
	SRIA1													1		4		
	SRIA2													5		8		
	SRIA3													5		8	3	
	SRIA4															3	3	
	ACIA1	436																
	ACIA2		665	11														
	ACIA3		21	1006														
	ACIA4				1228													
	ARIA1					424												
	ARIA2						674	6										
	ARIA3						10	728	3									
	ARIA4						11	5	864									
	Malicious ICMP									993								
	Smurf										772							
	SYN Flood											724		3		4		
	UDP Flood												763		1			
	Amap													1330		162	23	12
	Nmap										6	1			1486	3		
	PLC Scan											5		106		1361		1
	Malicious SSH								7					31		1	386	3
	Malicious Telnet													11	2	1	3	456

Table 6: CART confusion matrix produced by testset D_{100} part 2. Note blank cell denotes 0

As shown in Table 5 the optimal CART model displays high accuracy, and moderate precision for signaturing simple injection attacks. In this case, whilst most signatures are correctly classified, confusion has occurred due to the signaturing of packets according to: usage of Modbus and having a remote source. Beyond this criteria a degree of overfitting has occurred. In addition to this a small amount of confusion has been exhibited via recon/brute force attacks being classified as simple injection attacks, not a large issue as verdict of nature will be correct.

Also in Table 5 a minority of advanced injection packets were confused for Modbus packets and a few recon/brute force packets were confused for TCP/SSH packets. This result reiterates the concerns of McHugh [29] expressed about KDD–99. Whilst correct, it is an understandable issue that malicious traffic may share the same patterns as valid traffic, it is an intrinsic issue that we cannot perceive the intentions of an attacker using a valid protocol preceding exploitation.

In Table 6 a level of confusion in the form of Modbus traffic being predicted as malicious traffic was exhibited, displaying FN results and that the system was aggressive with classification of these packets. In a similar manner to Figure 5, some normal control protocols, namely TCP were confused for malicious traffic. Provided an appropriate voting matrix is implemented this should be rectified. Confusion was also displayed between Recon and Brute Force attacks.

5.4.4 Results

SBD Module		Test 1			Test 2		
Algorithm	Train Set	Data Set	Max Accuracy	Max κ	Data Set	Max Accuracy	Max κ
CART	C_{80}	C_{20}	0.986	0.978	D_{100}	0.987	0.980
NN	C_{80}	C_{20}	0.898	0.836	D_{100}	0.901	0.842
RF	C_{80}	C_{20}	0.888	0.813	D_{100}	0.881	0.800
LDA	C_{80}	C_{20}	0.802	0.688	D_{100}	0.797	0.679
MNLR	C_{80}	C_{20}	0.806	0.679	D_{100}	0.771	0.575
KNN	A_{100}	C_{20}	0.729	0.620	D_{100}	0.733	0.625

Table 7: Final SBD Module Results

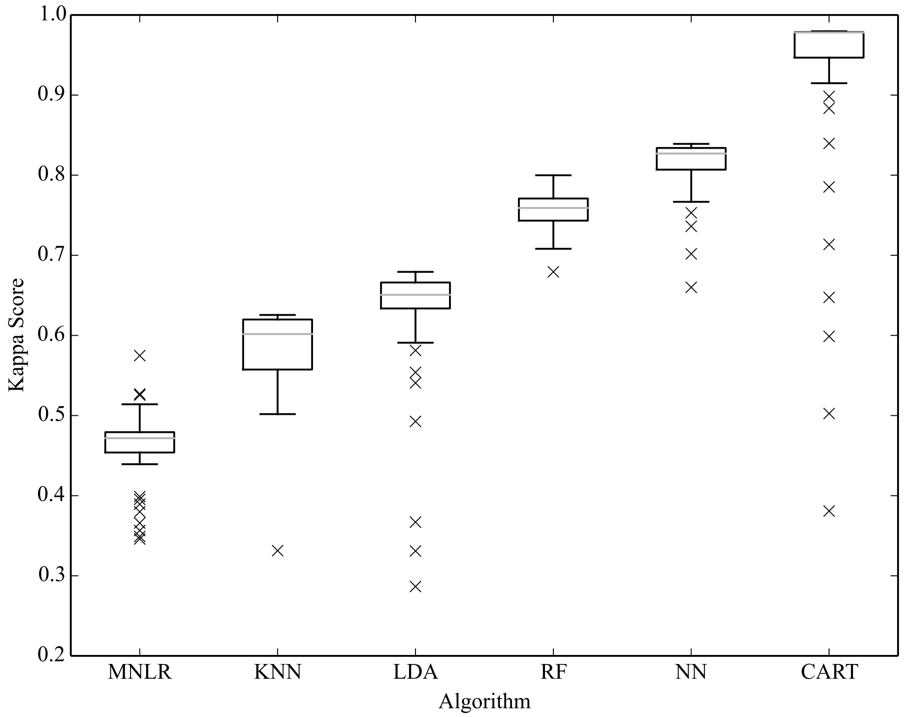


Figure 9: Box plot analysis displaying κ score for first $min(IVsize, 50)$ IVs when tested against D_{100}

Whilst the CART required a depth of 7 to assert dominance amongst the other algorithms, once it did it more than superseded the classification accuracy of the others; this algorithm is implemented into the SBD module. In this scenario more models were prone to overfitting than in the ABD module, with MNLR producing 3 outliers above the upper fence; showing the signaturing of packets as a harder task.

