

Evolving Intrusion Detection Rules on Mobile Ad Hoc Networks

Sevil Şen and John A. Clark

Department of Computer Science, University of York, YO10 5DD, UK
{ssen,jac}@cs.york.ac.uk

Abstract. Intrusion detection on mobile Ad Hoc Networks (MANETs) is in its early stages. In this paper, we show how grammatical evolution can be used to evolve detection programs for dropping attacks, a particularly important attack type for such networks.

Keywords: Grammatical evolution, intrusion detection, MANETs.

1 Introduction

A mobile ad hoc network (MANET) is a self-configuring network of mobile nodes connected by wireless links. MANETs do not have any fixed and pre-established infrastructure such as centralised management or base stations in wireless networks. The union of nodes forms an arbitrary network topology that changes frequently due to the mobility of the nodes. In addition, the nodes must cooperate with each other to provide essential networking. Mobile nodes that are within each other's radio range can communicate directly via wireless links, while those that are far apart must rely on other nodes to forward their messages. Since they provide communication even in the absence of a fixed infrastructure, they are very attractive for many applications such as rescue operations, tactical operations, environmental monitoring, conferences, and the like.

Attempts to design intrusion detection systems for MANETs to date are typically carried out entirely by the designer. However, humans are not particularly adept at selecting good choices when complex tradeoffs have to be made. Accordingly we propose to investigate the use of an artificial intelligence based learning technique to explore this difficult design space. In this paper, grammatical evolution (GE) is explored as a technique to detect known attacks on MANETs. Detection rules are evolved to detect a known type of attack on MANETs (dropping attack) and evaluated on networks with varying mobility and traffic patterns.

2 Grammatical Evolution in Intrusion Detection on MANETs

2.1 The Problem

In this paper, we use grammatical evolution [1] to evolve detection rules for dropping attacks on MANETs. In the dropping attack scenario malicious node(s)

drop data packets not destined for themselves to disrupt the network connection. Since malicious nodes need to be on a routing path to drop data packets, they have little reason to drop routing protocol control packets such as RREQ, RREP, and RERR messages used in route discovery and maintenance mechanisms of AODV. So, it is assumed that malicious nodes do not drop routing protocol control packets.

While packet losses usually occur due to congestion in wired networks, there can be other causes on MANETs. Major causes of packet losses on MANETs are given as wireless link transmission errors, mobility and congestion in [2].

Transmission errors depend on the physical characteristics of the channel and the terrain, and they can not be eliminated or reduced by improving routing protocols [2]. Packet losses due to mobility are the result of one of the main characteristics of MANETs. Mobility of the nodes changes network topology and frequently makes existing routes inactive. Situations like buffer overflows, broken links, and no route to the destination can occur due to mobility and cause packets to be dropped. Lastly, packet losses due to congestion occur when the demands exceeds the capacity of a communication link [2].

Mobility is given as the major cause of packet losses on AODV [2]. It is shown that more than 60% of total packet loss on AODV is due to mobility. We mainly aim to differentiate packet dropping due to malicious behaviour from packet dropping due to mobility in this paper.

2.2 The Method

We evolve a program to detect dropping attacks on MANETs. The evolved program is distributed to each node on the network. We assume that dropping attacks can be detected by the neighbours of the malicious node who sent/forwarded packets to the malicious node, but has not received any acknowledgement from it for a while. Moreover, an attack is assumed to be detected in a time interval Δ after it has occurred. Since features are gathered every time interval by each node locally, a sliding window mechanism, which includes all features in Δ , is applied for training and testing the evolved program.

The *detect window* shown in Figure 1 below is defined as the window that consists of the network features available during the period of length Δ after a dropping attack has occurred. For training purposes we assume that an ideal evolved IDS program should flag the occurrence of an attack precisely Δ seconds after the attack has finished (i.e. using the feature data available during the detect window), at the *detect point* t_D . It should flag "no attack" at or before the attack begins and also after more than Δ seconds after it has finished (i.e after the detect point). At all other times we do not care whether the evolved program flags "attack" or "no attack". This means that in the training process programs that can detect some attacks earlier than Δ units after they have finished are not punished for doing so. Additionally our training assumes that a network "returns to normality" at $\Delta+1$ seconds after an attack has finished. This is a very straightforward evaluation approach for experimental purposes. Other choices for desired flagging profiles are clearly possible.

