A Constraint-based Intrusion Detection System

MD Siam Hasan
Queen's University
Kingston, Canada
hasan@cs.queensu.ca

Thomas Dean
Queen's University
Kingston, Canada
trd@queensu.ca

Fahim T. Imam
Queen's University
Kingston, Canada
imam@cs.queensu.ca

Francisco Garcia
Universidad Complutense de Madrid
Madrid, Spain
franci04@ucm.es

Sylvain P. Leblanc
Royal Military College of Canada
Kingston, Canada
sylvain.leblanc@rmc.ca

Mohammad Zulkernine
Queen's University
Kingston, Canada
mzulker@cs.queensu.ca

ABSTRACT

The expressiveness of constraints has a potential to define network behavior and defend against complex network intrusions. This potential can be an integral part of an Intrusion Detection System (IDS) for defending networks against various attacks. The existing approaches of constraint logic programming have limitations when it comes to solving the network constraints in the presence of the continuous, constantly changing stream of network data. In this paper, we propose two variations of a tree-based constraint satisfaction technique to evaluate network constraints on continuous network data. A Domain Specific Language (DSL) is developed so that the IDS users can specify different intrusions related to their networks. We also present a prototype implementation of these techniques. We evaluate the performance and effectiveness of our approach against the network traffic data generated from an experimental network.

CCS CONCEPTS

• **Security and privacy** → **Intrusion detection systems**; • **Software and its engineering** → *Constraint and logic languages*; • **Networks** → Formal specifications;

KEYWORDS

Intrusion Detection System, Constraint Satisfaction Problem, Domain Specific Language

ACM Reference format:
MD Siam Hasan, Thomas Dean, Fahim T. Imam, Francisco Garcia, Sylvain P. Leblanc, and Mohammad Zulkernine. 2017. A Constraint-based Intrusion Detection System. In *Proceedings of ECBS '17, Larnaca, Cyprus, August 31-September 1, 2017,* 10 pages.
https://doi.org/10.1145/3123779.3123812

ECBS '17, August 31-September 1, 2017, Larnaca, Cyprus

ACM ISBN 978-1-4503-4843-0/17/08…$15.00
https://doi.org/10.1145/3123779.3123812

1 INTRODUCTION

Developing an effective network-based IDS is a critical area of research in network security. The basic approaches to developing the network-based IDSs can be divided into two broad categories: the signature-based IDS, and the anomaly-based IDS [7]. Popular IDSs are capable of recognizing attacks with known signatures. However, their success against the new, sophisticated attacks is less certain. A complex attack can be executed through multiple packets, exploiting the dependencies between the packets. In some cases, these packets are spread non-consecutively across a large window of network traffic. However, that does not mean that the IDS users, i.e., the network security specialists are not aware of these attacks. The specialists in Special Weapons And Tactics (SWAT) network security team [5], for instance, are actively involved in monitoring and detecting intrusions based on any abnormal traffic behaviour. Detecting intrusions in that case is handled by human experts with no automated IDS. These shortcomings suggest that we need a new kind of IDS.

Conventional IDSs use imperative rules to detect the misuse of a network. They often use a ring buffer of predetermined size, and if a rule applies to two packets that are far apart, the rules must explicitly save, retrieve and delete packet data from a data store. We have shown in our previous research [10] that constraints are capable of expressing complex network vulnerabilities in private networks such as those used in air traffic control or industrial control systems. As contrasted with general public networks, these networks are isolated and only use a small, limited number of network protocols. The limited number of protocols allow us to use network constraints to mathematically define the normal behavior of the network. To be more specific, conditions between multiple packets can be specified in a network constraint, which the constraint engine can evaluate by automatically matching packet data as it is received, automating the management of the storage of packet data. Defining the well-formedness of a network traffic as a set of high-level attributes allows multiple intrusions to be covered in a single constraint. This allows the network security analyst to concentrate on the health of the network as opposed to identifying specific vulnerabilities.

In this paper, we propose a new type of IDS which solves constraints to identify intrusions. It should be noted that the framework was initially implemented in our previous research [10]. We present the enhancement of our original framework and propose two tree-based techniques for solving network constraints. The framework

is now capable of handling incoming packets and identify different patterns by relating their field values. In our initial framework, all the constraints were hard-coded as we were only interested in determining their validity. In this research, a Domain Specific Language (DSL) is designed so that the framework can generate solutions from that language. However, we only show the direct mapping between the DSL code and actual implementation of constraints. Generating source code from the DSL is kept for future research. The IDS is evaluated against the traffic generated from an experimental private network running the Data Distribution Services (DDS). During the evaluation, the emphasis is given on several issues such as the validity of the constraints, successful detection of constraint failure, required time to generate errors, resource usage, and the reliability of the IDS. The final result also includes a comparison between the two proposed tree-based techniques.

This IDS is expected to detect the intrusions defined by an IDS user. For example, we construct a constraint which ensures the legitimacy of two entities while they are communicating with each other within a network. If the constraint fails, we can suspect that one or both of the entities are potentially malicious hosts in that network. The combination of several constraints similar to this latter one makes our IDS powerful enough to monitor the overall health of the whole network.

Paper Organization: The rest of the paper is organized as follows. Section 2 gives an overview of our framework emphasizing the constraint engine. Section 3 provides the details behind the necessity of adopting a new constraint satisfaction technique. Section 4 describes the attributes of a constraint. Section 5 is about the life cycle of a network constraint. We explain the two proposed tree-based techniques (naive and optimized) in Section 6. The following two sections demonstrate three examples of the DSL that we have developed to express the network constraints along with the sample snippets of their implementation. The evaluation environment and the findings are presented in Section 9. Section 10 reviews the research related to this paper. Finally, the last section provides the conclusion and states our future plans for this research.

