IMPLEMENTATION OF GENETIC ALGORITHMS INTO A NETWORK INTRUSION DETECTION SYSTEM (netGA), AND INTEGRATION INTO nProbe

Brian Eugene Lavender
B.S., California Polytechnic State University, San Luis Obispo, 1993

PROJECT

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

at

CALIFORNIA STATE UNIVERSITY, SACRAMENTO

FALL
2010

IMPLEMENTATION OF GENETIC ALGORITHMS INTO A NETWORK INTRUSION DETECTION SYSTEM (netGA), AND INTEGRATION INTO nProbe

A Project

by

Brian Eugene Lavender

__________________________________, Committee Chair
V. Scott Gordon, Ph.D.

__________________________________, Second Reader
Isaac Ghansah, Ph.D.

____
Date

Student: <u>Brian Eugene Lavender</u>

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the Project.

________________________________, Graduate Coordinator ______________________
Nikrouz Faroughi, Ph.D. Date

Department of Computer Science

Abstract

of

IMPLEMENTATION OF GENETIC ALGORITHMS INTO A NETWORK INTRUSION DETECTION SYSTEM (netGA), AND INTEGRATION INTO nProbe

by

Brian Eugene Lavender

netGA takes networking theory and artificial intelligence theory and combines them together to form an attack detection system. netGA is an implementation of the method proposed by the paper titled *A Software Implementation of a Genetic Algorithm Based Approach to Network Intrusion Detection* written by Ren Hui Gong and associates. It also includes an implementation of the resulting rules into a Network Intrusion Detection System (NIDS) called nProbe. The project brings together Genetic Algorithms from soft computing methods, also known as Artificial Intelligence, and a Network Intrusion Detection System (NIDS). In order to limit the project scope, data developed by DARPA, also used in Gong's paper, is used as training data for the Genetic Algorithms. The resulting tool is described and analyzed, and results and sample runs are presented.

_______________________________, Committee Chair
V. Scott Gordon, Ph.D.

____
Date

ACKNOWLEDGMENTS

I would like to thank the free software community for their commitment to make software that others might find useful. I would like to thank my mother for her moral support and my father for giving me curiosity and opening my eyes to exploration and learning new things.

TABLE OF CONTENTS

Page

Chapter

LIST OF FIGURES

Chapter 1

MOTIVATION[1]

My interest in this project started while I was working as a graduate student assistant at the Legislative Data Center. I was working with a system called OSSIM [OSSIM], a tool that aggregates output from various security tools, one being SNORT, with the objective of better determining whether a server had been attacked. To really understand OSSIM one needs to understand the tools that support it. I found a HOWTO for installing SNORT [HARPER]. I followed the HOWTO and everything seemed to go well, so I wanted to see if it worked. In order to test my new SNORT attack alerting tool I had to find a vulnerable server and an attack that would exploit it.

I had previously worked with an FTP server called WuFTP [WUFTP]. I recalled from about five years before that an alert came out on the security sites [SECURITYFOCUS] that a security analyst discovered WuFTP was vulnerable to a serious exploit. An attacker could send a carefully crafted packet to the WuFTP server and instantly the attacker could gain root level access (full control) to the target server. It was a serious. We had to quickly patch our WuFTP installation on the server where it ran. I quickly compiled a new version of WuFTP, and installed it. To our knowledge, no one discovered the server and exploited it, but we never actually knew. All we knew was that we installed the new patched version and that we hadn't noticed unusual activity, so we assumed that we had fixed it before the attackers had discovered it.

---

[1] I have taken the liberty of writing the first section in the first person. The remaining sections are written in the third person.

Here I was with my brand new attack detection tool, SNORT, capable of detecting this attack. Question was, would it work? So, I installed the old version of the vulnerable WuFTP server and on a second computer, I was ready with my attack, just like the crafty attacker searching the internet for vulnerable systems. The attack was available for download on the internet. I launched the attack from my remote computer, and as I had hoped, the SNORT tool detected my attack and alerted on it. SNORT identified the attack by matching the network traffic against the rule specifically written against my vulnerable installation of WuFTP. At the same time, my attack worked and I was able to gain root access (full control), but now I had a tool that detected it. This gave me great satisfaction. I had discovered a tool that could monitor an application by monitoring network traffic targeted towards it. Yet, SNORT used a specific rule created by an expert familiar with the WuFTP application and networking. Was there a way to automatically create these rules?

I explored SNORT further and I discovered SPADE had been written to statistically analyze traffic and alert on anomalies using Bayes Theorem. What I found so intriguing was that no one would have to write a specialized rule to identify the attack with SPADE. The SPADE tool would follow traffic, and when it detected anomalous traffic, it would alert on it. Step forward to my Artificial Intelligence class with Dr. Gordon where we explored different techniques to solve problems using techniques such as Artificial Neural Networks, Swarm Theory, Genetic Algorithms, and more. My curiosity led me to question whether we could adopt these same techniques to security and identification of attacks like SPADE had done.

Chapter 2

BACKGROUND

The following details the tools, techniques and theory coming from both the network and security side to build netGA.

2.1 SNORT

SNORT [SNORT] has become a popular Network Intrusion Detection System(NIDS). A search on the Google search engine [GOOGLE] for term “snort” results in a set that exceeds 1,000,000. Its main focus is a rule based detection system for identifying malicious traffic.

SNORT started as the pet project by Marty Roesch in November of 1998. Originally, he created it to examine network traffic on his cable modem. Later, he began to develop rules for identifying different types of traffic and alerting on them. Today, Sourcefire maintains the free software version of SNORT and distributes rule sets to registered users. There have been other efforts to create rule sets such as the SNORT bleeding rules. Below is an example snort rule taken from the chat rules found in current SNORT rule snapshot (snortrules-snapshot-2.8.tar.gz [SNORTrules]).

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-IIS
CodeRed v2 root.exe access"; flow:to_server, established;
uricontent:"/root.exe"; nocase; reference:url,
www.cert.org/advisories/CA-
2001-19.html; classtype:web-application-attack; sid:1256; rev:8;)
```

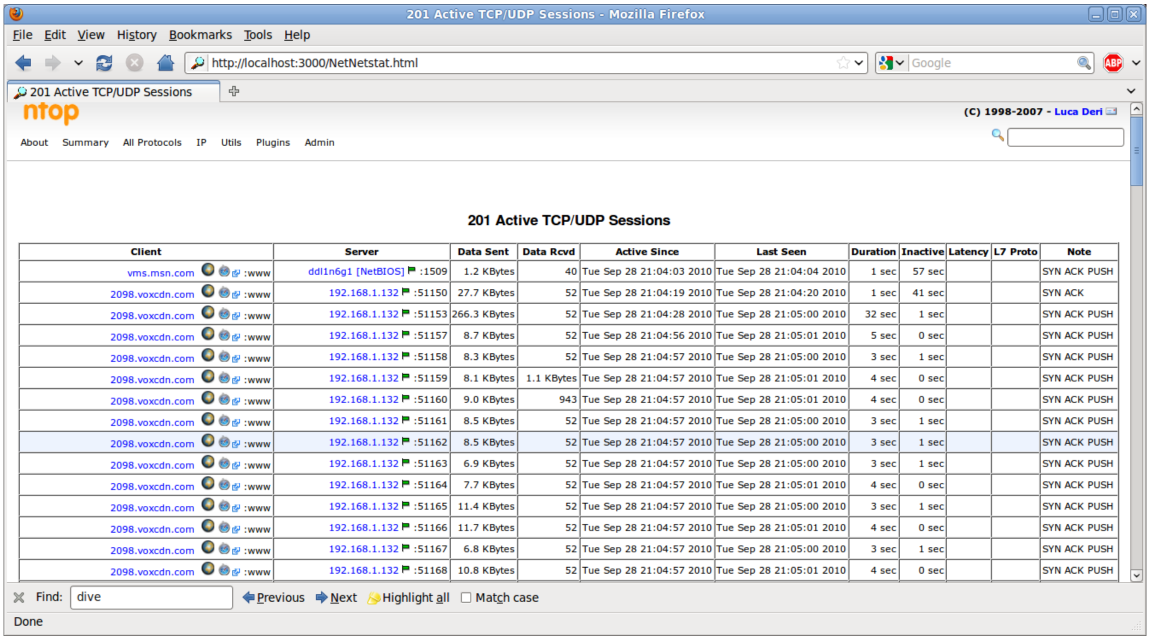
Figure 1: Sample SNORT Rule

The rule identifies the notorious CodeRed worm [Kohlenberg] that wrecked havoc on the internet in 2001. In order to develop this rule, an administrator trained in the SNORT rule syntax had to determine what traffic is not desirable, examine it for identifiable attributes, and then create the rule.

Beyond writing specific rules, SNORT has supported a modularized architecture allowing developers to write customized plug-ins for it. SPADE utilized this plug-in architecture for integrating its plug-in into SNORT (version 2.7.0). Unfortunately, at the time of this writing, SNORT (version 2.8.3) no longer maintains compatibility with the SPADE plug-in and during the course of this project the architecture of SNORT was in question and potentially slated for a complete rewrite.

2.2 NTOP and nProbe

NTOP [NTOP] is another popular network monitoring system. A search for “ntop” on Google generates over one million search result “hits”. It's original focus is not alerting on attacks, yet be able to present the state of network connections and corresponding statistics. It monitors the state of “Active TCP/UDP Sessions” (Figure 2) which plays a key role in the development of the netGA system. The name derives from the UNIX utility called “top” that shows statistics of running processes. Luca Deri and Stefano Suin developed NTOP along with contributions by other developers. NTOP has a series of web based graphical tools for viewing these “Active TCP/UDP Sessions”.



201 Active TCP/UDP Sessions

Client	Server	Data Sent	Data Rcvd	Active Since	Last Seen	Duration	Inactive	Latency	L7 Proto	Note
vms.msn.com :www	ddl1n6g1 [NetBIOS] :1509	1.2 KBytes	40	Tue Sep 28 21:04:03 2010	Tue Sep 28 21:04:04 2010	1 sec	57 sec			SYN ACK PUSH
2098.voxcdn.com :www	192.168.1.132 :51150	27.7 KBytes	52	Tue Sep 28 21:04:19 2010	Tue Sep 28 21:04:20 2010	1 sec	41 sec			SYN ACK
2098.voxcdn.com :www	192.168.1.132 :51153	266.3 KBytes	52	Tue Sep 28 21:04:28 2010	Tue Sep 28 21:05:00 2010	32 sec	1 sec			SYN ACK PUSH
2098.voxcdn.com :www	192.168.1.132 :51157	8.7 KBytes	52	Tue Sep 28 21:04:56 2010	Tue Sep 28 21:05:01 2010	5 sec	0 sec			SYN ACK PUSH
2098.voxcdn.com :www	192.168.1.132 :51158	8.3 KBytes	52	Tue Sep 28 21:04:57 2010	Tue Sep 28 21:05:00 2010	3 sec	1 sec			SYN ACK PUSH
2098.voxcdn.com :www	192.168.1.132 :51159	8.1 KBytes	1.1 KBytes	Tue Sep 28 21:04:57 2010	Tue Sep 28 21:05:01 2010	4 sec	0 sec			SYN ACK PUSH
2098.voxcdn.com :www	192.168.1.132 :51160	9.0 KBytes	943	Tue Sep 28 21:04:57 2010	Tue Sep 28 21:05:01 2010	4 sec	0 sec			SYN ACK PUSH
2098.voxcdn.com :www	192.168.1.132 :51161	8.5 KBytes	52	Tue Sep 28 21:04:57 2010	Tue Sep 28 21:05:00 2010	3 sec	1 sec			SYN ACK PUSH
2098.voxcdn.com :www	192.168.1.132 :51162	8.5 KBytes	52	Tue Sep 28 21:04:57 2010	Tue Sep 28 21:05:00 2010	3 sec	1 sec			SYN ACK PUSH
2098.voxcdn.com :www	192.168.1.132 :51163	6.9 KBytes	52	Tue Sep 28 21:04:57 2010	Tue Sep 28 21:05:00 2010	3 sec	1 sec			SYN ACK PUSH
2098.voxcdn.com :www	192.168.1.132 :51164	7.7 KBytes	52	Tue Sep 28 21:04:57 2010	Tue Sep 28 21:05:01 2010	4 sec	0 sec			SYN ACK PUSH
2098.voxcdn.com :www	192.168.1.132 :51165	11.4 KBytes	52	Tue Sep 28 21:04:57 2010	Tue Sep 28 21:05:00 2010	3 sec	1 sec			SYN ACK PUSH
2098.voxcdn.com :www	192.168.1.132 :51166	11.7 KBytes	52	Tue Sep 28 21:04:57 2010	Tue Sep 28 21:05:01 2010	4 sec	0 sec			SYN ACK PUSH
2098.voxcdn.com :www	192.168.1.132 :51167	6.8 KBytes	52	Tue Sep 28 21:04:57 2010	Tue Sep 28 21:05:00 2010	3 sec	1 sec			SYN ACK PUSH
2098.voxcdn.com :www	192.168.1.132 :51168	10.8 KBytes	52	Tue Sep 28 21:04:57 2010	Tue Sep 28 21:05:01 2010	4 sec	0 sec			SYN ACK PUSH

Figure 2: NTOP Connection Tracking Screen

Since network monitoring can occur at various points in the network, NTOP has a sister tool called nProbe that monitors traffic and sends data to NTOP, performing a sub function of NTOP and sending this data to a centralized NTOP process to perform aggregation of statistics of all the reporting probes. nProbe has a plug-in architecture allowing users to write plug-ins tapping into nProbe TCP tracking capability and providing additional functionality. The structure of the plug-in architecture is easy to follow and Luca Deri supported the development of netGA plug-in for nProbe. netGA uses the plug-in architecture provided by nProbe for integration of rules created by the Genetic Algorithm (GA).

2.3 Motivation for Artificial Intelligence and Network Intrusion Detection Integration

The primary focus of SNORT hasn't been on Artificial Intelligence methods, but has focused on developing explicit rules by a team of experts. At the same time, various researchers have performed studies using soft based computing for Network Intrusion Detection including Fuzzy Logic, Artificial Neural Networks (ANN), Probabilistic Reasoning, and Genetic Algorithms [Farshchi]. James Hoagland wrote Statistical Packet Anomaly Detection Engine SPADE [Farshchi] taking advantage of the plug-in type architecture of SNORT. It monitors traffic and maintains a statistical probability table for IP addresses and port destinations. When a packet arrives, SPADE calculates an anomaly score for the packet. Anomalous traffic generally occurs with an attack or malicious traffic. SPADE operates regardless of the rule set and uses probabilistic analysis to do its job.

Farshchi [Farshchi], in his analysis of SPADE, notes that while rule based analysis used by SNORT provides reliable results for detecting malicious traffic, it has two downsides. One, being that maintaining the rule sets can be a burden to the security professional. Two, rule based methods have no way of identifying new attacks for which no rule is available. In addition, he points to other Artificial Intelligence techniques such as Artificial Immune System, Control Loop Measurement, and Data Mining as effective methods for identifying malicious traffic. SPADE supports the idea that other Artificial Intelligence techniques can be incorporated into SNORT.

2.4 Genetic Algorithms

Genetic Algorithms is an optimization technique using an evolutionary process. A solution to a problem is represented as a data structure known as chromosome. The “goodness” of a solution is evaluated by an algorithm called a fitness function. A series of initial solutions is initially generated (random population) and through a combination of algorithms similar to an evolutionary process (often a combination of elitism, crossover, and mutation) the process works towards evolving solutions having better “goodness” as evaluated by the fitness function. The book *Artificial Intelligence, A Modern Approach[Norvig]* offers a detailed explanation of Genetic Algorithms. Genetic Algorithms follow the process listed below, which can also be seen in Figure 3 [Pohlheim]:

1. Initialize population
2. Calculate fitness of population.
3. Perform selection. Roulette wheel is technique that randomly selects chromosomes giving proportional weight to chromosomes with higher fitness.
4. Perform crossover
5. Perform mutation
6. If stopping criteria not met, go back to step 2.
7. Quit

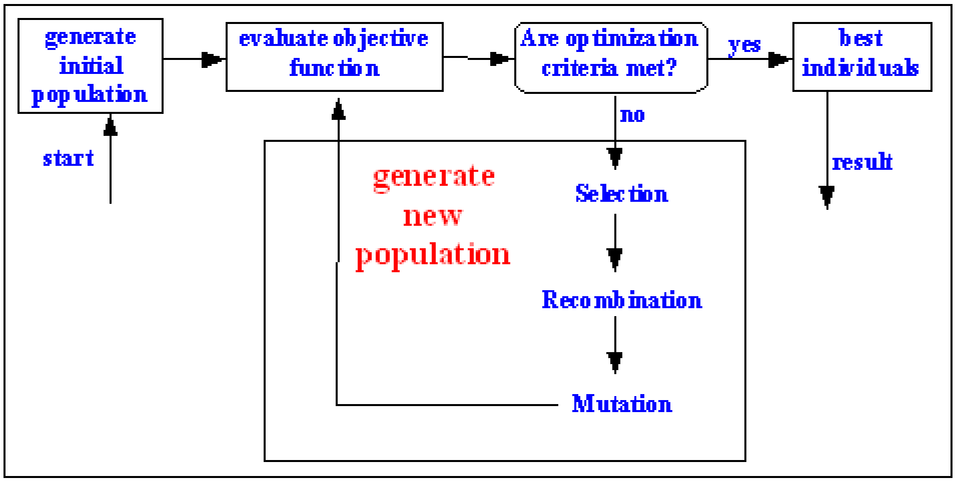


Figure 3: Structure of a Simple Genetic Algorithm (Pohlheim)

The basic concepts of Genetic Algorithms are simple, yet the process of choosing the gene representation, a good fitness function, and even application of the recombination [Whitley] can be the key to successful use of Genetic Algorithms.

2.5 Previous Genetic Research for Network Intrusion Detection

Wei Li [Li] wrote a proposal for using GA in a NIDS and Ren Hui Gong [Gong] followed with his implementation. Li set the foundation for creating a system using Genetic Algorithms that analyzes DARPA data sets, and Gong created a proposed implementation using ECJ [ECLab] (A Java-based Evolutionary Computation Research System). Gong provided pseudo code and class diagrams (one familiar with the ECJ library could probably implement the algorithm). Li proposed using DARPA data sets [DARPA] from MIT Lincoln Laboratory for training and testing.

In both Li's proposal and Gong's approach they create a fitness function and a chromosome type for the Genetic Algorithms.

2.6 DARPA Data Sets

A key dependency of the work done by Gong and Li and as will be shown with netGA is the usage of DARPA data sets for training data. Creating this training data is not a trivial task and is considered beyond the scope of this project. The MIT Lincoln laboratory provides an excellent description of the process followed for creating the data. This DARPA training data is actually a result of test network traffic data, a Sun Microsystems Solaris and the use of Sun's Basic Security Module[Sun]. The data sets used in both papers were created in 1998. Today's attacks have changed with regard to rule based systems, but the training data still works well for developing Genetic Algorithms.

There are two important pieces of data that are used in netGA. First, is the data contained in the file called *bsm.list* . The following snippet (Figure 4) identifies two normal connections and two attack connections (rcp and guess). This file has a list records each containing the following attributes: Connection Number, Starting Date, Starting Time, Duration, protocol, Source Port, Destination Port, Source IP Address, Destination IP Address, a zero or one field, and attack name (or a dash if it was a normal connection).

Normal Connection

```
118 01/23/1998 17:00:13 00:00:11 ftp 1892 21 192.168.1.30 192.168.0.20 0 -
```

Normal Connection

```
122 01/23/1998 17:00:31 00:00:00 smtp 1900 25 192.168.1.30 192.168.0.20 0 -
```

rcp Attack Connection