5.5 Static Rule Based Detection

The purpose of implementing static rule based detection is to enable companies to modify the underlying logic of the NIDPS to meet their specific needs; show how this system could generalise. After recieving a packet from NetfilterQueue the system iterates over the ruleset until a match is found or the the search of the list is complete; to reduce latency commonly matched rules should be placed near the top of the list.

5.5.1 Rule Structure

```
protocol src sport >>> dst dport

protocol = ip | tcp | udp | icmp
src = validip | !validip
sport = validport | !validport
dst = validip | !validip
dport = validport | !validport

validip = any | public | x.x.x.x/y
validport = any | 0 - 65535

Example: tcp 10.10.10.0/24 any >>> 10.10.3.0/24 502
```

Listing 2: Configuring a static rule

5.5.2 Rulesets used

ε = Alert on incoming ICMP packets from public and outgoing ICMP packets to public
μ = Alert public incoming UDP packets

Ruleset 1 – Block public incoming ICMP, UCP, and TCP traffic
Ruleset 2 – ε, μ, alert on public incoming TCP packets with the destination 10.10.(1|2|3).101
Ruleset 3 – ε, μ, alert on TCP traffic from public to internal ICS, alert traffic on port 22 coming from public IPs
Ruleset 4 – ε, μ, alert on TCP traffic to/from public IP addresses

Syntax for representations can be found in Appendix C

5.5.3 Results

Static Rules	Ruleset 1		Ruleset 2		Ruleset 3		Ruleset 4	
Dataset	Accuracy	κ	Accuracy	κ	Accuracy	κ	Accuracy	κ
A_{100}	0.488	0.185	0.537	0.232	0.546	0.242	0.770	0.524
B_{100}	0.661	0.321	0.693	0.385	0.699	0.398	0.847	0.694
C_{100}	0.844	0.486	0.855	0.531	0.859	0.547	0.950	0.861
D_{100}	0.844	0.485	0.856	0.532	0.859	0.546	0.951	0.862

Table 8: Final Static Ruleset Module Results

As shown in Table 8 the best performing ruleset was Ruleset 4 as it provides the best defence strategy against remote attackers. In addition to this Table 12 displays the low chance of FN in this static ruleset. Upon further analysis the source of FN in this case was an embedded ICMP layer in NTP packets containing ICMP Destination Unreachable messages. Whilst no

Actual			Prediction		
Nature	Class	Signature	Normal	Malicious	Accuracy
Normal	Operational	NTP	73	27	0.730
Malicious	Injection	ACIA1	436	0	0.000
		ACIA2	676	0	0.000
		ACIA3	1027	0	0.000
		ACIA4	1234	0	0.000
		ARIA1	424	0	0.000
		ARIA2	684	0	0.000
		ARIA3	752	0	0.000
		ARIA4	885	0	0.000

Table 9: Confusion matrix for static ruleset 4, rows where Accuracy == 1 omitted. For full matrix see Appendix D

advanced injection attacks were detected by this module it provides a useful starting layer in which to build the voting matrix of the NIDPS.

6 Testing

6.1 Offline Testing

Label	Case Block Configuration								U_{100}			
	1	2	3	4	5	6	7	8	Accuracy	F_1	κ	Safety Score
α	0	1	0	1	1	1	1	1	0.993	0.999	0.997	0.993
β	0	0	0	1	1	1	1	1	0.980	0.990	0.963	0.987
γ	0	1	0	1	0	1	1	1	0.967	0.982	0.926	0.980
Baseline	0	0	0	1	0	1	1	1	0.954	0.974	0.890	0.973
δ	0	0	1	1	0	1	1	1	0.954	0.970	0.875	0.973

Table 10: Offline analysis of the effectiveness of different block configurations

As identified in Section 2.6, specifically [3] in a hybrid NID(P)S an appropriate voting matrix is of paramount importance to ensure optimal performance. To aid in ascertaining the best blocking configuration for the NIDPS a new metric is defined to reward alerts when the blocking system fails. In Safety Score X = the ordered set of NIDPS verdicts and Y = the ordered set of correct behaviour for a packet.

