Initiating a Moving Target Network Defense with a Real-time Neuro-evolutionary Detector

Robert J. Smith
Dalhousie University,
6050 University Av., Halifax,
NS, Canada
rsmith@cs.dal.ca

A. Nur Zincir-Heywood
Dalhousie University,
6050 University Av., Halifax,
NS, Canada
zincir@cs.dal.ca

Malcolm I. Heywood
Dalhousie University,
6050 University Av., Halifax,
NS, Canada
mheywood@cs.dal.ca

John T. Jacobs
Raytheon Space and Airborne Systems,
6380 Hollister Av. Goleta,
California 93117-3114
John_T_Jacobs@raytheon.com

ABSTRACT

The moving network target defense (MTD) based approach to security aims to design and develop capabilities to dynamically change the attack surfaces to make it more difficult for attackers to strike. One such capability is to dynamically change the IP addresses of subnetworks in unpredictable ways in an attempt to disrupt the ability of an attacker to collect the necessary reconnaissance information to launch successful attacks. In particular, Denial of Service (DoS) and worms represent examples of distributed attacks that can potentially propagate through networks very quickly, but could also be disrupted by MTD. Conversely, MTD are also disruptive to regular users. For example, when IP addresses are changed dynamically it is no longer effective to use DNS caches for IP address resolutions before any communication can be performed. In this work we take another approach. We note that the deployment of MTD could be triggered through the use of light-weight intrusion detection. We demonstrate that the neuro-evolution of augmented topologies algorithm (NEAT) has the capacity to construct detectors that operate on packet data and produce sparse topologies, hence are real-time in operation. Benchmarking under examples of DoS and worm attacks indicates that NEAT detectors can be constructed from relatively small amounts of data and detect attacks $\approx$ 90% accuracy. Additional experiments with the open-ended evolution of code modules through genetic program teams provided detection rates approaching 100%. We believe that adopting such an approach to MTB a more specific deployment strategy that is less invasive to legitimate users, while disrupting the actions of malicious users.

GECCO'16 Companion, July 20 - 24, 2016, Denver, CO, USA

ISBN 978-1-4503-4323-7/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2908961.2931681

CCS Concepts

•**Computing methodologies** → **Generative and developmental approaches;** •**Security and privacy** → *Denial-of-service attacks; Firewalls;*

Keywords

Neural Evolution; Network Security; Moving Target Defense; Software Defined Networks; Genetic Programming

1. INTRODUCTION

Network security metaphors often take the form of walls, the more walls that are involved the more secure the system, i.e. defense in depth. Systems are secured by surrounding themselves by, say, two walls with varying degrees of security at each. Getting messages to the users of the inner wall requires a verification and obfuscation process typically handled by a firewall with designated messengers (proxies) speaking for the outside and inside worlds separately. However, even virtual walls tend to be difficult to adapt to new circumstances over time. Thus, while network technology is evolving at a rapid pace, the walls by which we secure that technology still tend to operate on the same notions from prior technological generations: sometimes operating efficiently is more important than protecting against attacks targeting the system. The notion of securing networks has become more important over time as the number of potential attack vectors increased through the proliferation of ubiquitous computing and globalized networking (such as the Internet). Thus even modern network configurations are largely static. Digital network components such as addresses and subnet configurations continue to be largely unchanged due to the convenience of having a fixed parameter set.

A Moving Target (Network) Defense (MTD) attempts to embody the concept that a malleable, changing system is more effective for combating attackers than a largely stationary one [1, 3, 16]. Different forms of 'moving target' defence have been proposed, for example, searching for 'good' computer configurations that are deemed more secure relative to a prior set of criteria [7]. In this research, given the overheads of performing MTD at the network level, we concentrate on identifying the condition(s) under which to

initiate a MTD for network reconfiguration under a software defined networking (SDN) approach. Thus, from the big picture perspective, having a static network configuration confers many advantages to the attacker, giving them time for planning attacks and gathering network information. However, if we provide our network with some form of effective MTD, we should be able to narrow the temporal factors of the information gathering process [5, 6]. That is to say, assuming that information for planning an attack has been collected, by the time that the attack is initiated, the network 'connectivity' has changed. Increasing product support for 'software defined networks' provides the opportunity to carry the concept of a malleable, changing network closer to that of an autonomous system.