```
125 01/23/1998 17:00:38 00:00:02 rsh 1023 1021 192.168.1.30 192.168.0.20 1 rcp
```

guess Attack Connection

```
126 01/23/1998 17:00:39 00:00:23 telnet 1906 23 192.168.1.30 192.168.0.20 1
guess
```

Figure 4: Sample DARPA Audit Data

The second is a network capture file named *sample_data01.tcpdump.* It contains the network data recording that generated the attacks. Thus, it will be used in the evaluation of the effectiveness of the rules created by the Genetic Algorithms.

2.7 The netGA Objective

netGA uses a series of Genetic Algorithm runs for generating rules for use in identifying attacks in a Network Intrusion Detection System using the DARPA set as training data. It closely follows the approach proposed by Gong and uses his same chromosome representation in the Genetic Algorithms. It also entails the development of a plug-in for nProbe. The plug-in loads the evolved rules from the Genetic Algorithm runs and matches them against traffic it listens to through a network wire tap. A corresponding network capture file, *sample_data01.tcpdump,* works as a playback to nProbe.

Chapter 3

netGA SYSTEM

netGA involves the use of Genetic Algorithms to generate rules to identify attacks and then the integration of the rules into nProbe for detection of network traffic. The following two subsections present the details for each.

3.1 Genetic Algorithm

The way that Genetic Algorithms are used with netGA is that rules are randomly created to match attacks encoded as a integer array with the seven elements shown in Figure 5. The first six attributes of the chromosome match the gene characteristics of an attack. The seventh attribute describes the attack type that the first six rules identify when they match. This representation uses the same approach as used by Gong.

	Feature Name	Format	Number of Genes
1	Duration	h:m:s	3
2	Protocol	Int	1
3	Source_port	Int	1
4	Destination_port	Int	1
5	Source_IP	a.b.c.d	4
6	Destination_IP	a.b.c.d	4
7	Attack_name	Int	1

Figure 5: Chromosome Representation for Rule

In order to evaluate a rule represented by a chromosome, the DARPA audit data is parsed and loaded into a list of audit connections (Figure 6). The sample data has five attack connections and five normal connections. The attributes loaded from the DARPA audit data directly match the attributes used in the chromosome representation.

	Duration				Protocol	SRC PORT	DST PRT	SRC IP				DST IP				Attack Type
		H	M	S				0	1	2	3	0	1	2	3	
1		0	0	11	ftp	1892	21	192	168	1	30	192	168	0	20	-
2		0	0	0	smtp	1900	25	192	168	1	30	192	168	0	20	-
3		0	0	2	rsh	1023	1021	192	168	1	30	192	168	0	20	rcp
4		0	0	23	telnet	1906	23	192	168	1	30	192	168	0	20	guess
5		0	0	14	rlogin	1022	513	192	168	1	30	192	168	0	20	rlogin
6		0	0	2	rsh	1022	1021	192	168	1	30	192	168	0	20	rsh
7		0	0	15	ftp	43549	21	192	168	0	40	192	168	0	20	-
8		0	0	40	telnet	1914	23	192	168	1	30	192	168	0	20	guess
9		0	1	24	telnet	43560	23	192	168	0	40	192	168	0	20	-
10		0	0	13	ftp	43566	21	192	168	0	40	192	168	0	20	-

Figure 6: DARPA Audit Data

The gene representation follows the simple rule *if A then B,* where if the first six attributes are logically and-ed together are true(A), then the rule matches the attack (B). Figure 7 illustrates the same representation of the chromosome in a horizontal layout for the rule. Rules can have wild card values in each of the fields. The sample chromosome representing a rule in Figure 7 has wild cards for the Hour, the source port, and the third octet of the source IP address. The attack type this rule identifies is an rsh attack. One can see from this this table that the three genes for duration sit in the first integer portion of the array index 0. The attributes for source IP (array index 4) and destination IP (array index 5) addresses also divide the integer into four sub portions for the gene representation. The netGA program uses a union to address these subsection areas while still utilizing a 32 bit integer portion of space for storage.

Figure 7 also illustrates index values at “index points” in the chromosome representation. There are a total of seventeen index points through chromosome representation. Crossover and mutation operations use these index points for their operations (shown later).

	Duration				Protocol	SRC PORT	DST PORT	SRC IP				DST IP				Attack Type	
		H	M	S				0	1	2	3	0	1	2	3		
		-1	0	3	rsh	-1	1021	192	168	-1	-1	192	168	0	20	rsh	
Arry Idx	0				1	2	3	4				5				6	
Cross Idx	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Figure 7: Chromosome Layout and Index Points

The fitness is evaluated by determining how many attack connections the rule matches (Figure 8).

$$support = |A \text{ and } B| \,/\, N$$

$$confidence = |A \text{ and } B| \,/\, |A|$$

$$fitness = w1 * support + w2 * confidence$$

Figure 8: Fitness Calculation

N represents the total number of connections. $|A|$ represents the number of connections where the rule matches the portion of connections matching the first six attributes (Figure 5). $|A \text{ and } B|$ represents the number of connections that rule matches in the audit data that matches the *if A then B* rule. w*1 and w2 weighting* parameters can be adjusted to fine tune the algorithm.

Gong described it as follows:

> "One of the nice properties of using this fitness function is that, by changing the weights w1 and w2, the approach can be used for either simply identifying network intrusions or precisely classifying the types of intrusions." [Gong]

netGA uses a the following weights: *w1 = 0.8, w2 = 0.2.*

	Duration				Protocol	SRC PORT	DST PRT	SRC IP				DST IP				Attack Type
		H	M	S				0	1	2	3	0	1	2	3	
1		0	0	11	ftp	1892	21	192	168	1	30	192	168	0	20	-
2		0	0	0	smtp	1900	25	192	168	1	30	192	168	0	20	-
3		0	0	2	rsh	1023	1021	192	168	1	30	192	168	0	20	rcp
4		0	0	23	telnet	1906	23	192	168	1	30	192	168	0	20	guess
5		0	0	14	rlogin	1022	513	192	168	1	30	192	168	0	20	rlogin
6		0	0	2	rsh	1022	1021	192	168	1	30	192	168	0	20	rsh
7		0	0	15	ftp	43549	21	192	168	0	40	192	168	0	20	-
8		0	0	40	telnet	1914	23	192	168	1	30	192	168	0	20	guess
9		0	1	24	telnet	43560	23	192	168	0	40	192	168	0	20	-
10		0	0	13	ftp	43566	21	192	168	0	40	192	168	0	20	-
	Chromosome for Individual (-1 is wildcard)															
		-1	**0**	**-1**	**rsh**	**-1**	**1021**	**192**	**168**	**-1**	**-1**	**192**	**168**	**0**	**-1**	**rsh**

Figure 9: Audit Data and Rule

Figure 9 shows a sample chromosome representing a rule that identifies an attack, and above the chromosome is the list of audit data. The matched connections are highlighted. The chromosome matches the first six attributes in lines 3 and 6. It matches the attack type, rsh, only on line 6. The fitness for this chromosome representing this rule is 0.42, illustrated in Figure 10. The fitness function is a key component to genetic algorithms. As can be seen in the Figure 9 example, rules that identify attacks in the audit data such as shown in the above example have higher fitness.

```
N = 10 connections.
|A| = 2
|A and B| = 1
w1 = 0.2
w2 = 0.8
fitness = w1 * support + w2 * confidence
support = | A and B | / N = 1 / 10 = 0.1
confidence = | A and B | / A = 1 /2 = 0.5
fitness = 0.2 * 0.1 + 0.5 * 0.8 = 0.42
```

Figure 10: Sample Fitness Calculation

The Genetic Algorithm process starts with the generation of 400 random rules, calculates the fitness of these random rules, and then goes through an evolution process (Figure 11). Most of the rules in the initial random set have a fitness of zero.

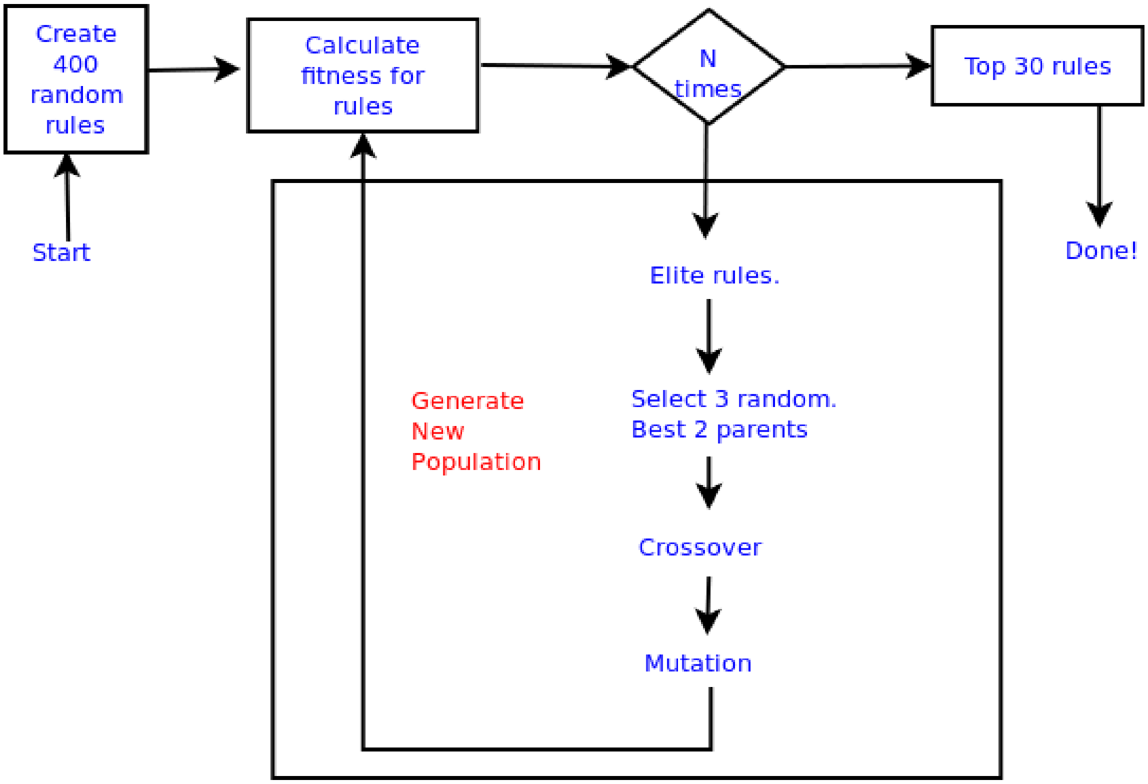


Figure 11: Genetic Algorithms Flowchart

Before generating random rules, the unique values in each field are identified. For example, the Source Port field of the sample DARPA audit data (Figure 6) contains the following five unique values out of the list ten audit connections as listed below:

```
21
25
1021
23
513
```

Thus, any chromosome for a successful rule will contain either one of these values or a wild card value of negative one. The netGA program allows the programmer to adjust the probability for the wild card value. So, if the programmer decides the wild card for this field should be 0.1, the remaining probability, 0.9, will be divided between the remaining unique values. In the above case, each unique value will have a 0.9/5.0 or 0.18 probability of being chosen for randomly generated individuals. netGA starts by generating a group of individuals. Figure 11 indicates 400 random individuals will be generated, but any even numbered group of individuals can be used in the population. Figure 12 shows four randomly generated individuals.

	Duration				Protocol	SRC PORT	DST PRT	SRC IP				DST IP				Attack Type
		H	M	S				0	1	2	3	0	1	2	3	
1		0	0	2	telnet	1900	23	192	-1	1	30	192	-1	0	20	guess
2		0	0	0	-1	1022	21	192	168	1	-1	192	168	0	20	rcp
3		0	1	15	rsh	43549	-1	-1	168	1	30	-1	168	0	20	guess
4		-1	0	23	-1	-1	-1	192	168	1	30	-1	-1	-1	-1	rsh

Figure 12: Random Individuals

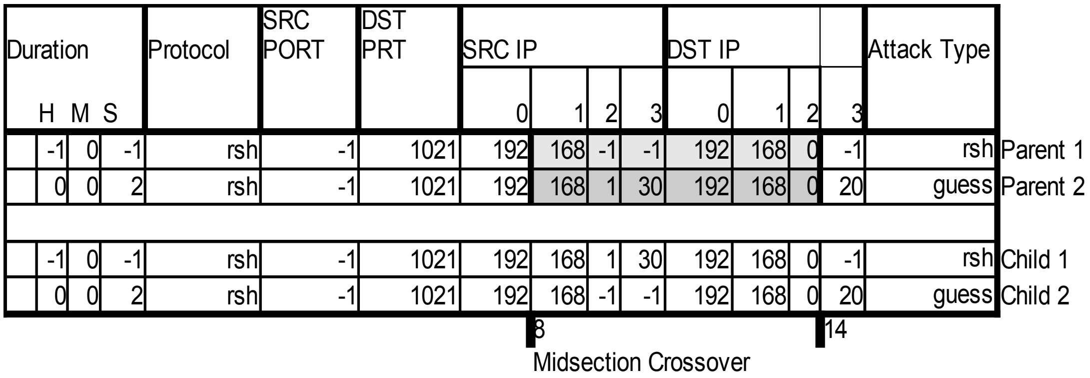
The initial group of random individuals is considered the old population once it enters the iterative loop. The area inside the box of Figure 11 is the process of generating a new population. In the sample above, it has an old population of four individuals, so the process followed in the box will generate 4 random individuals as well. The first step is that the two fittest individuals for each attack type are copied over into the new population.

The sample audit data contains the following unique attack types:

```
rsh
guess
rlogin
rcp
```

Assuming there are at least eight individuals with two of each attack type, the top two of each attack type would be copied over into the new population.

After the initial elite individuals are copied into the new population, the remaining are generated using crossover and applied mutation. Considering our initial population is 400 and the number of unique attack types were 4, then the new population would require 392 individuals to generate. For crossover, three individuals are chosen from the pool of the old population and the best two of three are used as “parents” for crossover. netGA uses a two point midsection crossover. The algorithm chooses two random cross section points from the *Cross Idx* list shown in Figure 7 and exchanges the midsection between the parents to form two new children (Figure 13) .



Duration				Protocol	SRC PORT	DST PRT	SRC IP				DST IP				Attack Type	
	H	M	S				0	1	2	3	0	1	2	3		
	-1	0	-1	rsh	-1	1021	192	168	-1	-1	192	168	0	-1	rsh	Parent 1
	0	0	2	rsh	-1	1021	192	168	1	30	192	168	0	20	guess	Parent 2
	-1	0	-1	rsh	-1	1021	192	168	1	30	192	168	0	-1	rsh	Child 1
	0	0	2	rsh	-1	1021	192	168	-1	-1	192	168	0	20	guess	Child 2

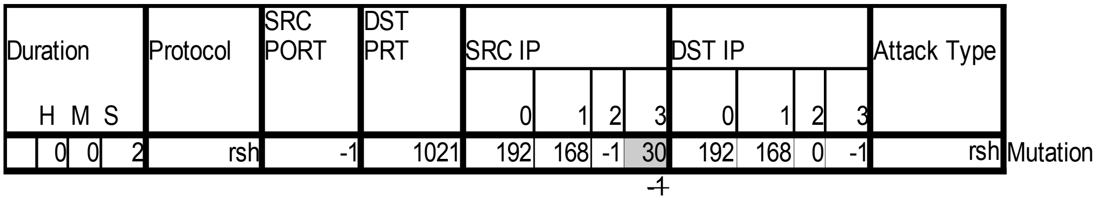
Figure 13: Sample Crossover

Mutation is an algorithm that iterates through the genes for an individual and and flips the field if the value comes up for that field. For each gene, it “rolls” the dice for that field and changes the value of that field to another unique value or a random value if the “roll” matches the probability. Figure 14 Illustrates this with a probability of 0.03.

```
SRC PORT x (0.03 probability) → gets chosen.
Choose new value randomly from (-1, 1892, 1900, 1023, 1906, 1022,
43549, 1914, 43560, 43566)
```

Figure 14: Mutation Pseudo-code

Figure 15 illustrates a sample mutation of the forth octet in the SRC IP gene changing from an initial value of 30 to the wild card entry of -1.



Duration				Protocol	SRC PORT	DST PRT	SRC IP				DST IP				Attack Type	
	H	M	S				0	1	2	3	0	1	2	3		
	0	0	2	rsh	-1	1021	192	168	-1	30	192	168	0	-1	rsh	Mutation
										-1						

Figure 15: Mutation Chromosome

3.2 Design Overview

The development approach of netGA closely matches the one used by Gong. NetGA creates individuals using unique values discovered for each gene during the load of audit data used in the creation of individuals when forming the initial population. netGA also utilizes elitism when producing a new population, where the best individuals are copied from the old population into the new population. Gong specifies an Evaluator class which could or could not be considered the equivalent of the elitism function in netGA. netGA runs for a fixed number of iterations when evolving individuals.

Gong's proposed approach uses the Java ECJ library [ECLab] while netGA uses the “C” programming language and Glib [GLIB] library. The overall approach for netGA is: (1) parses audit data, (2) produces a set of random rules, (3) goes through an iterative evolutionary process driving towards better individuals guided by the fitness function. At the end of a fixed number of iterations netGA prints out the top 30 rules. Options such as the number of iterations are hard coded into netGA. The procedural structure of netGA is shown in Figure 16.

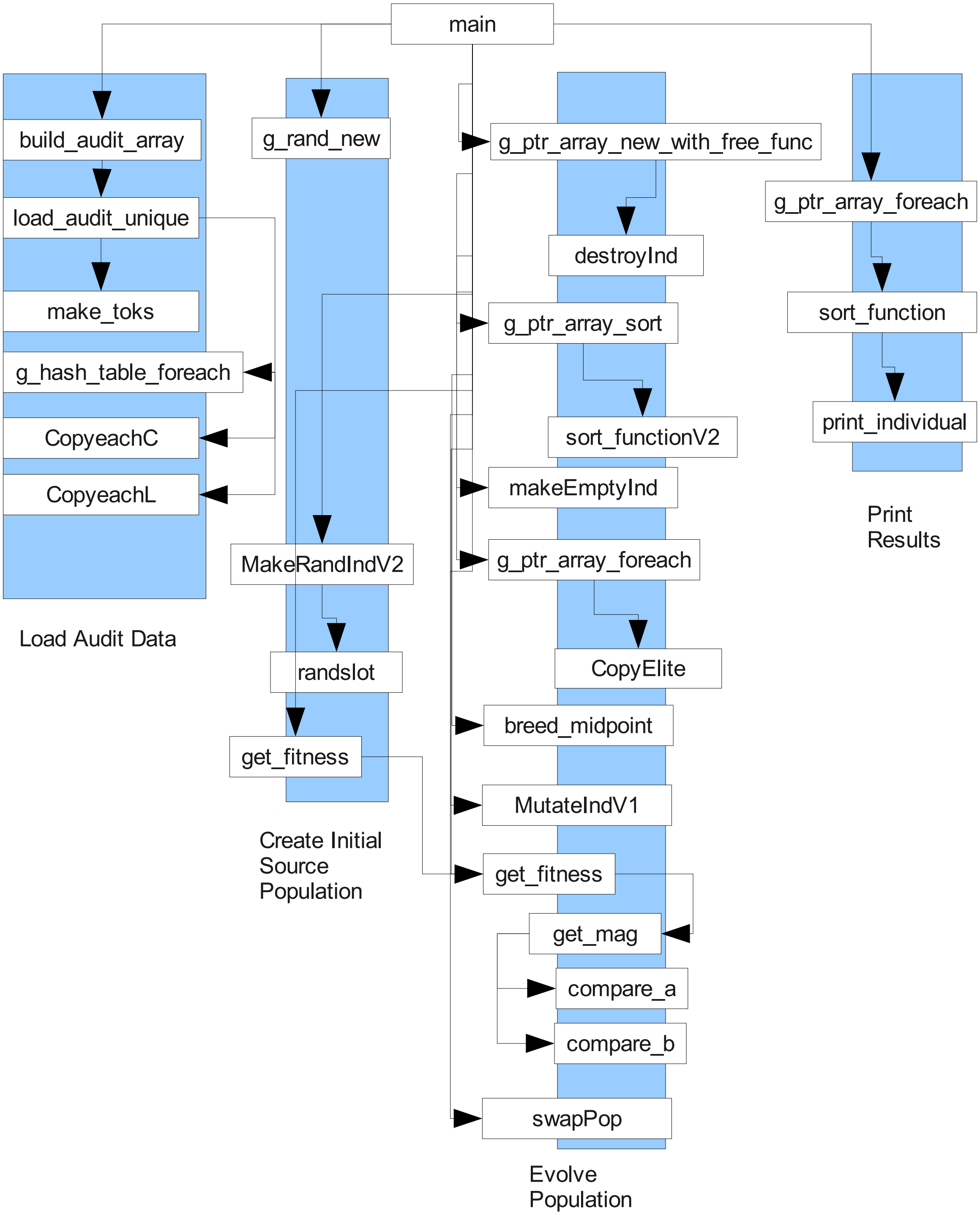


Figure 16: Function Calls in netGA

3.3 Primary Data Structures

The main representation of the chromosome, secondly known as an individual, and thirdly known as a set of rules is a seven integer array (below). netGA stores this seven integer array inside of a struct along with double for fitness and a string for an optional description. The following is the code for an individual:

