# Autonomous Rule Creation for Intrusion Detection

Todd Vollmer
Idaho National Laboratory
Idaho Falls, ID, USA
denis.vollmer@inl.gov

Jim Alves-Foss, Milos Manic
University of Idaho
Dept. Computer Science
Moscow, ID, USA
jimaf@uidaho.edu, misko@ieee.org

*Abstract*—**Many computational intelligence techniques for anomaly based network intrusion detection can be found in literature. Translating a newly discovered intrusion recognition criteria into a distributable rule can be a human intensive effort. This paper explores a multi-modal genetic algorithm solution for autonomous rule creation. This algorithm focuses on the process of creating rules once an intrusion has been identified, rather than the evolution of rules to provide a solution for intrusion detection. The algorithm was demonstrated on anomalous ICMP network packets (input) and Snort rules (output of the algorithm). Output rules were sorted according to a fitness value and any duplicates were removed. The experimental results on ten test cases demonstrated a 100 percent rule alert rate. Out of 33,804 test packets 3 produced false positives. Each test case produced a minimum of three rule variations that could be used as candidates for a production system.**

***Keywords-Intrusion detection; Computational intelligence; Genetic algorithms***

## I. Introduction

In the voluminous amounts of research on anomaly based intrusion detection systems, little consideration has been given to a posteriori communication mechanisms [1]. Anomaly based systems typically produce two types of output (anomalous or not, perhaps with a companion confidence score). Other systems provide a label associating the data with a known attack type i.e. Denial of Service [2]. If the ability to identify a novel intrusion is valuable in one network it seems reasonable that knowledge gleaned from those efforts would be useful in another. Given the numerous different implementations of anomaly detection and the resulting unique knowledge representations used to find anomalies, there does not appear to be an automated solution to translate the newly discovered detection criteria into a more widely accessible format. Indeed this knowledge transfer is typically performed by a human expert manually examining the traffic and creating a rule. The rule can subsequently be used as input to any system capable of deciphering the syntax.

There are two fundamental approaches for Network Intrusion Detection Systems (IDS): behavior and rule based. Behavior based systems typically maintain a model of normal system behavior and raise exceptions when parameters fall outside the norm. Computational intelligence algorithms such as neural networks and clustering have been shown to be effective solutions at identifying anomalous behaviors [3], [4]. Rule based systems use widely distributable predefined signatures to detect known network issues.

The cost of developing and maintaining rule sets is an important issue for the rule based systems. Human experts are required to create, test and distribute the rules. Given a network trace containing anomalous packets, an expert must investigate the numerous attributes that uniquely identify the attack. This involves laboriously examining the packets for information and creating a candidate rule. Correct rule creation is then a manual process of trial and error where each trial run is examined for a proper alert on a test file of captured network data. Finally, if the expert chooses to do so, the rule is submitted to a rule repository where it may be accepted into a public distribution system.

This paper explores a solution to autonomously create IDS rule sets utilizing evolutionary computation techniques. This is accomplished by implementing a Genetic Algorithm (GA) to autonomously create rules from identified network packets that are indicative of system misuse. These packets, used as input, originate from network traffic identified by a behavior based IDS. The resulting rules from processing the packets may not always be optimal for direct distribution but should provide a basis for reducing subsequent expert analysis effort. The system described here can be considered as a one way communication mechanism bridging the two types of intrusion detection systems.

The GA chosen for implementation provides multiple near optimal unique rules that are made available for further evaluation by a human expert. It would be a straightforward process to simply provide a single rule that contains all possible attributes of a given network packet. However this could lead to over-fitting of the rule to a single attack instance. In general, the more specific a rule is the more likely it is to eliminate false positives. However, if a rule is too specific it may become brittle in the sense that any minor variation in the attack may be missed. In addition, the more detailed a rule becomes, the more computational effort is required to process it.

The problem we are solving is similar in approach but different in context from past work in network intrusion detection. Previous work as in [5] was primarily concerned with developing a multiple rule set able to separate known behavior from unknown. Our effort is to produce a set of near optimal IDS rules for a single specific anomalous instance previously detected by a behavior based system. This is accomplished via use of a GA that has several unique characteristics differing from previous research efforts, namely the representation of the population as syntactically correct Snort rules (as opposed to binary or researcher created syntax) and a three part fitness function. This fitness function is designed to optimize the resulting rule sets for the Snort rule engine based on published best practices and characteristics of historical rule repositories.

