Modeling Distributed Network Attacks with Constraints

Pedro Salgueiro and Salvador Abreu

Abstract In this work we demonstrate how to model and perform the detection of Distributed Network attacks using NeMODe, a declarative system for Computer Network Intrusion Detection which provides a declarative Domain Specific Language for describing computer network intrusion signatures which span several network packets by stating constraints over network packets, thus, describing relations between several packets, in a declarative and expressive way.

Key words: Constraint Programming, Intrusion Detection Systems, Domain Specific Languages

1 Introduction

Maintaining the security of computer networks is a crucial task and plays an important role in keeping the users data safe and the network a safe place to work. Such task can be accomplished by Network Intrusion Detection System (IDS) e.g. Snort [1].

Distributed network attacks are very popular among hacker communities, since they are driven from several places and easily elude the detection mechanisms, since these attacks originate from several places at once, being difficult to identify the network traffic as an attack.

There are certain aspects that should be verified in order to maintain the security of the users data, as well as the quality and integrity of the services provided by a computer network. Being able to describe these aspects, together with a verification that they are met can be considered as an Network Intrusion Detection task.
Describing those conditions, in terms of properties which must be verified in the network traffic, also describe the desired or unwanted state of the network, which can be induced by a system intrusion or another form of malicious access.

While using a declarative programming approach, such as Constraint Programming [2] or Constraint-Based Local Search Programming (CBLS) [3], we can describe those condition, in an easier, natural and expressive way.

NeMODe Intrusion Detection System is a declarative system that provides a Domain Specific Language enabling an easy and very descriptive way to describe the network intrusion signatures by following the constraint programming methodologies. Besides using constraint programming paradigm to describe the desired network situation, it also relies constraint programming to perform the detection of such intrusions, providing several back-end detection mechanisms based on Constraint Programming, such as Propagation Based solvers, using Gecode, and Constraint-Based Local Search, using Adaptive Search. NeMODe system is further described in [4, 5].

This paper is organized as follows. Section 1 introduces the work and makes a brief description of Network Intrusion Detection Systems and Constraint Programming. Section 2 describes the NeMODe. Section 3 presents some distributed network attacks and how to describe them in NeMODe. Section 4 presents the experimental results, Sect. 5 evaluates NeMODe while detecting distributed network attacks, and Sect. 6 presents the conclusions and the future work.

Throughout this paper, we mention some TCP/IP and UDP/IP technical terms, such as packet flags, URG, ACK, PSH, RST, SYN, FIN, acknowledgment, source port, destination port, source address, destination address, payload, described in [6].

1.1 Intrusion Detection Systems

Network Intrusion Detection Systems are very important and one of the first lines of defense against network attacks or other type of malicious access, which constantly monitors the network traffic looking for anomalies or undesirable communications in order to keep the network a safe place.

There are several methods to perform network intrusion detection, but, among them two of them are more used [7]:

1. Based on the network intrusion signatures
2. Based on anomaly detection

On Network Intrusion Detection Systems based on signatures, the network attacks are described using their signatures, particular properties of network packets used to achieve the desired intrusion or attack, which are then looked in the network traffic. Intrusion Detection Systems based on anomaly detection, tries to understand the normal behavior of the systems by modeling its behavior using statistical methods and/or data mining approaches. The network behavior is then monitored, and if
considered anomalous according the network model, the network is probably under some kind of attack. NeMODe uses use an approach based on signatures.

Snort [1] is a widely used Network Intrusion Detection System, primarily designed to detect signatures that can be identified in a single network packet, using efficient pattern-matching techniques to detect the desired intrusion signature.

Although Snort provides some basic mechanisms which allow the writing of rules that spread over several network packets, such as the Stream4 or Flow preprocessors, they do so in a very limited and counter-intuitive way, not allowing the description of more complex relations between packets, such as the temporal distance between two packets.

Most of the work in the area of Intrusion Detection Systems consists in the development of faster detection methods [8], but there is also some work focused on how the network signatures are described and detected, such as in the work [9], where the authors present a declarative approach to specify intrusion signatures which are represented as a specialized graph, allowing the description of signatures that spread across several network packets.

1.2 Constraint Programming

Constraint Programming (CP) is a declarative programming paradigm consisting in the formulation of a solution to a problem specified as a Constraint Satisfaction Problem (CSP) [2], in which a number of variables are introduced, with well-specified domains and which describe the state of the system. A set of relations, called constraints, is then imposed on the variables which make up the problem. These constraints are understood to have to hold true for a particular set of bindings for the variables, resulting in a solution to the CSP.