```
typedef struct
{
  char desc[DESC_SZ];
  int chrome[7];
  double fitness;
} individual;
```

The *individual* type above is also used when loading the audit data. NetGA stores these individuals in a Glib audit list:

```
GSList *auditList;
```

The following starts with the description of storage of the connection into the individual type. The data that represents the connection duration is packed into an integer value using a 4 element char (8-bit) array to store the values of the hour, minute, and second for the array. The value for hour goes into byte[1], minute goes into byte[2], and second goes into byte[3]. byte[0] is not used. The char values allow for a value of 0 to 255. netGA considers 255 a wild card value of negative 1. All other values , 0 to 254 can be stored in the char. Once the individual values for the byte[4] array are assigned, the packed value can be assigned to the integer representation.

The following union is the code that represents the time_stamp:

```
typedef union {
  char byte[4];
  unsigned int tot;
} time_stamp;
```

The following code segment demonstrates how a duration of 0 hours, 3 minutes, and a wild card for the seconds is assigned and in turn assigned to the individual:

```
time_stamp foo;
individual bar;

foo.byte[0] = 0;
foo.byte[1] = 0;
foo.byte[2] = 3;
foo.byte[3] = -1;
bar.chrome[0] = foo.tot;
```

Each area of the chromosome that is used to represent data must be tracked when loading audit data. The routine that loads the audit data tracks unique values in the following Glib data structure:

```
GHashTable *myHTableL[NUM_HTABLES];
GHashTable *myHTableC[NUM_HTABLES][SUBH];
```

The hash data structure tracks only unique values as audit data is loaded. The constant defined for NUM_HTABLES is 7. The constant defined for SUBH is 4. Both these constants directly correlate to the chromosome *individual* type defined earlier that contains the 7 integer array. Each element can be decomposed into 4 (8-bit) char values.

The unique data is later loaded into a GLib sequence data structure that may be accessed via index value :

```
  GArray *myArrayL[NUM_HTABLES];
  GArray *myArrayC[NUM_HTABLES][SUBH];
```

The same type of union technique described earlier for the duration is used to represent the individual elements of the IP address. An IPv4 address occupies four octets or 32 bits, fitting nicely into the forth element of the array. The same approach for storage is used as the time stamp. If the octet has a value of 255, then the only way to represent this is with a wild card. For an 8-bit value -1 is equal to 255. This limits the rule, but allows for an 8 bit value. Gong uses the same approach to the gene representation:

```
    typedef union {
      char octet[4];
      unsigned int full;
    } IPAddr;
```

netGA uses enum types to represent attack types and service and for reference index values in the *individual* integer array for better code clarity. A separate string array holds the string representation for the corresponding attack or service type constant values. The data for these types is shown below:

```
    enum FILE_GENE_IDX{F_DURATION=3, F_SERVICE=4, F_SOURCE_PORT=5,
F_DEST_PORT=6, F_SRC_IP=7, F_DEST_IP=8, F_ATTACK=10};

    enum ARY_GENE_IDX{G_DURATION=0, G_SERVICE=1, G_SOURCE_PORT=2,
G_DEST_PORT=3, G_SRC_IP=4, G_DEST_IP=5, G_ATTACK=6};

    enum
SERVICE{EXEC=0,FINGER=1,FTP=2,RLOGIN=3,RSH=4,SMTP=5,TELNET=6,ENDP=7};

    enum ATTACK{NONE=0,GUESS_A=1, PORT_SCAN_A=2, RCP_A=3, RLOGIN_A=4,
RSH_A=5, FORMAT_CLEAR_A=6, FFB_CLEAR_A=7, END_A=8};
```

```
    char services[10][40] =
{"exec","finger","ftp","rlogin","rsh","smtp","telnet","endp"};


    char attacks[END_A][255];

    gint global_individual_count=0;

    void init_attacks() {
      strcpy(attacks[NONE],"none");
      strcpy(attacks[GUESS_A],"guess");
      strcpy(attacks[PORT_SCAN_A],"port-scan");
      strcpy(attacks[RCP_A],"rcp");
      strcpy(attacks[RLOGIN_A],"rlogin");
      strcpy(attacks[RSH_A],"rlogin");
      strcpy(attacks[FORMAT_CLEAR_A],"format_clear");
      strcpy(attacks[FFB_CLEAR_A],"ffb_clear");
      strcpy(attacks[END_A],"end");
    }
```

This seven byte representation makes for easy manipulation of individuals. The following is how netGA creates the Source IP part of the Random individual. *myIP* is an *IPAddr* type described above. The *randslot* function (described in the background section) chooses one of the unique values discovered in the loading of the audit data (range 0 to 254) or the wild card value (-1):

```
    // Source IP xxx.xxx.xxx.xxx
    //            0   1   2   3
    for (i=0; i<4; i++) {
    mySlot = randslot(rnd, garraysC[G_SRC_IP][i]->len, wcardProb);
    myIP.octet[i] = g_array_index (garraysC[G_SRC_IP][i], guchar,
mySlot);
    }
    tmpChrome[G_SRC_IP] = myIP.full;
```

After the individual octets for the IP address are assigned, the whole 32 bit value of the union is assigned to Source IP section of the chromosome.

netGA uses the following struct when copying the best individuals in each attack area:

```
typedef struct {
  enum ATTACK prevAttack;
  int count;
  GPtrArray  *popSrc, *popDest;
} prevData;
```

This structure is used by an iterator named *g_ptr_array_foreach* in GLib and works in conjunction with the *CopyElite* function which is also passed as an argument to the iterator. As the iterator proceeds through the list of individuals, a pointer to the data structure maintains information about the previous attack, copied elite count, and the source and destination populations between calls for the list of individuals. The documentation from the Glib library (reference) further describes the technique of this *user_data* structure.

3.4 Pseudo-code

The netGA program utilizes the functions and data structures following the pseudo-code shown in Figure 17. The netGA program has four main areas:

1. Load Audit Data
2. Create Initial Source Population
3. Evolve Population
4. Print Results

The four areas in *Genetic Algorithms Pseudo-code* (Figure 17) match the blocks shown in *Function Calls in netGA* (Figure 16). The following sections describe the netGA executable and how it works with the data structures and the functions.

```
Load Audit Data
    Open audit data file
    while audit file has records
        read record
        check fields of record against unique data sets
end Load Audit Data

Create Initial Source Population
    Generate N random individuals
        Create Individual from Unique data
        Calculate fitness of individual
end Generate Initial Population

Evolve Population
    Initialize Destination Population
    Sort Source Population on Attack Type then on Fitness
    Copy Elite Individuals to Destination Population
    do the following
        pick 3 random Individuals
        With 2 most fitest Individuals as Parents
            Breed 2 new children
            Apply Mutation to children
            Calculate fitness of children
            Add Individuals to Destination Population
        Swap Destination with Source Population
    for N minus number of Elite Individuals
end Evolve

Print Results
    Sort Source Population on Fitness
    Print Top 30 Individuals
end Print
```

Figure 17: Genetic Algorithms Pseudo-code

3.4.1 Load Audit Data

The job of the *Load Audit Data* section is to populate the list by parsing the audit data file and finding the unique values for each area of the *individual* array. The *build_audit_array* sets up the hashes and then makes a call to the *load_audit_unique* which opens the file of audit data and reads each line. It calls *make_toks* which makes tokens from the line and inserts the token values into the hash that maintains unique values for each token field. Upon return back to the *build_audit_array,* the function copies the unique values from a hash to an array so that the unique elements for each token field can be referenced by an array index value.

3.4.2 Create Initial Source Population

This section begins by initializing the random number generator using the built-in Glib function *g_rand_new.* This random number source provides a consistent source of entropy and if fed the same initial seed can replicate the same random path. With the unique values from the *Load Audit Data* (Figure 16) code section and the random number input, random individuals are created by a call to *MakeRandIndV2*. This function makes a call to *randslot* to choose a random slot and pick a random element from the set of unique values including the wild card value. The technique describing this algorithm was described in the Genetic Algorithm section.

3.4.3 Evolve Population

This section starts by creating a Glib array with the destroy function which acts as the new destination population. The source population is sorted using GLib's built in sorting function which is passed the *sort_functionV2* function for comparing elements. This resulting sort groups individuals by attack type and then sorts based upon fitness. GLib's built-in *g_ptr_array_foreach* utilizes the *CopyElite* function that in turn uses the *prev_data* type for copying the best two individuals for each attack type into the destination population. This block of population creates the same number of individuals as the old population, so the remaining number to create is N minus the number of elite individuals. This is represented by the *do* loop in the pseudo code. The code picks three random individuals by getting a random index in the array. The two top fittest individuals are used as parents to create two new children by calling *breed_midpoint.* The MutateIndV1 applies possible mutation. The *get_fitness* routine calculates the fitness and the new children are added to the destination population. At the end of this do loop, the destination population is swapped with the source and the loop is repeated (line 16). At the end of "N minus the number of elite operations" iterations, the evolve process stops.

3.4.4 Print Results

The final result exists in the source population, because the *swapPop* function is called before the end of the *Evolve Population* process. The population is sorted and the top 30 individuals are printed, regardless of attack type that the rules identify. The output is directly used as input to the plug-in.

The user is must redirect the output to the file and manually add the "-2" file. The suggested name of this file is *rules.txt,* as will be seen in the following section. This completes the Genetic Algorithms portion of generating the rules for the Network Intrusion Detection System. The rules are ready for utilization in the nProbe plug-in.

3.5 nProbe Integration

nProbe reads a configuration file specified as an option on the command line and reads the rules from that file (Figure 18). The rules file terminates with a "-2" on a single line.

```
0,0,23 telnet -1 23 192.168.1.-1 192.168.0.20 guess
399 fitness is 0.8063
-2
```

Figure 18: Sample Rules for nProbe plug-in

In order to use the nProbe with the netGA plug-in run it as follows:

```
nprobe --netGA "./rules.txt" <other options>
```

3.5.1 Design Overview

The netGA plug-in parses the rules specified in the configuration file specified at run-time. The *read_record1* function parses the first part of the rule attributes for identifying an attack (line 1 of Figure 18) and places the data in the following struct:

```
typedef struct {
  int dur_h;
  int dur_m;
  int dur_s;
  char protocol[16];
  int src_port;
  int dst_port;
  int srcIP[4];
  int dstIP[4];
  char attack[16];
} record1;
```

Each rule has a rule number and an associated fitness. *read_record2* parses line 2 of the rule information and stores it in the record2 struct:

```
typedef struct {
  int rulenum;
  float fitness;
} record2;
```

The *record2* struct holds holds the rule number and the fitness for the rule number. netGA uses a linked list to store all the rules it parses with each element of the list storing the *record1* and *record2* struct information previously parsed. The linked list is represented by the *record3* struct. The final rule uses the NULL pointer as the value for the "next" field in the struct.

The struct is shown below:

```
struct record3 {
  struct record3 *next;
  record1 r;
  record2 s;
};
```

As the IP address comes in on the wire, nProbe stores the value in a 32 bit integer variable. The union is used to access the individual octets of the IP Address. The netGA plug-in converts the IP address from network byte order to host byte order before assigning it to the *int* portion of the union. Then, the plug-in can access the individual octets by reading an element of the *octet* array. Below is the code used for the union:

```
typedef union {
  char octet[4];
  unsigned int full;
} IPAddr;
```

nProbe uses a template for specifying the configuration for the plug-in. nProbe scans the directory with plug-ins, and attempts to load the plug-ins via the name of the plug-in. Once it loads the dynamically loadable “.so” file, it searches for a struct named "netGAPlugin" (Figure 19) and then inspects the elements to determine how the plug-in operates, considered the configuration for the plug-in.

```
/* Plugin entrypoint */
static PluginInfo netGAPlugin = {
  NPROBE_REVISION,
  "NetGA",
  "0.1",
  "Genetic Algorithm rule matcher",
  "Brian E. Lavender",
  1 /*  always enabled */, 1, /* enabled */
  netGAPlugin_init,
  NULL, /* Term */
  netGAPlugin_conf,
  NULL,
  0, /* call packetFlowFctn for each packet */
  NULL,
  netGAPlugin_get_template,
  netGAPlugin_export,
  netGAPlugin_print,
  NULL,
  netGAPlugin_help
};
```

Figure 19: netGA plug-in Configuration struct

The struct has one critical area used by the netGA plug-in. This is the *netGAPlugin_init* value, which is the name of the function which starts the plug-in. The following section describes the functions (Figure 20) contained within the netGA plug-in and how they interact with each other.

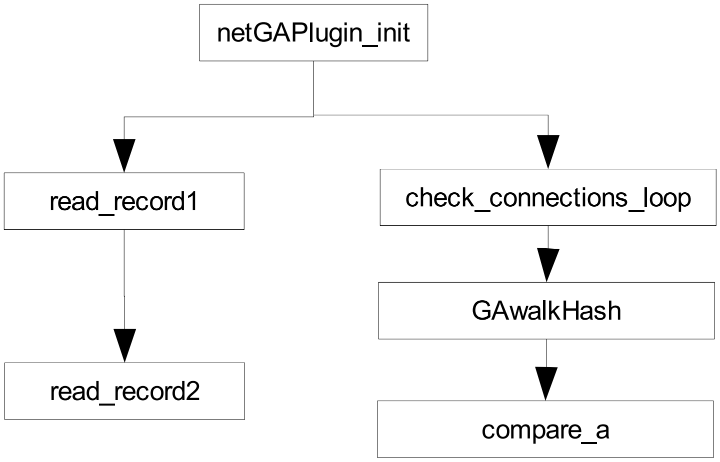


Figure 20: Plug-in Function Calls

The plug-in follows the basic pseudo code:

```
1. Load rules from configuration file.
2. Iterate over the set of connections comparing each connection
   attributes to the set of rules loaded in from rules.txt
   specified above
3. sleep one second.
4. Go back to step 2.
```

When nProbe starts, it scans the plug-ins folder searching for available loadable plug-ins. For each plug-in, it retrieves the *PluginInfo* struct with a name matching the name of the plug-in. For netGA, the plug-in is named netGAPlugin. Thus, nProbe searches for the *PluginInfo* type called *netGAPlugin.* The *netGAPlugin* variable contains the information so that nProbe knows how to manage the netGA plug-in. The *PluginInfo* type specifies an initialization function, configuration function, *netflow* template function, export function, print function, and a help function among some optional attributes. The *netGAPlugin_init* function is the critical function for the netGA plug-in. It loads the rules file, and creates a thread for performing the iterated task of checking rules against the set of active connections. The thread it creates calls the function named *check_connections_thread.* This thread sleeps one second and checks the list of rules loaded against the set of active connections (iteration code contributed by Luca Deri). The plug-in checks the duration of the connection, the source and destination IP addresses, and the source and destination ports against each rule in the set of rules loaded. The one part of the rules that the plug-in does not check is the protocol. The destination port could be used to determine the protocol assuming that the protocols ran on standard ports. For example, a web server often runs on port 80, but a user can specify any port for this service to run. While a rule may specify the protocol, the plug-in treats the protocol as equivalent to being a wild card. When a rule matches a connection, the plug-in prints to *stdout* the connection that it matched.

Chapter 4

RESULTS

The results section shows how to utilize the concrete implementation and also evaluates results. The following sections describe how to run, and observations gathered from sample runs of the netGA executable and operation of the plug-in with nProbe.

4.1 netGA Executable and Evaluation

Enter the source for the netGA executable (see Appendix) and compile it using the following command:

```
$ make
```

The DARPA data set contains a file named *bsm.list*. Put this file in the same directory as the netGA executable. The netGA executable sends its output to standard output. It is recommended to redirect the output to a file. The output begins with the sample random rules. The program evolves the rules and finishes by sending to standard output the top 30 rules. It prints a -2 then ends. To run the netGA with output redirection, use the following command:

```
$ netga > rules.txt
```

	H	M	S	Protocol	Src Port	Dest	Source IP Address				Destination				Attack	Fitness
1	0	0	5	telnet	-1	-1	-1	168	1	30	192	168	0	20	rcp	0
2	0	1	42	telnet	1832	513	192	168	0	30	-1	168	0	20	rcp	0
3	0	1	19	rsh	1022	23	192	168	1	30	192	168	0	20	rlogin	0
4	0	1	20	smtp	-1	23	192	168	1	30	10	168	0	20	rcp	0
5	0	1	14	rsh	43587	23	192	168	0	30	-1	168	0	20	rcp	0
6	0	0	-1	rlogin	-1	23	192	168	1	40	192	168	0	20	guess	0
7	0	0	20	rsh	1906	-1	192	168	0	30	10	168	0	20	rsh	0
8	0	0	-1	rlogin	1906	513	192	168	1	30	192	168	0	20	rcp	0
9	0	0	20	-1	-1	513	192	168	1	40	192	168	0	20	guess	0
10	0	0	14	telnet	1832	512	192	168	0	30	10	168	0	20	rlogin	0
11	0	0	11	exec	43497	-1	192	168	1	30	-1	168	0	20	rcp	0
12	-1	1	23	rlogin	-1	-1	192	168	1	40	192	168	0	20	rcp	0
13	0	1	48	telnet	1876	-1	192	168	1	30	-1	168	0	20	port-scan	0
14	0	0	-1	rsh	-1	-1	192	168	1	30	-1	168	0	20	rcp	0.2698
15	0	0	-1	rsh	1023	-1	192	168	1	30	-1	168	0	20	rcp	0.8031
16	-1	0	2	rsh	1023	-1	192	168	1	30	-1	168	0	20	rcp	0.8031
17	0	0	14	rlogin	-1	513	192	168	1	30	-1	168	0	20	rsh	0.8031
18	0	0	14	rlogin	-1	513	192	168	1	30	-1	168	0	20	rsh	0.8031
19	0	0	-1	-1	-1	512	192	168	1	30	192	168	0	20	port-scan	0.8031
20	0	0	-1	rsh	1023	-1	192	168	1	30	-1	168	0	20	rcp	0.8031
21	-1	0	2	rsh	1023	-1	192	168	1	30	192	-1	0	20	rcp	0.8031
22	-1	0	2	rsh	1023	-1	192	168	1	30	192	168	0	20	rcp	0.8031
23	-1	0	2	rsh	1023	-1	192	168	1	30	192	168	0	20	rcp	0.8031
24	0	0	23	telnet	-1	23	192	168	1	30	192	168	0	20	guess	0.8063
25	0	0	23	telnet	-1	23	192	168	1	30	192	168	0	20	guess	0.8063
26	0	0	5	-1	-1	-1	192	168	1	30	-1	168	0	20	port-scan	0.8063
27	0	0	5	-1	-1	-1	192	168	1	30	-1	168	0	20	port-scan	0.8063
28	0	0	23	telnet	-1	23	192	168	1	30	192	168	0	20	guess	0.8063
29	0	0	5	-1	-1	-1	192	168	1	30	-1	168	0	20	port-scan	0.8063
30	0	0	5	-1	-1	-1	192	168	1	30	-1	168	0	20	port-scan	0.8063

Figure 21: Rules Generated by netGA Executable

Depending upon the initialization of the random number using the g_rand_new() function will determine the output of the evolution process. Figure 21 shows results for a sample run. The first 13 rules as having a fitness of zero. Thus, these rules have no effectiveness in identifying attacks and won't provide any value as far as identifying attacks. Twenty program runs of the netGA executable consistently produced 18 or so rules that had a fitness greater than zero. These runs used 400 initial individuals and netGA went through 5000 evolutions. Even with a varied number of evolutions, the netGA executable continually produced 12 to 16 individuals with fitness greater than zero.

4.2 nProbe Plug-in Build and Evaluation

Patch nProbe source using the *diff* listing and the source listing for the netGA plug-in provided in the Appendix. Run the following commands to build the program:

```
$ ./configure --prefix=/usr/local/nprobe
$ make
$ su
# make install
```

Set the library path :

```
# export LD_LIBRARY_PATH=/usr/local/nprobe/lib
```

Set the path :

```
# export PATH=/usr/local/nprobe/bin:$PATH
```

Enable the dummy network interface, specific to Linux:

```
# modprobe dummy
```

Configure the dummy interface to listen to traffic to any destination:

```
# ifconfig dummy0 0.0.0.0
```

Start nProbe with options. Set the option for *--netGA* option so that it matches the name of your rule file. In the following example, it is named *rules.txt*. Add the "-b2" option in order to view debugging output. The "-L" option indicates that hosts in the 192.168.1.0/24 are in the local network.

Start nprobe using the following command:

```
# nprobe -b2 -i dummy0 --netGA "./rules.txt"   -L 192.168.0.0/24 -T \
"%L7_PROTO %IPV4_SRC_ADDR %IPV4_DST_ADDR %IPV4_NEXT_HOP %INPUT_SNMP \
%OUTPUT_SNMP %IN_PKTS %IN_BYTES %FIRST_SWITCHED %LAST_SWITCHED \
%L4_SRC_PORT %L4_DST_PORT %TCP_FLAGS %PROTOCOL %SRC_TOS %SRC_AS \
%DST_AS %SRC_MASK %DST_MASK %HTTP_URL %HTTP_RET_CODE %SMTP_MAIL_FROM \
%SMTP_RCPT_TO"  > foo.txt
```

rules.txt contains the following rule:

```
$ cat rules.txt
0,0,23 telnet -1 23 192.168.1.30 192.168.0.20 guess
399 fitness is 0.8063
-2
```

The DARPA data set contains the test playback stream in a file named sample_data01.tcpdump. Replay the network playback stream using tcpreplay as follows:

```
# tcpreplay -i dummy0 sample_data01.tcpdump
```

Tail the foo.txt output file to view results which include debugging information:

```
# tail -f foo.txt
```

This rule specified above for *rules.txt* matches 14 connections. In order to view the matched connections, you can grep the output file for "Match" and view the 11 previous lines to see what matches:

```
# grep -B11 Match foo.txt

NetGA TCP connection 192.168.1.30:1884->192.168.0.20:23
duration hours 0 minutes 0 seconds 23
rule hours 0 minutes 0 seconds 23
Src Rule IP 192.168.1.30
Src Test IP 192.168.1.30
```

```
Dst Rule IP 192.168.0.20
Dst Test IP 192.168.0.20
Src Rule Port -1
Src Test Port 1884
Dst Rule Port 23
Dst Test Port 23
Match <<<------------------------------>>>
```

Figure 22 illustrates the matches for a sample rule against a rule that identifies a guess attack. The rule matches a total of 14 different connections. The netGA plug-in for nProbe is unable to match against the protocol attribute in the rule, thus matching any protocol. The matched connections in Figure 22 all have destination port 23 which is the normal destination port (one could run a telnetd server on any port) for *telnet* protocol. While the plug-in can't match on the protocol, the fact the rule specifies port 23 as the destination port means that the rule has still worked despite this deficiency.

	Hours	Minute	Second	Protocol	Source IP	Destination IP	Source Port	Destination Port	Attack
Rule	0	0	23	telnet	192.168.1.-1	192.168.0.20	-1	23	guess
Matches									
1	0	0	23	*	192.168.1.30	192.168.0.20	1754	23	guess
2	0	0	23	*	192.168.1.30	192.168.0.20	1769	23	guess
3	0	0	23	*	192.168.1.30	192.168.0.20	1867	23	guess
4	0	0	23	*	192.168.1.30	192.168.0.20	1876	23	guess
5	0	0	23	*	192.168.1.30	192.168.0.20	1884	23	guess
6	0	0	23	*	192.168.1.30	192.168.0.20	1890	23	guess
7	0	0	23	*	192.168.1.30	192.168.0.20	1906	23	guess
8	0	0	23	*	192.168.1.30	192.168.0.20	1914	23	guess
9	0	0	23	*	192.168.1.30	192.168.0.20	1959	23	guess
10	0	0	23	*	192.168.1.30	192.168.0.20	1967	23	guess
11	0	0	23	*	192.168.1.30	192.168.0.20	1978	23	guess
12	0	0	23	*	192.168.1.30	192.168.0.20	2016	23	guess
13	0	0	23	*	192.168.1.30	192.168.0.20	2020	23	guess
14	0	0	23	*	192.168.1.30	192.168.0.20	1042	23	guess

Figure 22: Matches for Rule

Another rule evolved to match port-scan connections also matches *guess* connections. In this case, both connections satisfy the rule because the plug-in matches a rule as a connection accumulates time. The *port-scan* rule matches at 5 seconds, and the *guess* rule later matches at 23 seconds. The rules are not exclusive.

Rule 15 has the following parameters and identifies an rcp attack:

```
{-1,0,2,rsh,1023,-1,192,168,1,30,192,-1,0,20,rcp}
```

Because the netGA plug-in for nProbe can not match against the protocol, the rule becomes the equivalent to the following:

```
{-1,0,2,-1,1023,-1,192,168,1,30,192,-1,0,20,rcp}
```

This wild card matches a larger set of connections than originally intended including the matches also matched by the separate rule above for the guess type attack. The rule for the port-scan attack doesn't necessarily represent the *guess* attack though.

	Hours	Minutes	Sec-onds	Proto-col	Source IP	Destination IP	Source Port	Destination Port	Attack
Rule	0	0	5	-1	192.168.1.30	-1.168.0.20	-1	-1	port-scan
Matches									
1	0	0	5	*	192.168.1.30	192.168.0.20	1754	23	port-scan
2	0	0	5	*	192.168.1.30	192.168.0.20	1755	21	port-scan
3	0	0	5	*	192.168.1.30	192.168.0.20	1762	20	port-scan
4	0	0	5	*	192.168.1.30	192.168.0.20	1767	20	port-scan
5	0	0	5	*	192.168.1.30	192.168.0.20	1769	23	port-scan
6	0	0	5	*	192.168.1.30	192.168.0.20	1768	20	port-scan
7	0	0	5	*	192.168.1.30	192.168.0.20	1770	20	port-scan
8	0	0	5	*	192.168.1.30	192.168.0.20	1772	79	port-scan
9	0	0	5	*	192.168.1.30	192.168.0.20	1778	25	port-scan
10	0	0	5	*	192.168.1.30	192.168.0.20	1783	25	port-scan
11	0	0	5	*	192.168.1.30	192.168.0.20	1787	21	port-scan
12	0	0	5	*	192.168.1.30	192.168.0.20	1801	20	port-scan
13	0	0	5	*	192.168.1.30	192.168.0.20	1802	20	port-scan
14	0	0	5	*	192.168.1.30	192.168.0.20	1811	79	port-scan
15	0	0	5	*	192.168.1.30	192.168.0.20	1820	79	port-scan
16	0	0	5	*	192.168.1.30	192.168.0.20	1826	25	port-scan
17	0	0	5	*	192.168.1.30	192.168.0.20	1832	25	port-scan
18	0	0	5	*	192.168.1.30	192.168.0.20	1834	79	port-scan
19	0	0	5	*	192.168.1.30	192.168.0.20	1841	79	port-scan
20	0	0	5	*	192.168.1.30	192.168.0.20	1847	79	port-scan
21	0	0	5	*	192.168.1.30	192.168.0.20	1850	21	port-scan
22	0	0	5	*	192.168.1.30	192.168.0.20	1850	21	port-scan
23	0	0	5	*	192.168.1.30	192.168.0.20	1854	20	port-scan
24	0	0	5	*	192.168.1.30	192.168.0.20	1855	79	port-scan
25	0	0	5	*	192.168.1.30	192.168.0.20	1856	20	port-scan
26	0	0	5	*	192.168.1.30	192.168.0.20	1858	20	port-scan
27	0	0	5	*	192.168.1.30	192.168.0.20	1863	20	port-scan
28	0	0	5	*	192.168.1.30	192.168.0.20	1867	23	port-scan
29	0	0	5	*	192.168.1.30	192.168.0.20	1876	23	port-scan
30	0	0	5	*	192.168.1.30	192.168.0.20	1884	23	port-scan
31	0	0	5	*	192.168.1.30	192.168.0.20	1890	23	port-scan
32	0	0	5	*	192.168.1.30	192.168.0.20	1892	21	port-scan
33	0	0	5	*	192.168.1.30	192.168.0.20	1893	20	port-scan
34	0	0	5	*	192.168.1.30	192.168.0.20	1894	20	port-scan
35	0	0	5	*	192.168.1.30	192.168.0.20	1895	20	port-scan
36	0	0	5	*	192.168.1.30	192.168.0.20	1900	25	port-scan
37	0	0	5	*	192.168.1.30	192.168.0.20	1023	514	port-scan
38	0	0	5	*	192.168.1.30	192.168.0.20	1906	23	port-scan
39	0	0	5	*	192.168.1.30	192.168.0.20	1022	513	port-scan

Figure 23: Port-Scan Rule Results

40	0	0	5	*	192.168.1.30	192.168.0.20	1022	514	port-scan
41	0	0	5	*	192.168.1.30	192.168.0.20	1914	23	port-scan
42	0	0	5	*	192.168.1.30	192.168.0.20	1917	113	port-scan
43	0	0	5	*	192.168.1.30	192.168.0.20	1932	21	port-scan
44	0	0	5	*	192.168.1.30	192.168.0.20	1933	79	port-scan
45	0	0	5	*	192.168.1.30	192.168.0.20	1937	20	port-scan
46	0	0	5	*	192.168.1.30	192.168.0.20	1938	20	port-scan
47	0	0	5	*	192.168.1.30	192.168.0.20	1940	20	port-scan
48	0	0	5	*	192.168.1.30	192.168.0.20	1939	79	port-scan
49	0	0	5	*	192.168.1.30	192.168.0.20	1942	20	port-scan
50	0	0	5	*	192.168.1.30	192.168.0.20	1943	20	port-scan
51	0	0	5	*	192.168.1.30	192.168.0.20	1946	79	port-scan
52	0	0	5	*	192.168.1.30	192.168.0.20	1959	23	port-scan
53	0	0	5	*	192.168.1.30	192.168.0.20	1967	23	port-scan
54	0	0	5	*	192.168.1.30	192.168.0.20	1976	25	port-scan
55	0	0	5	*	192.168.1.30	192.168.0.20	1978	23	port-scan
56	0	0	5	*	192.168.1.30	192.168.0.20	1984	21	port-scan
57	0	0	5	*	192.168.1.30	192.168.0.20	1987	20	port-scan
58	0	0	5	*	192.168.1.30	192.168.0.20	1990	20	port-scan
59	0	0	5	*	192.168.1.30	192.168.0.20	1992	20	port-scan
60	0	0	5	*	192.168.1.30	192.168.0.20	2016	23	port-scan
61	0	0	5	*	192.168.1.30	192.168.0.20	2023	79	port-scan
62	0	0	5	*	192.168.1.30	192.168.0.20	2024	80	port-scan
63	0	0	5	*	192.168.1.30	192.168.0.20	2026	110	port-scan
64	0	0	5	*	192.168.1.30	192.168.0.20	2025	111	port-scan
65	0	0	5	*	192.168.1.30	192.168.0.20	2032	512	port-scan
66	0	0	5	*	192.168.1.30	192.168.0.20	2031	513	port-scan
67	0	0	5	*	192.168.1.30	192.168.0.20	2030	514	port-scan
68	0	0	5	*	192.168.1.30	192.168.0.20	2029	515	port-scan
69	0	0	5	*	192.168.1.30	192.168.0.20	2033	2049	port-scan
70	0	0	5	*	192.168.1.30	192.168.0.20	2034	3000	port-scan
71	0	0	5	*	192.168.1.30	192.168.0.20	2022	21	port-scan
72	0	0	5	*	192.168.1.30	192.168.0.20	2021	22	port-scan
73	0	0	5	*	192.168.1.30	192.168.0.20	2020	23	port-scan
74	0	0	5	*	192.168.1.30	192.168.0.20	2028	109	port-scan
75	0	0	5	*	192.168.1.30	192.168.0.20	2035	6000	port-scan
76	0	0	5	*	192.168.1.30	192.168.0.20	1042	23	port-scan
77	0	0	5	*	192.168.1.30	192.168.0.20	1048	25	port-scan
78	0	0	5	*	192.168.1.30	192.168.0.20	1050	79	port-scan

Figure 24: Port-Scan Results Continued

Chapter 5

FUTURE WORK

The netGA project has numerous areas to build upon. The netGA executable has a modular architecture, so a programmer can easily modify its code. The same applies to the nProbe plug-in as well. The project brings the following ideas to mind that could be good extensions:

1. Integrate with nProbe protocol analyzer for layer 7 of the network protocol. nProbe has a separate layer 7 analyzer, but currently, the netGA plug-in does not have access to it. Luca Deri, author of nProbe, indicated that the netGA plug-in would have to “piggy back” on the layer 7 plug-in. This would add capability that the netGA plug-in could match on the layer 7 attribute as is currently missing with the current chromosome representation.
2. Make exclusive rules. A rule intended for a duration of 23 seconds matches a connection of 23 seconds and only 23 seconds, not one of 5 seconds too as the duration of the connection progresses.
3. Find a better technique to match multiple types of attacks. While the current elitism attack produces a result set of varied types of attacks, the population never converges in a single direction.
4. There is a slow memory leak in the netGA executable that should be fixed.
5. Make gene representation so that it can match values of 255 in each area of the octet.

6. Build or find an audit system instead of using DARPA audit data.
7. Modify the nProbe plug-in so that it can read rules with a signal or socket instead of just at start up.
8. Run tests varying the parameter weights for *w1* and *w2* in the fitness function.

Chapter 6

CONCLUSION

This project provided a successful implementation of a concrete solution representing most of the techniques proposed by Gong. It also provides a successful implementation into the network analysis tool called nProbe. To summarize the netGA executable, it loads the audit data, and effectively executes the algorithms specified in the pseudo-code in the design overview section that closely represents the pseudo-code presented by Gong. The plug-in also executes the code as specified in the pseudo-code presented in the design overview section.

The following are some of the areas where the genetic algorithms netGA executable and nProbe plug-in could use improvement. The netGA program is capable of generating rules, but the population only generates a few more rules than number of elite individuals. While the rules that do have a fitness greater than zero are effective, the population doesn't build upon itself. Other techniques should be investigated. The plug-in works well when the rule it is utilizing is not dependent upon the protocol, or when the destination port matches the standard port the protocol usually runs on. In other areas, some rules match many connections that they shouldn't.

The project helps illustrate the implementation proposed by Gong and provides a solid foundation for others to build upon.

APPENDIX

Source Code

netGA executable

```
./compare.c
#include <string.h>
#include <glib.h>
#include <glib/gprintf.h>
#include <stdlib.h>
#include "types.h"
#include "compare.h"
#include "print.h"
#include "service_attacks.h"

#ifndef SWAP_4
#define SWAP_4(x) ( ((x) << 24) | \
         (((x) << 8) & 0x00ff0000) | \
         (((x) >> 8) & 0x0000ff00) | \
         ((x) >> 24) )
#endif



// Idx Feature Name    Format Number of Genes
//              byte  0 1 2 3
//  0  Duration         h:m:s         3
//  1  Protocol         Int           1
//  2  Source_port      Int           1
//  3  Destination_port Int           1
//                byte  0 1 2 3
//  4  Source_IP        a.b.c.d       4
//                byte  0 1 2 3
//  5  Destination_IP   a.b.c.d       4
//  6  Attack_name      Int           1
//
// Chromosome length 7

// myEvolve - individual from evovlved data
// myAudit - individual from Audit data
// return match variable
// 0 - no match
// 1 - match
gboolean compare_a(individual *trainer, individual *myAudit) {
  // assume we have a match
  int match = TRUE;
  int i, j;

  time_stamp tmpTimeE, tmpTimeA;
  //  IPAddr tmpIPE, tmpIPA;

```

```

  //  g_printf("match is %d\n",match);

  for (i=0; i<6; i++) {
    switch (i) {
    case 0: // Duration
    case 4: // Source IP
    case 5: // Destination IP
      // PART 0 of chromosome - Duration
      //  g_printf("Chrome %d\n",i);
      tmpTimeE.tot = trainer->chrome[i];
      tmpTimeA.tot = myAudit->chrome[i];

      // g_printf("a tot %x\n", tmpTimeE.tot);
      // g_printf("b tot %x\n", tmpTimeA.tot);
      // Assumes that the first byte of duration is -1
      for (j = 0; j<4; j++) {
  // We want to see if it doesn't match.
  if ( !
       ( tmpTimeE.byte[j] == -1 || tmpTimeE.byte[j] == tmpTimeA.byte[j] )
       )
    match = FALSE;

  /*    g_printf("chrome %d match is %d %d %d\n",i,match, */
  /*           tmpTimeE.byte[j], */
  /*           tmpTimeA.byte[j]); */
      }
      break;

    case 1: // Protocol
    case 2: // Source Port
    case 3: // Dest Port

      if ( !
     ( trainer->chrome[i] == -1 || trainer->chrome[i] == myAudit->chrome[i] )
     )
  match = FALSE;
      // g_printf("chrome %d match is %d\n",i,match);
      break;
    default:
      ;
    }
  }

  return match;
}

gboolean compare_b(individual *trainer, individual *myAudit) {
  // assume we have a match
  gboolean match = TRUE;
  if ( !
```

```
       //       ( trainer->chrome[G_ATTACK] == -1 || trainer->chrome[G_ATTACK] == myAudit->chrome[G_ATTACK] )
       (  trainer->chrome[G_ATTACK] == myAudit->chrome[G_ATTACK] )
       )
    match = FALSE;
  return match;

}


gint get_mag(GSList *auditList, individual *trainer, guint *mag_AandB,
       guint *magA ) {
  GSList *iterator = NULL;
  individual *auditItem;
  gint n = 0;

  for (iterator = auditList; iterator; iterator = iterator->next) {
    auditItem = (individual*)iterator->data;


    if ( compare_a(trainer,auditItem) && compare_b(trainer,auditItem) )
      (*mag_AandB)++;

    if ( compare_a(trainer,auditItem) )
      (*magA)++;
    n++;
  }

  return n;