As a consequence of the population representation a unique distance method was required. The cardinality of the rule genotype attributes is variable. Gene features may be removed or added as Snort rules are not limited to a fixed number of fields. There are optional fields and most production rules contain a variety of them. Because of this, mutation and crossover required special consideration as well. The presented GA solution accommodates these features.

The input data is different than prior work as well. In this case a single instance of labeled data is presented to the system and the rule set is evolved to detect it. The question is can this be done without producing a rule that provides a positive response to network traffic that is not an anomaly (false positive).

## II. Background

Genetic Algorithms are an effective heuristic search technique inspired by concepts of evolutionary biology. They became popular with the published work of John Holland in the 1970's. For an evolutionary algorithm to be categorized as a GA it needs a population representation of possible solutions, variation operators, selection and replacement mechanisms. When optimizing multi-modal functions a conventional GA's population tends to converge to one of the optimal, or near optimal points.

A specific implementation of a GA called Restricted Tournament Selection (RTS) provides a solution for maintaining several optimal solutions in the population [6]. This modification is possible because a GA utilizes a population of many (hundreds, thousands or more) possible solutions. This is effective in situations when the fitness function is not capable of representing the fitness with a high fidelity. An evaluation function is used to determine the 'fitness' of individuals in a population. This fitness is a measure indicating how well the individual solves a given problem.

The RTS algorithms ability to maintain several fit solutions will be leveraged to produce unique rules that alert on a given anomalous packet. This assumes that the solution surface for generating rules is not uni-modal. There are several syntax models available for the rule format. Because of its widespread use the Snort rule format was implemented.

### A. Snort

Snort is an open source IDS created by Martin Roesch [7]. It is capable of performing protocol analysis, content searching/matching and utilizing predefined signatures. Several signature rule sets are available for use including those officially approved by the Sourcefire Vulnerability Research Team (VRT) and those contributed by other communities. Three sources for acquiring rules were used in this project. The first two sets are VRT certified rules and community rules available online at http://www.snort.org. The third set was obtained from emerging threats and is available online at http://www.emergingthreats.net. All of these sets combined to define 16,181 rules including 146 ICMP specific rules.

Snort supports a simple rule language that matches against network packets, generating alerts or log messages. Rules are broken into two logical areas: rule headers and rule options [8]. Rule headers contain required protocol fields that every rule must have and rule options contain a list of optional information used to refine a match. The rule field format and an example rule are as follows:

```
<action><protocol><sourceIP><sourcePort><direction>
<destIP> <destPort> (<rule options>)

alert tcp any any -> 192.168.1.0 21 (content:"USER")
```

The rule action tells Snort what to do when a match occurs. A common action, as shown in the example, is to log information to an alert file. The protocol field specifies one of four possible values: TCP, UDP, ICMP and IP. Each value has options specific to the protocol available for use in the option section. The source IP address field can contain the keyword "any", a single IP address or a CIDR (Classless Inter-Domain Routing) block. CIDR blocks allow for specifying ranges of IP addresses. Port numbers may be specified as a single static port, a range or use the keyword "any".

There are two direction operators. One specifies that the source and destination portions of a rule must match the appropriate items from a packet. The bidirectional operator indicates that the source and destination sections can match either portion of a packet. This allows for tracking two way conversations as seen between a typical client and server application.

Rule options provide further refinement of matching parameters and tie the rule to a rule identification system. There are four major categories of rule options: general, payload, non-payload and post-detection. General options provide information about the rule such as reference information, rule identification and specific log messages. Payload options examine data contained in the packet data such as content matching expressions. Non-payload options provide matching specifications against packet header data outside of ports and IP addresses. Options include fragment offsets, time-to-live values and specific IP options.

### B. Snort Rule Processing

This section briefly describes how Snort internally processes rules and a required modification to that process. The rule can be seen as a Boolean truth statement. In order for Snort to identify a match, a logical and of all positive field matches is utilized. Upon discovery of a false condition in the *and* evaluation, further processing of that rule is halted.

Snort builds a tree data structure used to compare rule values against packet features. Each mandatory field in a rule is stored in rule tree node (RTN). An OTN (optional tree node) is associated with an RTN and used to store optional rule fields. If multiple rules have the same RTN fields they are only represented by a single node. This optimization feature allows for removal of multiple rules from consideration once a negative match occurs.