2 THE IDS FRAMEWORK

Figure 1 shows the framework of our IDS. In this paper, we focus on the development of the constraint engine that evaluates network constraints for identifying intrusions. The other modules in the framework help the engine to evaluate the network constraints. The parser of the framework validates the incoming packets from the network and passes those packets along with their contexts (e.g., IP address, Port number) to the entry points of the constraint engine. These entry points or callbacks will be generated automatically from the protocol specifications (future work). For messages of each protocol, we have separate entry point or callback function. An IDS user specifies the security concerns about a network as part of the Protocol Specs module. This high-level description of the constraints are converted into an intermediate, low-level constraint details by the ontological process. The Generator generates the actual constraint tree and the code to handle their life cycles based on the constraint data from the entry points. Currently, all the constraint trees are manually hard-coded in the engine. The evaluator checks the application specific environmental constraints. Using the environmental constraints, we ensure that the static behavior of elements in a DDS service (e.g, a publisher's topic ID) are not altered during the runtime of the IDS. The IDS that we have developed has two modes: checking and learning. In order to gather the information about the network architecture and the configuration settings, we run the IDS in a learning mode. In this mode, no constraints are evaluated and the information required to evaluate the environmental constraints is collected. Figure 1 shows the data flow between the Evaluator and the Environmental Information. In the learning mode, the evaluator stores the environmental information (red arrow) and uses the information in checking mode (black arrow). Based on the packet information and the environmental knowledge, the engine evaluates the hard-coded constraint trees. In case of a constraint violation, the evaluator generates an alert. We extend the specification of a constraint to include the life cycle and how the constraint binds to the network data.

3 CONSTRAINT SATISFACTION

In order to find a suitable constraint satisfaction technique, we focus on different kinds of search mechanisms in a Constraint Satisfaction Problem (CSP). Two types of search mechanisms are currently available: complete and incomplete. The complete algorithm searches the full solution space using backtracking. The incomplete algorithm performs a partial search on the solution space and has good results with Anti-Monotonicity (AM)[1] constraints. The computational time and memory management of the latter algorithm are appropriate for using constraints in a real-time environment. Another way to use the incomplete algorithm is the compact prefix tree structure which holds information about patterns. However, maintaining a large decision tree containing a high volume of frequent patterns requires managing a large amount of memory. We choose the tree-based technique for checking constraints with dynamic information. The IDS framework uses a model which has been developed for checking multi-packet constraints. One significant part of this feature is to dynamically search packets in a network traffic. Finding patterns or violating constraints on a continuous data stream is not a novel idea [23]. However, this technique has never been tried to check network constraints.

A naive approach for building optimal trees is to grow a full (accurate) tree and then employ an algorithm which is based on dynamic programming to prune away suboptimal portions of a tree until that constraint is satisfied. This is a time-consuming and memory inefficient technique and is not feasible for checking constraints based on dynamic information. Another drawback of this approach is to constantly update the search tree with incoming packet attributes.

An optimized version of the constraint tree is to create efficient propagators for all the constraints. The scope of a constraint on a particular node may contain both CSP variables and meta-variables of the children of that node. This type of tree is used to implement those constraints that are expressed as logical combinations of other constraints. The major challenge of applying this technique in a network is handling repeated variables. For keeping a constraint tree persistent, if a separate copy has to be created every single time, then the memory management becomes infeasible. Constant

[1]This type of constraint fails when a subset of the condition is not satisfied.

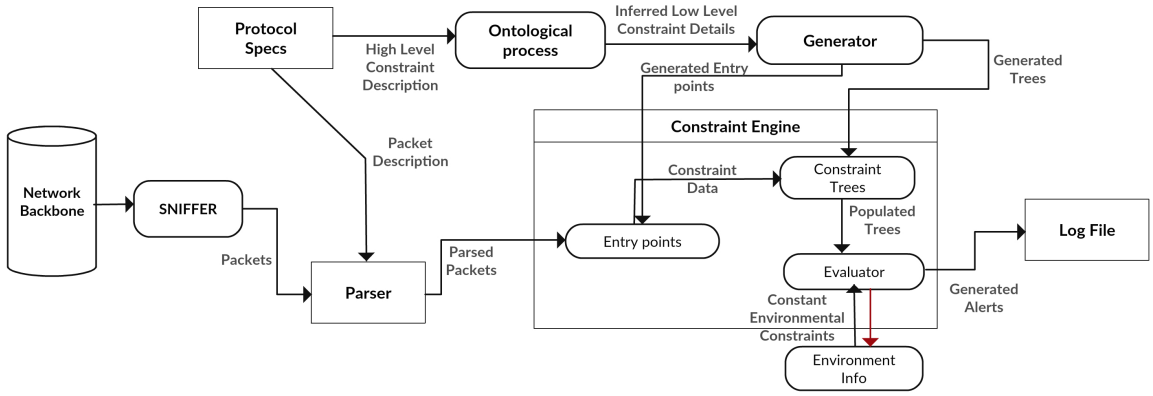

Figure 1: The Constraint-based IDS Framework.

updates of the same variables in different locations of the tree requires frequent memory operations. The solution to this problem is to reduce the number of frequent patterns by merging nearest constraints and to decrease the frequency of memory shuffling. When it comes to reducing the number of constraints, we can apply a few propagation algorithms such as the Generalized Arc Consistency (GAC) [19]. However, they are not effective to prune away a subdomain where variables are repeating. In this situation, constraints with polynomial-time GAC propagators turns into an NP-hard problem to enforce the GAC, for example, the Global Cardinality Constraint [16]. Besides these, no other optimization algorithm has been proposed which addresses our problem.

From the network security point of view, a tree-based constraint satisfaction technique can be compared with the state-based security testing on packets [26]. However, in a state-based approach, the scope of expressing multi-packet constraints is limited.

Apart from the dynamic rules [17] or events in existing IDS rule engines, a few research contributions can be found demonstrating the process of expressing vulnerabilities in consecutive packets. None of them provides a pragmatic solution for the proposed network constraints as they lack the ability to bind conditions between multiple packets and do not possess the liveness property [2].

4 ATTRIBUTES OF A CONSTRAINT

Environmental (EV): This is part of a constraint which checks information that is part of the deployment of an application. For example, in an Air Traffic Control (ATC) system, only the systems that act as the interfaces to the radar hardware may introduce radar data into the network system. Similarly, the flight plan information can only be originated from certain dedicated systems in the network. While this information may be manually entered based on the specific network architecture or from configuration files, the information may also be inferred when checking the constraints against a known, clean system.