In this research, we propose to treat the MTD as a policy which is enacted once suspicious behaviour is identified. The MTD is only initiated relative to the detection of specific network behaviours. Recent demonstrations of MTD have focused on the capacity to resist Worm attacks (e.g., [5, 6]). Hence, if we can demonstrate robust detection of such attacks, a software defined network can then explicitly introduce defensive actions such as a complete MTD (or MTD targeting specific services / ports) depending on the policy assumptions. To do so, we construct a packet generator for a cross-section of packet 'types', and then assume a neuro-evolutionary approach for constructing detectors. We demonstrate that rather than aiming to learn everything, we can build 'expert' detectors which can identify different types of attacks. An analysis of possible contradictions in the labels provided by each detector is then used to construct a detector for resolving exceptions, or real-time forensics. Adopting such a strategy enables us to incrementally build new detectors for different types of attacks and specifically resolve exceptions when they appear. The resulting framework is benchmarked through a simulated SDN, where an SDN provides the flexibility necessary to support the automation of network counter-measures in practice.

2. BACKGROUND

Moving Target Defense (MTD) attempts to address key issues with static network configuration [3]. One of the major criticisms of classic network design and security is that the addresses are typically statically assigned by a naming service (e.g. DNS). This issue is likely due in part to the limited IPv4 address space. However, there are several proposed solutions to overcome the static addressing issue. One such work attempted to create a temporal naming system which would create identifiers for host pairs on a network and regularly rotate identifiers which have been in use for extended periods of time [1, 3, 16]. By creating a real-time unique mapping for hosts and IP addresses they could limit the exposure of intra-network information gathering by rearranging the way hosts communicate within a given network based on pre-determined temporal parameters. Another approach is to provide rotational addressing using software defined networks and OpenFlow controllers [5].

The bottom line is that each host is associated with different *unique* subsets of (pseudo) IP addresses. This implies that only host X may communicate with host Y using this IP address. Moreover, the interval of time for which this host-IP 'binding' holds is also a stochastic parameter. Earlier works for MTD also introduced a temporal scheme for 'IP hopping' [1] or were incompatible with current network protocols [16]. Conversely, Jafar *et al.* maintain compatibility with current IPv4 and IPv6 networks and also introduce a spatial element to the randomization process for constructing (pseudo) IP addresses.

The underlying approach of [5, 6] is to create a name-to-address mapping that is a function of both source identity as well as time of the request. In their work, each source needs a range of addresses for the temporal hops. The wider the range of such addresses is the greater the potential for hiding the true identity of network structure. Conversely, if addresses are just periodically switched, the same association between nodes exists. Jafar *et al.* avoid this because hosts are also subject to different periods of time during which their name-to-address mapping holds. This is based on the host spatial locations.

The network infrastructure to support such a process, consists of two components, a controller and a gateway. A gateway performs the translation from real IP to pseudo IP addresses used on a subnet. It is the controller that acts as the name server responsible for distributing spatiotemporal mappings between host pairs and their (pseudo) IP addresses. Thus, each time that communication between a pair of hosts is necessary, the gateway is contacted in order to find the relevant pseudo address. Every message sent has to be verified by the name server before communication can take place. The penalty for this is the extra cost of having to repeatedly communicate with gateways due to the temporal nature of the translation between real to pseudo IP addresses, i.e. the changing of addresses on the institutional (local) network quickly becomes irrelevant. Moreover, even under a MTD, the gateway has to assume a static address in order for each host to contact it. This means, that the gateway can always be identified, turning it into a static target.

3. METHODOLOGY

Previous research with a MTD articulated a basic goal of reacting to Worm attacks, but without actually simulating the action of the proposed framework on a network [5, 6]. Hence, a policy is established for modifying the network (the MTD), but as a consequence introduce limitations in terms of responsiveness (as reviewed in Section 2). In this work, we will take a data driven approach to first detecting the types of attack and then on detection, we will initiate a MTD as a policy decision using a simulated SDN. Given the emphasis of MTD with regards to combating Worm attacks we will use the detection of such attacks as the basis for evaluating our MTD initiator.

Detectors will operate on TCP packet header and payload information in order to provide real-time operation for the detectors. Detectors will be developed for specific classes of attacks using the data formats discussed in Section 3.1

3.1 Packet Format

TCP packet format is modelled in terms of 5 categories of information:

1. Source IP – 4 bytes
2. Destination IP – 4 bytes
3. Source Port – 4 bytes
4. Destination Port – 4 Bytes

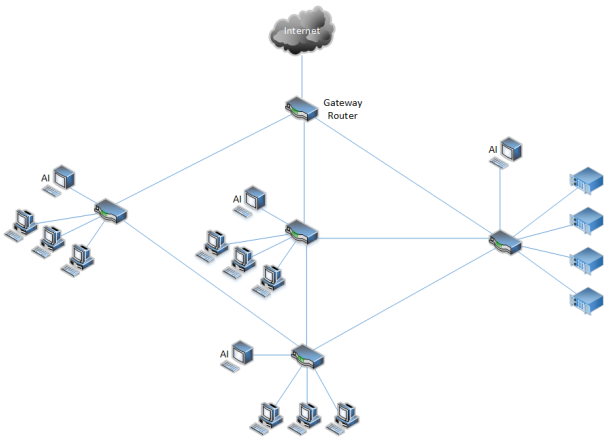

Figure 1: Architecture of a SDN-based simulated network for data generation and evaluation. Detectors (or 'AI') appear at each host. Gateway node represents the location for a name server (DNS). Under MTD all requests for (pseudo) IP would first have to visit the gateway as no caching at a host is possible.