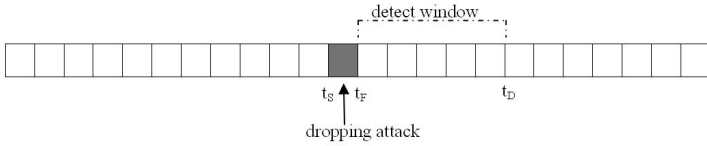


Fig. 1. Detect Window

Grammar. The grammar used to evolve a program to detect dropping attacks on MANETs and raise an alarm is defined in Table 1:

Table 1. BNF grammar used for the problem

S	=	<code>
<code>	::=	if(<cond>) raise_alarm()
<cond>	::=	<cond> <set-op> <cond> <expr> <rel-op> <expr>
<expr>	::=	<expr> <op> <expr> (<expr> <op> <expr>) <pre-op> (<expr>) <pre-op2> (<expr>, <expr>) <var>
<op>	::=	+ - / *
<pre-op>	::=	sin cos log ln sqrt abs exp ceil floor
<pre-op2>	::=	max min pow percent
<rel-op>	::=	< ≤ > ≥ == !=
<set-op>	::=	and or
<var>	::=	feature set in Appendix A

Features used in the grammar are given in Appendix A. We use both mobility-related features as well as packet-related features as input to the evolution system. While some of these features give information about mobility directly (such as changes in the number of neighbours), some of them can be the result of mobility (such as added routes in the last period). Packet-related features include routing protocol control packets and transport protocol packets. AODV[5], which is one of the most commonly used on-demand routing protocols on MANETs, is used in this paper. Because TCP expects acknowledgement packets from the destination and lack of acknowledgements may indicate dropping attacks, it is chosen as a transport layer protocol. All features are gathered periodically at every second by each node.

Fitness Function. As evaluation measures we use detection rate (the ratio of correctly detected intrusions to the total intrusions on the network) and false positives rate (the ratio of normal activities which are incorrectly marked as intrusions to the total normal activities on the network). Low false positive rate is as important as high detection rate for a good intrusion detection system. That’s why the constant k ($=4$) in the fitness function is used for decreasing the false positive rate.

$$Fitness = detection\ rate - k * false\ positive\ rate \tag{1}$$

GE Parameters. The parameters used in GE are given in Table 2.

Table 2. GE Tableau for detecting dropping attacks on MANETs

Objective:	Find a program to detect dropping attacks on MANETs
Non-Terminal Operators:	The binary operators +, -, *, /, pow, min, max, percent The unary operators sin, cos, log, ln, sqrt, abs, exp, ceil
Terminal Operators:	The feature set in Appendix A
A Fitness cases:	The given sample of network data marked malicious or non-malicious
Raw Fitness:	The detection rate over the fitness cases subtract the false positive rate over the fitness cases
Standardised Fitness:	Same as raw fitness
Parameters	Populations Size = 100 Termination when Generations= 1000 Prob. Mutation = 0.02, Prob. Crossover = 0.9 Steady State

2.3 Experiment and Results

The evolved program is evaluated on the networks simulated by ns-2 [3]. Mobility of the nodes is simulated by the Random Waypoint model which is created using BonnMotion [4]. In the Random Waypoint model, each node moves from its current location to a random new location with random speed and pause time in determined speed/pause time limits [6]. Different network scenarios are created with different mobility levels and traffic loads. The parameters of the network simulation are given in Table 3.

The algorithm is evolved using the training data collected from a network under medium mobility with 30 TCP connections. The same network with dropping attacks and without attacks are used for training to reduce false positives. Evolved programs are evaluated on different network scenarios and the results are presented in Table 4. There are two different networks under medium mobility in Table, which are simulated with different mobility and traffic patterns, since one of them is used for evolution. In the results, false positives increase in proportion to the mobility (as expected). False positives also increase under high traffic loads (which can be a source of non-malicious packet loss).