$$score(x_{alert}, x_{block}, y_{block}) = \begin{cases} 1 & x_{block} = y_{block} \\ \frac{1}{2} & x_{block} = \text{False} \land x_{alert} = \text{True} \land y_{block} = \text{True} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

$$\text{Safety Score} = \frac{1}{|X|} \cdot \sum_{\substack{x_a, x_b \in X \\ y_b \in Y}} score(x_a, x_b, y_b)$$

With respect to the baseline configuration the following block configurations were altered to true, as in Table 10:
γ (Case 2) – Considering the NIDPS in the optimal case; assume that the rules that have been configured are correct, however they may be sparse, furthermore the ABD module, proven less effective at analysing traffic nature than SBD at signaturing traffic, assume SBD is correct
δ (Case 3) – ABD proven to be less effective than SBD module in Table 4 Table 7
β (Case 5) – In Table 12 static ruleset 4 was correct in nearly all cases; should provide few FN results.
α (Case 2 + 5) – Combine the successes of γ and β relative to the baseline to create a better block configuration.

Multiprocessing:
Note: multiprocessing yielded a an average per packet latency addition of 115.983 ms, compared to the average added latency of 3.239 ms for the single threaded variant. This is most likely the case because processing the packet in the NIDPS is relatively inexpensive operation

in comparison to the Inter-Process Communciation overhead of spawning a process in Python.

6.2 Online Testing

In this investigation a TP result is one that is normal and is predicted normal, therefore a FP (Type 1) result is one where the nature of the traffic is malicious and it is classified as normal, whereas a FN (Type 2) error is one that incorrectly classifies normal traffic. As stated in Section 5.3.1 and Section 6.1 F_1 score can be interpreted as a measure of the flow of normal data relative to packets incorrectly classified as a result of Type 1 and Type 2 errors.

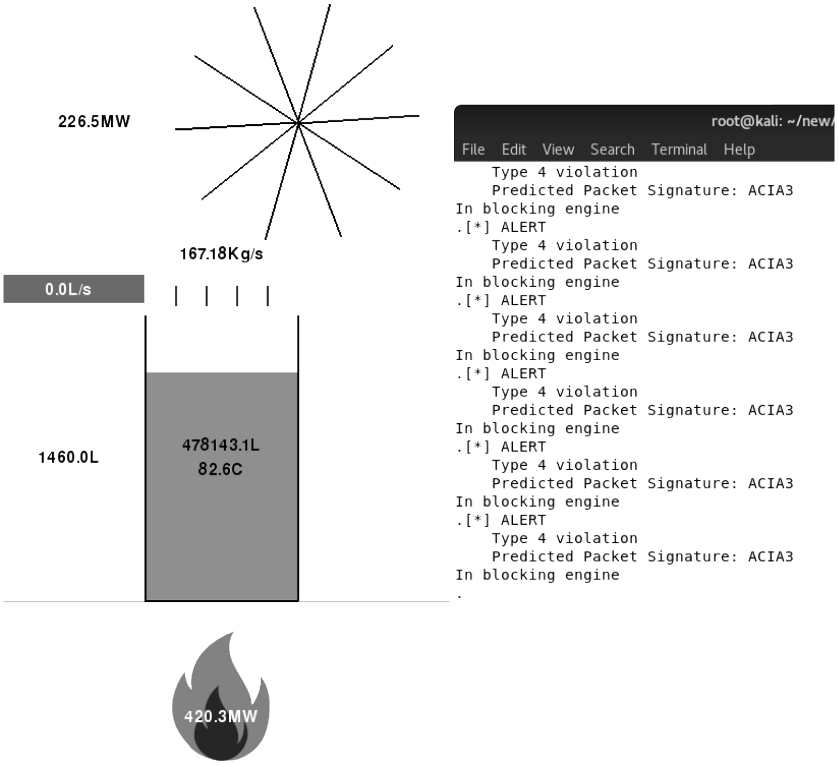
Although this metric gives a useful indication of how flow is impacted by the classification criteria of the model, it was found that, when running critical processes under TCP using configuration α, the generation of Type 2 errors was caused more harm than good, compromising the availability of the ICS as TCP connections timed out following multiple incorrectly classified instances.

Upon further inspection, the confusion matrix of α (see Table 13) showed that whilst the accuracy of the system was superior to the next best alternative β, a larger amount of Type 2 errors were inflicted on D_{100}.