The optimization aspect of this rule tree structure implementation is detrimental to our proposed GA fitness algorithm discussed in section 3. If the Snort engine processes a rule section that proves false, that rule is no longer considered for a possible match. This is a logical performance enhancement that speeds the execution of the rule engine. However this short circuit of rule evaluation prevents using a modified Snort source as a basis for rule fitness evaluation.

An attempt was made to encourage the GA to craft good rules to reduce human expert analysis effort spent on examining the rules produced. Good being a subjective term we have defined it in respect to three measurements. First the rule has to be able to recognize the packet as being an anomaly. Second it should conform to a grammar checker called dumbpig produced by Ward [9]. This tool parses a rule, reports on badly formatted entries, incorrect usage, and alerts to possible performance issues. Finally the rule should be similar in the number and type of fields used in existing rules. The assumption being that these rules have been vetted by the community of experts and are therefore worthy of emulation.

## III. Experimental Approach

This section describes the implementation details of the final solution. The input data processing and its representation are presented. This data is fed to a GA implemented with RTS. A fitness function implemented in three parts is described. The resulting rules produced by the GA are then sorted by fitness and the top three rules are presented as possible solutions.

### *A. Input Processing*

It is assumed that the anomalous network traffic will already have been identified in advance. Systems such as those described by Linda et al [3] and Taylor with Alves-Foss [4] are capable of isolating this kind of traffic. The network traffic data is expected to be contained in a PCAP formatted file. PCAP data files have become industry standard and are the output of an application programming interface library called libpcap. Utilizing the Perl CPAN module (Net::Pcap) based on this library, the information is read into the program memory space. Having network data stored in files, as opposed to real time capture on a network interface, enables offline processing. However the PCAP library is capable of performing both functions.

After the packets are read into memory each one is parsed and stored in a data structure. For ICMP packets, this structure includes the following: source IP address, destination IP address, ICMP id, type, code, sequence number and packet size. These are all fields used in the GA population representation described in the next section.

### *B. Genetic Algorithm Description*

A pseudo-code implementation of a GA is presented in Fig. 1; numbered lines are described more fully later.

One of the first tasks in building a GA is to decide upon a representation of the solution population (line 1) and create a number of individuals in that population. Each individual is stored as a representation of a variable length list of Snort rule fields. A Perl associative array maintains these mixed type values. The field key, type and acceptable range values are shown in Table 1. Field key values are taken directly from the Snort rule syntax definitions. The ranges may include values that are not allowed according to specifications but are technically allowable within the data type or are specific to Snort rule processing. For instance, the src and dst fields can contain a variable name that Snort will replace with a configurable value at runtime.

Each individual in this population can be represented using a variable length vector $\vec{v}_i = (\mathrm{x}_0, \ldots, \mathrm{x}_\mathrm{n})$ of mixed data type values x. The total population is the set of vectors $P_T = \{\vec{v}_0, \vec{v}_1, \ldots, \vec{v}_T\}$. For the experiment described here the population size T was fixed at 200. This value was chosen as a reasonable tradeoff between time efficiency and solution convergence. The first generation of individuals was populated by randomly generating values in the domain range of the given fields with a random number of options.

The selection of a field's inclusion in a rule was not uniformly random in all cases. An analysis of the fields present in the 146 ICMP specific Snort rules was performed. Statistics were compiled on the type and frequency of rule option values present. An assumption was made that these rules, having been vetted and accepted into the official repositories by experts, exhibit desirable characteristics worthy of emulation.

Whenever a decision needs to be made about a field's inclusion a random number $r$ is generated in the range 0 to N where N is the number of rules. Given a frequency count $fc$ of a given field f from the set of rules N such that $fc <$ cardinality (N), the field $f$ is included in a rule if the generated $r$ is less than $fc$. In other words, a field is randomly selected for inclusion proportional to the relative

```
1 Create an initial population
2 LOOP while below execution count
   3 select individuals as parents
   4 create children from parent
   (crossover/mutation)
   5 select and replace individuals with children
   6 update fitness values
End LOOP
```

Figure 1. Genetic Algorithm

TABLE 1: RULE POPULATION FIELDS

| Field | Type | Range |
|---|---|---|
| proto | string | 'icmp' |
| src | string or CIDR | '$SNET' or 0.0.0.0/0 |
| sport | string or integer | 'any' |
| dst | string or CIDR | '$DNET' or 0.0.0.0/0 |
| dport | integer | 'any' |
| itype | integer | 0-255 |
| icode | integer | 0-255 |
| icmp_id | integer | 0-65535 |
| icmp_seq | integer | 0-65535 |
| dsize | integer | 0-65535 |
| content | string | Any text |

frequency of its presence in the original 146 ICMP rules. For example, the dsize field occurred 13 times in the set of 146 rules. Therefore close to a 9% chance exists that this field will be picked for inclusion.