There are several types of constraint solvers, in this work we use: (1) Propagation Based solvers; and (2) Constraint Based Local Search (CBLS).

1.3 Propagation-Based solvers

Problems in Propagation-Based [2] solvers are described by stating constraints over each variable that composes the problem, which states what values are allowed to be assigned to each variable, then, the constraint solver will propagate all the constraints and reduce the domain of each network variables in order to satisfy all the constraints and instantiate the variables that compose the problem with valid results, thus reaching a solution to the initial problem.

Gecode [10] is a constraint solver library based on propagation, implemented in C++ and designed to be interfaced with other systems or programming languages.
1.4 Constraint Based Local Search

CBLS [3] is a fundamental approach to solve combinatorial problems such as Constraint Satisfaction Problems. CBLS is a method that can solve very large problems, although not a complete algorithm and unable to provide a complete or optimal solution. Usually, this approach initiates with an initial, candidate solution to the problem which is then iteratively improved through small modifications until some criteria is satisfied. The modifications to the candidate solution is usually driven by heuristics that guide the solver to a solution.

Adaptive Search (AS) [11] is a Constraint Based Local Search [3] algorithm, taking into account the structure of the problem and using variable-based information to design general heuristics which help solve the problem. The iterative repairs to the candidate solution in Adaptive Search are based on variable and constraint error information which seeks to reduce errors on the variables used to model the problem.

2 Intrusion Detection with Constraints

Detecting Network Intrusions with constraints consists on identifying a set of network packets in the network traffic, which identify and makes proof of the desired network signature attack, matching the desired network signature described through the use of constraints stated over a set of network packet variables, thus describing relations between several network packets.

In order to use the constraint programming mechanism to perform Network Intrusion Detection, there is the need to model the desired signature as a Constraint Satisfaction Problem (CSP). A CSP which models a network situation is composed by a set of variables, \( V \), representing the network packets involved in the description of the network situation; the domain of the network packet variables, \( D \); and a set of constraints, \( C \), which relates the variables in order to describe the network situation. We call such a CSP a network CSP. On a network CSP, each network packet variable is a tuple of integer variables, 19 variables for TCP/IP packets and 12 variables for UDP packets, representing the significant fields of a network packet necessary to model the intrusion signatures used in our experiments.

The domain of the network packet variables, \( D \), are the values actually seen on the network traffic window, which is a set of tuples of 19 integer values (for the TCP variables) and 12 integer values (for the UDP variables), each tuple representing a network packet actually observed on the traffic window and each integer value represents each field relevant to intrusion detection. The packets payload is stored separately in an array containing the payload of all packets seen on the traffic window.

---

1 Here, we are only considering the “interesting” fields in TCP/IP packets, from an IDS point-of-view.

2 Here, we are only considering the “interesting” fields in UDP packets, from an IDS point-of-view.
The correspondence between the packet and its payload is achieved by matching the packet number, \( i \), which is the first variable in the tuple representing the packets and the \( i^{th} \) position of the array containing the payloads.

Listing 1 shows a representation of such CSP, where \( P \) represents the set of network packet variables, where \( P_{n \times z} \) is each of the individual integer variables of the network packet variable, in a total of \( z \) fields for each network of the \( n \) variables, with \( z = 19 \) for TCP packets and \( z = 12 \) for UDP packets. \( D \) is the network traffic window, where \( D_i = (V_{i,1}, \ldots, V_{i,z}) \in D \) is one of the real network packets on the network traffic window, which is part of the domain of the packet variables \( P \). \( Data \) is the payloads of the network packets present in the network window, where \( Data_i \) is the payload of the packet \( P_i = (V_{i,1}, \ldots, V_{i,z}) \in D \).

The associated domains of the network packet variables is represented by \( \forall P_i \in P \Rightarrow P_i \in D \), forcing all variables belonging to \( P \) to obtain values from the set of packets in the network window \( D \).

A solution to a network CSP, if it exists, is an assignment of network packet values, \( D_i = (V_{i,1}, \ldots, V_{i,z}) \in D \), to each packet variable, \( P_i = (P_{j,1}, \ldots, P_{j,z}) \in P \), that models the desired situation, thus identifying the network packets that identify the intrusion being detected.