Table 3. The parameters of the network simulation

network dimensions	1000x500
number of nodes	50
packet traffic	TCP with 20 and 30 connections
speed	0-20 m/sec
pause time	40,20,5 sec (low, medium and high mobility)
routing protocol	AODV
radio propagation	two-way ground model with 250m transmission range
local link connectivity	AODV periodic hello messages
simulation time	1000 sec (testing), 2000 sec (training)

Table 4. The experiment results

Scenarios	Detection Rate	False Positive Rate
Low mobility, 20 TCP connections	79.59%	3.81%
Low mobility, 30 TCP connections	93.85%	5.25%
Medium mobility, 20 TCP connections	92.45%	3.95%
Medium mobility, 30 TCP connections	87.04%	6.30%
Medium mobility, 20 TCP connections	90.48%	4.07%
Medium mobility, 30 TCP connections (training)	82.64%	5.53%
High mobility, 20 TCP connections	83.33%	5.05%
High mobility, 30 TCP connections	84.38%	6.22%

3 Conclusion

Grammatical evolution essentially "grows" intrusion detection programs by evaluating populations of potential programs and subjecting them to a variety of genetically inspired operators. The results show that our grammatical evolution technique has significant potential for evolving efficient detectors from such discrimination attacks. The approach has good chances of generalizing: we aim now to simulate a variety of attacks and employ the same grammatical evolution technique to evolve detectors for those attacks.

One interesting feature of the approach is that we can evolve detectors that can be evaluated with respect to how well they detect a range of attacks rather than a single specific attack. Though we have not done so here, it should also be possible to employ multi-objective evaluation mechanisms to explore optimal tradeoffs between resources consumed by programs (e.g. memory and power) and detection efficacy. This is very difficult for a human designer to address.

References

1. Ryan, C., Colline, J.J., O'Neill, M.: Grammatical Evolution: Evolving Programs for an Arbitrary Language. In: Banzhaf, W., Poli, R., Schoenauer, M., Fogarty, T.C. (eds.) EuroGP 1998. LNCS, vol. 1391, pp. 83–95. Springer, Heidelberg (1998)
2. Lu, Y., Zhong, Y., Bhargava, B.: Packet Loss in Mobile Ad Hoc Networks. Technical Report, Dept. of Comp. Sci, Purdue Uni., TR 03-009 (2003)
3. The Network Simulator: ns-2, <http://www.isi.edu/nsnam/ns>
4. BonnMotion: A Mobility Scenario Generation and Analysis Tool, <http://web.informatik.uni-bonn.de/IV/Mitarbeiter/dewaal/BonnMotion/>
5. Perkins, C.E., Royer, E.M.: Ad-hoc On-Demand Distance Vector Routing. In: Proc. of the 2nd IEEE Workshop on Mobile Computer Systems and Applications, pp. 90–100 (1999)
6. Camp, T., Boleng, J., Davies, V.: A Survey of Mobility Models for Ad Hoc Network Research. *Wireless Communication and Mobile Computing*, 483–502 (2002)

Appendix A. The Features

neighbours	number of neighbours
added_neighbours	number of added neighbours
removed_neighbours	number of removed neighbours
active_routes	number of active routes
repaired_routes	number of routes under repair
invalidated_routes	number of invalidated routes
addedroutes_disc	number of added routes by route discovery mechanism
addedroutes_notice	number of added routes by overhearing
updated_routes	number of updated routes (modifying hop count, sequence number)
addedroutes_repaired	number of added routes under repair
invroutes_timeout	number of invalidated routes due to expiry
invroutes_other	number of invalidated routes due to other reasons
recv_rreqPs	number of received route request packets destined to this node
recvF_rreqPs	number of received route request packets to be forwarded by this node
send_rreqPs	number of broadcasted route request packets from this node
frw_rreqPs	number of forwarded route request packets from this node
recv_rrepPs	number of received route reply packets destined to this node
recvF_rrepPs	number of received route reply packets to be forwarded by this node
send_rrepPs	number of initiated route reply packets from this node
frw_rrepPs	number of forwarded route reply packets from this node
recvB_rerrPs	number of received broadcast route error packets (to be forwarded or not)
send_rerrPs	number of broadcasted route error packets from this node
recv_aodvPs	number of received total routing protocol packets
recvF_aodvPs	number of received total routing protocol packets to be forwarded
send_aodvPs	number of initiated total routing protocol packets from this node
frw_aodvPs	number of forwarded total routing protocol packets by this node
recvF_dataPs	number of TCP data packets forwarded by this node (not acknowledged)
send_dataPs	number of TCP data packets initiated by this node (not acknowledged)