After an initial population is created, a series of variations and replacement selections must take place on some of the population individuals. This process repeats until some acceptable solution or predefined iteration limit is reached. For this project a fixed number of 1000 iterations were selected. After looping (line 2) the defined number of iterations, processing terminated and the fittest individuals maintained by the RTS selection algorithm were isolated as the final best rule set. The details of variation and selection that occur in this loop are presented next.

A steady state population model was implemented. This means that for a given iteration in the loop, only a maximum of two individuals in the original population are selected for replacement. This is in contrast to a generational model where the entire population is replaced by the offspring. In this case, two individuals were selected as parents and two offspring were created from them. Subsequently, two more individuals were then selected for replacement by the new offspring creating a new generation.

The two candidate parent individuals were randomly selected independent of any fitness or distance measure (line 3). Crossover between the two parents to create two new offspring utilized uniform crossover. Creation of these two offspring is a two step process consisting of rule header creation and rule option creation.

Rule header creation consists of randomly choosing a field from either parent and copying that value into the child. For the given ICMP problem, there is not much variability in the header portion and header creation is not that important relative to the option fields.

Each option field of the new offspring rule is a copy of a field selected from a given parent. For each offspring a primary parent of the original two candidates is selected. Each option field of that primary parent is then considered for inclusion in the new child. A random number from a uniform distribution over 0.0-1.0 is generated. If the number is greater than 0.5 than the primary parents value is utilized; otherwise consideration is given to retrieving the information from the second parent. At this point, if the second parent contains the option field, it is copied into the child. However, if the field does not exist in the second parent, then the default action is to revert to the value from the primary parent. The net result of this is a child that contains the same number of options as the primary parent but with potentially different values from the secondary parent. Variation in child option count is left as a possibility in the mutation operator.

After crossover, mutation occurs on both children prior to replacement selection (line 4). As was the case in crossover, rule header mutation and rule option mutation behave differently.

There is a 25% independent chance of mutating the rule header of a single child. Once selected for mutation the header src or dst IP field is randomly chosen for change. This change consists of randomly choosing between the three available options: any, Snort IP variable ($DNET or $SNET) and the test packet IP address (source or destination as appropriate).

Option mutation is considered separately from header mutation and it has a 25% chance of occurring as well on each option in the child. The reason for this separation of mutation is a result of programming convenience and need not occur in this manner. The integer domain values such as icode, itype, icmp_id and icmp_seq are processed the same according to their domain ranges. A random value within a window bracketing the original value is chosen. If this operation results in a value outside of the acceptable domain range defined in Table 1, the value is set equal to the closest boundary value. Content keyword mutation occurs by randomly selecting a range of the test packets data load and transforming any non printable characters into hex representation. Finally, with a 10% chance, it is possible that any of the options are simply removed from the rule. A future enhancement for consideration would be to add a nonexistent parent option field to the rule.

Once mutation is performed the two resulting offspring are evaluated to replace candidates in the population. This is a critical step for the maintenance of a multi-modal solution set. Uni-modal GA solutions using tournament selection randomly pick two individuals for replacement. These values are replaced if the new children have better fitness values. RTS instead picks from a window size *w*, an individual that is closest to the new child (line 5). The size of the population contained in w is defined by empirical testing and each member is drawn from the original population using a uniformly random selection process. For this project *w* is set at twenty-five. Determination of the best value for *w* was not examined exhaustively and there could be a better value. Closeness is determined by a distance function. After determination of the closest individual to the candidate child a competition is held based on fitness between the child and selected individual. The one with the best fitness is selected for inclusion into the solution population.

### C. *Three Part Fitness Function*

In order to rank the individuals' fitness an evaluation function called the fitness function was defined. The fitness function is a critical component of a GA as it is a primary source for determining an individual's selection for survival and evolution. As was described at the end of the background section, three criteria were identified to judge a 'good' rule. The criteria are described as complete rule match, partial rule match and grammar check. Each criterion is implemented as a function that returns a numeric value. This sum of all three values constitutes the fitness value with a larger value indicating a higher fitness (line 6). Details of each criterion function are presented next.