Listing 1 Representation of a network CSP

\[
P = \{(P_1, \ldots, P_{z}), \ldots, (P_{n,1}, \ldots, P_{n,z})\}
\]
\[
D = \{(V_{1,1}, \ldots, V_{1,z}), \ldots, (V_{x,1}, \ldots, V_{x,z})\}
\]
\[
Data = \{Data_1, \ldots, Data_x\}
\]
\[
\forall P_i \in P \Rightarrow P_i \in D
\]

2.1 A DSL to describe network signatures

The NeMODe Intrusion Detection System is a declarative system that provides a Domain Specific Language, following the constraint programming methodologies, enabling an easy and very descriptive way to describe the intrusion signatures that spread across several network packets by allowing to state of constraints over network entities and express relations across several network packets.

This Domain Specific Language (DSL) will then translate the program into constraints which are then solved by several constraint solving techniques, including Propagation-based systems, such as Gecode and Constraint-Based Local Search (CBLS), such as Adaptive Search. This DSL is further described in [4, 5].
2.2 Architecture

NeMODe is composed by a compiler, which reads a NeMODe program and parses it into a semantic model. Then, based on that semantic model, it is generated code for each of the available back-ends in system.

After all recognizers have been generated, each generated back-end receives as input the network traffic and produces a valid solution, if the intrusion described as a NeMODe Program exists on the network traffic that was given as input to each back-end detection mechanism.

All back-ends available in the system work in parallel, each one producing a solution to the problem. In a final step, the best solution produced is selected, which is simply the first solution to be produced.

Fig. 1 represents the architecture of the system and how the data flows between each component.

3 Examples

So far, we have worked with some simple network intrusion signatures: (1) a distributed port-scan attack, and (2) a distributed SYN flood attack. All of these intrusion patterns can be described using NeMODe and the generated code was successful in finding the desired situations in the network traffic logs.

3.1 Distributed port-scan

A port-scan is an important step before a network attack, scanning for available services on a given computer, allowing to discover the potential vulnerabilities of the victim, helping the attackers to better the attack. A distributed port-scan is made from several computers, geographically distributed, all scanning the services of a single computer.
The port-scan attack can be described by a set of constraints between two network packets, the packet that initiates the TCP/IP connection and the packet that closes it, which closes the connection in a very short time after the TCP connection had been initiated. In order to detect if the network is under such attack we monitor if there are many of these sets of networks packet to appear on a network traffic in a short period of time.

Listing 2 A port-scan attack using NeMODe

```plaintext
1 portscan {
2   C = { packet(A), tcp(A), syn(A), nak(A),
3         packet(B), tcp(B), rst(B),
4         time(A) - time(B) < usecs(100),
5         related(A,B) },
6   R := repeat(5,C),
7   max_interval(R) < usecs(500)
8 }
```

3.2 Distributed SYN flood attack

A Distributed SYN flood attack happens when several attackers, from several places, join forces to drive a distributed attack in order to initiate more TCP/IP connections than the server can handle and then ignoring the replies from the server, forcing the server to have a large number of half open connections in standby, which leads the service to stop when this number reach the limit of number of connections. This attack can be detected if a large number of connections is made from a single machine to other in a very short time interval. Listing 3 shows how a SYN flood attack can be described using NeMODe.

Listing 3 A SYN flood attack programmed with NeMODe

```plaintext
syn_flood {
1   C = { tcp_packet(A), syn(A), nak(A) },
2   R := repeat(30,C),
3   max_interval(R) < usecs(500)
4 }
```

4 Experimental Results

Among others, we have tested the examples of Sect. 3, a Distributed SYN flood and a Distributed port-scan. All these intrusions were successfully described using
NeMODe and valid Gecode and Adaptive Search code was produced and then executed in order to validate the code and ensure that it could indeed find the desired network intrusions.

The code was then run on a dedicated computer, an HP Proliant DL380 G4 with two Intel(R) Xeon(TM) CPU 3.40GHz and with 4 GB of memory, running Debian GNU/Linux 4.0 with Linux kernel version 2.6.18-5. The Adaptive Search was also run on an IBM BladeCenter H equipped with QS21 dual-Cell/BE blades, each with two 3.2 GHz processors, 2GB of RAM, running RHEL Server release 5.2, since it was recently ported to the Cell/BE architecture, presented in [12].