}

void get_fitness(GSList *auditList, individual *trainer, gdouble w1,
         gdouble w2, guint N) {
  guint countAandB=0,  countA=0;
  double fitness ;
  double support;
  double confidence;
  // Check the training individual
  get_mag(auditList, trainer, &countAandB, &countA);

  // Must not divide by 0
  if ( countA > 0 && N > 0 ) {
    support = countAandB / (double)N;
    confidence = countAandB / (double) countA;
    fitness = w1 * support + w2 * confidence;
  } else
    fitness = 0.0;
  // Assign the fitness
  trainer->fitness = fitness;
```

```
  set_string_individual(trainer);
}

gint sort_function(gconstpointer a, gconstpointer b) {
  individual **pia, **pib;

  gdouble fitness_a, fitness_b, delta;
  pia = (individual **) a;
  pib = (individual **) b;

  fitness_a = (*pia)->fitness;
  fitness_b = (*pib)->fitness;
  delta = fitness_a - fitness_b;

  //  g_print("fitness a: %.4f b: %.4f\n", fitness_a, fitness_b);

  if ( delta < 0.001 ) // they are equal
    return 0;

  if ( fitness_a < fitness_b )
    return -1;
  else
    return 1;
}

gint sort_functionV2(gconstpointer a, gconstpointer b) {
  individual **pia, **pib;

  gdouble fitness_a, fitness_b, delta;
  pia = (individual **) a;
  pib = (individual **) b;

  fitness_a = (*pia)->fitness;
  fitness_b = (*pib)->fitness;
  delta = fitness_a - fitness_b;

  //  g_print("fitness a: %.4f b: %.4f\n", fitness_a, fitness_b);

  if ( (*pia)->chrome[G_ATTACK] < (*pib)->chrome[G_ATTACK] )
    return -1;
  else if ( (*pia)->chrome[G_ATTACK] > (*pib)->chrome[G_ATTACK] )
    return 1;
  else {

/*     if ( delta < 0.001 ) // they are equal */
/*       return 0; */

    if ( fitness_a < fitness_b ) {
      return 1;
    } else if ( fitness_a > fitness_b ) {
      return -1;
    }

```

```
    return 0;

  }
  // Should not get here
}

void destroyInd(gpointer myInd) {
  individual *pInd;
  gdouble fitnessInd;
  pInd = (individual *)myInd;
  fitnessInd = pInd->fitness;
  g_slice_free(individual, pInd );
  global_individual_count--;

}

void normalize(gint *a, gint*b) {
  gint tmp;

  if ( *a > *b ) {
    tmp = *a;
    *a = *b;
    *b = tmp;
  }
}


gint get_crossByte(guint randInt) {
  gint idx;
  gint offset;
  guint base;
  gint rValue = -1;

  switch(randInt) {

  case 0: // left edge of chromosome storage
  case 1: // left edge of chromosome
  case 2:
  case 3:
    base = 0;
    idx = 0;
    break;
  case 4:
    base = 4;
    idx = 1;
    break;
  case 5:
    base = 5;
    idx = 2;
    break;
  case 6:
    base = 6;
    idx = 3;
```

```
    break;
  case 7:
  case 8:
  case 9:
  case 10:
    base = 7;
    idx = 4;
    break;
  case 11:
  case 12:
  case 13:
  case 14:
    base = 11;
    idx = 5;
    break;
  case 15:
    base = 15;
    idx = 6;
    break;
  case 16: // right edge of chromosome
    base = 16;
    idx = 7; //
    break;

  default:
    base = 0;
    idx = 0;
    ;
    //g_print("Default\n");
  }

  //g_print("randInt is %d\n",randInt);

  offset = randInt - base;
  rValue = idx * 4 + offset;


    return rValue;
}

void breed_v1(GRand *rnd, individual *parent1, individual *parent2,
     individual *child1, individual *child2 ) {

  gint randInt, whichbyte;

  // Pick a random integer between [1,17)
  randInt = g_rand_int_range(rnd,1,17);

  whichbyte = get_crossByte(randInt );


  if ( whichbyte == 1 || whichbyte == 28 ) {
    //g_print("No crossover\n");
```

```
    g_memmove(child1->chrome,parent1->chrome, NUM_GENE*4 );
    g_memmove(child2->chrome,parent2->chrome, NUM_GENE*4 );
  }
  else {

    g_memmove(child1->chrome,parent1->chrome, whichbyte );
    g_memmove((char *)(child1->chrome) + whichbyte ,
        (char *)(parent2->chrome) + whichbyte ,
        NUM_GENE*4 - whichbyte );

    g_memmove(child2->chrome,parent2->chrome, whichbyte );
    g_memmove((char *)(child2->chrome) + whichbyte ,
        (char *)(parent1->chrome) + whichbyte ,
        NUM_GENE*4 - whichbyte );


  }

}

void breed_v2(GRand *rnd, individual *parent1, individual *parent2,
     individual *child1, individual *child2 ) {

  gint randInt, whichbyte;
  individual t1, t2;
  gboolean putBack = FALSE;

  // If parents are different types, don't cross over certain data

  if (parent1->chrome[G_ATTACK] != parent2->chrome[G_ATTACK]) {
    // Stash away parent data
    g_memmove(&t1, parent1, sizeof(individual));
    g_memmove(&t2, parent2, sizeof(individual));
    putBack = TRUE;
  }

  // Pick a random integer between [1,17)
  randInt = g_rand_int_range(rnd,1,17);

  whichbyte = get_crossByte(randInt );

  if ( whichbyte == 1 || whichbyte == 28 ) {
    //g_print("No crossover\n");
    g_memmove(child1->chrome,parent1->chrome, NUM_GENE*4 );
    g_memmove(child2->chrome,parent2->chrome, NUM_GENE*4 );
  }
  else {
    g_memmove(child1->chrome,parent1->chrome, whichbyte );
    g_memmove((char *)(child1->chrome) + whichbyte ,
        (char *)(parent2->chrome) + whichbyte ,
        NUM_GENE*4 - whichbyte );

    g_memmove(child2->chrome,parent2->chrome, whichbyte );
```

```
    g_memmove((char *)(child2->chrome) + whichbyte ,
        (char *)(parent1->chrome) + whichbyte ,
        NUM_GENE*4 - whichbyte );

  }

  // putBack if true
  if (putBack) {
    // Fix back up child 1.
    child1->chrome[G_DEST_IP] = t1.chrome[G_DEST_IP];
    child1->chrome[G_SERVICE] = t1.chrome[G_SERVICE];
    child1->chrome[G_ATTACK] = t1.chrome[G_ATTACK];

    // Fix back up child 2.
    child2->chrome[G_DEST_IP] = t2.chrome[G_DEST_IP];
    child2->chrome[G_SERVICE] = t2.chrome[G_SERVICE];
    child2->chrome[G_ATTACK] = t2.chrome[G_ATTACK];
  }

}

void breed_midpoint(GRand *rnd, individual *parent1, individual *parent2,
            individual *child1, individual *child2 ) {

  gint randInt1, randInt2;
  gint whichbyte1, whichbyte2, delta1, delta2, delta3;
  randInt1 = g_rand_int_range(rnd, 1,17);
  randInt2 = g_rand_int_range(rnd, 1,17);

  whichbyte1 = get_crossByte(randInt1);
  whichbyte2 = get_crossByte(randInt2);

  // Make sure that whichbyte2 is greater than whichbyte1
  normalize(&whichbyte1, &whichbyte2);

  //  g_print("Bytes  %d %d\n", whichbyte1, whichbyte2);

  delta1 =  whichbyte1;
  delta2 = whichbyte2 - whichbyte1;
  delta3 = NUM_GENE*4 - whichbyte2;


  if (delta1 < 0 || delta2 < 0 || delta3 < 0) {
    g_print("Negative delta! %d %d %d\n",delta1, delta2, delta3);
    exit (-1);
  }

  // Child 1
  if (delta1 > 0)
    g_memmove((char *)(child1->chrome), (char *)(parent1->chrome), delta1 );

```

```
  if (delta2 >0)
    g_memmove((char *)(child1->chrome) + whichbyte1 ,
        (char *)(parent2->chrome) + whichbyte1  ,
        delta2 );

  if (delta3 > 0)
    g_memmove((char *)(child1->chrome) + whichbyte2,
        (char *)(parent1->chrome) + whichbyte2,
        delta3 );

  // Child 2
  if (delta1 > 0)
    g_memmove(child2->chrome, parent2->chrome, delta1 );

  if (delta2 > 0)
  g_memmove((char *)(child2->chrome) + whichbyte1 ,
      (char *)(parent1->chrome) + whichbyte1  ,
      delta2 );

  if (delta3 > 0)
    g_memmove((char *)child2->chrome + whichbyte2,
        (char *)parent2->chrome + whichbyte2,
        delta3 );

}
```

./compare.h

```
#ifndef COMPARE_H
#define COMPARE_H
#include "types.h"
extern gint global_individual_count;
gboolean compare_a(individual *trainer, individual *myAudit);
gboolean compare_b(individual *trainer, individual *myAudit);
gint get_mag(GSList *auditList, individual *trainer, guint
*mag_AandB,
       guint *magA);
void get_fitness(GSList *auditList, individual *trainer, gdouble
w1,
         gdouble w2, guint N);
gint sort_function(gconstpointer a, gconstpointer b);
gint sort_functionV2(gconstpointer a, gconstpointer b);
void destroyInd(gpointer myInd);
gint get_crossByte(guint randInt);
void breed_v1(GRand *rnd, individual *parent1, individual *parent2,
     individual *child1, individual *child2 );
void breed_v2(GRand *rnd, individual *parent1, individual *parent2,
     individual *child1, individual *child2 );
void breed_midpoint(GRand *rnd, individual *parent1, individual
*parent2,
            individual *child1, individual *child2);
#endif


```

```


./netga.c
 */



#include <string.h>
#include <stdlib.h>
#include <glib.h>
#include <glib/gprintf.h>
#include "types.h"
#include "print.h"
#include "rand.h"
#include "read_bsm.h"
#include "service_attacks.h"
#include "compare.h"

#ifndef SWAP_4
#define SWAP_4(x) ( ((x) << 24) | \
         (((x) << 8) & 0x00ff0000) | \
         (((x) >> 8) & 0x0000ff00) | \
         ((x) >> 24) )
#endif

extern gint global_individual_count;

typedef struct {
  enum ATTACK prevAttack;
  int count;
  GPtrArray  *popSrc, *popDest;
} prevData;


// Used with array iterator
void copyElite(gpointer a, gpointer userdata) {
  individual *trainer, *child1;
  prevData *getTwo;

  trainer = (individual *)a;
  getTwo = (prevData *)userdata;

  // I don't know why this gives a warning.
  // FIXME - Should the cast be required?
  if (trainer->chrome[G_ATTACK] != (int)getTwo->prevAttack) {
    getTwo->count = 0;
    getTwo->prevAttack = trainer->chrome[G_ATTACK];
  }

  if (getTwo->count < 2) {
    //g_print("Copy elite\n");
    child1 = makeEmptyInd();
```

```c
    g_memmove( child1, trainer, sizeof(individual) );
    g_ptr_array_add(getTwo->popDest, child1);
  }

  getTwo->count++;
}


int main() {
  GSList *auditList = NULL;
  GPtrArray  *popSrc, *popDest, *threeRand; // Array of individuals
  gdouble w1 = 0.2; // These constants are suggested per the paper
  gdouble w2 = 0.8;
  gdouble mutateProb = 0.05; // could be configurable
  gdouble wildCardProb = 0.05;
  gint nElite ;
  gint nPop = 400; // Should be even
  gint nEvolutions = 5000;
  gint array_len;
  gint i,j,k;

  // evolve keeps track of the number of times through the evolution cycle
  gint  evolve;

  prevData getTwo;

  GRand *rnd; // Random number entropy
  //  gdouble test_fit;

  GArray *myArrayL[NUM_HTABLES];
  GArray *myArrayC[NUM_HTABLES][SUBH];

  individual *trainer, *parent1, *parent2, *child1, *child2; // Random Individual
  guint nAuditRecords = 0; // number of audit records

  char *myfile = "./bsm.list";
  //char *myfile = "./bsm_pres1.list";

  //g_mem_set_vtable(glib_mem_profiler_table);
  //g_atexit(g_mem_profile);

  rnd = g_rand_new();

  //g_print("Audit data pulled from %s\n",myfile);


  // Load the audit list
  nAuditRecords = build_audit_array(&auditList, myArrayL, myArrayC, myfile);

  // Build an array of training individuals now
```

```

  // Initialize the array
  popSrc = g_ptr_array_new_with_free_func(destroyInd);

  i=0;
  while (i < nPop) {
    makeRandIndV2(rnd, wildCardProb, &trainer, myArrayL, myArrayC);
    get_fitness(auditList, trainer, w1, w2, nAuditRecords);
    //if ( trainer->fitness > 0.01 ) {
    g_ptr_array_add(popSrc, trainer);
    i++;
    //} else
    //destroyInd(trainer);
  }

  g_ptr_array_sort(popSrc, sort_functionV2);


  array_len = popSrc->len;
  g_print("Initial population length %d\n",popSrc->len);
  for (j = 0; j < array_len  ; j++) {
    //g_print("j is %d\n",j);
    trainer = g_ptr_array_index(popSrc, j);
    g_print("%02d %s\n",j,trainer->desc);
    g_print("fitness %.04f\n",trainer->fitness);
  }
  g_print("========================================\n");




  // Start evolution process

  // m keeps track of the number of times through the cycle
  evolve = 0;
  do {

    popDest = g_ptr_array_new_with_free_func(destroyInd);

    // Sort source population
    g_ptr_array_sort(popSrc, sort_functionV2);

    // Copy over elite.
    // Set up the user_data structure.
    // Maybe I should do this a different way.
    getTwo.prevAttack = NONE; // Initialize the previous attack type to NONE.
    getTwo.count = 0; // Previous attack count.
    getTwo.popSrc = popSrc;
    getTwo.popDest = popDest;

    g_ptr_array_foreach(popSrc, copyElite, &getTwo);

```

```
        // Check if we have an even elite population.
        // If not, add one.
        if ( popDest->len % 2 == 1 ) {
          j = g_rand_int_range(rnd, 0, nPop);
          trainer = g_ptr_array_index(popSrc, j);
          child1 = makeEmptyInd();
          g_memmove( child1, trainer, sizeof(individual) );
          g_ptr_array_add(popDest, child1);
        }


        /*       array_len = popDest->len; */
        /*       g_print("Elite population length %d\n",popDest->len); */
        /*       for (j = 0; j < array_len  ; j++) { */
        /*         trainer = g_ptr_array_index(popDest, j); */
        /*         g_print("%s\n",trainer->desc); */
        /*         g_print("fitness %.04f\n",trainer->fitness); */
        /*       } */

        //exit(0);

        nElite = popDest->len;

        // Evolve for remainder of population
        for (k = 0; k < (nPop - nElite) /2 ; k++) {

          //threeRand = g_ptr_array_new_with_free_func(destroyInd);


          threeRand = g_ptr_array_new();
          for (i=0; i< 3 ;i++) {
      j = g_rand_int_range(rnd, 0, nPop);
      trainer = g_ptr_array_index(popSrc, j);
      g_ptr_array_add(threeRand, trainer);
          }

          g_ptr_array_sort(threeRand, sort_function);

          //g_print("Two top individuals are the following\n");

          // fitness goes lowest to highest (0,1,2). Thus elements 1
and 2
          parent1 = g_ptr_array_index(threeRand, 1);
          parent2 = g_ptr_array_index(threeRand, 2);

          g_ptr_array_free(threeRand, FALSE);

          child1 = makeEmptyInd();
          child2 = makeEmptyInd();

          // Uncomment for No breed. Just test
          //g_memmove(child1->chrome, parent1->chrome, NUM_GENE*4 );
          //g_memmove(child2->chrome, parent2->chrome, NUM_GENE*4 );
```

```

      breed_midpoint(rnd, parent1, parent2, child1, child2);

      mutateIndV1(rnd, mutateProb, wildCardProb, child1, myArrayL, myArrayC);
      mutateIndV1(rnd, mutateProb, wildCardProb, child2, myArrayL, myArrayC);

      get_fitness(auditList, child1, w1, w2, nAuditRecords);
      get_fitness(auditList, child2, w1, w2, nAuditRecords);

      g_ptr_array_add(popDest, child1);
      g_ptr_array_add(popDest, child2);

    }

    // Get fitness of the 10th highest
    //trainer = g_ptr_array_index(popDest, (popDest->len) - 10);
    //test_fit = trainer->fitness;

    // Shallow copy
    swapPop(&popDest,&popSrc);

    g_ptr_array_free(popDest, TRUE);
    evolve++;

  } while ( evolve < nEvolutions );

  g_ptr_array_sort(popSrc, sort_function);

  // Print the top 30 individuals
  g_print("Top 30 individuals are the following\n");
  for (i=0, j= (popSrc->len) - 30 ; i<30 ; i++,j++) {
       trainer = g_ptr_array_index(popSrc, j);
       print_individual(trainer);
       g_print("%d fitness is %.04f\n",j,trainer->fitness);
  }

  g_ptr_array_free(popSrc, TRUE);
  g_slist_free(auditList);
  g_rand_free(rnd);

  //  g_print("The net individuals should match the number of audit records %d.\n",
  //      nAuditRecords);
  // g_print("Count is %d.\n",global_individual_count);
  g_print("-2\n");


  return 0;

}
```

```
./print.c
#include <glib.h>
#include <glib/gprintf.h>
#include <stdlib.h>

#include "types.h"
#include "print.h"
#include "service_attacks.h"

#ifndef SWAP_4
#define SWAP_4(x) ( ((x) << 24) | \
         (((x) << 8) & 0x00ff0000) | \
         (((x) >> 8) & 0x0000ff00) | \
         ((x) >> 24) )
#endif

void display_list(GSList *list)
{
    GSList *iterator = NULL;
    individual *myInd;


    // g_printf("print the data:\n");
    //print the list data
    for (iterator = list; iterator; iterator = iterator->next) {
        myInd = (individual*)iterator->data;
  g_printf("%s\n",myInd->desc);

  print_individual(myInd);
  g_printf("\n");
    }

}

void display_array(GPtrArray *myArray)
{
  guint i;
  individual *trainer;
  gdouble myFit;

  for (i=0; i < (myArray->len) ;  i++ ) {
       trainer = g_ptr_array_index(myArray, i);
       print_individual(trainer);
       myFit = trainer->fitness;
       //g_print("fitness is %.04f\n",myFit);
  }
}

void print_individual( individual *myInd ) {
  set_string_individual(myInd);
  g_printf("%s\n",myInd->desc);
}

```

```
void set_string_individual( individual *myInd ) {
  int i,j,k;
  guchar uv;
  char tmp[DESC_SZ];
  time_stamp tmpS;

  // Set description to empty string
  *(myInd->desc) = '\0';

  for (i=0; i<7; i++)  {
    switch (i) {

    case 0: // Duration
      //tmp = SWAP_4(myInd->chrome[i]);
      //g_printf("%08x ", tmp);
      tmpS.tot = myInd->chrome[i];
      for (j=1; j< 4; j++) {
  uv = (guchar)tmpS.byte[j];
  if ( uv < 255 ) // Check if it is neg1
    k = uv;
  else
    k = -1;
  //g_snprintf(tmp,DESC_SZ, "%02d",k);
  g_snprintf(tmp,DESC_SZ, "%d",k);
  g_strlcat(myInd->desc,tmp,DESC_SZ);
  if (j<3)
    g_snprintf(tmp,DESC_SZ, ",");
  else
    g_snprintf(tmp,DESC_SZ, " ");
  g_strlcat(myInd->desc,tmp,DESC_SZ);