Blocking configuration β showed more promise on implementation as during the session no timeout occurred and it was able to block ACIA class attacks and limit potential damages to the ICS, an example of the output is shown in Whilst however this does prevent a vector for an extremely effective DoS as systems are shut down, this can be seen in 10

Whilst networked attacks were blocked effectively during qualitative analysis of the NIDPS, and the detection of ARIA was usually successful, prevention of ARIA requires further consideration to be appropriately treated, as the response to ARIA that should be taken by a NIDPS is complete system shutdown rather than a pause on control input as seen with ACIA. This is due to the nature of ARIA masking the state of the ICS so no device will be able to accurately feed commands to it; all input would be guesswork. The best case scenario for handling ARIA (leading to a system DoS) is the termination of device connections in the following order MM, MServ (sanitise), wait for MSlav to act on the new state of MServ (not guaranteed as this involves commands passing through the compromised connection), then terminate all other connections. Any other execution pattern risks MSlav being left in an unsafe state that would cause all water to evaporate or an overflow. This is the scenario for a TN result. If a FN result was interpreted (likely to happen at some point given the accuracy of the modules being used) and this system kill execution pattern was used, the system will suffer a DoS for no reason.

As a result of this ARIA packets are added as an exception that get flagged only; overriding all case verdicts.



```
    Type 4 violation
    Predicted Packet Signature: ACIA3
In blocking engine
.[*] ALERT
    Type 4 violation
    Predicted Packet Signature: ACIA3
In blocking engine
.[*] ALERT
    Type 4 violation
    Predicted Packet Signature: ACIA3
In blocking engine
.[*] ALERT
    Type 4 violation
    Predicted Packet Signature: ACIA3
In blocking engine
.[*] ALERT
    Type 4 violation
    Predicted Packet Signature: ACIA3
In blocking engine
.[*] ALERT
    Type 4 violation
    Predicted Packet Signature: ACIA3
In blocking engine
.
```

Figure 10: Live NIDPS

7 Evaluation

Overall, this project has provided a useful insight into the viability of implementing a NIDPS, whose logic is trained using ML, in critical ICSs. In 99.3% of cases, the amalgamation of simple and complex classifiers is able to ascertain traffic nature allowing/blocking it as appropriate. However, in the case of CNI 99.3% accuracy is unsatisfactory; the incorrect classification of traffic, and compromise of availability, in an ICS has the potential to cause more damage than an attack itself.

In the case of misclassified traffic, in the domain of ICSs, Type 2 errors proved the most detrimental to system availability, causing both system downtime of the ICS and the persistence of incorrect, unsafe states. Advanced attacks on ICS require a lot of time, resources and in the case of this model insider access to implement. Whilst this NIDPS can detect such malicious insider threats, including command and response injection into the holding registers of the PLC, this type of system can only offer a kill switch to prevent command injection, and cannot guarantee protection against response injection. This combined with statistically unlikely, but over a long periods inevitable, Type 2 error generation show that the current implementation using the aforementioned algorithms lacks the accuracy required to run in an ICS. Whilst this is the case regarding the algorithms used in this paper, as the fields of ML and AI advance, a layered model similar to this may one day meet the requirements of CNI well enough to be safely implemented.

Actual			α			β		
Nature	Class	Signature	Normal	Malicious	Accuracy	Normal	Malicious	Accuracy
Normal	Operational	IPv6 TCP	20	20	0.5000	40	0	1.0000
		NTP	148	52	0.7400	148	52	0.7400
		Modbus	105175	40	0.9996	105204	11	0.9999
		TCP	17442	60	0.9966	17498	4	0.9998
	Technical	SSH	1245	9	0.9928	1252	2	0.9984
Malicious	Injection	ACIA3	0	1427	1.0000	128	1299	0.9103
		ACIA4	5	1729	0.9971	1676	58	0.0334
		ARIA1	0	599	1.0000	23	576	0.9610
		ARIA2	4	955	0.9958	39	920	0.9593
		ARIA3	12	1040	0.9886	42	1010	0.9601
		ARIA4	0	1235	1.0000	413	822	0.6656
	Recon	PLC Scan	1	2117	0.9995	1	2117	0.9995

Table 11: Confusion matricies of block configurations α and β, rows where Accuracy of α and β == 1 omitted. For full matrix see Appendix E