First a rule should be able to recognize the packet as being an anomaly. This is tested by running Snort with a candidate rule on a test packet and evaluating the result. A system call to the Snort command line was created that sends the output to a comma separated file. This output file is then read for existence of an alert related to the rule. If this exists a value of 10.0 is returned, otherwise the value 0.0 is returned. This relatively large value was chosen to promote the importance of rules that cause an actual alert.

Second the rule is checked for a bad format, incorrect structure and possible performance issues. The Perl based dumbpig grammar checker was incorporated into the project code base to perform this check [9]. A function call with the rule data results in a floating point value between 0.0 and 1.0. The fewer issues the function finds the greater the return value.

The final evaluation is executed even when a rule does not completely match an evaluation packet. As was described in the background section, Snort rule evaluation short circuits processing a given rule definition whenever it finds a non match. In response to this behavior, a Snort rule evaluation process was recreated without this aspect. Instead, the processing was modified to track matches on all possible rule fields. For each field a Boolean value is maintained with 1 indicating match and 0 not. After a complete pass evaluating all fields in a rule, the final value is computed according to:

$$F(x) = \sum_{i=1}^{N} match(x_i) \tag{1}$$

where N is the number of fields to compare and match is the function that returns the Boolean result of the comparison.

The final fitness evaluation of a candidate rule is then computed as a sum of all three criteria. This fitness value is stored with a reference to the rule and only updated as needed. As this is the final computation step, execution resumes at the beginning of the loop and continues as appropriate.

### D. *Output Processing*

The output of the GA described in the previous section is a set of rules along with their respective fitness values. These rules are sorted according to the fitness values with any duplicate rules removed. The resulting top three (highest fitness value) rules are then proposed as possible rule definitions to be distributed. This assumes that the top rules induce Snort to alert on the related packet. It was found in our testing that this occurred in all ten tests cases, with an average 28% of the final rules producing a match

### E. *Complexity Analysis*

Rule generation in the manner described was considered to be an offline process so initially minimal consideration was given to runtime performance. However the complexity of the RTS GA is N x *w* where *w* is the size of the selection window [1]. The complexity analysis of our implementation is complicated by the three part fitness test. Specifically the call to the Snort executable is an unknown quantity. The runtime performance of Snort varies greatly depending upon the nature of the rule set and volume of network traffic. In this project, the observed process run time of just the Snort binary was less than 1 second in all test cases on a desktop DELL with an Intel E5430 CPU and 4 GB's of RAM. The average run time of the test cases was 57 seconds. This reflects the heavy usage of string manipulation procedures. In addition, the project was implemented in PERL with a focus on program correctness and process visibility instead of runtime performance.

## IV. Test Data And Results

This section describes the results of running the system on two test data sets. A subset of the resulting rules that compiled the best fitness (largest) values are presented in addition to information on fitness and algorithm progression.

### A. *ICMP Test Data*

ICMP rules and characteristics were the primary focus of a test data set created to show a proof of concept. Three packet creation tools: Nemesis, packETH and ISIC were utilized to provide two sets of test network data. The first two tools provided customized individual packets designed to trigger specific rules. The third tool, ISIC, was used to create a large set of packets composed of random values to test for false positives. The use of these tools and the resulting test data sets are described in this section.

Nemesis and packETH are network packet crafting and injection tools [10],[11]. Details of an individual ICMP packet can be specified making them well suited for creating and reproducing test scenarios. They are similar in functionality but packETH features a graphical user interface while Nemesis is a command line tool. An example Nemesis Linux command line used to create a packet is shown below.

```
> nemesis -i 7 -s 0 -d 11 -d lo
```

This command will create an ICMP packet with an itype of 7, a sequence number of 0 and an icmp_id value in the header of 11. Subsequently, the packet will be placed on the lo or loopback interface of the machine. With a packet capture tool attached to this interface, the data can then be captured into a static file and reused for later testing by replaying the file.

The individual test packets created using the tools in the manner just described were captured and stored as a PCAP data file. A total of ten different test packets were created to trigger ten different Snort rules. Each rule was chosen for its rule keyword variety and membership in a rule class. The packet specifics and class types are presented in Table 2 in Snort rule format.