Single Packet (SP): A single packet constraint can be compared to detecting signatures in a single packet envelope. Mostly this constraint satisfies hard rules set by the administrators. The tree of this type is short-lived and does not require the liveliness property [2].

Multi Packet (MP): A multiple packet constraint models the dependencies which occur between multiple packets. The packets are expected to arrive in a specific order. For example, when a network file needs to be modified, first thing the Network File System (NFS) does is to open the handle to that file. After getting a valid file handle, the number of bytes requested can be written to the file. In the end, the handle is gracefully terminated by NFS. During this full operation, a constraint can ensure that the file handle is kept open by the NFS. This type of constraint is used to detect the complex attack patterns. Trees for this type of constraint have a longer life, and are used to correlate data from packets that may be separated by many unrelated packets.

Kept-Alive (KA): Other than the EV constraint which must be kept for the run-time of the IDS, we need to define the living time of a constraint tree in order to manage the runtime environment of the IDS. Without this attribute, it is impossible to manage the memory operation, the run-time, and the space requirements of the IDS.

Chained Sequence (CS): Some of the constraints can be co-related and merged into a single one in order to improve the performance of the IDS. If the precondition of one constraint is already satisfied by another constraint, then they can be considered as the constraints with chained-sequence.

It should be noted that with the exception of the SP and MP, none of the attributes are mutually exclusive. Most of the constraints are multi-packet and kept-alive constraints.

5 LIFE CYCLE OF A CONSTRAINT

Finding patterns or pushing constraints in a continuous stream is not a novel idea [23]. However, this technique has never been tried to satisfy stand alone constraints. We adapt this approach by instantiating the trees as the packets are evaluated. We identify four steps in the life cycle of a network constraint.

Instantiate (I): Instantiate is the first step in the life cycle of a constraint tree. This is used when the first packet in the sequence or the only packet for a single packet constraint is encountered. A copy of the constraint tree is created and the information from the packet is added to the tree. The tree is then cached for two reasons. First, it helps find immediate next step in a constraint to

find the instance of the tree. Second, a duplicate tree is identified if a network entity restarts a sequence.

Bind (B): A bind step is triggered by the arrival of each packet in a multipacket sequence. The data relevant to a constraint is added to the packet, accumulating the relevant data for later use. If the constraint tree does not exist already, i.e., from the instantiate or previous bind step, a violation is logged.

Evaluate (E): Once all of the data from the packets are collected in the constraint tree, a constraint can be evaluated. A failure to evaluate the constraint indicates that something abnormal has occurred on the network. Similar to the bind step, the IDS generates errors if the constraint tree cannot be found in this step.

Destroy (D): This is the final step of a constraint life cycle. Based on the original constraint, we can determine that all the values stored in a constraint tree are no longer needed, and the tree can be removed from the cache. Destroying a constraint tree increases memory efficiency of the proposed techniques and reduces the look up operations of existing constraint trees in the cache.

6 TWO TREE-BASED TECHNIQUES

We propose two tree-based techniques for solving the constraints. In the naive technique, each constraint is represented as a constraint tree structure encoded in an array. Array indexes are used instead of pointers, allowing the tree to be entirely self-contained in the array.

Figure 2 shows how the tree skeleton of constraint **C5** looks in both techniques. This tree has four non-leaf and six leaf nodes. The naive technique needs to store the complete constraint tree compared to the optimized technique storing only four leaf nodes from the instantiate and bind steps of the constraint tree.

The instantiate step involves creating a copy of the array to which data is added from each packet that triggers a bind step. The constraint tree is evaluated by a recursive depth first traversal [11] of the tree. We compare the naive technique to an optimized implementation. In this case, the code for the evaluation step is generated automatically from a constraint specification. While the data may be dynamic, the constraints themselves are static. Instead of managing a full constraint tree, only an array representing the leaf nodes from the first two steps is allocated and cached in the instantiate step. The code representing a constraint is generated and used to evaluate the constraint using the data stored in the array representing the leaf nodes.

7 THE DSL CODE OF CONSTRAINTS

Earlier we proposed an optimized technique to satisfy a network constraint. However, we still do not show how the structure of a constraint is hard-coded beforehand. The constraints developed by us are descriptions of the intrusions in a private network running DDS based on two protocols: the Internet Group Management Protocol (IGMP) [9] and the Real-Time Publisher Subscriber Protocol (RTPS) [24]. These descriptions are made in natural language and can not be transformed into a solution. We therefore needed an intermediate DSL which can represent the constraints that can be converted directly into an actual source code structure. Since the optimized technique does not evaluate a constraint with a generic algorithm, the language should possess the ability to represent the life cycle of a constraint. During designing the DSL, we consider the following:

- The packet data structures passed by the parser
- The callbacks generated by the parser to identify a packet and message type
- The environmental data specific to a set of constraints.

The language we propose is a low-level specification, which is only used for an intermediate step for generating constraint tree and is not built with the intention of user expressiveness. We do consider the comprehensibility of this language. To explain the language better, we chose three network constraints developed in our previous work [10] and express them in our proposed DSL.

7.1 Valid Entities

The RTPS protocol [24] is a UDP protocol which is commonly used to implement the DDS. A general data exchange involves publisher and subscribers first sending an identification message called a participant message, followed by either a publisher or a subscriber message which identifies the topics that they either provide or require. The RTPS makes extensive use of multicasting in order to reduce the network traffic. Thus a publisher or a subscriber must first join a multicast group before they can establish a communication. The constraint recognizes this behavior is

All the subscribers and publishers must be valid members of at least the IGMP multicast group. These participants must send their membership reports to the group addresses (to join) before publishing or subscribing to in a topic - C5.

The DDS participants (publishers and subscribers) must first announce to the network that they are participants to a multicast address. In order to receive similar messages, the participants must also be members of the multicast group. Thus we instantiate this constraint when a participant sends IGMP messages to join a multicast group. The instantiated tree is cached key derived from the source IP address and the multicast address.