5. Payload – 32 Bytes (8×4)

Thus, from a detector's perspective packets consist of 48 bytes or 12 integers. Potentially these values provide 12 attributes from which to construct the detector. However, a neural representation will be assumed for the detector in which case each neurone performs the operation: $y = f(net)$ and $net = \sum w_i x_i$ where $f(\cdot)$ is the sigmoid activation function. Learning gradients will only exist if the attributes are normalized such that most of the variation in attributes appears in the transition region of the activation function. With this in mind, all 12 attributes are normalized to the interval $[-1.0, 1.0]$.

Three classes of packets will be generated: Normal, LAND, and Worm. **Normal traffic packets** were designed to communicate data between hosts on a $10.0.0.X$ network where $1 \leq X \leq 17$. Specifically, four hosts appear on each of three routers and one router has five hosts: $4 \times 3 + 5 = 17$ (Figure 1). This corresponds to the complexity that might appear in a class C network within a university campus or government / commercial organization. The ports could be anything in the port range, the IPs were always on the network, and the payload represented hashes on a database of text phases.

LAND packets represent DoS attacks in which the objective is to trick host(s) into opening all their ports. At the packet level this amounts to explicitly requesting a source and destination IP address with the same value.[1] All the remaining attributes assumed properties that overlapped with the definitions used for a normal packet. **Worm packets** were designed to specifically attack port 139 (i.e. as per the Sasser worm) with a buffer overflow, i.e. an internal propagation or *IntProp* exploit.[2] The payload in this case takes the form of ASCII character sequences (as opposed to the hashed text of normal or LAND packets). In short, we are emphasizing light-weight detection at the packet level. More detailed detection could take place at the behavioural level (say, in terms of network flow preprocessing), but only at the expense of no longer supporting real-time operation. Moreover, features might also be designed to 'recognize' specific attacks, however, doing so also potentially leads to brittle 'signature style' detectors that can be evaded by mimicry style attacks (e.g., [8]). In this work, we therefore require our light-weight detector to recognize general properties within single packets that can potentially mark it as malicious. Given that flagging an attack would then initiate a MTD, additional forensic analysis could be performed in parallel to resolve the nature of attacks in greater detail.

[1]http://www.juniper.net/techpubs/en_US/junos12.1/topics/concept/denial-of-service-network-land-attack-understanding.html

[2]http://www.iss.net/threats/172.html

3.2 Neuro-evolution

Two independent detectors are evolved one for detecting each threat. In this case, the NEAT framework was assumed for each detector (Section 3.2.1). The question then arises as to how to combine the two detectors (building a combined detector is possible, but implies that any updates to the detector requires a complete reconstruction). One possible approach could be to evolve a third network, the gate, which learns under what conditions to switch/mix between either of the 'experts'. In practice, a simpler solution was found by using the output of each expert to explicitly flag ambiguity. That is to say, a committee architecture using gating is appropriate when ambiguity exists between the expert outcomes. However, in this setting the cases corresponding to erroneous detection were most likely to be flagged by the ability of detectors to actually flag ambiguity.

In the following, we first summarize the properties that make neuro-evolution through NEAT particularly useful. Then, we establish the design approach adopted for disambiguation of detectors. Section 4 will present the empirical findings within the context of the system design established below.

3.2.1 NEAT

Neuro-evolution of Augmenting Topologies (NEAT) is an approach to neuro-evolution in which crossover context is explicitly supported [12, 13], i.e. previous to the NEAT algorithm, there was no basis for contextual information. Thus, arbitrary switching groups of neurones between two parents would result in children with very ineffectual behaviours. Conversely, NEAT makes use of a genotype that explicitly marks unique links (between nodes) and therefore provides the basis for enforcing minimal levels of structural context between pairs of parents. Moreover, the NEAT algorithm leverages the concept of complexification, where the initial population is limited to single neurones, and incrementally lets the neural network (NN) topology 'complexify' as the search process progresses. In short, instead of relying on user-made neural networks with which the algorithm attempts to properly re-weight, the algorithm begins with some base neurone (typically a fully connected bipartite graph between input nodes and output nodes) and attempts to incrementally modify the network structure by adding nodes throughout the learning process. This means that if the task is linearly separable, then such solutions will be found first. Conversely, if the task is not linearly separable, then NEAT will incrementally 'complexify' the topology as opposed to the user having to guess an appropriate non-linear representation.