Snort was exercised with the ten hand crafted test packets against 146 ICMP rules from the three sources mentioned in the background section. These test packets were crafted to match all conditions of ten specific rules that were categorized into a variety of rule classes. Each individual packet was run against the original set of rules to ensure that a one to one relationship of packet to rule existed. In other words a single packet matched with a single rule. However this was not entirely possible as some of the rule definitions are broad enough to alert on only a few attributes. For instance test packet 8 triggered an alert on a rule that only specified itype 8 and icode 0. This is a generic ping rule and is technically correct. This type of issue aside Snort positively identified 100% of the test packets with no false positives or false negatives.

A second large test set of random valued ICMP packets was created to evaluate the system for false positives. ISIC is a generic utility used to test the stability of an IP stack [12]. It is capable of creating a large number of test packets containing random values. The random values are correct in that they fall within a field's acceptable data range. However, the value may not be currently in use or match a prerequisite implied by a setting in another field. These packets are then typically sent to a target while observing for any anomalous results. For this project 23 megabytes of data containing 33,794 ICMP packets were created.

In the same manner as was described in the hand crafted packet test creation, the random test set was run against Snort and the original rules. This resulted in a total of 31,929 alerts. Eighty-eight percent (28,293) of the alerts were triggered by a single rule designed to find undefined codes.

TABLE 2: ICMP PACKET DETAILS.

| | Packet Details | Class Type |
|---|---|---|
| 0 | icmp_id:667;itype:0;content:"ficken" | attempted-dos |
| 1 | same as #1 except random IP's and IP identification field. | attempted-dos |
| 2 | dsize:0;itype:8;icode:0 | attempted-recon |
| 3 | icode:0;itype:5 | bad-unknown |
| 4 | icode:2;itype:3; | misc-activity |
| 5 | icode:2;itype:3; content:"\|28 00 00 50 00 00 00 00 F9 57 1F 30 00 00 00 00 00 00 00 00 00 00 00 00\|" | attempted-user |
| 6 | icode:0;itype:8;dsize:20; content:"abcde12345fghij6789"; | trojan-activity |
| 7 | itype:8;icode:0;dsize:32;content:"abcdefgh ijklmnopqr\|0000\|";depth:22; | trojan-activity |
| 8 | icmp_id:123;icmp_seq:0;itype:0; content:"shell bound to port"; | attempted-dos |
| 9 | icode:0; itype:40; | misc-activity |

The remaining 3,636 alerts were produced by 51 unique rules. Of this set, 6 alerts were generated from 4 rules that were used to craft matches in the first test set. These were carefully noted for the evaluation phase as genuine alerts. They should be ignored when testing for false positives as indeed they are not.

After generating the two data sets, the original ten rule definitions used for the hand crafted packets were saved. These rules then became one basis for the final evaluation of the evolved rules correctness. It was not expected that the created rules exactly match the originals but they should be similar in content and behavior.

## B. *Test Results*

In order to show the progression of the algorithms search for a set of optimal rules, Table 3 provides average fitness information, largest final rule fitness and the number of times a child was used to replace a member of the population. The columns are labeled with a test packet identifier of P0-P9. It can be seen in all test cases that the average final fitness values are higher than the initial average fitness. The fitness value of the final best solution (rule) in each case is approximately two or three times that of the final average. The child creation step did produce an individual, on average, every five iterations that improved fitness as indicated by the replacement count.

Fig. 2 shows the top three fittest individuals for test case P0. For clarity the original rule definition used to create the test packet is included as the last line and is labeled as src. For formatting reasons the action, protocol, sid, classtype and rev number have been removed. These values are metadata information and do not contribute to Snorts recognition engine execution. As can be seen there is a variety in the rule header composure. The specific IP addresses were retrieved from the test packet headers. Retaining these in a production rule may not add value but this depends on knowledge not used as input into this system. A rule option commonality can be observed in the inclusion of similar content values and itype fields. The first rule contains an icode field that does not appear in the original rule. As this field was not identified in the original