Constraint C5: Packet sequence

I : IGMP V2Report or V3Report
B : RTPS Participant
E : RTPS Publisher or Subscriber
D : IGMP V3Leave or V2Leave

The bind step is triggered when a participant message is received from the same machine (IP address) to the same multicast group. For this constraint, there is no additional information beyond the fact that the participant packet has been observed. If the constraint tree cannot be retrieved using the source and multicast address, then an IGMP join message was not previously received. This results in a constraint violation.

The evaluation step is triggered when either a publisher or subscriber packet is found. The constraint is used to detect spurious DDS packets that are not associated with the appropriate multicast group. When the host exits normally, it leaves the multicast group by sending an IGMP leave message. At this stage, the constraint tree is removed from the cache. The DSL code for this constraint is shown below:

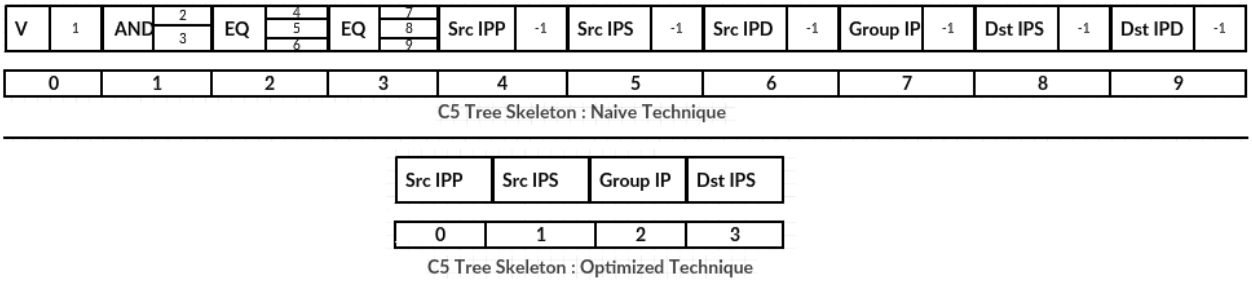

Figure 2: Constraint C5 Tree Skeleton.

```
CONSTRAINT C5
 V (AND (EQ (SrcIPJ, SrcIPP) EQ (GroupIPJ, DstIPP))
INSTATNTIATE
 IGMP Packet.Type is V2Member or
 Packet.Type is V3Member
 if V2Member
  SrcIPJ = Packet.SrcIP
  GroupIPJ = Packet.Groupaddress
  Key = Packet.SrcIP, Packet.Groupaddress
 endif
 if V3Member
  loop until Packet.groupRecordInfo
    SrcIPJ = Packet.SrcIP
    GroupIPJ = Packet.GroupRecord
    Key = Packet.SrcIP, Packet.GroupRecord
  end loop
 endif
BIND
 RTPS Packet.SUBMSG contains DATAPSUB
 SEARCH Packet.SrcIP, Packet.DstIP
 SrcIPP= Packet.SrcIP
 DstIPP= Packet.DstIP
 Key = Packet.SrcIP, Packet.Groupaddress
EVALUATE
 RTPS Packet.SUBMSG contains DATARSUB or
 Packet.SUBMSG contains DATAWSUB
 SEARCH Packet.SrcIP, Packet.DstIP
 EVAL Packet.SrcIP, Packet.DstIP
DESTROY
 IGMP Packet.Type is V2Leave or
 Packet.Type is V3Leave
 if V2Leave
  SEARCH Packet.SrcIP, Packet.Groupaddress
 endif
 if V3Leave
  loop Packet.GroupRecordInfo
    SEARCH Packet.SrcIP, Packet.Groupaddress
 endif
 END
```

7.2 Static Environment

While it is possible for a DDS publisher to change the topic it transmits, it is not normal in a private network. The interface to the radar in an ATC environment will not start sending flight plan information. In addition, the network configuration such as IP addresses also remains consistent. **C8** is an EV constraint. It fails whenever a publisher deviates from the behavior previously configured in the configuration settings of the ATC network.

A topic key is only published from a predefined set of publishers - C8.

Constraint C8: Packet sequence

I : Initialize
E : RTPS Publisher
D : Finalize

In this case, the constraints are checked against a known configuration that contains a list which associates the publishers IP address with the topic name and quality of service (QoS). A set of constraint tree is initialized from the configuration for each publisher that contains the topic names and QoS. When a publisher packet is encountered, the IP address, topic name, publisher Entity and QoS are compared against the list values.

```
CONSTRAINT C8
 V (AND (EQ (SrcIP) EQ (Topic) EQ(Entity) EQ(QoS))
INSTANTIATE Initialize
 SrcIP = FactC8.SrcIP
 Topic = HashKey(FactC8.TopicName)
 Entity = FactC8.entityID
 QoS = struct(FactC8.QoS)
 Key = FactC8.SrcIP,
 HashKey(FactC8.TopicName), FactC8.entityID
BIND
EVALUATE
 RTPS Packet.SUBMSG contains DATAWSUB
 EVAL Packet.SrcIP,
 HashKey(Packet.TOPICPARMS.TOPICNAME),
 Packet.entityID, struct(Packet.QoS)
DESTROY
END
```

To minimize manual intervention, the constraint engine can be run in a learning mode on a known clean environment in which the constraint records the information rather than validating the information.

7.3 Healthy Communication Channel

The RTPS protocol identifies the application data sent from a publisher to a subscriber using a writer entity key. This key is generated dynamically by the publisher each time the DDS application is run. The publisher packet links the topic name to the writer entity key. Thus to determine if a data packet from a particular IP address is legitimate, the IDS must have first cached the writer entity key from the previous publisher packet. In addition, the DDS framework may choose to transmit data packets directly using a unicast address instead of using a more general multicast address. Constraint **C11** is used to validate the communication channel between a publisher and a subscriber.

Data of certain topic is considered valid if it is produced from a valid publisher and consumed by a valid subscriber - C11.