Table 1: Interpreting the outcome from the LAND (DoS) and Internal Propagation (worm) detection 'expert' neural networks. As long as both experts do not label the same packet as an attack no corrective action is necessary. If both experts label the packet as an attack a third 'disambiguation' network is deployed

NEAT expert		Interpretation	Action
LAND (DoS)	Int Prop (Worm)		
0	0	Normal packet	valid
0	1	Worm packet	valid
1	0	LAND packet	valid
1	1	Ambiguous	invalid

Since the initial development of NEAT, there have been multiple frameworks developed for evolving neural networks with contextual information (e.g., EANT [10], rtNEAT [11]). However, in this research the original NEAT algorithm is considered complex enough for the purposes of this task. The specific code distribution assumed in this work is that of jNEAT.[3]

3.2.2 Committee machines

Real-time operation requires a 'light-weight' detector, implying that the detectors should be as efficient as possible. Thus, increasing accuracy of the detector(s) by large numbers of classifiers in parallel to construct an ensemble against which majority voting is performed would not be appropriate. Conversely, assuming a gating architecture as popularized by the 'mixtures-of-experts' framework (e.g., [4]) could potentially represent a viable approach that also remains light-weight. The gate represents a neural network that learns under what conditions to switch and / or mix between the labels suggested by the experts. However, such architectures provide most leverage when experts consistently disagree over a significant portion of the classification task. Instead, for the most part the experts – NEAT trained to recognize single attack types – are accurate. What potentially could also be corrected, without reference to some external oracle, is the case when they both attempt to label a packet as different types of attack, see Table 1, or a form of forensic disambiguation.

Training of the expert networks is performed using 320 training packets and 400 validation packets each. At the end of this process the ambiguous scenario from Table 1 accounts for $\leq 6\%$ of the data expert networks employed for training. We can then use the trained expert networks to bootstrap the data generation process such that $40,000$ packets are created that conform to the 'ambiguous' expert outcome.[4] These packets were then divided into $10,000$ each for training and validation and $20,000$ for test purposes. Training of each expert network was performed for 400 generations whereas $1,000$ generations were used to evolve the disambiguation neural network.

[3]http://nn.cs.utexas.edu/?jneat

[4]Implies that roughly $800,000$ packets were created to do so.

In all cases the fitness function takes the form of the multi-class detection rate, where the per class detection rate has the form:

$$DR_c = \frac{tp_c}{tp_c + fn_c} \quad (1)$$

where tp_c, fn_c are the counts for true positive and false negative rates w.r.t. class c.

The multi-class detection rate now has the form:

$$DR = \frac{1}{C} \sum_{c=[1,\ldots,C]} DR_c \quad (2)$$

Under the expert networks the $C = 2$ or normal and malicious, whereas under the disambiguation network the two classes take the form of LAND and IntProp.

3.3 Teams of Programs

A second framework in which task decomposition forms a central theme is that of teams of programs (a genetic programming paradigm). Specifically, the SBB framework will be assumed where this enables the size of teams to be discovered as well as the programs [2].[5] SBB employs two populations, a team population and a program population. Each individual from the team population (tm_i) defines a candidate team by indexing a subset of programs from the program population. In assuming a variable length representation, team size is evolved. Programs (sym_j) learn context for suggesting a class label. With respect to team, tm_i, and packet, p_k, only program $sym \in tm_i$ with maximum output under packet p_k wins the right to suggest its label. Such a framework has demonstrated the capacity to decompose classification tasks through cooperative coevolution [9].

4. EMPIRICAL STUDY

4.1 Parameterization

As detailed in Section 3.1, the state of our packet structure is expressed as a vector of 12 floating point values derived from the data in our custom designed packets. Each integer represents a component of the custom packet, including source and destination IP addresses, source and destination ports, sequence number, acknowledgement number, and a payload. The integers were then divided by the maximum value a signed integer could represent to produce a float in the interval $[0.0, 1.0]$ (see Section 3.1 for the rationale).

There is potentially a large number of parameters that could be tuned when evolving neural networks with NEAT, a complete declaration is provided in Table 2. However, most of these follow from the defaults provided in the jNEAT implementation. One area in which we diverged from the typical parameterization was with respect to the disambiguation network. In particular, although NEAT initializes with networks defined by a single neurone, each neurone is fully connected. Parameters were therefore modified for the disambiguation network in order to encourage the removal of links, thus promoting the design of neural network topologies that did not necessarily index all the attribute space. From the larger topological perspective, this also introduced a bias to finding sparse topologies.