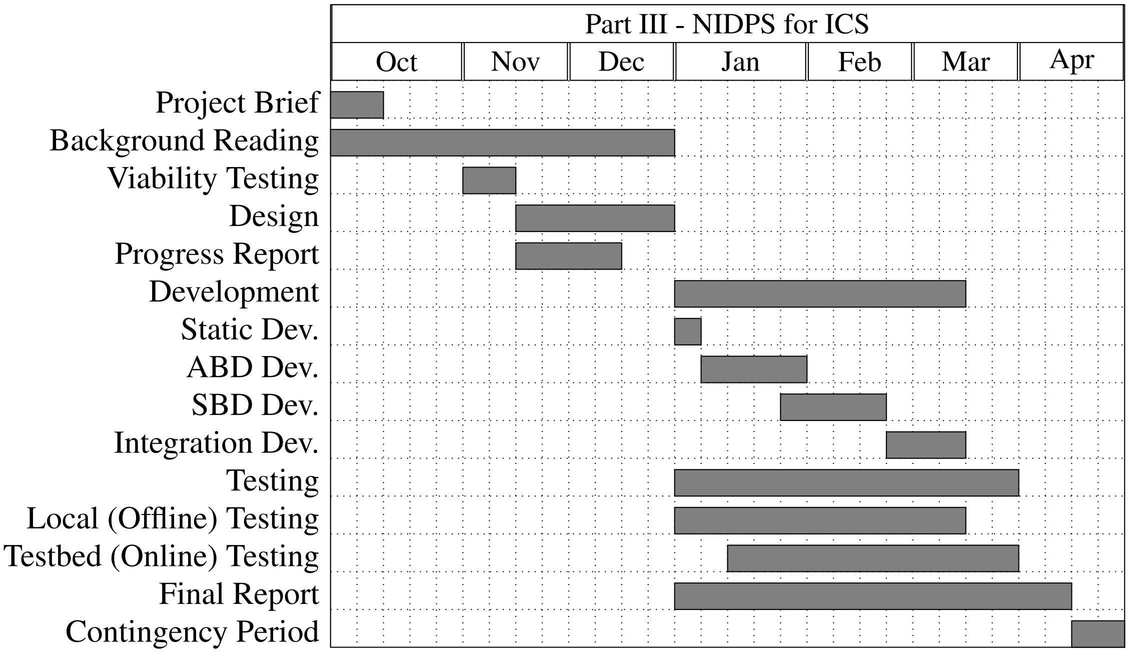
With the increased automation of the world that we live in, it is certain that the use of ICSs will increase, with which more CNI will become connected to the Internet. As a result of this technological revolution, it has never been more important to use new, innovative techniques to secure both legacy and future control systems to ensure that economic, political and social damage is not inflicted on us technology that we have become so reliant upon.

8 Further Proposals

The focus of this project was on building a NIDPS for ICS, the following proposals detail how the current project could be extended in the future:

- Apply PCA to the MLP in the ABD to see how it performs and improve execution times. (To classify 1 packet the MLP was up to 3 times slower than CART)
- Add support for IPv6 addressing
- Add a novel method for alerts to be conveyed wirelessly [43]
- This attempt involves stateless security, would keeping track of previous packets improve the model
- Validate sets using cross-validation instead of percentage split.
- Create a distributed IDS network so attacks can be learnt and signatures shared autonomously among the network.

9 Gantt Chart



10 References

[1] Shodan, “Shodan - Industrial Control Systems,” 2017. [Online]. Available: https://www.shodan.io/report/btZPXVti

[2] K. Stouffer, J. Falco, and K. Scarfone, “Guide to industrial control systems (ICS) security,” *NIST special publication*, vol. 800, no. 82, p. 16, 2011.

[3] O. Depren, M. Topallar, E. Anarim, and M. K. Ciliz, “An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks,” *Expert Systems with Applications*, vol. 29, no. 4, pp. 713–722, 2005.

[4] E. Gallo, “Jean Simon Bonnemain (1743-1830) and the Origins of Hot Water Central Heating,” in *Proceedings of the Second International Congress on Construction History*. Construction History Society, 2006, pp. 1043–1060.

[5] R. A. Gupta and M.-Y. Chow, “Networked control system: Overview and research trends,” *IEEE transactions on industrial electronics*, vol. 57, no. 7, pp. 2527–2535, 2010.

[6] K. T. Erickson, “Programmable logic controllers,” *IEEE potentials*, vol. 15, no. 1, pp. 14–17, 1996.

[7] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolff, “A brief history of the Internet,” *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 5, pp. 22–31, 2009.

[8] A. D. Papadopoulos, A. Tanzman, R. A. Baker Jr, R. G. Belliardi, and D. J. W. Dube, “System for remotely accessing an industrial control system over a commercial communications network,” may 2000.

[9] R. Bodenheim, J. Butts, S. Dunlap, and B. Mullins, “Evaluation of the ability of the Shodan search engine to identify Internet-facing industrial control devices,” *International Journal of Critical Infrastructure Protection*, vol. 7, no. 2, pp. 114–123, 2014.

[10] S. Karnouskos, “Stuxnet worm impact on industrial cyber-physical system security,” in *IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society*. IEEE, 2011, pp. 4490–4494.