TABLE 3: FITNESS AND RUN DATA

| | Initial Avg Fitness | Final Avg Fitness | Best Fitness | Replace |
|---|---|---|---|---|
| P0 | 5.05 | 7.95 | 31 | 145 |
| P1 | 5.23 | 10.49 | 31 | 207 |
| P2 | 2.5 | 8.92 | 22 | 185 |
| P3 | 2.63 | 9.06 | 22 | 191 |
| P4 | 2.6 | 10.35 | 22 | 224 |
| P5 | 4.6 | 10.24 | 30.6 | 209 |
| P6 | 5.14 | 11.5 | 30.6 | 258 |
| P7 | 5.22 | 11.86 | 30.72 | 270 |
| P8 | 4.97 | 9.12 | 26.47 | 180 |
| P9 | 2.46 | 9.54 | 22 | 218 |

```
235.130.217.126/32 any -> any any
          (itype:0; content:"ficken|0A|"; icode:0;)
$SNET any -> 140.53.42.24/32 any
          (itype:0; content:"cken";)
$SNET any -> any any
          (content:"fick"; dsize:7;)
(src) $HOME_NET any -> $EXTERNAL_NET any
          (icmp_id:667; itype:0; content:"ficken";)
```

Figure 2. Packet 0 Generated Rules

rule a default value was supplied when creating the packet. This value was manually verified as being correct for the test packet. Because of space considerations the other top three rules generated from each of the remaining nine cases are not shown.

A simple test of rule correctness involved running the ten test anomaly packets against the corresponding set of top three generated rules. All of the packets were recognized and generated the appropriate Snort alerts for a 100% positive identification rate for each of the top three rules for the 10 tests. This was expected based on the composition of the fitness evaluation function. It should be noted that all three of the rules contain valid identifying fields. A post processing step could involve any combination of the defined fields to create a valid rule.

A possibly more interesting test concerns the occurrence of false positives. It has been observed that a simple rule definition could contain just one field that matches a large number of packets. This would certainly produce a large positive identification rate. A set of random ICMP test packets was created to test for this scenario. In addition to the random packets, the ten hand crafted packets were used since they were readily available. For a given rule generated by our algorithm, only one of the crafted packets should generate an alert.

As can be seen in Fig. 3 the false positive rate was very low. The vertical axis indicates the number of false positives for a given test rule set. A total of four false positives were generated from the set of 30 generated rules over the more than 33,000 test packets. Three false positives came from the hand crafted packets. None of the false positives were generated from the single top fittest rule for each test case.

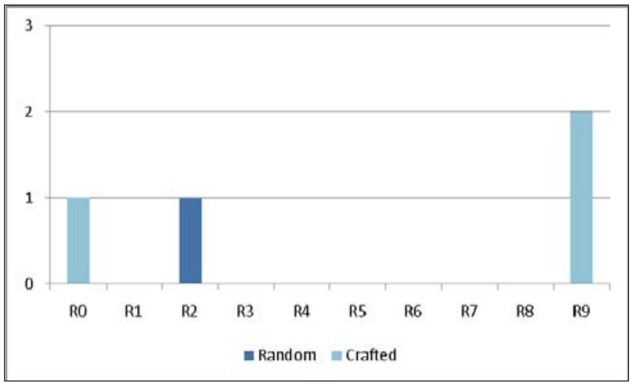

Figure 3. False Positive Chart

## V. Related Work

This section reviews the work from four different papers that relate to autonomous rule creation. The first two discuss the use of GA's and their applicability to network intrusion detection. The third paper is relevant in that Snort rules are created for a specific malware signature. The final paper reviews the capability of a multi-modal GA to find and maintain multiple optimal solutions.

Goyal and Kumar [13] were concerned with identification of malicious computer network connections. The scope of their experiment focused on generating rules for six attack types belonging to two different classes: Denial of Service and Probes. The labeled KDD 99 Cup data set was used as training and testing input data for a simple genetic algorithm. Eight fields from the data set were used in the encoding of the population: protocol, type of service, flag (error or normal connection), duration of bytes sent, duration of the connection, percentage of connections to different hosts, number of operations on access control files and number of outbound commands in an ftp control session. The rules generated produced from the trained GA were able to correctly identify the test data with ninety-two percent accuracy.

GAs and decision trees have also been used to produce rules classifying network connections [14]. The source and destination IP address, port numbers and network protocol type from a database were used as input to evolve a solution in a crowding form of GA. The initial random population was created with chromosomes representing these input values. A training set of connections was marked by a human expert as normal or anomalous. Subsequently, the GA was executed utilizing a fitness function that compared the evolved rule to this training set. Partial or complete matches for an anomaly were rewarded and normal connection matches were penalized. Consequently the resulting rule set was biased towards anomaly recognition rules. No empirical testing of these rules sets was reported.