[5]jSBB implementation employed from https://web.cs.dal.ca/~mheywood/Code/

Table 2: **NEAT Parameters. Both Expert NN and Disambiguation NN were generated using the same parameters.**

Parameter	Value
p_trait_param_mut_prob	0.5
p_trait_mutation_power	1.0
p_linktrait_mut_sig	1.0
p_nodetrait_mut_sig	0.5
p_weight_mut_power	2.5
p_recur_prob	0.0
p_disjoint_coeff	1.0
p_excess_coeff	1.0
p_mutdiff_coeff	0.4
p_compat_threshold	3.0
p_age_significance	1.0
p_survival_thresh	0.2
p_mutate_only_prob	0.25
p_mutate_random_trait_prob	0.1
p_mutate_link_trait_prob	0.1
p_mutate_node_trait_prob	0.1
p_mutate_link_weights_prob	0.9
p_mutate_toggle_enable_prob	0.2
p_mutate_gene_reenable_prob	0.05
p_mutate_add_node_prob	0.03
p_mutate_add_link_prob	0.08
p_interspecies_mate_rate	0.0010
p_mate_multipoint_prob	0.3
p_mate_multipoint_avg_prob	0.3
p_mate_singlepoint_prob	0.3
p_mate_only_prob	0.2
p_recur_only_prob	0.0
p_pop_size	500
p_dropoff_age	50
p_newlink_tries	50
p_print_every	10
p_babies_stolen	0
p_num_runs	1
p_num_trait_params	8
epoch	1,000

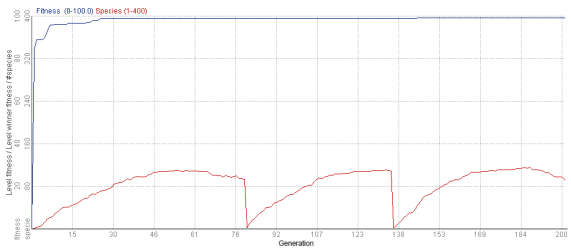

Figure 2: Example fitness (upper curve) and speciation (lower curve) for the LAND expert over time. Note the periodic resetting of species as plateauing / stagnation is detected.

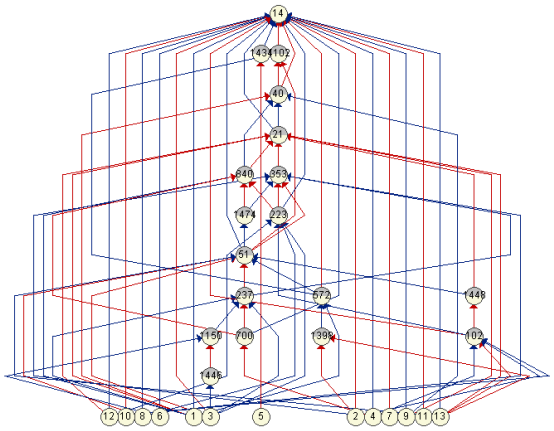

Figure 3: Example NN topology representing the LAND expert.

4.2 Results with NEAT

As there are a total of three independent networks, we will consider each one alone and then look at the combined results.

4.2.1 LAND Expert

The LAND network needs to distinguish the difference between normal and LAND (i.e., DoS) attacks (see Section 3.1). Figure 3 provides an example of a typical training curve, and Figure 2 provides an illustrative topology of a champion solution post training. From the resulting topology, it is clear that the task is not linearly separable, but likewise the resulting network is also reasonably sparse. Table 3 summarizes the confusion matrix for test performance in terms of the median over 20 independent runs. We also note the performance under test of the best individual as identified from training, where this reflects the performance of the individual that would be deployed in practice.

4.2.2 Internal Propagation Expert

The Internal Propagation network needs to distinguish the difference between normal and Internal Propagation or IntProp (i.e., worm) attacks (see Section 3.1). Figure 5 provides an example of a typical training curve, and Figure 4 provides an illustrative topology of a champion solution post training. Clearly a more complex topology results under the

Table 3: Confusion matrix for LAND NN. Mean performance (%) of 20,000 Test packets (best individual). Results averaged over 20 runs.