[11] R. M. Lee, M. J. Assante, and T. Conway, “German Steel Mill Cyber Attack,” 2014. [Online]. Available: https://ics.sans.org/media/ICS-CPPE-case-Study-2-German-Steelworks_Facility.pdf

[12] A. Soullié, “Industrial Control Systems : Pentesting PLCs 101,” *BlackHat EU 14*, vol. 1, 2014.

[13] T. Morris, R. Vaughn, and Y. Dandass, “A retrofit network intrusion detection system for MODBUS RTU and ASCII industrial control systems,” in *System Science (HICSS), 2012 45th Hawaii International Conference on*. IEEE, 2012, pp. 2338–2345.

[14] T. H. Morris and W. Gao, "Industrial control system cyber attacks," in *Proceedings of the 1st International Symposium for ICS & SCADA Cyber Security Research*, 2013, p. 22.

[15] B. Drury, *Control Techniques Drives and Controls Handbook (2nd Edition)*. Institution of Engineering and Technology, 2009.

[16] R. Langner, *Robust Control System Networks: How to Achieve Reliable Control After Stuxnet*. Momentum Press, 2011.

[17] P. Théron, *Critical Information Infrastructure Protection and Resilience in the ICT Sector*. IGI Global, 2013. [Online]. Available: https://books.google.co.uk/books?id=P8ueBQAAQBAJ

[18] E. J. M. Colbert and A. Kott, *Cyber-security of SCADA and Other Industrial Control Systems*, ser. Advances in Information Security. Springer International Publishing, 2016. [Online]. Available: https://books.google.co.uk/books?id=4ZTlDAAAQBAJ

[19] R. Bace and P. Mell, "NIST special publication on intrusion detection systems," NIST, Tech. Rep., 2001.

[20] X. Wang, D. S. Reeves, S. F. Wu, and J. Yuill, "Sleepy Watermark Tracing: An Active Network-Based Intrusion Response Framework." in *Sec*. Springer, 2001, pp. 369–384.

[21] P. A. Porras and A. Valdes, "Live Traffic Analysis of TCP/IP Gateways." in *NDSS*, 1998.

[22] W. Jardine, S. Frey, B. Green, and A. Rashid, "SENAMI: Selective Non-Invasive Active Monitoring for ICS Intrusion Detection," in *Proceedings of the 2Nd ACM Workshop on Cyber-Physical Systems Security and Privacy*, ser. CPS-SPC '16. New York, NY, USA: ACM, 2016, pp. 23–34.

[23] Cisco, "Snort," 2017. [Online]. Available: https://www.snort.org

[24] MIT, "Kdd cup 1999," October 2007. [Online]. Available: https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[25] L. Portnoy, E. Eskin, and S. Stolfo, "Intrusion detection with unlabeled data using clustering," in *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001*, 2001, pp. 5–8.

[26] V. A. Golovko, L. U. Vaitsekhovich, P. A. Kochurko, and U. S. Rubanau, "Dimensionality reduction and attack recognition using neural network approaches," in *Neural Networks, 2007. IJCNN 2007. International Joint Conference on*. IEEE, 2007, pp. 2734–2739.

[27] S.-J. Horng, M.-Y. Su, Y.-H. Chen, T.-W. Kao, R.-J. Chen, J.-L. Lai, and C. D. Perkasa, "A novel intrusion detection system based on hierarchical clustering and support vector machines," *Expert Systems with Applications*, vol. 38, no. 1, pp. 306–313, 2011.

[28] L. M. Ibrahim, D. T. Basheer, and M. S. Mahmod, "A comparison study for intrusion database (kdd99, nsl-kdd) based on self organization map (som) artificial neural network," *Journal of Engineering Science and Technology*, vol. 8, no. 1, pp. 107–119, 2013.

[29] J. McHugh, “Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory,” *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 4, pp. 262–294, Nov. 2000.

[30] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” in *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*. IEEE, 2009, pp. 1–6.

[31] B. Bencsáth, G. Pék, L. Buttyán, and M. Félegyházi, “The cousins of stuxnet: Duqu, flame, and gauss,” *Future Internet*, vol. 4, no. 4, pp. 971–1003, 2012.

[32] S. Kumar, A. Viinikainen, and T. Hamalainen, “Machine learning classification model for Network based Intrusion Detection System,” in *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, 2016, pp. 242–249.

[33] ModbusPal, “Modbuspal - a java modbus simulator,” 2015. [Online]. Available: https://sourceforge.net/projects/modbuspal/

[34] M. Fox, “Netfilterqueue,” 2018. [Online]. Available: https://github.com/kti/python-netfilterqueue

[35] P. Biondi, “Scapy: explore the net with new eyes,” EADS Corporate Research Center, Tech. Rep., 2005.