Wang, Jha and Ganapathy describe a tool called NetSpy that automatically generates network signatures for specific instances of spyware [15]. The spyware signature generation section most closely resembles the work presented in this paper. Network traffic is identified as belonging to a particular piece of malware. A modified version of the Longest Common Subsequence algorithm is used to produce regular expressions. The expressions are then added to a Snort rule as a payload detection option. The most significant difference in their approach to the one presented in this paper is the rule generation algorithm.

Sing and Deb compared the performance of eight multi-modal optimization algorithms based on evolutionary algorithms [16]. Multi-modal problems involve finding more than one optimal solution for a given task. Modifications to simple GAs such as crowding, RTS, clearing, fitness sharing and species conserving were presented. These methods using real encoded populations were tested against three mathematical functions. The computational complexity ranged from O(N) to $O(N^2)$. Among the approaches examined RTS, deterministic crowding, original clearing and a proposed modified clearing method found more optima.

## VI. Conclusion

The primary goal of devising a solution to decrease human effort in creating Snort rules based on anomalous network traffic was accomplished. All of the ten hand crafted test packets resulted in a set of rules that caused Snort to alert. Not all of the rules were unique, but in each case there were at least three unique rules and as many as eight. Testing showed that these rules were specific to the packets and produced only four false positives from 33,804 test packets. These successful results indicate that the generated rules can be used by analysts as the basis for production rules.

A key enabling technology was the use of a multi-modal GA. A critical condition in the performance of a GA's search capability is the ability of the fitness function to accurately indicate progress in exploration of the solution set. For this project a three part fitness function was developed. It proved to be sufficient in aiding generation of rules that caused alerts on test cases. Further refinement of this function and the addition of more rule options may increase the capability of the system to define robust rules. In addition defining the capability to include session information instead of single packet rules could be explored. Finally expanding the domain of the test packets to UDP and TCP would provide a broader coverage of the anomalous network possibilities.

## References

[1] P. Gogoi, B Borah and D.K. Bhattacharyya, "Anomaly Detection Analysis of Intrusion Data using Supervised & Unsupervised Approach", Jour. Of Convergence Information Technology, Vol 5, No. 1, Feb. 2010

[2] T. Vollmer, M. Manic, "Computationally Efficient Neural Network Intrusion Security Awareness", In Proc. of the 2nd IEEE Symposium on Resilience Control Systems, Idaho Falls, Idaho, Aug. 11-13, 2009.

[3] O. Linda, T. Vollmer and M. Manic, "Neural Network Based Intrusion Detection For Critical Infrastructures", In Proc. of IJCNN09, June 2009.

[4] C. Taylor and J. Alves-Foss, "An empirical analysis of NATE: Network Analysis of Anomalous Traffic Events", In Proc. of the 2002 workshop on New security paradigms, 2002, pp 18-26.

[5] F. Gonzalez, D. Dasgupta,"An Immunogenetic Approach to Intrusion Detection", Genetic and Evolutionary Computation Conference (GECCO), New York, New York: July, 2002.

[6] G.R. Harik, "Finding multimodal solutions using restricted tournament selection", In Proc. of the 6th International Conf. on Genetic Algorithms, San Francisco, CA, USA, pp. 24-31, 1995.

[7] M. Roesch. "Snort – lightweight intrusion detection for networks", In Proc. of USENIX LISA '99, Nov. 1999.

[8] M. Roesch, "Writing Snort Rules: How To write Snort rules and keep your sanity", http://www.snort.org.

[9] L. Ward, dumbpig, online: http://leonward.wordpress.com/dumbpig.

[10] J. Nathan, Nemesis, http://nemesis.sourceforge.net.

[11] M. Jemec, packETH – Ethernet packet generator, http://packeth.sourceforge.net/.

[12] S. Xiao, ISIC – IP Stack Integrity Checker, http://isic.sourceforge.net/.

[13] A. Goyal and C. Kumar, "GA-NIDS: A Genetic Algorithm based Network Intrusion Detection System", not published, Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, 2007.

[14] C. Sinclair, L. Pierce and S. Matzner, "An Application of Machine Learning to Network Intrusion Detection", In Proc. of the 15th Annual Computer Security Applications Conference, 1999.

[15] H. Wang, S. Jha and V. Ganapathy, "NetSpy: Automatic Generation of Spyware Signatures for NIDS",in Proc. Of 22nd Ann. Comp. Security Applications Conf., 2006

[16] G. Singh and K. Deb, "Comparison of Multi-Modal Optimization Algorithms Based on Evolutionary Algorithms", In Proc. Of GECCO'06, July 2006.