      }
      break;
    case 4: // source IP
    case 5: // destination IP
      tmpS.tot = myInd->chrome[i];
      for (j=0; j< 4; j++) {
  uv = (guchar)tmpS.byte[j];
  if ( uv < 255 ) // Check if it is neg1
    k = uv;
  else
    k = -1;
  //g_printf("%03d", k);
  //g_snprintf(tmp,DESC_SZ, "%03d",k);
  g_snprintf(tmp,DESC_SZ, "%d",k);
  g_strlcat(myInd->desc,tmp,DESC_SZ);
  if (j<3)
    //g_print(".");
    g_snprintf(tmp,DESC_SZ, ".");
  else
    g_snprintf(tmp,DESC_SZ, " ");
  g_strlcat(myInd->desc,tmp,DESC_SZ);
      }
```

```

    break;
  case 1: // protocol
    switch (myInd->chrome[i]) {
    case EXEC:
g_snprintf(tmp,DESC_SZ, "  exec");
break;
    case FINGER:
g_snprintf(tmp,DESC_SZ, "finger");
break;
    case FTP:
g_snprintf(tmp,DESC_SZ,"   ftp" );
break;
    case RLOGIN:
g_snprintf(tmp,DESC_SZ, "rlogin" );
break;
    case RSH:
g_snprintf(tmp,DESC_SZ, "   rsh" );
break;
    case SMTP:
g_snprintf(tmp,DESC_SZ, "  smtp" );
break;
    case TELNET:
g_snprintf(tmp,DESC_SZ,"telnet" );
break;
    default:
//g_snprintf(tmp,DESC_SZ, " %05d", myInd->chrome[i]);
g_snprintf(tmp,DESC_SZ, " %d", myInd->chrome[i]);

    }
    g_strlcat(myInd->desc,tmp,DESC_SZ);
    g_strlcat(myInd->desc," ",DESC_SZ);
    break;

  case 6: // attack
    switch (myInd->chrome[i]) {
    case NONE:
g_snprintf(tmp,DESC_SZ, "none");
break;
    case GUESS_A:
g_snprintf(tmp,DESC_SZ, "guess");
break;
    case PORT_SCAN_A:
g_snprintf(tmp,DESC_SZ, "port-scan");
break;
    case RCP_A:
g_snprintf(tmp,DESC_SZ, "rcp");
break;
    case RLOGIN_A:
g_snprintf(tmp,DESC_SZ, "rlogin");
break;
    case RSH_A:
g_snprintf(tmp,DESC_SZ, "rsh");
```

```
  break;
      case FORMAT_CLEAR_A:
  g_snprintf(tmp,DESC_SZ, "format_clear");
  break;
      case FFB_CLEAR_A:
  g_snprintf(tmp,DESC_SZ, "ffb_clear");
  break;
      default:
  g_snprintf(tmp,DESC_SZ, "Unknown");
  //exit(1);
      }
      g_strlcat(myInd->desc,tmp,DESC_SZ);
      break;
    case 2:
    case 3:
      //g_snprintf(tmp,DESC_SZ, "%08d ", myInd->chrome[i] );
      g_snprintf(tmp,DESC_SZ, "%d ", myInd->chrome[i] );
      g_strlcat(myInd->desc,tmp,DESC_SZ);
      break;
    default:
      g_snprintf(tmp,DESC_SZ, "%08x ", myInd->chrome[i] );
      g_strlcat(myInd->desc,tmp,DESC_SZ);
    }
  }

}
```

./print.h

```
void display_array(GPtrArray *myArray);
void display_list(GSList *list);
void print_individual( individual *myInd );
void set_string_individual( individual *myInd );


```

./rand.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <glib.h>
#include "types.h"
#include "print.h"
#include "service_attacks.h"
#include "rand.h"

#define BUF_SZ 80


#ifndef SWAP_4
#define SWAP_4(x) ( ((x) << 24) |          \
            (((x) << 8) & 0x00ff0000) | \
            (((x) >> 8) & 0x0000ff00) | \
            ((x) >> 24) )
```

```
#endif


// returns an array index
// input
// numUniqueP1 - number of elements plus the wildcard element.
//               The wildcard element is assumed to be in index 0.
//               {-1,5,3,98,34,4}
// wcardProb   - Probability that you want the wildcard chosen.
guint randslot(GRand *rnd, guint numUniqueP1, double wcardProb ) {
  gdouble indProb;
  gdouble randVal;
  guint slot;

  indProb = (1.0 - wcardProb )/ ( numUniqueP1 - 1 );
  //  g_printf("Individual probability minus WCARD is %.6f\n",indProb);

  // Get a random number between [0,1)
  randVal = g_rand_double(rnd);

  if (randVal < wcardProb ) {
    //g_printf("Got a wildcard\n");
    slot = 0;
  } else {
    slot =  (randVal - wcardProb)/ ( 1 - wcardProb) * (numUniqueP1 - 1) + 1;

  }
  return slot;
}



void makeRandIndV1(GRand *rnd, individual **myInd)
{
  int tmp;
  time_stamp myDuration;
  guint tmpChrome[NUM_GENE];


  // Create a random chromosome

  // Time
  myDuration.tot = g_rand_int(rnd);
  myDuration.byte[0] = 0xff; // zero out first byte

  // copy myDuration to tmpChrome
  tmpChrome[0] = myDuration.tot;

  // protocol Loock at header file for protocol definitions
  // TODO - allow for a wildcard
  tmpChrome[1] = g_rand_int_range(rnd, 0, ENDP);
```

```

  // Src port. Max port number is range is [1, 2^16 -1 ]
  tmpChrome[2] = g_rand_int_range(rnd,1,0x10000 );

  // Dst port. Max port number is range [1 , 2^16 -1 ]
  tmpChrome[3] = g_rand_int_range(rnd,1,0x10000 );

  // full range on source IP
  tmpChrome[4] = g_rand_int(rnd);

  // full range on dest IP address
  tmpChrome[5] = g_rand_int(rnd);

  // random number [ 0 , END_A ]
  // which attack should this match?
  tmp = g_rand_int_range(rnd, 0, END_A );
  tmpChrome[6] = tmp;


  // Malloc an individual
  *myInd = g_slice_new0(individual);
  global_individual_count++;
  //*myInd = (individual *)g_malloc0(sizeof(individual));


  // Copy the temp chromosome into the individual
  g_memmove((*myInd)->chrome, tmpChrome, 4 * NUM_GENE);
  g_snprintf((*myInd)->desc, DESC_SZ,  "Training Chromosome" );



}


void makeRandIndV2(GRand *rnd, double wcardProb, individual **myInd,
           GArray *garraysL[NUM_HTABLES],
           GArray *garraysC[NUM_HTABLES][SUBH] ) {
  // TODO: stub for code
  gint i; // loop counter
  guint tmpChrome[NUM_GENE]; // tempChrome created
  time_stamp myT; // temporary time stamp data
  IPAddr myIP; // temporary IP address
  guint mySlot; // random slot number picked

  //  time_stamp myT; // temporary time stamp data
  // IPAddr myIP; // temporary IP address
  myT.byte[0] = 0xff; // not used, all should be -1

  // Hours, Minutes, Seconds
  for (i=1; i<4; i++) {
    mySlot = randslot(rnd, garraysC[G_DURATION][i]->len, wcardProb);
```

```
    myT.byte[i] = g_array_index (garraysC[G_DURATION][i], guchar, mySlot);
  }

  tmpChrome[G_DURATION] = myT.tot; // duration

  // Service
  mySlot = randslot(rnd, garraysL[G_SERVICE]->len, wcardProb);
  tmpChrome[G_SERVICE] = g_array_index (garraysL[G_SERVICE], guint, mySlot);

  // Source Port
  mySlot = randslot(rnd, garraysL[G_SOURCE_PORT]->len, wcardProb);
  tmpChrome[G_SOURCE_PORT] = g_array_index (garraysL[G_SOURCE_PORT], guint, mySlot);

  // Dest Port
  mySlot = randslot(rnd, garraysL[G_DEST_PORT]->len, wcardProb);
  tmpChrome[G_DEST_PORT] = g_array_index (garraysL[G_DEST_PORT], guint, mySlot);

  // Source IP xxx.xxx.xxx.xxx
  //             0   1   2   3
  for (i=0; i<4; i++) {
    mySlot = randslot(rnd, garraysC[G_SRC_IP][i]->len, wcardProb);
    myIP.octet[i] = g_array_index (garraysC[G_SRC_IP][i], guchar, mySlot);
  }
  tmpChrome[G_SRC_IP] = myIP.full; // source IP

  // Dest IP xxx.xxx.xxx.xxx
  //           0   1   2   3
  for (i=0; i<4; i++) {
    mySlot = randslot(rnd, garraysC[G_DEST_IP][i]->len, wcardProb);
    myIP.octet[i] = g_array_index (garraysC[G_DEST_IP][i], guchar, mySlot);
  }
  tmpChrome[G_DEST_IP] = myIP.full; // dest IP

  // Attack. Do not select wildcard, so put its probability at 0.
  mySlot = randslot(rnd, garraysL[G_ATTACK]->len, 0.0 );
  tmpChrome[G_ATTACK] = g_array_index (garraysL[G_ATTACK], guint, mySlot);

  *myInd = g_slice_new0(individual);
  global_individual_count++;

  // Copy the temp chromosome into the individual
  g_memmove( (*myInd)->chrome, tmpChrome, 4 * NUM_GENE);
  g_snprintf((*myInd)->desc, DESC_SZ,  "Training Chromosome" );

}

```

```

individual* makeEmptyInd(void) {
  individual *a;
  a = g_slice_new0(individual);
  global_individual_count++;
  return a;
}

void swapPop(GPtrArray **a, GPtrArray **b ) {
  GPtrArray *tmp;
  tmp = *a;
  *a = *b;
  *b = tmp;

}

void mutateIndV1(GRand *rnd, double mutateProb, double wcardProb, individual *myInd,
           GArray *garraysL[NUM_HTABLES],
           GArray *garraysC[NUM_HTABLES][SUBH] ) {
  // TODO: stub for code
  gint i; // loop counter
  guint tmpChrome[NUM_GENE]; // tempChrome created
  time_stamp myT; // temporary time stamp data
  IPAddr myIP; // temporary IP address
  guint mySlot; // random slot number picked

  gdouble myRandD;




  // Copy the chromosome into the tmpChrome
  g_memmove( tmpChrome, myInd->chrome, 4 * NUM_GENE);

  myT.tot = tmpChrome[G_DURATION];
  //  time_stamp myT; // temporary time stamp data
  // IPAddr myIP; // temporary IP address
  myT.byte[0] = 0xff; // not used, all should be -1

  // Hours, Minutes, Seconds
  for (i=1; i<4; i++) {
    myRandD = g_rand_double(rnd);
    if ( myRandD < mutateProb ) {
      mySlot = randslot(rnd, garraysC[G_DURATION][i]->len, wcardProb);
      myT.byte[i] = g_array_index (garraysC[G_DURATION][i], guchar, mySlot);
    }
  }

  tmpChrome[G_DURATION] = myT.tot; // duration

```

```
// Service
myRandD = g_rand_double(rnd);
if ( myRandD < mutateProb ) {
  mySlot = randslot(rnd, garraysL[G_SERVICE]->len, wcardProb);
  tmpChrome[G_SERVICE] = g_array_index (garraysL[G_SERVICE], guint, mySlot);
}

// Source Port
myRandD = g_rand_double(rnd);
if ( myRandD < mutateProb ) {
  mySlot = randslot(rnd, garraysL[G_SOURCE_PORT]->len, wcardProb);
  tmpChrome[G_SOURCE_PORT] = g_array_index (garraysL[G_SOURCE_PORT], guint, mySlot);
}

// Dest Port
myRandD = g_rand_double(rnd);
if ( myRandD < mutateProb ) {
  mySlot = randslot(rnd, garraysL[G_DEST_PORT]->len, wcardProb);
  tmpChrome[G_DEST_PORT] = g_array_index (garraysL[G_DEST_PORT], guint, mySlot);
}

// Source IP xxx.xxx.xxx.xxx
//            0   1   2   3
myIP.full = tmpChrome[G_SRC_IP];
for (i=0; i<4; i++) {
  myRandD = g_rand_double(rnd);
  if ( myRandD < mutateProb ) {
    mySlot = randslot(rnd, garraysC[G_SRC_IP][i]->len, wcardProb);
    myIP.octet[i] = g_array_index (garraysC[G_SRC_IP][i], guchar, mySlot);
  }
}
tmpChrome[G_SRC_IP] = myIP.full; // source IP

// Dest IP xxx.xxx.xxx.xxx
//          0   1   2   3
myIP.full = tmpChrome[G_DEST_IP];
for (i=0; i<4; i++) {
  myRandD = g_rand_double(rnd);
  if ( myRandD < mutateProb ) {
    mySlot = randslot(rnd, garraysC[G_DEST_IP][i]->len, wcardProb);
    myIP.octet[i] = g_array_index (garraysC[G_DEST_IP][i], guchar, mySlot);
  }
}
tmpChrome[G_DEST_IP] = myIP.full; // dest IP

```

```
  // Attack. Do not select wildcard, so put its probability at 0.
  myRandD = g_rand_double(rnd);
  if ( myRandD < mutateProb ) {
    mySlot = randslot(rnd, garraysL[G_ATTACK]->len, 0.0 );
    tmpChrome[G_ATTACK] = g_array_index (garraysL[G_ATTACK], guint, mySlot);
  }

  // Copy the temp chromosome into the individual
  g_memmove( myInd->chrome, tmpChrome, 4 * NUM_GENE);

}

./rand.h
#include "types.h"
extern gint global_individual_count;
void makeRandIndV1(GRand *rnd, individual **myInd);
void makeRandIndV2(GRand *rnd, double wcardProb, individual **myInd,
           GArray *garraysL[NUM_HTABLES],
           GArray *garraysC[NUM_HTABLES][SUBH] );
individual* makeEmptyInd();
void mutateIndV1(GRand *rnd, double mutateProb, double wcardProb, individual *myInd,
         GArray *garraysL[NUM_HTABLES],
         GArray *garraysC[NUM_HTABLES][SUBH] );
guint randslot(GRand *rnd, guint numUniqueP1, double wcardProb );
void swapPop(GPtrArray **a, GPtrArray **b) ;

./read_bsm.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <glib.h>
#include "types.h"
#include "print.h"
#include "read_bsm.h"
#include "service_attacks.h"

#define BUF_SZ 80

#ifndef SWAP_4
#define SWAP_4(x) ( ((x) << 24) | \
         (((x) << 8) & 0x00ff0000) | \
         (((x) >> 8) & 0x0000ff00) | \
         ((x) >> 24) )
#endif

void destroyL_key(gpointer foo) {
  g_slice_free1(sizeof(gint),foo);
}

```

```
void destroyL_value(gpointer foo) {
  g_slice_free1(sizeof(gint),foo);
}

/**
 * g_char_hash:
 * @v: a pointer to a #gchar key
 *
 * Converts a pointer to a #gchar to a hash value.
 * It can be passed to g_hash_table_new() as the @hash_func parameter,
 * when using pointers to integers values as keys in a #GHashTable.
 *
 * Returns: a hash value corresponding to the key.
 */
guint
g_char_hash (gconstpointer v)
{
  // Copy the value of unsigned char to unsigned int
  guchar a;
  guint b;
  a = *(const guchar*) v;
  b = a;
  return b;
}

gboolean
g_char_equal (gconstpointer v1,
        gconstpointer v2)
{
  return *((const guchar*) v1) == *((const guchar*) v2) ;
}

void
destroyC_key(gpointer foo) {
  g_slice_free1(sizeof(gchar),foo);
}


void updateL( GHashTable *uniqueL, GHashTable *uniqueC[], guint value ) {
  guint *c, *d; // Pointers to key and value for insertion
  gchar *cc;
  guint newcnt; // new count
  guint *lkey, *lvalue; // lookup key and lookup value
  //  gchar *lckey;
  guint initcnt = 1; // initial count
  IPAddr myUnion;
  int i;

  myUnion.full = value;

  c = (guint *)g_slice_alloc0(  sizeof(guint) );
```

```
  d = (guint *)g_slice_alloc0(  sizeof(guint) );
  g_memmove(c, &value, sizeof(gint));

  if ( g_hash_table_lookup_extended( uniqueL, c, (gpointer)&lkey,
                         (gpointer)&lvalue) ) {
    newcnt = *lvalue + 1;
    g_memmove(d, &newcnt,sizeof(guint));
  } else {
    g_memmove(d, &initcnt,sizeof(guint));
  }
  // FIXME: check to see that insert succeeded. If not, free c and d.
  g_hash_table_insert(uniqueL, c, d);

  for (i=0; i<SUBH; i++) {
    cc = (gchar *)g_slice_alloc0(  sizeof(gchar) );
    d = (guint *)g_slice_alloc0(  sizeof(guint) );
    g_memmove(cc, &(myUnion.octet[i]),sizeof(gchar));

    if ( g_hash_table_lookup_extended( uniqueC[i], cc, (gpointer)&lkey,
                           (gpointer)&lvalue) ) {
      newcnt = *lvalue + 1;
      g_memmove(d, &newcnt,sizeof(guint));
    } else {
      g_memmove(d, &initcnt,sizeof(guint));
    }
    g_hash_table_insert(uniqueC[i], cc, d);
  }

}

// foo - raw line for input string
// length - length of input string
// myind - pointer to individual to be created.
void make_toks(char *foo, int length, individual *myind, GHashTable *uniqueL[NUM_HTABLES], GHashTable *uniqueC[NUM_HTABLES][SUBH])
{
  time_stamp ts;
  IPAddr ip;
  char *svptr1, *svptr2;
  char delims[] = " ";
  char durdelim[] = ":";
  char ipdelim[] = ".";
  char *result = NULL;
  char *result2 = NULL;
  int idx = 0;
  int idx2 = 0;
  int i;
  int matched;
  int pv; // parsed value
  char tmp[BUF_SZ];
  gboolean getUnique;
```

```

init_attacks();

if (uniqueL != NULL && uniqueC != NULL )
  getUnique = TRUE;
else
  getUnique = FALSE;

foo[length - 1] = '\0';

result = strtok_r( foo, delims, &svptr1 );

while( result != NULL ) {
  switch (idx) {

  case F_DURATION: // duration
    ts.byte[0] = -1;
    idx2 = 1;
    strncpy(tmp, result, BUF_SZ);
    result2 = strtok_r( tmp , durdelim, &svptr2 );
    while( result2 != NULL ) {
ts.byte[idx2] = atoi(result2);
result2 = strtok_r(NULL, durdelim, &svptr2 );
idx2++;
    }
    myind->chrome[G_DURATION] = ts.tot;
    if (getUnique)
updateL(uniqueL[G_DURATION], uniqueC[G_DURATION], myind->chrome[G_DURATION]);
    break; // end of duration

  case F_SERVICE: // service
    for (i = 0; i < ENDP; i++) {
if ( strncmp(services[i], result, strlen( services[i] ) ) == 0 ) {
  myind->chrome[G_SERVICE] = i;
  if (getUnique)
    updateL(uniqueL[G_SERVICE], uniqueC[G_SERVICE], myind->chrome[G_SERVICE]);
}
    }
    break;// end of service

  case F_SOURCE_PORT: // source port
    pv = atoi(result);
    myind->chrome[G_SOURCE_PORT] = pv;
    if (getUnique)
updateL(uniqueL[G_SOURCE_PORT], uniqueC[G_SOURCE_PORT], myind->chrome[G_SOURCE_PORT]);
    break; // end of source port

  case F_DEST_PORT: // destination port
    myind->chrome[G_DEST_PORT] = atoi(result);
```

```
      if (getUnique)
  updateL(uniqueL[G_DEST_PORT], uniqueC[G_DEST_PORT], myind->chrome[G_DEST_PORT]);
      break; // end of destination port

    case F_SRC_IP: // Source IP
      idx2 = 0;
      strncpy(tmp, result, BUF_SZ); // copy up to the size of the target buffer
      result2 = strtok_r( tmp , ipdelim, &svptr2 );
      while( result2 != NULL ) {
  ip.octet[idx2] = atoi(result2);
  result2 = strtok_r(NULL, ipdelim, &svptr2 );
  idx2++;
      }
      myind->chrome[G_SRC_IP] = ip.full;
      if (getUnique)
  updateL(uniqueL[G_SRC_IP], uniqueC[G_SRC_IP], myind->chrome[G_SRC_IP]);
      break; // end of Source IP

    case F_DEST_IP: // Dest IP
      idx2 = 0;
      strncpy(tmp, result, BUF_SZ); // copy up to the size of the target buffer
      result2 = strtok_r( tmp , ipdelim, &svptr2 );
      while( result2 != NULL ) {
  ip.octet[idx2] = atoi(result2);
  result2 = strtok_r(NULL, ipdelim, &svptr2 );
  idx2++;
      }
      myind->chrome[G_DEST_IP] = ip.full;
      if (getUnique)
  updateL(uniqueL[G_DEST_IP], uniqueC[G_DEST_IP], myind->chrome[G_DEST_IP]);
      break; // End of Dest IP

    case F_ATTACK: // attack name
      // FIXME : Is -1 good for this result?
      myind->chrome[G_ATTACK] = NONE;
      for ( i = 1; i < END_A; i++) {
  if ( strncmp(attacks[i], result, strlen( attacks[i] ) ) == 0 ) {
    matched = 1;
    myind->chrome[G_ATTACK] = i;
    if (getUnique)
      updateL(uniqueL[G_ATTACK], uniqueC[G_ATTACK], myind->chrome[G_ATTACK]);
  }
      }
      break; // end of attack

    default:
      // Do Nothing
```

```
      ;
    }
      //g_printf("|");
    idx++;
    result = strtok_r( NULL, delims, &svptr1 );
  }
    //g_printf("\n");

}


int load_audit(GSList **auditList, char *myfile )
{

  return load_audit_unique(auditList, myfile, NULL, NULL);
}


gint load_audit_unique(GSList **auditList, char *myfile, GHashTable *uniqueL[NUM_HTABLES], GHashTable *uniqueC[NUM_HTABLES][SUBH])
{
  FILE *input; // input file
  char *rd_buf; // buffer for reading input
  int nchars; // number of characters read
  gint nRecords = 0;
  size_t cur_sz = BUF_SZ;
  individual *myInd;

  // Empty the list
  if (*auditList != NULL)
    {
      g_slist_free(*auditList);
      *auditList = NULL;
    }

  // Open the file
  //g_printf("file to open %s\n",myfile);

  input = fopen(myfile, "r");
  if (input == NULL)
    {
      perror("Failed to open file");
      exit(-1);
    }
  // buffer for reading input
  rd_buf = (char *) malloc( BUF_SZ * sizeof(char) );

  nchars = getline(&rd_buf, &cur_sz, input);

  while (nchars != -1) {
    myInd = g_slice_new0(individual);
    global_individual_count++;