Since either a publisher or a subscriber may broadcast first, both orders must be checked by the constraint. However, in both cases, the data is transferred via unicast communication. The constraint checks that the data carried by a unicast application data packet is a topic understood by both the publisher (as identified by the data packet source IP address) and the subscriber (as identified by the data packet destination IP address).

Constraint C11: Packet sequence

I : RTPS Publisher or Subscriber
B : RTPS Subscriber or Publisher
E : RTPS Unicast Data
D : IGMP V3Leave or V2Leave

This constraint is instantiated by either a publisher or a subscriber packet and stores the source address of the publisher or destination address of the subscriber into the constraint tree. For a publisher packet, the entity key is also stored in the constraint tree. The tree is cached using the publisher's IP address (available as the destination IP address of the subscriber's packet) and the topic name.

The bind step is triggered by the opposite packet of the instantiate step to insert either the subscriber's IP address or the publisher's IP address and entity key into the constraint tree. The bound tree is cached using a four element key: publisher's IP address, publisher's writer entity key, subscriber's IP address, and the topic name.

Data packets are used to evaluate the constraint tree. The source and destination IP addresses, topic name and entity key are used to retrieve the constraint tree for evaluation. The constraint succeeds if the entity key is the same used by a publisher when it established the unicast connection with a subscriber. The DSL code for this constraint is shown on the right.

```
CONSTRAINT C11
V (AND (EQ (SrcIPPu) (EQ (entityIDPu) (EQ (SrcIPSu))
INSTANTIATE
 RTPS Packet.SUBMSG contains DATAWSUB or
 RTPS Packet.SUBMSG contains DATARSUB
 if Packet.SUBMSG contains DATAWSUB
  if SEARCH HashKey(Packet.TOPICPARMS.TOPICNAME),
  Packet.DstIP
   BINDCALL Packet.SrcIP, Packet.entityID
  else
   SrcIPPu = Packet.SrcIP
   entityIDPu = Packet.entityID
  endif
  Key = HashKey(Packet.TOPICPARMS.TOPICNAME),
  Packet.SrcIP
 endif
 if Packet.SUBMSG contains DATARSUB
  if SEARCH HashKey(Packet.TOPICPARMS.TOPICNAME),
  Packet.DstIP
   BINDCALL Packet.SrcIP
  else
   SrcIPSu = Packet.SrcIP
  endif
  Key=HashKey(Packet.TOPICPARMS.TOPICNAME),
  Packet.DstIP
 endif
BIND
 RTPS Packet.SUBMSG contains DATAWSUB or
 RTPS Packet.SUBMSG contains DATARSUB
 if RTPS Packet.SUBMSG contains DATARSUB
  SEARCH HashKey(Packet.TOPICPARMS.TOPICNAME),
  Packet.DstIP
  SrcIPSu = Packet.SrcIP
  Key=HashKey(Packet.TOPICPARMS.TOPICNAME),
  Packet.SrcIP
 endif
 if RTPS Packet.SUBMSG contains DATAWSUB
  SEARCH HashKey(Packet.TOPICPARMS.TOPICNAME),
  Packet.DstIP
  SrcIPPu = Packet.SrcIP
  entityIDPu = Packet.entityID
  Key=HashKey(Packet.TOPICPARMS.TOPICNAME),
  Packet.DstIP
 endif
EVALUATE
 RTPS Packet.SUBMSG contains DATASUB
 if IsUnicastComm(Packet.SrcIP,Packet.DstIP)
  SEARCH HashKey(Packet.TOPICPARMS.TOPICNAME),
  Packet.SrcIP
  EVAL Packet.SrcIP,Packet.entityID,Packet.DstIP
 endif
DESTROY
 IGMP Packet.Type is V2Leave or
 IGMP Packet.Type is V3Leave
 SEARCH Packet.SrcIP
END
```

We show that our proposed constraints are multi-packet constraints dependent on multiple protocols. Since these constraints are not similar, it can be deduced that the proposed DSL is capable of expressing all the network constraints related to DDS mechanism in a private network.

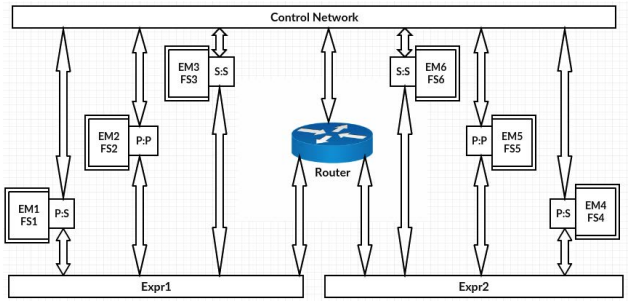

Figure 3: The Simulator Architecture.

8 THE DSL AND SOLUTION CODE

Figure 4 corresponds to the C code that handles the constraint **C11**. As mentioned already, the solution is yet to be generated from the DSL, we only show that the DSL code can be directly mapped to the constraints solution code.

The constraint tree is instantiated at Line 8 or 11 when the IDS receives a publisher or a subscriber packet. Both functions can jump to bind step if the tree is already instantiated by the counterpart entity. Line 15, 16, and 17 show that **C11** is only evaluated when the IDS receives a *DATA* packet and the data is transferred from a publisher to a subscriber via unicast communication. Figure 4 also shows the C code for evaluating constraint **C11** at Line 21. It starts with looking for an already bound constraint tree in the cache at Line 24. Later, at Line 25, 26, and 27 the values from the packet are matched with the leaf nodes of the tree. If it does not match, then the constraint is considered to be violated. Finally, at Line 35 and 41, we destroy the constraint tree when the IDS encounters a *V2Leave* or *V3Leave* message.

9 EVALUATION

In order to evaluate our approach, we developed an experimental network which acts as a simulator of an ATC system. Experiments are conducted on several PCAP[2] files captured from the simulator. We test the IDS with three constraints described earlier.