Actual Label	NEAT expert	
	LAND	Normal
LAND	86.8 (93.8)	13.2 (6.2)
Normal	15.3 (9.0)	84.7 (91.0)

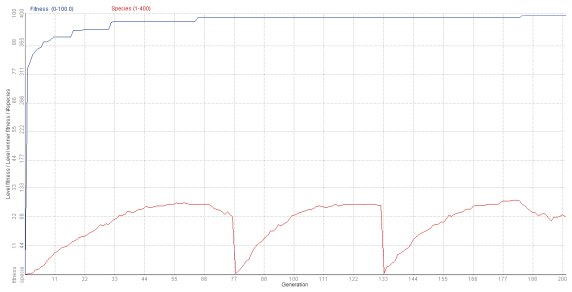

Figure 4: Example fitness (upper curve) and speciation (lower curve) for the Internal Propagation expert over time. Note the periodic resetting of species as plateauing / stagnation is detected.

LAND attack, implying that the identification of an internal propagation (IntProp) attack is a more demanding task than detecting LAND attacks. Table 4 summarizes the confusion matrix for test performance in terms of the median over 20 independent runs. We also note the performance under test of the best individual as identified from training, where this reflects the performance of the individual that would be deployed in practice.

4.2.3 Disambiguation Network

The Disambiguation network needs to distinguish the difference between LAND and Internal Propagation attacks, or the 'conflict' case of Table 1, Section 3.1. Figure 7 provides an example of a typical training curve, and Figure 6 provides an illustrative topology of a champion solution post training. This is the most demanding task, however, sparseness is also widely apparent. Table 5 summarizes the confusion matrix for test performance in terms of the median over 20 independent runs. We also note the performance under test of the best individual as identified from training, where this reflects the performance of the individual that would be deployed in practice.

Table 4: Confusion matrix for IntProp NN. Mean performance (%) of 20,000 Test packets (best individual). Results averaged over 20 runs.

Actual Label	NEAT expert	
	IntProp	Normal
IntProp	82.9 (92.9)	17.1 (7.1)
Normal	12.7 (7.6)	87.3 (92.4)

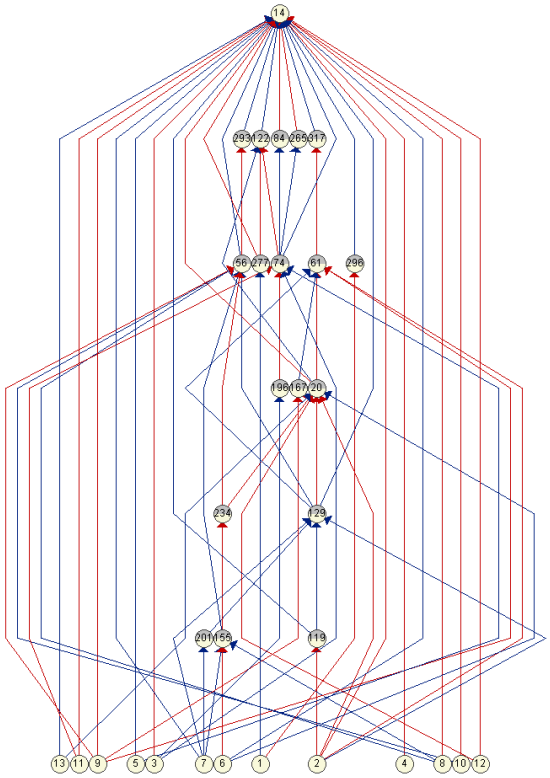

Figure 5: Example NN topology representing the IntProp expert.

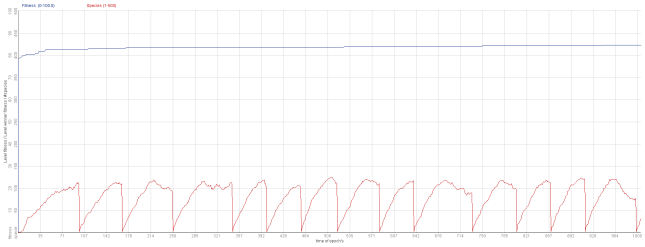

Figure 6: Example fitness (upper curve) and speciation (lower curve) for the Disambiguation NN over time. Note the periodic resetting of species as plateauing / stagnation is detected.

Table 5: Confusion matrix for Disambiguation NN. Mean performance (%) of 20,000 Test packets (best). Results averaged over 20 runs.