```

```
    if ( rd_buf[nchars-1] == '\n')
      rd_buf[nchars-1] = '\0'; // get rid of newline. Probably won't work on PC though \n\r
    // string description
    g_snprintf(myInd->desc, DESC_SZ, "%s",rd_buf );
    // g_printf("Read this %s\n",rd_buf);

    make_toks(rd_buf, nchars, myInd, uniqueL, uniqueC);
    *auditList = g_slist_append(*auditList, myInd);
    nRecords++;
    nchars = getline(&rd_buf, &cur_sz, input);
  }

  free(rd_buf);


  if (fclose(input) != 0) {
    perror("Failed to close");
    exit(-1);
  }
  return nRecords;

}

void copyeachL(gpointer a, gpointer b, gpointer userdata) {
  GArray * myArray;
  myArray = (GArray *)userdata;
  g_array_append_val( myArray, *(guint *)a);
}

void copyeachC(gpointer a, gpointer b, gpointer userdata) {
  GArray * myArray;
  myArray = (GArray *)userdata;
  g_array_append_val( myArray, *(guchar *)a);
}


int build_audit_array(GSList **auditList, GArray *arrayL[NUM_HTABLES],
              GArray *arrayC[NUM_HTABLES][SUBH],
              char *myfile) {

  gint nRecords = 0; // Number of records in audit data
  gint i,j; // loop variables
  gint neg1 = -1;
  //  GArray *tmpArray;
  GHashTable *myHTableL[NUM_HTABLES];
  GHashTable *myHTableC[NUM_HTABLES][SUBH];

  // Initialize each array and put wildcard at the beginning

  for (i=0; i< NUM_HTABLES;i++) {
    //tmpArray = g_array_new(FALSE, FALSE, sizeof(guint));
```

```
    arrayL[i] = g_array_new(FALSE, FALSE, sizeof(guint));
    g_array_append_val (arrayL[i], neg1 );
    // use memcpy here instead!
    //arrayL[i] = tmpArray;
  }

  for (i=0; i< NUM_HTABLES;i++)
    for (j=0; j<SUBH; j++) {
      //tmpArray = g_array_new(FALSE, FALSE, sizeof(guchar));
      arrayC[i][j] = g_array_new(FALSE, FALSE, sizeof(guchar));
      g_array_append_val (arrayC[i][j], neg1 );
      // use memcpy here instead!
      //arrayC[i][j] = tmpArray;
    }

  for (i=0; i< NUM_HTABLES;i++)
    myHTableL[i] = g_hash_table_new_full(g_int_hash,g_int_equal,
                           (GDestroyNotify)destroyL_key,
                           (GDestroyNotify)destroyL_value);

  for (i=0; i< NUM_HTABLES;i++)
    for (j=0; j<SUBH; j++)
      myHTableC[i][j] =
g_hash_table_new_full(g_char_hash,g_char_equal,
                                (GDestroyNotify)destroyC_key,
                                (GDestroyNotify)destroyL_value);

  nRecords = load_audit_unique(auditList, myfile, myHTableL,
myHTableC);

  // Load the data into arrays

  // Duration
  // -1, Hours, Min, Sec
  //  0    1     2    3
  for (i=0; i< SUBH; i++)
    g_hash_table_foreach(myHTableC[G_DURATION][i], copyeachC,
               arrayC[G_DURATION][i]);

  // Service
  g_hash_table_foreach(myHTableL[G_SERVICE], copyeachL,
               arrayL[G_SERVICE]);


  // Source Port
  g_hash_table_foreach(myHTableL[G_SOURCE_PORT], copyeachL,
               arrayL[G_SOURCE_PORT]);

  // Dest Port
  g_hash_table_foreach(myHTableL[G_DEST_PORT], copyeachL,
               arrayL[G_DEST_PORT]);

  // Source IP Address
```

```
  // xxx, xxx, xxx, xxx
  //  0    1    2    3
  for (i=0; i< SUBH; i++)
    g_hash_table_foreach(myHTableC[G_SRC_IP][i], copyeachC,
               arrayC[G_SRC_IP][i]);

  // Dest IP Address
  // xxx, xxx, xxx, xxx
  //  0    1    2    3
  for (i=0; i< SUBH; i++)
    g_hash_table_foreach(myHTableC[G_DEST_IP][i], copyeachC,
               arrayC[G_DEST_IP][i]);

  // Attack
  g_hash_table_foreach(myHTableL[G_ATTACK], copyeachL,
               arrayL[G_ATTACK]);


  // free Hash tables. Don't want to leak memory
  for (i=0; i< NUM_HTABLES;i++)
    g_hash_table_destroy(myHTableL[i]);

  for (i=0; i< NUM_HTABLES;i++)
    for (j=0; j<SUBH; j++)
      g_hash_table_destroy(myHTableC[i][j]);

  return nRecords;
}

./read_bsm.h
#ifndef READ_BSM_H
#define READ_BSM_H
extern gint global_individual_count;
void make_toks(char *foo, int length, individual *myind, GHashTable
*uniqueL[NUM_HTABLES], GHashTable *uniqueC[NUM_HTABLES][SUBH]);
int load_audit(GSList **auditList, char *myfile);
int load_audit_unique(GSList **auditList, char *myfile, GHashTable
*uniqueL[NUM_HTABLES], GHashTable *uniqueC[NUM_HTABLES][SUBH]);
void destroyL_key(gpointer foo);
void destroyL_value(gpointer foo);
guint g_char_hash (gconstpointer v);
gboolean g_char_equal (gconstpointer v1, gconstpointer v2);
void destroyC_key(gpointer foo);
gint build_audit_array(GSList **auditList, GArray
*arrayL[NUM_HTABLES] ,
               GArray *arrayC[NUM_HTABLES][SUBH] ,
              char *myfile);
#endif


./service_attacks.c
#include <string.h>
#include <glib.h>
```

```
#include "service_attacks.h"


// Service list - static array

char services[10][40] =
{"exec","finger","ftp","rlogin","rsh","smtp","telnet","endp"};

// Attack lists - static array

// This array has to match the actual strings in the bsm.list file
char attacks[END_A][255];

gint global_individual_count=0;

void init_attacks() {
  strcpy(attacks[NONE],"none");
  strcpy(attacks[GUESS_A],"guess");
  strcpy(attacks[PORT_SCAN_A],"port-scan");
  strcpy(attacks[RCP_A],"rcp");
  strcpy(attacks[RLOGIN_A],"rlogin");
  strcpy(attacks[RSH_A],"rlogin");
  strcpy(attacks[FORMAT_CLEAR_A],"format_clear");
  strcpy(attacks[FFB_CLEAR_A],"ffb_clear");
  strcpy(attacks[END_A],"end");
}




./service_attacks.h
#ifndef SERVICE_ATTACKS_H
#define SERVICE_ATTACKS_H

#define NUM_GENE 7

enum FILE_GENE_IDX{F_DURATION=3, F_SERVICE=4, F_SOURCE_PORT=5,
F_DEST_PORT=6,
           F_SRC_IP=7, F_DEST_IP=8, F_ATTACK=10};

enum ARY_GENE_IDX{G_DURATION=0, G_SERVICE=1, G_SOURCE_PORT=2,
G_DEST_PORT=3,
          G_SRC_IP=4, G_DEST_IP=5, G_ATTACK=6};

enum
SERVICE{EXEC=0,FINGER=1,FTP=2,RLOGIN=3,RSH=4,SMTP=5,TELNET=6,ENDP=7};

enum ATTACK{NONE=0,GUESS_A=1, PORT_SCAN_A=2, RCP_A=3, RLOGIN_A=4,
RSH_A=5,
      FORMAT_CLEAR_A=6, FFB_CLEAR_A=7, END_A=8};

extern char services[10][40];
extern char attacks[END_A][255];
```

```

void init_attacks();
#endif
```

./types.h

```
#ifndef TYPES_H

#define TYPES_H

#define DESC_SZ 91

#define NUM_HTABLES 9 // number of main genes
#define SUBH 4 // number of sub-elements per chromosome

typedef struct
{
  char desc[DESC_SZ];
  int chrome[7]; // Attack is the last element
  double fitness;
} individual;

typedef union {
  char byte[4];
  unsigned int     tot;
} time_stamp;

typedef union {
  char octet[4];
  unsigned int full;
} IPAddr;

//enum attack{NEPTUNE=1,PASSWD_GUESS=2};
//enum protocol{FINGER=2,TELNET=3};



#endif
```

./Makefile

```

TEST_WHICHBYTE_OBJS := test_crossbyte.o compare.o

test_crossbyte:       $(TEST_WHICHBYTE_OBJS)
  gcc $(LDFLAGS) $(TEST_WHICHBYTE_OBJS) -o $@

TEST_GET_COUNTS_OBJS := test_get_counts.o read_bsm.o print.o service_attacks.o  rand.o compare.o

test_get_counts:       $(TEST_GET_COUNTS_OBJS)
  gcc $(LDFLAGS) $(TEST_GET_COUNTS_OBJS) -o $@

TEST_GET_MAG_OBJS := test_get_mag.o read_bsm.o print.o service_attacks.o rand.o compare.o
```

```

test_get_mag:       $(TEST_GET_MAG_OBJS)
  echo "Test for Magnitude of |A and B| and |A|" ; \
  gcc $(LDFLAGS) $(TEST_GET_MAG_OBJS) -o $@

TEST_FITNESS_OBJS := test_fitness.o read_bsm.o print.o service_attacks.o rand.o compare.o

test_fitness:       $(TEST_FITNESS_OBJS)
  echo "Test fitness" ; \
  gcc $(LDFLAGS) $(TEST_FITNESS_OBJS) -o $@

TEST_POPULATION_OBJS := test_population.o read_bsm.o print.o service_attacks.o  rand.o compare.o

test_population:       $(TEST_POPULATION_OBJS)
  gcc $(LDFLAGS) $(TEST_POPULATION_OBJS) -o $@


test_read_bsm.o: test_read_bsm.c


read_bsm.o: read_bsm.c read_bsm.h types.h print.h service_attacks.h
  gcc $(CFLAGS) read_bsm.c

TEST_RAND_OBS := test_rand.o rand.o print.o service_attacks.o

test_rand: $(TEST_RAND_OBS)
  gcc $(LDFLAGS) $(TEST_RAND_OBS) -o test_rand

test_rand.o: test_rand.c types.h print.h service_attacks.h rand.h
  gcc $(CFLAGS) test_rand.c

rand.o: rand.c rand.h types.h print.h service_attacks.h


testList: testList.o

testprec: testprec.o

TEST_COMPARE_OBJS := test_compare.o compare.o print.o service_attacks.o

test_compare: $(TEST_COMPARE_OBJS)
  gcc $(LDFLAGS) $(TEST_COMPARE_OBJS) -o $@

TEST_CROSSOVER_OBJS := test_crossover.o compare.o print.o service_attacks.o

test_crossover: $(TEST_CROSSOVER_OBJS)
  gcc $(LDFLAGS) $(TEST_CROSSOVER_OBJS) -o $@

TEST_MUTATE_OBJS := test_mutate.o read_bsm.o compare.o print.o
```

```
service_attacks.o rand.o

test_mutate: $(TEST_MUTATE_OBJS)
  gcc $(LDFLAGS) $(TEST_MUTATE_OBJS) -o $@


TEST_ARRAY_PTR_OBS := rand.o test_array_ptr.o print.o

test_array_ptr: $(TEST_ARRAY_PTR_OBS)
  gcc $(LDFLAGS) $(TEST_ARRAY_PTR_OBS) -o test_array_ptr


compare.o: compare.c compare.h

print.o: print.h print.c types.h service_attacks.h


service_attacks.o: service_attacks.h


test_compare.o: test_compare.c

TEST_GA2_OBJS :=  read_bsm.o print.o service_attacks.o rand.o compare.o

test_ga2:  test_ga2.o     $(TEST_GA2_OBJS)
  gcc $(LDFLAGS) test_ga2.o $(TEST_GA2_OBJS) -o $@

test_ga3:  test_ga3.o     $(TEST_GA2_OBJS)
  gcc $(LDFLAGS) test_ga3.o $(TEST_GA2_OBJS) -o $@

netga:  netga.o     $(TEST_GA2_OBJS)
  gcc $(LDFLAGS) netga.o $(TEST_GA2_OBJS) -o $@


clean:
  rm -f *.o test_read_bsm test_read_bsm2 test_read_bsm3 test_read_bsm4  \
        test_compare test_rand test_fitness test_get_counts \
  test_get_mag test_array_ptr test_crossover test_mutate \
  test_population test_ga test_ga2 test_ga3 test_crossbyte core \
  netga *~

.c.o:
  gcc $(CFLAGS) -o $*.o $<

```

nProbe Plug-in

```
./nprobe.diff
index 22ff2f7..da956dd 100644
--- a/plugin.c
+++ b/plugin.c
@@ -46,6 +46,7 @@ extern PluginInfo* smtpPluginEntryFctn(void);
 #else
 static char *pluginDirs[] = { "./plugins",
                    "/usr/local/lib/nprobe/plugins",
+                         "/usr/local/nprobe/lib/nprobe/plugins",
                    NULL };
 #endif

diff --git a/plugins/Makefile.am b/plugins/Makefile.am
index 909495b..2001aa3 100644
--- a/plugins/Makefile.am
+++ b/plugins/Makefile.am
@@ -50,7 +50,8 @@ noinst_PROGRAMS = \
                   sipPlugin.so \
                   rtpPlugin.so \
                   dumpPlugin.so \
-         l7Plugin.so
+         l7Plugin.so \
+         netGAPlugin.so

 lib_LTLIBRARIES = \
                   libdbPlugin.la \
@@ -60,7 +61,25 @@ lib_LTLIBRARIES = \
                   libsipPlugin.la \
                   librtpPlugin.la \
                   libdumpPlugin.la \
-         libl7Plugin.la
+         libl7Plugin.la \
+         libnetGAPlugin.la
+
+######################################################
+
+libnetGAPlugin_la_SOURCES = netGAPlugin.c
+libnetGAPlugin_la_LDFLAGS = -shared -release @PACKAGE_VERSION@ @DYN_FLAGS@
+libnetGAPlugin_la_CFLAGS = $(AM_CFLAGS)
+
+.libs/libnetGAPlugin.so@SO_VERSION_PATCH@:
+ @if test -f libnetGAPlugin_la-netGAPlugin.o; then \
+    $(CC) @MAKE_SHARED_LIBRARY_PARM@ -o .libs/libnetGAPlugin.so@SO_VERSION_PATCH@ libnetGAPlugin_la-netGAPlugin.o; \
+    else \
+    $(CC) @MAKE_SHARED_LIBRARY_PARM@ -o .libs/libnetGAPlugin.so@SO_VERSION_PATCH@ netGAPlugin.o; \
+    fi
+
```

```
+netGAPlugin.so$(EXEEXT): .libs/libnetGAPlugin.so@SO_VERSION_PATCH@
+ @$(LN_S) .libs/libnetGAPlugin.so netGAPlugin.so$(EXEEXT)
+

 ######################################################