9.1 Environment

The Canadian Automated Air Traffic Control System (CAATS)[3] uses the RTPS protocol to share data between devices, controlers and ATC centers. We adapted the Euroscope ATC simulator[6] to use the RTPS protocol[15]. Figure 3 shows the simulator as deployed at Queen's University. The network architecture is built based on the proposal from Lemay et al. [13]. Six DDS machines *FS1*, *FS2*, *FS3*, *FS4*, *FS5*, and *FS6* are connected by three networks. One network, labeled *control*, connects to all the machines and the router and is used to control the experiment. Two networks, *Exp1* and *Exp2* carry the experimental data. Both of the machines, the router, and the networks are deployed in the KVM (Kernel-based Virtual Machine) environment [22]. Each machine in this network is performing dual role of Data Distribution service (DDS). They can be a publisher-publisher (P:P) or publisher-subscriber (P:S) or subscriber-subscriber (S:S). The normal operation between these machines produces RTPS network traffic. For generating IGMP data, these machines are placed on separate network segments so that the router is used for multi-casting. With the help of Wireshark on the router, we capture the traffic from one of the experimental networks.

[2]Network traffic captured file which can be understood by an application capable of reading the format such as Tcpdump, Wireshark.

9.2 Detection Results

The first part of the experiment is to run the IDS against the PCAP files to verify that all the occurrences of three constraints could be successfully checked. Since each PCAP file contained the full process of a DDS running with multiple publishers and subscribers, we expect a high frequency of successful evaluation for the three constraints.

Table 1 shows the number of constraints evaluated on each PCAP files. Since these PCAP files are from a network without intrusions, no constrains are violated. From this table, we can deduce that the frequency of evaluating constraints is directly proportional to the number of packets it is processing.

The IDS was then tested with the malformed PCAP files. We inserted some malicious sequences of packets in the PCAP files which should fail three constraints. All the illegitimate patterns were captured and violations were logged. Due to lack of knowledge on penetrating the protocols, we could not execute actual attacks to the experimental network.

We also discovered that for one of the PCAP files, two of the machines were in an inconsistent state. One of them was a publisher that sent application level data message *(DATA)* without previously sending publisher message *(DATA(W))*. The surplus data packets caused constraint **C11** to fail. Examining the trace in detail also revealed that the other machine, a subscriber did not send subscriber messages *(DATA(R))*.

9.3 Performance

After confirming the successful evaluation of the network constraints, our next goal is to determine the speed of their satisfaction. The optimized technique runs faster than the naive one because it reduces space requirement and run-time complexity as follows: time complexity = (NT * NC * TT); and space complexity = NE (NT * TT + NL + NLE); where,

- NT = Number of trees, NC = Number of constraints
- NL = Number of non-leaf nodes, NLE = NL in evaluations
- TT = Tree traversal time, NE = Number of evaluations

Running as a single thread on a third generation core i5-3337U Mobile 2.7 GHz laptop with 7.7 GB of RAM, the IDS takes between 11 to 13 seconds to process the Small PCAP file with an average throughput of 1160 MB/sec. The IDS runs on Ubuntu 16.04 operating system with kernel version of 4.10.11. Figure 5 shows the

```
void FULL_RTPS_callback(FULL_RTPS * r, PDU * thePDU){
   struct HeaderInfo *h = thePDU->header;
   for(int i = 0; i < r->submsgcount; i++){
     if (r->submsg[i].type == DATAWSUB_RTPS_VAL){
         TOPICS_RTPS* topic_parms = r->submsg[i].ptr.datawsub_rtps.serializeddata;
         unsigned int entity_id = topic_parms->topicdata.ptr.entityid;
         unsigned int topic_key = topic_parms->topicdata.ptr.topickey;
         instantiateC11Pub(topic_key, h->srcIP, h->dstIP, entity_id);
     }
     else if (r->submsg[i].type == DATARSUB_RTPS_VAL){
         TOPICS_RTPS* topic_parms = r->submsg[i].ptr.datarsub_rtps.serializeddata;
         unsigned int topic_key = topic_parms->topicdata.ptr.topickey;
         instantiateC11Sub(topic_key, h->dstIP, h->srcIP, entity_id);
     }
     else if (r->submsg[i].type == DATASUB_RTPS_VAL){
         if (IsUniCastChannel(h->srcIP, h->dstIP)
            evaluateC11(topic_key, h->srcIP, h->dstIP, entity_id);
     }
   }
}
int evaluateC11(unsigned int topic, unsigned int PublisherIP,
                unsigned int SubscriberIP, unsigned int entityID)
{
    unsigned int* c11boundtree = GetC11BoundTreefromCache(topic, PublisherIP, SubscriberIP);
    if (c11boundtree[0] == PubliserIP) {
       if (c11boundtree[1] == SubscriberIP) {
         if (c11boundtree[2] == entityID)
            return 1;
       }
   }
   return 0;
}
void V2Leave_IGMP_callback(V2Leave_IGMP *v, PDU * thePDU) {
      struct HeaderInfo *h = thePDU->header;
      destroyC11(h->srcIP, h->srcPort);
}
void V3Report_IGMP_callback(V3Report_IGMP *v, PDU * thePDU) {
   struct HeaderInfo *h = thePDU->header;
   for (int i = 0; i < v->numgrps; i++){
      if (v->IsLeave())
          destroyC11(h->srcIP);
   }
}
```

Figure 4: Corresponding C code for Constraint C11

comparison of runtime of the two techniques proposed in this paper. The naive technique is slower than the optimized technique during processing the small PCAP file.

We also present a relation between throughputs of the data sets and runtime of the IDS (adopting optimized technique) in Figure 6. The runtime of the IDS is inversely proportional to the throughput of the PCAP files. The performance of the IDS remains constant with larger files as all the PCAP files have similar throughputs. This indicates that the IDS is capable of handling large network.

9.4 Scalability

We have already mentioned that our IDS achieves good runtime in terms of evaluating the constraints. However, the question remains whether the IDS can process PCAP files with more throughputs at a constant speed. The determination of memory consumption of our IDS is necessary to decide whether this IDS is applicable in a network or not.