Actual Label	NEAT expert	
	IntProp	Normal
IntProp	83.3 (90.8)	16.6 (9.2)
Normal	16.6 (9.7)	83.4 (90.3)

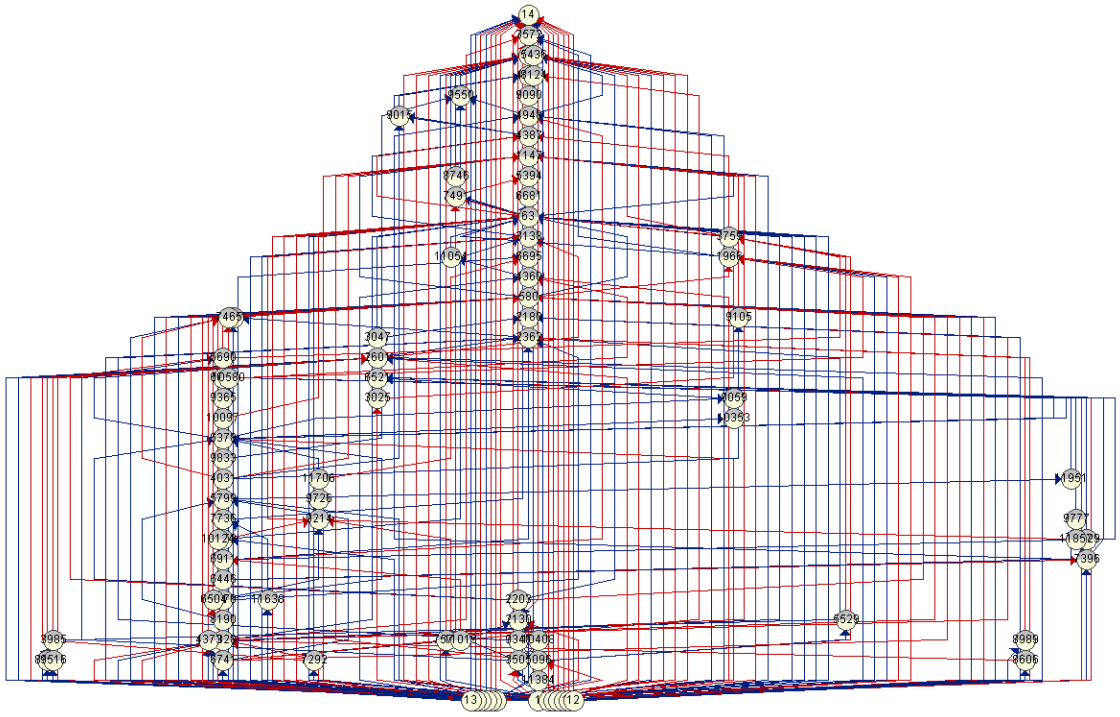

Figure 7: Example NN topology representing the Disambiguation neural network.

Table 6: Relative complexity of each NN.

Neural Network	Mean node count	Std. Deviation
LAND (DoS)	13.3	3.39
IntProp (worm)	17.8	2.71
Disambiguation	65.5	11.24

Table 7: NEAT – Overall confusion matrix. Mean performance (%) of 20,000 Test packets (best individual). Results averaged over 20 runs.

Actual Label	NEAT		
	Normal	LAND	IntProp
Normal	80.8 (90.8)	10.7 (0.5)	8.5 (8.7)
LAND	4.5 (10.0)	91.5 (82.1)	4.0 (7.9)
IntProp	15.9 (0.6)	14.5 (6.8)	69.5 (92.6)

4.2.4 Combined performance

Sections 4.2.1 through 4.2.3 discussed performance on an individual-wise basis. Here, we review the overall 'system' performance. The relative complexity of each NN comprising the overall architecture is summarized in terms of the respective node counts (Table 6). These results reinforce the observations made in the earlier sections, with LAND detection representing the simplest task and the final disambiguation step being the most complex.

Overall detection rates for the system under unseen test conditions are summarized in Table 7. This reflects the average Detection rate of each class under test conditions: 10,000 normal; 5,000 LAND; 5,000 IntProp. packets. None of this data was used to construct models. Also summarized is the best of model performance, as in the test performance of the best individual as identified from the training conditions. In short, the model deployed would be capable of detecting Normal and IntProp (worm) packets with detection rates $> 90\%$ and LAND (DoS) packets with $> 80\%$. Note that this refers to attacks that are characterized (identified) at the packet level on a packet-by-packet basis. Attacks characterized across multiple packets would imply detection through, say, flow style features (at the loss of real-time operation).

4.3 Teaming GP

Section 4.2 demonstrated that the task is not linearly separable (solutions would take the form of a single neurone) and indicated that task decomposition is beneficial (total of three neural networks deployed in combination). As noted in Section 3.3, the SBB framework has the potential to take this further as the process of discovering the amount of 'modularity' is in itself performed through evolution.

Table 8 summarizes the resulting performance of SBB as deployed under test conditions. Considerable benefits appear to be availed through pursuing open-ended task decomposition as opposed to assuming a prior class wise decomposition, as under NEAT. Naturally, we make no claims regarding the generality of this outcome, and adopting other forms of neuro-evolution might give similar results.

Table 8: Teaming GP – Overall confusion matrix. Mean performance (%) of 20,000 Test packets (best individual). Results averaged over 20 runs.