[36] VectorsMarket, “Flame - free nature icons,” 2018. [Online]. Available: https://www.flaticon.com/free-icon/flame_426833

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[38] C. F. Dormann, J. Elith, S. Bacher, C. Buchmann, G. Carl, G. Carré, J. R. G. Marquéz, B. Gruber, B. Lafourcade, P. J. Leitão *et al.*, “Collinearity: a review of methods to deal with it and a simulation study evaluating their performance,” *Ecography*, vol. 36, no. 1, pp. 27–46, 2013.

[39] Z. Tan, A. Jamdagni, X. He, and P. Nanda, “Network intrusion detection based on lda for payload feature selection,” in *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*. IEEE, 2010, pp. 1545–1549.

[40] P. Laskov, P. Düssel, C. Schäfer, and K. Rieck, “Learning intrusion detection: supervised or unsupervised?” in *International Conference on Image Analysis and Processing*. Springer, 2005, pp. 50–57.

[41] M. Panda and M. R. Patra, “Network intrusion detection using naive bayes,” *International journal of computer science and network security*, vol. 7, no. 12, pp. 258–263, 2007.

[42] F. Chollet *et al.*, “Keras,” 2015. [Online]. Available: https://keras.io

[43] K. Khakpour and M. Shenassa, “Industrial control using wireless sensor networks,” in *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*. IEEE, 2008, pp. 1–5.

A Original Project Brief

Problem

A modern Industrial Control System (ICS) can be defined as a Cyber-Physical System (CPS) that uses networked computers and sensors to automatically control and monitor industrial processes. Such processes consist of anything from general manufacturing to water treatment.

Both the concept of an ICS and the invention of the first Programmable Logic Controller (PLC) precede the first transmission of a message over the Internet (then ARPANET); therefore in this period ICSs, were air-gapped networks. By being in a "safe" environment it would take a remarkable effort, including physical access, to exploit a system. This remarkable effort came in the form of Stuxnet, a malicious worm, designed to interrupt Iran?s nuclear program. Stuxnet was a pivotal moment for the cyber security community as it highlighted the need for defence against skilled attackers in ICS environments.

As increasing numbers of devices have become interconnected via the Internet so have ICSs. Shodan.io [1] identifies 100,000 Internet connected ICSs at the time of writing, presenting many avenues for the remote exploitation of control systems, such attacks having the potential for catastrophic damage.

To attempt to rectify the state of ICS security, NIST published a guide [2] detailing methods to secure ICSs. One such method was the implementation of Intrusion Detection/Prevention Systems (IDSs/IPSs). These IDSs/IPSs can be categorised as either:

- Signature based – requiring predefined rules to determine what constitutes an intrusion; meaning that exploitation via new methods or valid commands could still occur.
- Anomaly detecting – based off the notion of normal traffic, abnormal traffic is detected/blocked, potentially catastrophic in the instance of false positive errors.

Goals

This project aims to build a layered Network Intrusion Detection and Prevention System (NIDPS), similar to that of Depren et al. [3], incorporating a signature based IDS, an anomaly detecting IDS and a known attack classification tree, using a voting system to determine threats. From this the NIDPS will be able to accept and relay or reject and drop valid/invalid traffic; protecting the ICS from malicious control traffic.

In order to achieve the aforementioned goal the following will be required:

- Research the inner workings of ICSs and their common threats; providing a deeper understanding of how the NIDPS might parse and relay/not relay network traffic within the given ICS.
- The design and implementation of a NIDPS capable of meeting the real-time requirements of an ICS; having little trade-off between this and system security.

- Development and implementation of a standard behaviour for the ICS representation that the IDS will be protecting, along with attack scenarios that will attempt to bypass the NIDPS in order to control the ICS.
- Virtual testing, then deployment the NIDPS into the ICS environment firstly operating under normal behaviour, then in a scenario where it is under attack.
- Evaluation of the effectiveness of the NIDPS, noting any room for future improvement.

Scope

The focus of the project will be on active network traffic analysis within the ICS setting. To evaluate the effectiveness of the NIDPS an environment representative of an ICS will be used; this will provide a realistic scenario that exposes the true performance of the NIDPS.

B Archive Tree

```
root
├── ICS Live
│   ├── Attacker
│   ├── Devops
│   ├── Gateway
│   ├── Injection
│   ├── MLModules
│   ├── Master
│   ├── NIDPS
│   └── Simulation
├── NIDPS
│   ├── DetectionModules
│   └── optimalmodels
└── Results
    ├── ABD
    └── SBD
```