Table 1: Evaluation Results of Three Constraints

PCAP File	Number of Packets	Number of Evaluations
Small File (1.7 GB)	10,707,581	6,787,031
Medium File (3 GB)	21,415,162	28,999,751
Large File (7 GB)	31,946,324	47,115,568

Table 2: Resource Usage of Three Constraints

PCAP File	Number of Trees	RAM Usage	Heap Size	Memory Shuffled
Small File (1.7 GB)	109	656 KB	4.6 KB	2497 MB
Medium File (3 GB)	120 (+9.1%)	678 KB	4.6 KB	3576 MB
Large File (7 GB)	138 (+13%)	696 KB	4.6 KB	4788 MB

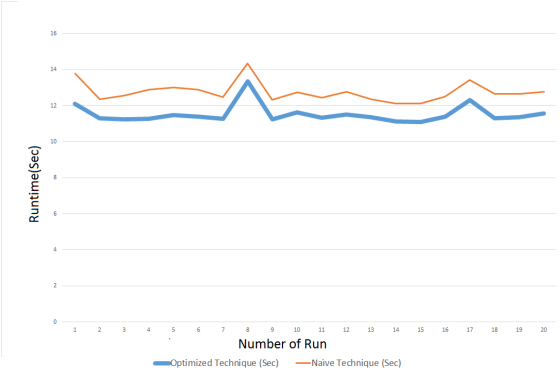

Figure 5: Naive vs. Optimized Technique Comparison.

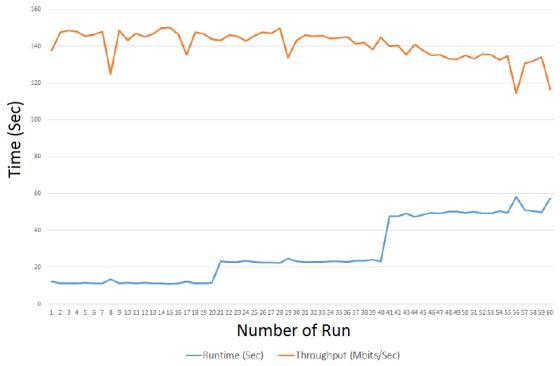

Figure 6: IDS Runtime vs. PCAP Throughputs.

Table 2 shows the number of trees generated by the IDS is directly proportional to the size of the PCAP files. It slowly increases with the number of packets. The percentage of growth of the constraint trees indicates that the IDS can be used in large scale. The other metrics in the table represent memory operations that occurred during each evaluation. First, we calculated the RAM usage of the IDS which is under 1MB on each occasion. The RAM usage also shows a slow escalation in contrast with the size of the PCAPs. The important issue to notice here is that the heap allocation of the IDS remained constant. It means that the process did not need to allocate more memory after the initialization. We computed this value by using *sbrk* [14] system calls to monitor the heap size. The last metric demonstrates the amount of memory used by the IDS to generate alerts. Overall, the purpose of these values is to verify that there is no sudden increase in memory operation.

9.5 Reliability

Since our IDS is given privileges to monitor all the traffic, it can be a possible entry point for an attacker to initiate an attack. Tuning an IDS to detect inside attacks is not easy [8]. If an attacker can detect vulnerabilities in the source code, the IDS becomes an reliable and insecure. At the final stage of the experiment, we decided to examine the reliability of the IDS source code. We executed the Valgrind *memcheck* [25] feature to locate evidence of memory leaks and the possibility of a buffer overflow in the full solution. With multiple runs over the PCAP files, the full process passed each time without showing any memory leaks or allocation errors.

The results show that this IDS is resourceful to be used in a network and does not require significant memory to operate. However, we did not calculate common evaluation metrics such as false positive rates [21], false alarms and plot the ROC curve of our IDS.

10 RELATED WORK

Detecting intrusions by analyzing the network behavior is not a novel approach. Numerous intrusion detection techniques can be found based on this idea [1]. However, using constraints instead of rules for expressing vulnerabilities is not that common. The effectiveness of the proposed approach depends on how good we are to solve these constraints. Several researches shows different ways of solving IDS rules. On the contrary, none can be found to discuss any solution related to network constraints.

Network traffic involves a continuous stream of data. Satisfying network constraints can be compared to finding frequent patterns in the data stream. A few research are presented related to constraints and data mining. Lini Susan et al. survey the available approaches of mining constraints on the data streams [12]. Rob Potharst and A. J. Feelders present the idea of using classification tree for solving monotonic constraint [18] with frequent patterns.

The main challenge of this paper is to find a technique which can be used to satisfy constraints on continuous data stream instead of static datasets and execute the full operation at high speed. Although previously mentioned work show related mechanisms

to our IDS, they are not capable of handling a network traffic. In a related effort, Andreia Silva and Cláudia Antunes show us the process of managing constraint tree to perform data mining [23]. The main drawback of their approach is constant update of prefix tree which causes huge memory overhead. Similar to this research, they also show two algorithms: naive and efficient one to achieve their desired results.

For the second approach we proposed a DSL for expressing the network constraints. The idea of gaining run-time performance depends on this proposed DSL. Expressing intrusions in a network through constraints is not a new idea. We find one particular research where they develop critical rules for SCADA system with Programmable Logic Controller (PLC) systems in a private network [4]. It shows a rule language set to define the normal pattern of operations in that system. However, they do not show the mapping between the rule set and actual solution. Another approach is presented by Pedro Salgueiro and Salvador Abreu where they show a DSL to express network constraints [20]. They achieve parallel processing of several constraints. However, their approach is not applicable for multi-packet constraints which require liveness property. The DSL developed in this paper is capable of expressing multi-packet constraints.

11 CONCLUSIONS AND FUTURE WORK

In this paper, we have presented an IDS to monitor the network traffic in an effective manner. We have shown the functional enhancements of our initial framework by proposing two tree-based techniques to satisfy network constraints. We have explained an intermediate DSL needed to evaluate network constraints. The DSL is not meant to be used by an IDS user and is generated from protocol specification language. The proposed DSL is capable of expressing all the network constraints developed in this thesis. We have also demonstrated that all the constraints are evaluated in an experimental private network. The data sets produced by the network has a throughput of a realistic network traffic. We have also shown that our IDS can detect some network irregularities using the constraints demonstrating that it is capable of monitoring the overall health of a network.