Actual Label	NEAT		
	Normal	LAND	IntProp
Normal	92.3 (99.8)	4.0 (0.02)	3.7 (0.14)
LAND	0.27 (0)	99.5 (100)	0.2 (0)
IntProp	0.42 (0)	0.23 (0)	99.3 (100)

5. CONCLUSION

MTD when applied to dynamically modifying the network address space is currently assumed to be applied on a continuous basis. However, there are several costs involved, not least that host machines can no longer cache their IP. This implies that every IP request has to be referred to the DNS (inserting a bottleneck in any communication) while implying that the DNS identity has to be static. In this work, we consider MTD to be a policy deployed under conditions as identified by a suitable light-weight detector. Light-weight implies that the detector is required to operate from packet data as opposed to flow data. We investigate two examples of exploit within this context: DoS and worms (code that self propagates). Naturally, such a detector would act as the first wall in a multifaceted defense, with other detectors and / or stochastic mechanism acting as triggers for initiating a MTD.

The NEAT algorithm is shown to be an effective framework for identifying sparse neural network topologies for detecting the two primary attack types. Particular attention is paid to resolving which type of attack has been initiated. From the MTD initiation perspective, just the two 'expert' networks is sufficient for determining when to initiate MTD. Investigating the open-ended evolution of task decomposition through the coevolution of programs indicated further improvements.

Future work could consider alternative types of detector and benchmark with light-weight signature based detectors. Previous research has demonstrated that various machine learning methods are capable of providing effective detectors [14, 15]. However, MTD represents an area in which a detector may operate more interactively with counter-measures with the goal of providing a more autonomous system, opening the opportunity to maintain the operation of a network as opposed to merely closing ports (denying access to legitimate and illegitimate users alike).

6. ACKNOWLEDGEMENTS

This research is supported by Raytheon SAS. The research is conducted as part of the Dalhousie NIMS Lab at: https://projects.cs.dal.ca/projectx/.

7. REFERENCES

[1] S. Antonatos, P. Akritidis, E. P. Markatos, and K. G. Anagnostakis. Defending against hitlist worms using network address space randomization. *Computer Networks*, 51(12):3471–3490, 2007.

[2] J. A. Doucette, A. R. McIntyre, P. Lichodzijewski, and M. I. Heywood. Symbiotic coevolutionary genetic programming. *Genetic Programming and Evolvable Machines*, 13:71–101, 2012.

[3] M. Dunlop, S. Groat, W. Urbanski, R. Marchany, and J. Tront. Mt6d: A moving target ipv6 defense. In *Military Communications Conference, (MILCOM)*, pages 1321–1326, 2011.

[4] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.

[5] J. H. Jafarian, E. Al-Shaer, and Q. Duan. Openflow random host mutation: Transparent moving target defense using software defined networking. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, pages 127–132, 2012.

[6] J. H. H. Jafarian, E. Al-Shaer, and Q. Duan. Spatio-temporal address mutation for proactive cyber agility against sophisticated attackers. In *Proceedings of the First ACM Workshop on Moving Target Defense*, pages 69–78, 2014.

[7] D. J. John, R. W. Smith, W. H. T. abd D. A. Cañas, and E. W. Fulp. Evolutionary based moving target cyber defense. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, pages 1261–1268, 2014.

[8] H. G. Kayacik, A. N. Zincir-Heywood, and M. I. Heywood. Evolutionary computation as an artificial attacker: generating evasion attacks for detector vulnerability testing. *Evolutionary Intelligence*, 4:243–266, 2011.

[9] P. Lichodzijewski and M. I. Heywood. Symbiosis, complexification and simplicity under GP. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, pages 853–860, 2010.

[10] J. H. Metzen, M. Edgington, Y. Kassahun, and F. Kirchner. Performance evaluation of EANT in the robocup keepaway benchmark. In *IEEE International Conference on Machine Learning and Applications*, pages 342–347, 2007.

[11] K. Stanley, B. Bryant, and R. Miikkulainen. Real-time neuroevolution in the nero video game. *IEEE Transactions on Evolutionary Computation*, 9(6):653–668, 2005.

[12] K. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[13] K. Stanley and R. Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21:63–100, 2004.

[14] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin. Intrusion detection by machine learning: A review. *Expert Systems with Applications*, 36:11994–12000, 2009.

[15] S. X. Wu and W. Banzhaf. The use of computational intelligence in intrusion detection: A review. *Applied Soft Computing*, 10(1):1–35, 2010.

[16] J. Yackoskia, P. Xie, H. Bullen, J. Li, and K. Sun. A self-shielding dynamic network architecture. In *Military Communications Conference, (MILCOM)*, pages 1381–1386, 2011.