C All Static Rulesets

```
//proto src sport >>> dst dport
//Static Ruleset 1
icmp public any >>> any any
udp public any >>> any any
tcp public any >>> any any

//Static Ruleset 2
icmp public any >>> any any
icmp any any >>> public any
udp public any >>> any any
tcp public any >>> 10.10.1.101/32 any
tcp public any >>> 10.10.2.101/32 any
tcp public any >>> 10.10.3.101/32 any

//Static Ruleset 3
icmp public any >>> any any
icmp any any >>> public any
udp public any >>> any any
tcp public any >>> 10.10.1.0/24 any
tcp public any >>> 10.10.2.0/24 any
tcp public any >>> 10.10.3.0/24 any
tcp public any >>> 10.10.4.0/24 any
tcp public any >>> any 22

//Static Ruleset 4
icmp public any >>> any any
icmp any any >>> public any
udp public any >>> any any
tcp any any >>> public any
tcp public any >>> any any
```

Listing 3: Static rules used

D Complete Confusion Matrix Static Ruleset 4

Actual			Prediction		
Nature	Class	Signature	Normal	Malicious	Accuracy
Normal	Operational	ARP	1132	0	1.000
		DNS	2408	0	1.000
		IPv6 TCP	20	0	1.000
		NTP	73	27	0.730
		Modbus	75215	0	1.000
		TCP	12502	0	1.000
	Technical	Diagnostic ICMP	914	0	1.000
		SSH	904	0	1.000
Malicious	Injection	SCIA1	0	972	1.000
		SCIA2	0	1669	1.000
		SCIA3	0	2451	1.000
		SCIA4	0	3195	1.000
		SRIA1	0	1021	1.000
		SRIA2	0	1779	1.000
		SRIA3	0	2483	1.000
		SRIA4	0	3296	1.000
		ACIA1	436	0	0.000
		ACIA2	676	0	0.000
		ACIA3	1027	0	0.000
		ACIA4	1234	0	0.000
		ARIA1	424	0	0.000
		ARIA2	684	0	0.000
		ARIA3	752	0	0.000
		ARIA4	885	0	0.000
	DoS	Malicious ICMP	0	993	1.000
		Smurf	0	772	1.000
		SYN Flood	0	731	1.000
		UDP Flood	0	773	1.000
	Recon	Amap	0	1549	1.000
		Nmap	0	1499	1.000
		PLC Scan	0	1518	1.000
	Brute Force	Malicious SSH	0	492	1.000
		Malicious Telnet	0	494	1.000

Table 12: Confusion matrix for static ruleset 4

E Complete Blocking Matrices

Actual			α			β		
Nature	Class	Signature	Normal	Malicious	Accuracy	Normal	Malicious	Accuracy
Normal	Operational	ARP	1592	0	1.0000	1592	0	1.0000
		DNS	3358	0	1.0000	3358	0	1.0000
		IPv6 TCP	20	20	0.5000	40	0	1.0000
		NTP	148	52	0.7400	148	52	0.7400
		Modbus	105175	40	0.9996	105204	11	0.9999
		TCP	17442	60	0.9966	17498	4	0.9998
	Technical	Diagnostic ICMP	1264	0	1.0000	1264	0	1.0000
		SSH	1245	9	0.9928	1252	2	0.9984
Malicious	Injection	SCIA1	0	1382	1.0000	0	1382	1.0000
		SCIA2	0	2369	1.0000	0	2369	1.0000
		SCIA3	0	3451	1.0000	0	3451	1.0000
		SCIA4	0	4495	1.0000	0	4495	1.0000
		SRIA1	0	1431	1.0000	0	1431	1.0000
		SRIA2	0	2479	1.0000	0	2479	1.0000
		SRIA3	0	3483	1.0000	0	3483	1.0000
		SRIA4	0	4596	1.0000	0	4596	1.0000
		ACIA1	0	611	1.0000	0	611	1.0000
		ACIA2	0	951	1.0000	0	951	1.0000
		ACIA3	0	1427	1.0000	128	1299	0.9103
		ACIA4	5	1729	0.9971	1676	58	0.0334
		ARIA1	0	599	1.0000	23	576	0.9610
		ARIA2	4	955	0.9958	39	920	0.9593
		ARIA3	12	1040	0.9886	42	1010	0.9601
		ARIA4	0	1235	1.0000	413	822	0.6656
	DoS	Malicious ICMP	0	1393	1.0000	0	1393	1.0000
		Smurf	0	1072	1.0000	0	1072	1.0000
		SYN Flood	0	1031	1.0000	0	1031	1.0000
		UDP Flood	0	1073	1.0000	0	1073	1.0000
	Recon	Amap	0	2149	1.0000	0	2149	1.0000
		Nmap	0	2099	1.0000	0	2099	1.0000
		PLC Scan	1	2117	0.9995	1	2117	0.9995
	Brute Force	Malicious SSH	0	692	1.0000	0	692	1.0000
		Malicious Telnet	0	694	1.0000	0	694	1.0000

Table 13: Confusion matricies of block configurations α and β