```

plugins/netGAPlugin.c

```
         ((x) >> 24) )
#else
#define ptohs(x) *(u_int16_t *)(x)
#define ptohl(x) *(u_int32 *)(x)
#endif

#define FALSE 0
#define TRUE 1


typedef union {
  char octet[4];
  unsigned int full;
} IPAddr;


typedef struct {
  int dur_h;
  int dur_m;
  int dur_s;
  char protocol[16];
  int src_port;
  int dst_port;
  int srcIP[4];
  int dstIP[4];
  char attack[16];
} record1;

typedef struct {
  int rulenum;
  float fitness;
} record2;

struct record3 {
  struct record3 *next;
  record1 r;
  record2 s;
};

#define NETGA_OPT "--netGA"

static V9V10TemplateElementId netGAPlugin_template[] = {
  /* Nothing to export into a template for now */
  { FLOW_TEMPLATE, NTOP_ENTERPRISE_ID, 0, 0, 0, 0, NULL, NULL }
};
```

```

static PluginInfo netGAPlugin; /* Forward */
static void* check_connections_loop(void *notUsed);
static pthread_mutex_t check_connections_mutex;
static pthread_t check_connections_thread;
static void GAwalkHash(u_int32_t hash_idx, struct record3 *rule);

void read_record2(record2 *r,  FILE *fp) {
  fscanf(fp,"%d", &r->rulenum);
  fscanf(fp,"%*s");
  fscanf(fp,"%*s");
  fscanf(fp,"%f",&r->fitness);
  //printf("%d fitness is %g\n",r->rulenum,r->fitness);

}

int read_record1(record1 *r, FILE *fp) {
  fscanf(fp,"%d",&r->dur_h);
  if (r->dur_h == -2)
    return 1;

  fscanf(fp,"%*c");
  fscanf(fp,"%d",&r->dur_m);
  fscanf(fp,"%*c");
  fscanf(fp,"%d",&r->dur_s);
  //printf("%d,%d,%d\n",r->dur_h,r->dur_m,r->dur_s);
  fscanf(fp,"%15s",r->protocol);

  //  printf("|%s|\n", r->protocol);
  fscanf(fp,"%d",&r->src_port);
  fscanf(fp,"%d",&r->dst_port);

  fscanf(fp,"%d",&r->srcIP[0]);
  fscanf(fp,"%*c");
  fscanf(fp,"%d",&r->srcIP[1]);
  fscanf(fp,"%*c");
  fscanf(fp,"%d",&r->srcIP[2]);
  fscanf(fp,"%*c");
  fscanf(fp,"%d",&r->srcIP[3]);

  fscanf(fp,"%d",&r->dstIP[0]);
  fscanf(fp,"%*c");
  fscanf(fp,"%d",&r->dstIP[1]);
  fscanf(fp,"%*c");
  fscanf(fp,"%d",&r->dstIP[2]);
  fscanf(fp,"%*c");
  fscanf(fp,"%d",&r->dstIP[3]);
  //  printf("%d.%d.%d.%d %d.%d.%d.%d\n",r->srcIP[0],r->srcIP[1],r->srcIP[2],
  //     r->srcIP[3], r->dstIP[0], r->dstIP[1], r->dstIP[2], r->dstIP[3]);

  fscanf(fp,"%15s", r->attack);
```

```
  //printf("|%s|\n", r->attack);


  return 0;
}

void netGAPlugin_init(int argc, char *argv[]) {
  int i;
  struct record3 *zrule;
  struct record3 *list = NULL;
  struct record3 *tmp;
  struct record3 *head = NULL;
  FILE *fp;
  char *arg = NULL;

  traceEvent(TRACE_INFO, "netGA plugin init");

  // Sample rule from evaluation data
  // 0,0,23 telnet 1884 23 192.168.1.30 192.168.0.20 guess
  //  fitness 0.8031

  // Adjust rule for shorter duration
  // 0,0,23 telnet 1884 23 192.168.1.30 192.168.0.20 guess
  //  fitness 0.8031

  zrule = (struct record3 *) malloc( sizeof(struct record3) );
  memset(zrule, 0x00, sizeof(struct record3) );

  zrule->r.dur_h = 0;
  zrule->r.dur_m = 0;
  zrule->r.dur_s = 5;
  strncpy(zrule->r.protocol,"telnet",16);
  zrule->r.src_port = -1;
  zrule->r.dst_port = 23;

  zrule->r.srcIP[0] = 192;
  zrule->r.srcIP[1] = 168;
  zrule->r.srcIP[2] = 1;
  zrule->r.srcIP[3] = 30;

  zrule->r.dstIP[0] = 192;
  zrule->r.dstIP[1] = 168;
  zrule->r.dstIP[2] = 0;
  zrule->r.dstIP[3] = 20;

  strncpy(zrule->r.attack,"guess",16);
  zrule->next = NULL;
  // FIXME - free memory for record3

  if((argc == 2) && (argv[1][0] != '-')) {
    traceEvent(TRACE_INFO, "Initializing netGA plugin\n argv[0] %s
argv[1] %s argv[2] %s\n",argv[0], argv[1], argv[2] );
    FILE * fd;
```

```
  char   line[256];

  fd = fopen(argv[1], "r");
  if(fd == NULL) {
    traceEvent(TRACE_ERROR, "Unable to read config. file %s", argv[1]);
    fclose(fd);
    return;
  }

  while(fgets(line, sizeof(line), fd)) {
    char * p = NULL;

    if(strncmp(line, NETGA_OPT, strlen(NETGA_OPT)) == 0) {
int sz = strlen(line)+2;
arg = malloc(sz);
if(arg == NULL) {
  traceEvent(TRACE_ERROR, "Not enough memory?");
  fclose(fd);
  return;
}
p = strchr(line,'\n');
if(p) *p='\0';
p = strchr(line,'=');
snprintf(arg, sz, "%s", p+1);

    }
  }

  fclose(fd);
} else {
  for(i=0; i<argc; i++)

    if(strncmp(argv[i], NETGA_OPT, strlen(NETGA_OPT)) == 0) {
char *netga_arg = argv[i+1];
int sz = strlen(netga_arg)+2;

if(argv[i][strlen(NETGA_OPT)] == '=') {
  netga_arg = &argv[i][strlen(NETGA_OPT)+1];
} else
  netga_arg = argv[i+1];

if(netga_arg == NULL) {
  traceEvent(TRACE_ERROR, "Bad format specified for --netGA parameter");
  return;
}

sz = strlen(netga_arg)+2;

arg = malloc(sz);
if(arg == NULL) {
  traceEvent(TRACE_ERROR, "Not enough memory?");
```

```
   return;
  }

  snprintf(arg, sz, "%s", netga_arg);
    }
  }

  traceEvent(TRACE_INFO, "netGA %s \n", arg );
  // Initialize check thread

  fp = fopen(arg, "r");
  if (fp == NULL) {
    traceEvent(TRACE_ERROR, "Failed to open rules file disabling %s \n", arg );
    return;
  }

  i = 0;
  for (;;) {
   tmp = (struct record3 *) malloc(sizeof (struct record3));
   if(read_record1(&tmp->r,fp) != 0)
    {
  free(tmp);
  break;
    }
   read_record2(&tmp->s,fp);
   if (tmp->s.fitness < 0.0001) {
    free(tmp);
    continue;
   }
   tmp->next = NULL;
   if (list == NULL)
    head = list = tmp;
   else {
    list->next = tmp;
    list = tmp;
   }
   i++;
  }
  fclose(fp);

  if (i==0) {
   traceEvent(TRACE_ERROR, "Unable to parse any rules from %s\n", arg );
   return;
  } else
   traceEvent(TRACE_INFO, "parsed %d rules from %s\n",i, arg );

  i = 0;
  for (tmp = head;tmp != NULL; tmp = tmp->next) {
   // check rule against pool
   traceEvent(TRACE_INFO, "Rule %d protocol %s\n",i,tmp->r.protocol);
```

```
    i++;
  }

  // use zrule for hard coded testing
  pthread_create(&check_connections_thread, NULL,
check_connections_loop, head);
}

/* ********************************************** */

static V9V10TemplateElementId* netGAPlugin_conf(void) {
  traceEvent(TRACE_INFO, "netGA template configured");
  return(netGAPlugin_template);
}

static V9V10TemplateElementId* netGAPlugin_get_template(char*
template_name) {
  int i;

  for(i=0; netGAPlugin_template[i].templateElementId != 0; i++) {
    if(!strcmp(template_name,
netGAPlugin_template[i].templateElementName)) {
      return(&netGAPlugin_template[i]);
    }
  }

  return(NULL); /* Unknown */
}

/* ********************************************** */

static int netGAPlugin_export(void *pluginData,
V9V10TemplateElementId *theTemplate, int direction,
                  FlowHashBucket *bkt, char *outBuffer,
                  u_int* outBufferBegin, u_int* outBufferMax) {

  // traceEvent(TRACE_ERROR, " +++ dbPlugin_export()");

  return(-1); /* Not handled */
}

/* ********************************************** */

static int netGAPlugin_print(void *pluginData,
V9V10TemplateElementId *theTemplate, int direction,
              FlowHashBucket *bkt, char *line_buffer, u_int
line_buffer_len) {
  return(-1); /* Not handled */
}


static void netGAPlugin_help(void) {
  printf("  --netGA=<rules file> \n");
```

```

}

/* Plugin entrypoint */
static PluginInfo netGAPlugin = {
  NPROBE_REVISION,
  "NetGA",
  "0.1",
  "Genetic Algorithm rule matcher",
  "Brian E. Lavender",
  1 /*  always enabled */, 1, /* enabled */
  netGAPlugin_init,
  NULL, /* Term */
  netGAPlugin_conf,
  NULL,
  0, /* call packetFlowFctn for each packet */
  NULL,
  netGAPlugin_get_template,
  netGAPlugin_export,
  netGAPlugin_print,
  NULL,
  netGAPlugin_help
};

/* ************************************************ */

/* Plugin entry fctn */
#ifdef MAKE_STATIC_PLUGINS
PluginInfo* netGAPluginEntryFctn(void)
#else
     PluginInfo* PluginEntryFctn(void)
#endif
{
  return(&netGAPlugin);
}


// return match variable
// 0 - no match
// 1 - match
int compare_a(FlowHashBucket *myBucket, struct record3 *zrule,
time_t tNow) {

  int match ;
  int i, j;
  char buf1[256] = { 0 };

  int totDurationSecs;
  int durationHours;
  int durationMinutes;
  int durationSeconds;

  IPAddr tmpIPSrcAudit,  tmpIPDstAudit;
```

```

  // assume we have a match
  match = TRUE;


  totDurationSecs = (myBucket->flowTimers).firstSeenSent.tv_sec != 0 ? tNow - (myBucket->flowTimers).firstSeenSent.tv_sec : 0;
  durationSeconds = totDurationSecs % 60;
  durationMinutes = (totDurationSecs % 3600) / 60;
  durationHours = totDurationSecs / 3600;

  if ( !
       ( zrule->r.dur_h == -1 || zrule->r.dur_h == durationHours )
       )
    match = FALSE;
  if
    ( !
      ( zrule->r.dur_m == -1 || zrule->r.dur_m == durationMinutes )
      )
    match = FALSE;
  if
    ( !
      ( zrule->r.dur_s == -1 || zrule->r.dur_s == durationSeconds )
      )
    match = FALSE;


  tmpIPSrcAudit.full = ptohl(myBucket->src->host.ipType.ipv4);
  for (j = 0; j<4; j++) {
    // We want to see if it doesn't match.
    if ( !
   ( zrule->r.srcIP[j] == -1 || (unsigned char)zrule->r.srcIP[j] == (unsigned char)tmpIPSrcAudit.octet[j] )
   )
      match = FALSE;
  }


  tmpIPDstAudit.full = ptohl(myBucket->dst->host.ipType.ipv4);
  for (j = 0; j<4; j++) {
    // We want to see if it doesn't match.
    if ( !
   ( zrule->r.dstIP[j] == -1 || (unsigned char)zrule->r.dstIP[j] == (unsigned char)tmpIPDstAudit.octet[j] )
   )
      match = FALSE;
  }


  if ( !
       ( zrule->r.src_port == -1 || zrule->r.src_port ==  myBucket->sport )
       )
```

```
    match = FALSE;

  if ( !
       ( zrule->r.dst_port == -1 || zrule->r.dst_port ==  myBucket->dport )
       )
    match = FALSE;

  // g_printf("chrome %d match is %d\n",i,match);

  if (match == TRUE) {
    printf("duration hours %d minutes %d seconds %d\n", durationHours,
     durationMinutes, durationSeconds);
    printf("rule hours %d minutes %d seconds %d\n", zrule->r.dur_h,
     zrule->r.dur_m, zrule->r.dur_s);

    printf("Src Rule IP %u.%u.%u.%u\n", zrule->r.srcIP[0],
     zrule->r.srcIP[1],
     zrule->r.srcIP[2],
     zrule->r.srcIP[3]);
    /* printf("Src Test IP %d.%d.%d.%d\n",tmpIPSrcAudit.octet[0], */
    /*     tmpIPSrcAudit.octet[1], tmpIPSrcAudit.octet[2], */
    /*     tmpIPSrcAudit.octet[3]); */
    printf("Src Test IP %s\n",_intoa(myBucket->src->host, buf1, sizeof(buf1)) );

    printf("Dst Rule IP %d.%d.%d.%d\n", zrule->r.dstIP[0], zrule->r.dstIP[1],
     zrule->r.dstIP[2], zrule->r.dstIP[3] );
    printf("Dst Test IP %s\n",_intoa(myBucket->dst->host, buf1, sizeof(buf1)) );
    printf("Src Rule Port %d\n", zrule->r.src_port);
    printf("Src Test Port %u\n", myBucket->sport);
    printf("Dst Rule Port %d\n", zrule->r.dst_port);
    printf("Dst Test Port %u\n", myBucket->dport);
  }


  return match;
}


static void* check_connections_loop(void* rule_arg) {
  struct record3 *zrule;
  struct record3 *tmp;
  long idx;

  zrule = (struct record3 *) rule_arg;
  /* Wait until all the data structures have been allocated */
  while(readWriteGlobals->theFlowHash[readOnlyGlobals.numPcapThreads-1] == NULL) ntop_sleep(1);
```

```

  while ((readWriteGlobals->shutdownInProgress == 0)
    && (readWriteGlobals->stopPacketCapture == 0)) {

    for(idx=0; idx<readOnlyGlobals.numPcapThreads; idx++) {
      //traceEvent(TRACE_INFO, "Check pool doogy pcap %d", idx );

      for (tmp = zrule;tmp != NULL; tmp = tmp->next) {
  // check rule against pool
  GAwalkHash(idx, tmp);
      }

    }

    // Check pool against set of rules here.
    // Report any matches
    ntop_sleep(1); // sleep 1 second between checks.
  }
  return(NULL);
}

static void GAwalkHash(u_int32_t hash_idx, struct record3 *zrule) {
  u_int walkIndex, mutex_idx = 0, old_mutex_idx = mutex_idx+1; /* This to make sure that we lock
                                              the mutex at the first run */
  char buf1[256] = { 0 };
  char buf2[256] = { 0 };
  FlowHashBucket *myPrevBucket, *myBucket;
  time_t now = time(NULL);
  u_int num_lock = 0, num_unlock = 0;
  IPAddr myIP, tmpIPtest;
  time_t tNow;


#ifdef DEBUG_EXPORT
  printf("Begin walkHash(%d)\n", hash_idx);
#endif

  tNow = time(NULL);

  for(walkIndex=0; walkIndex < readOnlyGlobals.flowHashSize; walkIndex++) {
    /* traceEvent(TRACE_INFO, "walkHash(%d)", walkIndex); */

    old_mutex_idx = mutex_idx;
    mutex_idx = walkIndex % MAX_HASH_MUTEXES;

    if(!readOnlyGlobals.rebuild_hash) {
      if(mutex_idx != old_mutex_idx)
  pthread_rwlock_wrlock(&readWriteGlobals->flowHashRwLock[hash_idx][mutex_idx]), num_lock++;
    } else {
```

```
      if(readWriteGlobals->thePrevFlowHash == NULL) return; /* Too early */
    }

    myPrevBucket = NULL;

    if((!readOnlyGlobals.rebuild_hash)
       || readOnlyGlobals.pcapFile /* We're reading from a dump file */)
      myBucket = readWriteGlobals->theFlowHash[hash_idx][walkIndex];
    else
      myBucket = readWriteGlobals->thePrevFlowHash[hash_idx][walkIndex];

     while(myBucket != NULL) {
#ifdef ENABLE_MAGIC
      if(myBucket->magic != 67) {
  printf("Error (2): magic error detected (magic=%d)\n", myBucket->magic);
      }
#endif

      if(readWriteGlobals->shutdownInProgress) {
  if(!readOnlyGlobals.rebuild_hash) {
    pthread_rwlock_unlock(&readWriteGlobals->flowHashRwLock[hash_idx][mutex_idx]);
    return;
  }
      }

  /* *** Do something with myBucket *** */

      if ( myBucket->proto == 6) // TCP connection
  {
    printf("NetGA TCP connection %s:%d->%s:%d\n",_intoa(myBucket->src->host, buf1, sizeof(buf1)), myBucket->sport,
         _intoa(myBucket->dst->host, buf2, sizeof(buf2)), myBucket->dport );
    // Is this an IPv4 connection?
    if ( myBucket->src->host.ipVersion == 4 ) {
      //printf("NetGA IPv4 connection\n");

      if ( compare_a(myBucket, zrule, tNow ) == TRUE )
        printf("Match <<<------------------------------>>>\n");

    }

  }

  /* Move to the next bucket */
  myPrevBucket = myBucket;
  myBucket = myBucket->next;
```

```
  } /* while */

  if(!readOnlyGlobals.rebuild_hash) {
   if(mutex_idx != old_mutex_idx) {
 pthread_rwlock_unlock(&readWriteGlobals->flowHashRwLock[hash_idx][mutex_idx]), num_unlock++;
   }
  }
 } /* for */
}
```

REFERENCES

OSSIM: Open Source Security Information Management, http://www.ossim.net (accessed in November 9, 2010).

HARPER: Harper, Patrick, Snort, Apache, SSL, PHP, MySQL, and BASE Install on CentOS 4, RHEL 4 or Fedora Core, http://www.internetsecurityguru.com/documents/Snort_Base_Minimal.pdf (accessed in 2005).

WUFTP: WU-FTPD Development Group, http://wu-ftpd.therockgarden.ca/ (accessed in November 9, 2010).

SECURITYFOCUS: Security Focus, http://www.securityfocus.com/ (accessed in November 9, 2010).

SNORT: SNORT, http://www.snort.org (accessed in November 9, 2010).

GOOGLE: Google, Google, http://www.google.com/ (accessed in November 9, 2010).

SNORTrules: SourceFire, Inc., Snort Rules, https://www.snort.org/snort-rules/ (accessed in August 29, 2009).

Kohlenberg: Toby Kohlenberg; Brian Caswell; Jay Beale; Andrew R Baker, *Snort: IDS and IPS Toolkit*, ISBN 1-59749-099-7, pp 180-183, 304-305, 2007.

NTOP: NTOP - network top, http://www.ntop.org/overview.html (accessed in November 9, 2010).

Farshchi: Farshchi, Jamil, Intrusion Detection FAQ: Statistical based approach to Intrusion Detection, http://www.sans.org/resources/idfaq/statistic_ids.php (accessed in August 29, 2009).

Norvig: Norvig, Peter. Russel, Stuart, *Artificial Intelligence, A Modern Approach*, ISBN 0137903952, pp 116-119, 2002.

Pohlheim: Pohlheim, Hartmut, “Genetic and Evolutionary Algorithms: Principles, Methods and Algorithms.”Genetic and Evolutionary Algorithm Toolbox, http://www.geatbx.com/docu/algindex.html (accessed in August 29, 2009).

Whitley: Whitley, Darrell, A Genetic Algorithm Tutorial, *Statistics and Computing 4,* pp 64-85, 1994.

Li: Li, Wei, Using Genetic Algorithm for Network Detection, ,United States Department of Energy Cyber Security Group 2004 Training Conference , May 24-27, 2004 .

Gong: Ren Hui Gong, Mohammad Zulkernine, Purang Abolmaesumi, “A Software Implementation of a Genetic Algorithm Based Approach to Network Intrusion Detection”, *Proceedings of the Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks* , 2005.

ECLab: Luke, Sean et al., A Java-based Evolutionary Computation (ECJ) and Genetic Programming Research System, http://cs.gmu.edu/~eclab/projects/ecj/ (accessed in August 2009).

DARPA: , MIT Lincoln Laboratory, DARPA datasets, MIT,USA, http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/index.html (accessed in August 29, 2009).

Sun: , Security Audit, http://www.sun.com/software/security/audit/ (accessed in August 29, 2009).

GLIB: GNOME, GLib Reference Manual, http://library.gnome.org/devel/glib/ (accessed in November 16, 2010).