The next goal of this research is to generate the constraint code automatically from our DSL. Also, we will determine the correlation between user specification and the DSL. We will introduce more anomalous traffic in our simulation and calculate the success rate of detecting attacks in case of constraint failure. We will also define the threat models needed to run generalized experiments in DDS environment. In the future, the generalized principles of this IDS will also be considered for public networks with more protocols.

ACKNOWLEDGMENTS

This research is funded in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Department of National Defense (DND), Canada.

REFERENCES

[1] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. 2016. A survey of network anomaly detection techniques. *J. Network and Computer Applications* 60 (2016), 19–31.

[2] Bowen Alpern and Fred B. Schneider. 1986. Recognizing Safety and Liveness. In *Distributed Computing*, Vol. 2. 117–126.

[3] NAV CANADA. *AV CANADA: Media - The Canadian Automated Air Traffic System.* http://www.navcanada.ca/EN/media/Pages/publications-corporate-direct-route-story-1.aspx.

[4] Andrea Carcano, Igor Nai Fovino, Marcelo Masera, and Alberto Trombetta. 2009. State-Based Network Intrusion Detection Systems for SCADA Protocols: A Proof of Concept. In *Proc. of the Critical Information Infrastructures Security (CRITIS), 4th International Workshop*, Bonn, Germany. 138–150.

[5] Lesley Carhart. Incident Response Team. https://www.tunnelsup.com/who-makes-up-an-computer-security-incident-response-team/. (????).

[6] Gergely Csernak. *EuroScope - power of control.* http://www.euroscope.hu.

[7] Dorothy E. Denning. 1987. An Intrusion-Detection Model. In *IEEE Trans. Software Eng.*, Vol. 13. 222–232.

[8] Nathan Einwechter. 2002. Preventing Insider Attacks. https://www.symantec.com/connect/articles/preventing-and-detecting-insider-attacks-using-ids. (2002).

[9] B. Fenner, H. He, B. Haberman, and H. Sandick. 2006. Internet Group Management Protocol (IGMP) Multicast Listener Discovery (MLD)-based Multicast Forwarding (IGMP/MLD Proxying). https://tools.ietf.org/html/rfc4605. (2006).

[10] Md Siam Hasan, Ali ElShakankiry, Thomas Dean, and Mohammad Zulkernine. 2016. Intrusion Detection in a Private Network by Satisfying Constraints. In *Proc. of the 14th Annual Conference on Privacy, Security and Trust.* 623–628.

[11] Ms. Avinash Kaur, Ms. Purva Sharma, and Ms. Apurva Verma. 2014. A Appraisal Paper on Breadth first search, Depth First search and Red Black Tree. In *International Journal of Scientific and Research Publications*, Vol. 4.

[12] Lini Susan Kurien, Sreekumar K, and Minu Kk. 2014. Survey on Constrained based Data Stream Mining. In *International Journal of Computer Applications (0975-8887)*, Vol. 107. 12–15.

[13] Antoine Lemay, José M. Fernandez, and Scott Knight. 2013. An Isolated Virtual Cluster for SCADA Network Security Research. In *Proc. of the 1st International Symposium for ICS & SCADA Cyber Security Research 2013 (ICS-CSR)*, Leicester, UK. 88–96.

[14] Fabian Monrose, Marc Dacier, Gregory Blanc, and Joaquín García-Alfaro. 2016. Research in Attacks, Intrusions, and Defenses. In *Proc. of the 19th International Symposium (RAID)*, Paris, France, Vol. 9854.

[15] Louis-Philippe Morel, Jose M. Fernandez, Simon Guigui, and Thomas R Dean. 2016. *Adapting a Virtual Flight Simulator to DDS.* Technical Report. Ecole Polytechnique.

[16] Peter Nightingale. 2011. The Extended Global Cardinality Constraint: An Empirical Survey. In *Artificial Intelligence Journal (AIJ)*, Vol. 175. 586–614.

[17] O'Reilly. 2007. Write Your Own Snort Rules. http://archive.oreilly.com/pub/h/1393. (2007).

[18] Rob Potharst and A. J. Feelders. 2002. Classification Trees for Problems with Monotonicity Constraints. In *Special Interest Group (SIG) on Knowledge Discovery and Data Mining Explorations (SIGKDD)*, Vol. 4. 1–10.

[19] Jean-Charles Régin. 1996. Generalized Arc Consistency for Global Cardinality Constraint. In *Proc. of the Thirteenth National Conference on Artificial Intelligence (AAAI) and Eighth Innovative Applications of Artificial Intelligence Conference (IAAI)*, Portland, Oregon, USA, Vol. 1. 209–215.

[20] Pedro Salgueiro and Salvador Abreu. 2010. On using Constraints for Network Intrusion Detection. In *INForum - Computer Science Symposium.* 637–648.

[21] Carla Sauvanaud, Kahina Lazri, Mohamed Kaâniche, and Karama Kanoun. 2016. Anomaly Detection and Root Cause Localization in Virtual Network Functions. In *Proc. of the 27th IEEE International Symposium on Software Reliability Engineering (ISSRE)*, Ottawa, Ontario, Canada. 196–206.

[22] Ryan Shea and Jiangchuan Liu. 2012. Network Interface Virtualization: Challenges and Solutions. In *IEEE Network*, Vol. 26. 28–34.

[23] Andreia Silva and Cláudia Antunes. 2013. Pushing Constraints into Data Streams. In *Proc. of the 2nd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications (BigMine)*, Chicago, Illinois, USA. 79–86.

[24] H. Schulzrinne snf A. Rao and R. Lanphier. 1998. Real Time Publisher Subscriber Protocol (RTPS). https://www.ietf.org/rfc/rfc2326.txt. (1998).

[25] Valgrind. 2007. The Valgrind Quick Start Guide. http://valgrind.org/docs/manual/quick-start.html. (2007).

[26] Songtao Zhang, Thomas R. Dean, and Scott Knight. 2006. A Lightweight Approach to State Based Security Testing. In *Proc. of the 2006 conference of the Centre for Advanced Studies on Collaborative Research*, Toronto, Ontario, Canada. 341–344.