For the Distributed Port-scan attack, we created a log file composed of 400 TCP network packets while a computer was being under a Distributed Port-scan attack, being used as the network traffic. The used signature describes 52 TCP packet variables.

In the case of the Distributed SYN flood attack, we created a log file composed of 100 TCP network packets while a computer was being under a Distributed SYN flood attack, which was used as the network traffic. The signature used to model the problem was composed by 30 TCP network packet variables.

Table 1 presents the time (user time, in milliseconds) required to find the desired network situations for the attacks presented in is present work, using Gecode and Adaptive Search on a x86 architecture and also using Adaptive Search on the Cell/BE architecture. The number of network packet variables used to model the signatures and the number of network packets used to search the intrusions are also presented. The times presented are the average of 128 runs.

<table>
<thead>
<tr>
<th>Intrusion to detect</th>
<th>Signature size</th>
<th>Window Size</th>
<th>Gecode x86 (ms)</th>
<th>A.S. x86 (ms)</th>
<th>A.S. Cell (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port-scan</td>
<td>52</td>
<td>400</td>
<td>127.3</td>
<td>674.9</td>
<td>62.5</td>
</tr>
<tr>
<td>SYN flood</td>
<td>30</td>
<td>100</td>
<td>56.6</td>
<td>6.1</td>
<td>22.5</td>
</tr>
</tbody>
</table>

5 Evaluation

The experimental results described in Sect. 4 shows that the performance varies in a great scale depending on the problem and the recognizer.

Table 1 shows that the Port-scan takes longer to be recognized than the Distributed SYN flood. This behavior is explained by the complexity of the problems, which is directly related to the way the problems are modeled, since the Port-scan is modeled by using more variables than the SYN flood attack, also, the constraints used to model the Port-scan are more complex than the ones used in the SYN flood, making the Port-scan much more complex than the SYN flood attack.
In the x86 architecture, the Port-scan performs better in Gecode than on Adaptive-Search, as for the SYN flood-attack, the opposite is verified. This shows that in the x86 architecture Gecode performs better on more complex, but when the complexity of problems decrease, Adaptive Search performs better.

While the performance of Adaptive Search Cell/Be version the x86 version can’t be directly compared, it’s possible to conclude that more complex problems takes advantage of the Cell/BE architecture and the performance doesn’t degrades as much when the complexity increases.

As for the SYN flood in the Cell/BE A.S. version it presents a worst performance, which could be explained by signature used to model the problem, which is much simpler than the Port-scan attack. Also, the network packets that make part of the attack are much closer together in the network packet window, making the search of solution much simpler. Do to these two facts, the problem gets much less complex, not being able to take full advantage of the Cell/BE architecture.

The results obtained, are quite good, allowing us to start the detection of intrusions in real network traffic instead of log files.

6 Conclusions and Future Work

In this work we describe how to model two distributed network attacks in NeMODe, a system for Network Intrusion Detection based on Constraint Programming, allowing an intuitive an expressive way to describe such situation due to the possibility of the specification of relations between several network packets in an easy way.

This work shows that we can easily describe distributed network attacks or network situations, which spread across several network packets, using a declarative approach, and, from that single description, generate several Constraint Programming based network situation recognizers, using different Constraint Programming paradigms, which actually detect the desired intrusions, if they exist in the network traffic.

With this work, we also proved that we can easily describe and detect signatures which spread across several network packets, something which is hard to achieve in systems like Snort. Although the intrusions mentioned in this work can be detected with other intrusion detection systems, they are modeled/described with out relating several network packets, usually being described by specifying a set properties over a single network packet, which could lead to a large number of false positives.

The results obtained are very promising, providing a platform to start performing network intrusion detection on a live network traffic link in a near future, a very important future step.

We still need to model more network situations as a CSP to better evaluate the performance of the system. We also need to better evaluate the the work presented in this paper by comparing the obtained results with systems like Snort. Also, we have plans to implement new back-end detection mechanisms using different constraint programming paradigms.
Acknowledgments

Pedro Salgueiro acknowledges FCT –Fundação para a Ciência e a Tecnologia– for supporting him with scholarship SFRH/BD/35581/2007. The IBM QS21 dual-Cell/BE blades used in this work were donated by IBM Corporation, in the context of a SUR (Shared University Research) grant awarded to Universidade de Évora and CENTRIA.

References